abc

# Array and Array Operations

1. Which of these best describes an array?
a) A data structure that shows a hierarchical behavior
b) Container of objects of similar types
c) Arrays are immutable once initialised
d) Array is not a data structure

**Answer: b**
Explanation: Array contains elements only of the same type.

2. How do you initialize an array in C?
a) int arr[3] = (1,2,3);
b) int arr(3) = {1,2,3};
c) int arr[3] = {1,2,3};
d) int arr(3) = (1,2,3);
**Answer: c**
Explanation: This is the syntax to initialize an array in C.

3. How do you instantiate an array in Java?
a) int arr[] = new int(3);
b) int arr[];
c) int arr[] = new int[3];
d) int arr() = new int(3);

**Answer: c**
Explanation: Note that int arr[]; is declaration whereas int arr[] = new int[3]; is to instantiate an array.

4. Which of the following is the correct way to declare a multidimensional array in Java?
a) int[] arr;
b) int arr[[]];
c) int[][]arr;
d) int[[]] arr;
**Answer: c**
Explanation: The syntax to declare multidimensional array in java is either int[][] arr; or int arr[][];

7. When does the ArrayIndexOutOfBoundsException occur?
a) Compile-time
b) Run-time
c) Not an error
d) Not an exception at all

**Answer: b**
Explanation: ArrayIndexOutOfBoundsException is a run-time exception and the compilation is error-free.

8. Which of the following concepts make extensive use of arrays?
a) Binary trees
b) Scheduling of processes
c) Caching
d) Spatial locality

abc

**Answer: d**
Explanation: Whenever a particular memory location is referred to, it is likely that the locations nearby are also referred, arrays are stored as contiguous blocks in memory, so if you want to access array elements, spatial locality makes it to access quickly
.
9. What are the advantages of arrays?
a) Objects of mixed data types can be stored
b) Elements in an array cannot be sorted
c) Index of first element of an array is 1
d) Easier to store elements of same data type
**Answer: d**
Explanation: Arrays store elements of the same data type and present in continuous memory locations.

10. What are the disadvantages of arrays?
a) Data structure like queue or stack cannot be implemented
b) There are chances of wastage of memory space if elements inserted in an array are lesser than the allocated size
c) Index value of an array can be negative
d) Elements are sequentially accessed
**Answer: b**
Explanation: Arrays are of fixed size. If we insert elements less than the allocated size, unoccupied positions can't be used again. Wastage will occur in memory.

11. Assuming int is of 4bytes, what is the size of int arr[15];?
a) 15
b) 19
c) 11
d) 60
**Answer: d**
Explanation: Since there are 15 int elements and each int is of 4bytes, we get 15*4 = 60bytes.

12. In general, the index of the first element in an array is _____
a) 0
b) -1
c) 2
d) 1
**Answer: a**
Explanation: In general, Array Indexing starts from 0. Thus, the index of the first element in an array is 0.

13. Elements in an array are accessed _____
a) randomly
b) sequentially
c) exponentially
d) logarithmically

**Answer: a**

abc

Explanation: Elements in an array are accessed randomly. In Linked lists, elements are accessed sequentially.

# Stack Operations – 1

1. Process of inserting an element in stack is called _____
   a) Create
   b) Push
   c) Evaluation
   d) Pop
   **Answer: b**
   Explanation: Push operation allows users to insert elements in the stack. If the stack is filled completely and trying to perform push operation stack – overflow can happen.

2. Process of removing an element from stack is called _____
   a) Create
   b) Push
   c) Evaluation
   d) Pop
   **Answer: d**
   Explanation: Elements in the stack are removed using pop operation. Pop operation removes the top most element in the stack i.e. last entered element.

3. In a stack, if a user tries to remove an element from an empty stack it is called _____
   a) Underflow
   b) Empty collection
   c) Overflow
   d) Garbage Collection
   **Answer: a**
   Explanation: Underflow occurs when the user performs a pop operation on an empty stack. Overflow occurs when the stack is full and the user performs a push operation. Garbage Collection is used to recover the memory occupied by objects that are no longer used.

4. Pushing an element into stack already having five elements and stack size of 5, then stack becomes _____
   a) Overflow
   b) Crash
   c) Underflow
   d) User flow
   **Answer: a**
   Explanation: The stack is filled with 5 elements and pushing one more element causes a stack overflow. This results in overwriting memory, code and loss of unsaved work on the computer.

5. Entries in a stack are “ordered”. What is the meaning of this statement?
   a) A collection of stacks is sortable
   b) Stack entries may be compared with the ‘<‘ operation
   c) The entries are stored in a linked list

abc

d) There is a Sequential entry that is one by one
**Answer: d**
Explanation: In stack data structure, elements are added one by one using push operation. Stack follows LIFO Principle i.e. Last In First Out(LIFO).

6. Which of the following is not the application of stack?
a) A parentheses balancing program
b) Tracking of local variables at run time
c) Compiler Syntax Analyzer
d) Data Transfer between two asynchronous process
**Answer: d**
Explanation: Data transfer between the two asynchronous process uses the queue data structure for synchronisation. The rest are all stack applications.

7. Consider the usual algorithm for determining whether a sequence of parentheses is balanced. The maximum number of parentheses that appear on the stack AT ANY ONE TIME when the algorithm analyzes: (()(())(())) ?
a) 1
b) 2
c) 3
d) 4 or more
**Answer: c**
Explanation: In the entire parenthesis balancing method when the incoming token is a left parenthesis it is pushed into stack. A right parenthesis makes pop operation to delete the elements in stack till we get left parenthesis as top most element. 3 elements are there in stack before right parentheses comes. Therefore, maximum number of elements in stack at run time is 3.

8. Consider the usual algorithm for determining whether a sequence of parentheses is balanced. Suppose that you run the algorithm on a sequence that contains 2 left parentheses and 3 right parentheses (in some order). The maximum number of parentheses that appear on the stack AT ANY ONE TIME during the computation?
a) 1
b) 2
c) 3
d) 4 or more
**Answer: b**
Explanation: In the entire parenthesis balancing method when the incoming token is a left parenthesis it is pushed into stack. A right parenthesis makes pop operation to delete the elements in stack till we get left parenthesis as top most element. 2 left parenthesis are pushed whereas one right parenthesis

abc

removes one of left parenthesis. 2 elements are there before right parenthesis which is the maximum number of elements in stack at run time.

9. What is the value of the postfix expression 6 3 2 4 + – *?
   a) 1
   b) 40
   c) 74
   d) -18
   **Answer: d**
   Explanation: Postfix Expression is (6*(3-(2+4))) which results -18 as output.

10. Here is an infix expression: 4 + 3*(6*3-12). Suppose that we are using the usual stack algorithm to convert the expression from infix to postfix notation. The maximum number of symbols that will appear on the stack AT ONE TIME during the conversion of this expression?
    a) 1
    b) 2
    c) 3
    d) 4
    **Answer: d**
    Explanation: When we perform the conversion from infix to postfix expression +, *, (, * symbols are placed inside the stack. A maximum of 4 symbols are identified during the entire conversion.



# Stack Operations – 2

1. The postfix form of the expression (A+ B)*(C*D- E)*F / G is?
a) AB+ CD*E – FG /**
b) AB + CD* E – F **G /
c) AB + CD* E – *F *G /
d) AB + CDE * – * F *G /
**Answer: c**
Explanation: (((A+ B)*(C*D- E)*F) / G) is converted to postfix expression as
(AB+(*(C*D- E)*F )/ G)
(AB+CD*E-*F) / G
(AB+CD*E-*F * G/). Thus Postfix expression is AB+CD*E-*F*G/

2. The data structure required to check whether an expression contains a balanced parenthesis is?
a) Stack
b) Queue
c) Array
d) Tree

abc

**Answer: a**
Explanation: The stack is a simple data structure in which elements are added and removed based on the LIFO principle. Open parenthesis is pushed into the stack and a closed parenthesis pops out elements till the top element of the stack is its corresponding open parenthesis. If the stack is empty, parenthesis is balanced otherwise it is unbalanced.

3. What data structure would you mostly likely see in non recursive implementation of a recursive algorithm?
a) Linked List
b) Stack
c) Queue
d) Tree
**Answer: b**
Explanation: In recursive algorithms, the order in which the recursive process comes back is the reverse of the order in which it goes forward during execution. The compiler uses the stack data structure to implement recursion. In the forwarding phase, the values of local variables, parameters and the return address are pushed into the stack at each recursion level. In the backing-out phase, the stacked address is popped and used to execute the rest of the code.

4. The process of accessing data stored in a serial access memory is similar to manipulating data on a _____
a) Heap
b) Binary Tree
c) Array
d) Stack
**Answer: d**
Explanation: In serial access memory data records are stored one after the other in which they are created and are accessed sequentially. In stack data structure, elements are accessed sequentially. Stack data structure resembles the serial access memory.

5. The postfix form of A*B+C/D is?
a) *AB/CD+
b) AB*CD/+
c) A*BC+/D
d) ABCD+/*
**Answer: b**
Explanation: Infix expression is (A*B)+(C/D)
AB*+(C/D)
AB*CD/+. Thus postfix expression is AB*CD/+

6. Which data structure is needed to convert infix notation to postfix notation?
a) Branch

abc

b) Tree
c) Queue
d) Stack
**Answer: d**
Explanation: The Stack data structure is used to convert infix expression to postfix expression. The purpose of stack is to reverse the order of the operators in the expression. It also serves as a storage structure, as no operator can be printed until both of its operands have appeared.

7. The prefix form of A-B/ (C * D ^ E) is?
a) -/*^ACBDE
b) -ABCD*^DE
c) -A/B*C^DE
d) -A/BC*^DE
**Answer: c**
Explanation: Infix Expression is (A-B)/(C*D^E)
(-A/B)(C*D^E)
-A/B*C^DE. Thus prefix expression is -A/B*C^DE.

8. What is the result of the following operation?
**Top (Push (S, X))**
a) X
b) X+S
c) S
d) XS
**Answer: a**
Explanation: The function Push(S,X) pushes the value X in the stack S. Top() function gives the value which entered last. X entered into stack S at last.

9. The prefix form of an infix expression (p + q) – (r * t) is?
a) + pq – *rt
b) – +pqr * t
c) – +pq * rt
d) – + * pqrt
**Answer: c**
Explanation: Given Infix Expression is ((p+q)-(r*t))
(+pq)-(r*t)
(-+pq)(r*t)
-+pq*rt. Thus prefix expression is -+pq*rt.

10. Which data structure is used for implementing recursion?
a) Queue
b) Stack

abc

c) Array
d) List
**Answer: b**
Explanation: Stacks are used for the implementation of Recursion.

# Stack Operations – 3

1. The result of evaluating the postfix expression 5, 4, 6, +, *, 4, 9, 3, /, +, * is?
a) 600
b) 350
c) 650
d) 588
**Answer: b**
Explanation: The postfix expression is evaluated using stack. We will get the infix expression as
(5*(4+6))*(4+9/3). On solving the Infix Expression, we get
(5*(10))*(4+3)
= 50*7
= 350.
2. Convert the following infix expressions into its equivalent postfix expressions.
**(A + B ∧D)/(E – F)+G**
a) (A B D ∧ + E F – / G +)
b) (A B D +∧ E F – / G +)
c) (A B D ∧ + E F/- G +)
d) (A B D E F + ∧ / – G +)
**Answer: a**
Explanation: The given infix expression is (A + B ∧D)/(E – F)+G.
(A B D ^ + ) / (E – F) +G
(A B D ^ + E F – ) + G. ‘/’ is present in stack.
A B D ^ + E F – / G +. Thus Postfix Expression is A B D ^ + E F – / G +.

3. Convert the following Infix expression to Postfix form using a stack.
**x + y * z + (p * q + r) * s**, Follow usual precedence rule and assume that the expression is legal.
a) xyz*+pq*r+s*+
b) xyz*+pq*r+s+*
c) xyz+*pq*r+s*+
d) xyzp+**qr+s*+
**Answer: a**
Explanation: The Infix Expression is x + y * z + (p * q + r) * s.
(x y z ) + (p * q + r) * s. ‘+’, ‘*’ are present in stack.
(x y z * + p q * r) * s. ‘+’ is present in stack.
x y z * + p q * r + s * +. Thus Postfix Expression is x y z * + p q * r + s * +.

abc

4. Which of the following statement(s) about stack data structure is/are NOT correct?
a) Linked List are used for implementing Stacks
b) Top of the Stack always contain the new node
c) Stack is the FIFO data structure
d) Null link is present in the last node at the bottom of the stack
**Answer: c**
Explanation: Stack follows LIFO.

6. Which of the following is not an inherent application of stack?
a) Reversing a string
b) Evaluation of postfix expression
c) Implementation of recursion
d) Job scheduling
**Answer: d**
Explanation: Job Scheduling is not performed using stacks.

7. The type of expression in which operator succeeds its operands is?
a) Infix Expression
b) Prefix Expression
c) Postfix Expression
d) Both Prefix and Postfix Expressions
**Answer: c**
Explanation: The expression in which operator succeeds its operands is called postfix expression. The expression in which operator precedes the operands is called prefix expression. If an operator is present between two operands, then it is called infix expressions.

8. Assume that the operators +,-, X are left associative and ^ is right associative. The order of precedence (from highest to lowest) is ^, X, +, -. The postfix expression for the infix expression a + b X c – d ^ e ^ f is?
a) abc X+ def ^^ –
b) abc X+ de^f^ –
c) ab+c Xd – e ^f^
d) -+aXbc^ ^def
**Answer: b**
Explanation: Given Infix Expression is a + b X c – d ^ e ^ f.
(a b c X +) (d ^ e ^ f). '–' is present in stack.
(a b c X + d e ^ f ^ -). Thus the final expression is (a b c X + d e ^ f ^ -).

9. If the elements "A", "B", "C" and "D" are placed in a stack and are deleted one at a time, what is the order of removal?

abc

a) ABCD
b) DCBA
c) DCAB
d) ABDC
**Answer: b**
Explanation: Stack follows LIFO(Last In First Out). So the removal order of elements are DCBA.

# Queue Operations



1. A linear list of elements in which deletion can be done from one end (front) and insertion can take place only at the other end (rear) is known as _____
a) Queue
b) Stack
c) Tree
d) Linked list

**Answer: a**
Explanation: Linear list of elements in which deletion is done at front side and insertion at rear side is called Queue. In stack we will delete the last entered element first.

2. The data structure required for Breadth First Traversal on a graph is?
a) Stack
b) Array
c) Queue
d) Tree
**Answer: c**
Explanation: In Breadth First Search Traversal, BFS, starting vertex is first taken and adjacent vertices which are unvisited are also taken. Again, the first vertex which was added as an unvisited adjacent vertex list will be considered to add further unvisited vertices of the graph. To get the first unvisited vertex we need to follows First In First Out principle. Queue uses FIFO principle.

3. A queue follows _____
a) FIFO (First In First Out) principle
b) LIFO (Last In First Out) principle
c) Ordered array
d) Linear tree
**Answer: a**
Explanation: Element first added in queue will be deleted first which is FIFO principle.

abc

4. Circular Queue is also known as _____
a) Ring Buffer
b) Square Buffer
c) Rectangle Buffer
d) Curve Buffer
**Answer: a**
Explanation: Circular Queue is also called as Ring Buffer. Circular Queue is a linear data structure in which last position is connected back to the first position to make a circle. It forms a ring structure.

5. If the elements “A”, “B”, “C” and “D” are placed in a queue and are deleted one at a time, in what order will they be removed?
a) ABCD
b) DCBA
c) DCAB
d) ABDC

**Answer: a**
Explanation: Queue follows FIFO approach. i.e. First in First Out Approach. So, the order of removal elements are ABCD.

6. A data structure in which elements can be inserted or deleted at/from both ends but not in the middle is?
a) Queue
b) Circular queue
c) Dequeue
d) Priority queue
**Answer: c**
Explanation: In dequeuer, we can insert or delete elements from both the ends. In queue, we will follow first in first out principle for insertion and deletion of elements. Element with least priority will be deleted in a priority queue.

7. A normal queue, if implemented using an array of size MAX_SIZE, gets full when?
a) Rear = MAX_SIZE – 1
b) Front = (rear + 1)mod MAX_SIZE
c) Front = rear + 1
d) Rear = front
**Answer: a**
Explanation: When Rear = MAX_SIZE – 1, there will be no space left for the elements to be added in queue. Thus queue becomes full.

abc

8. Queues serve major role in _____
a) Simulation of recursion
b) Simulation of arbitrary linked list
c) Simulation of limited resource allocation
d) Simulation of heap sort
**Answer: c**
Explanation: Simulation of recursion uses stack data structure. Simulation of arbitrary linked lists uses linked lists. Simulation of resource allocation uses queue as first entered data needs to be given first priority during resource allocation. Simulation of heap sort uses heap data structure.

9. Which of the following is not the type of queue?
a) Ordinary queue
b) Single ended queue
c) Circular queue
d) Priority queue
**Answer: b**
Explanation: Queue always has two ends. So, single ended queue is not the type of queue.



# Singly Linked List Operations – 1

1. A linear collection of data elements where the linear node is given by means of pointer is called?
   a) Linked list
   b) Node list
   c) Primitive list
   d) Unordered list
   **Answer: a**
   Explanation: In Linked list each node has its own data and the address of next node. These nodes are linked by using pointers. Node list is an object that consists of a list of all nodes in a document with in a particular selected set of nodes.

2. Consider an implementation of unsorted singly linked list. Suppose it has its representation with a head pointer only. Given the representation, which of the following operation can be implemented in O(1) time?

```
i) Insertion at the front of the linked list
ii) Insertion at the end of the linked list
iii) Deletion of the front node of the linked list
iv) Deletion of the last node of the linked list
```

a) I and II
b) I and III
c) I, II and III
d) I, II and IV

abc

**Answer: b**
Explanation: We know the head node in the given linked list. Insertion and deletion of elements at the front of the linked list completes in O (1) time whereas for insertion and deletion at the last node requires to traverse through every node in the linked list. Suppose there are n elements in a linked list, we need to traverse through each node. Hence time complexity becomes O(n).

3. In linked list each node contains a minimum of two fields. One field is data field to store the data second field is?
a) Pointer to character
b) Pointer to integer
c) Pointer to node
d) Node

**Answer: c**
Explanation: Each node in a linked list contains data and a pointer (reference) to the next node. Second field contains pointer to node.

4. What would be the asymptotic time complexity to add a node at the end of singly linked list, if the pointer is initially pointing to the head of the list?
a) O(1)
b) O(n)
c) θ(n)
d) θ(1)
**Answer: c**
Explanation: In case of a linked list having n elements, we need to travel through every node of the list to add the element at the end of the list. Thus asymptotic time complexity is θ(n).

5. What would be the asymptotic time complexity to insert an element at the front of the linked list (head is known)?
a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(n^3)$

**Answer: a**
Explanation: To add an element at the front of the linked list, we will create a new node which holds the data to be added to the linked list and pointer which points to head position in the linked list. The entire thing happens within O (1) time. Thus the asymptotic time complexity is O (1).

6. What would be the asymptotic time complexity to find an element in the linked list?
a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(n^4)$

**Answer: b**

abc

Explanation: If the required element is in the last position, we need to traverse the entire linked list. This will take O (n) time to search the element.

7. What would be the asymptotic time complexity to insert an element at the second position in the linked list?
a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(n^3)$
**Answer: a**
Explanation: A new node is created with the required element. The pointer of the new node points the node to which the head node of the linked list is also pointing. The head node pointer is changed and it points to the new node which we created earlier. The entire process completes in O (1) time. Thus the asymptotic time complexity to insert an element in the second position of the linked list is O (1).

8. The concatenation of two lists can be performed in O(1) time. Which of the following variation of the linked list can be used?
a) Singly linked list
b) Doubly linked list
c) Circular doubly linked list
d) Array implementation of list
**Answer: c**
Explanation: We can easily concatenate two lists in O (1) time using singly or doubly linked list, provided that we have a pointer to the last node at least one of the lists. But in case of circular doubly linked lists, we will break the link in both the lists and hook them together. Thus circular doubly linked list concatenates two lists in O (1) time.

9. Consider the following definition in c programming language.

```
struct node
{
    int data;
    struct node * next;
}
typedef struct node NODE;
NODE *ptr;
```

Which of the following c code is used to create new node?
a) ptr = (NODE*)malloc(sizeof(NODE));
b) ptr = (NODE*)malloc(NODE);
c) ptr = (NODE*)malloc(sizeof(NODE*));
d) ptr = (NODE)malloc(sizeof(NODE));
**Answer: a**
Explanation: As it represents the right way to create a node

abc

# Singly Linked List Operations – 2

1. What kind of linked list is best to answer questions like “What is the item at position n?”
   a) Singly linked list
   b) Doubly linked list
   c) Circular linked list
   d) Array implementation of linked list
   **Answer: d**
   Explanation: Arrays provide random access to elements by providing the index value within square brackets. In the linked list, we need to traverse through each element until we reach the nth position. Time taken to access an element represented in arrays is less than the singly, doubly and circular linked lists. Thus, array implementation is used to access the item at the position n.

2. Linked lists are not suitable for the implementation of _____
   a) Insertion sort
   b) Radix sort
   c) Polynomial manipulation
   d) Binary search
   **Answer: d**
   Explanation: It cannot be implemented using linked lists.

3. Linked list is considered as an example of _____ type of memory allocation.
   a) Dynamic
   b) Static
   c) Compile time
   d) Heap
   **Answer: a**
   Explanation: As memory is allocated at the run time.

4. In Linked List implementation, a node carries information regarding _____
   a) Data
   b) Link
   c) Data and Link
   d) Node
   **Answer: b**
   Explanation: A linked list is a collection of objects linked together by references from an object to another object. By convention these objects are names as nodes. Linked list consists of nodes where each node contains one or more data fields and a reference(link) to the next node.

5. Linked list data structure offers considerable saving in _____
   a) Computational Time
   b) Space Utilization
   c) Space Utilization and Computational Time

abc

d) Speed Utilization
**Answer: c**
Explanation: Linked lists saves both space and time.

6. Which of the following points is/are not true about Linked List data structure when it is compared with an array?
a) Arrays have better cache locality that can make them better in terms of performance
b) It is easy to insert and delete elements in Linked List
c) Random access is not allowed in a typical implementation of Linked Lists
d) Access of elements in linked list takes less time than compared to arrays
**Answer: d**
Explanation: To access an element in a linked list, we need to traverse every element until we reach the desired element. This will take more time than arrays as arrays provide random access to its elements.

7. What does the following function do for a given Linked List with first node as head?

```
void funl(struct node* head)
{
    if(head == NULL)
    return;
    funl(head->next);
    printf("%d  ", head->data);
}
```

a)Prints all nodes of linked lists
b) Prints all nodes of linked list in reverse order
c) Prints alternate nodes of Linked List
d) Prints alternate nodes in reverse order
**Answer: b**
Explanation: fun1() prints the given Linked List in reverse manner.
For Linked List 1->2->3->4->5, fun1() prints 5->4->3->2->1.

8. Which of the following sorting algorithms can be used to sort a random linked list with minimum time complexity?
a) Insertion Sort
b) Quick Sort
c) Heap Sort
d) Merge Sort
**Answer: d**
Explanation: Both Merge sort and Insertion sort can be used for linked lists.

abc

The slow random-access performance of a linked list makes other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible. Since worst case time complexity of Merge Sort is O(nLogn) and Insertion sort is $O(n^2)$, merge sort is preferred.

# Singly Linked List Operations – 3

1. The following function reverse() is supposed to reverse a singly linked list. There is one line missing at the end of the function.

```
/* Link list node */
struct node
{
    int data;
    struct node* next;
};

/* head ref is a double pointer which points to head (or start) pointer
  of linked list */
static void reverse(struct node** head_ref)
{
    struct node* prev   = NULL;
    struct node* current = *head_ref;
    struct node* next;
    while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    /*ADD A STATEMENT HERE*/
}
```

What should be added in place of "/*ADD A STATEMENT HERE*/", so that the function correctly reverses a linked list.
a) *head_ref = prev;
b) *head_ref = current;
c) *head_ref = next;
d) *head_ref = NULL;
**Answer: a**
Explanation: *head_ref = prev; At the end of while loop, the prev pointer points to the last node of original linked list.
We need to change *head_ref so that the head pointer now starts pointing to the last node.



2. What is the output of following function for start pointing to first node of following linked list?

abc

```
1->2->3->4->5->6
void fun(struct node* start)
{
    if(start == NULL)
    return;
    printf("%d  ", start->data);
    if(start->next != NULL )
    fun(start->next->next);
    printf("%d  ", start->data);
}
```

a) 1 4 6 6 4 1
b) 1 3 5 1 3 5
c) 1 2 3 5
d) 1 3 5 5 3 1
**Answer: d**
Explanation: fun() prints alternate nodes of the given Linked List, first from head to end, and then from end to head.
If Linked List has even number of nodes, then skips the last node.

3. The following C function takes a simply-linked list as an input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank. Choose the correct alternative to replace the blank line.

```
typedef struct node
{
    int value;
    struct node *next;
}Node;

Node *move_to_front(Node *head)
{
    Node *p, *q;
    if ((head == NULL) || (head->next == NULL))
    return head;
    q = NULL; p = head;
    while (p->next !=NULL)
    {
        q = p;
        p = p->next;
    }
    ____________________________
    return head;
}
```

a)q = NULL; p->next = head; head = p;
b) q->next = NULL; head = p; p->next = head;
c) head = p; p->next = q; q->next = NULL;
d) q->next = NULL; p->next = head; head = p;
**Answer: d**
Explanation: When while loop completes its execution, node ‘p’ refers to the last node whereas the ‘q’ node refers to the node before ‘p’ in the linked list.

abc

q->next=NULL makes q as the last node. p->next=head places p as the first node. the head must be modified to ‘p’ as ‘p’ is the starting node of the list (head=p). Thus the sequence of steps are q->next=NULL, p->next=head, head=p.

4. The following C function takes a single-linked list of integers as a parameter and rearranges the elements of the list. The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after the function completes execution?

```
struct node
{
    int value;
    struct node *next;
};
void rearrange(struct node *list)
{
    struct node *p, * q;
    int temp;
    if ((!list) || !list->next)
      return;
    p = list;
    q = list->next;
    while(q)
    {
         temp = p->value;
         p->value = q->value;
         q->value = temp;
         p = q->next;
         q = p?p->next:0;
    }
}
```

a) 1, 2, 3, 4, 5, 6, 7
b) 2, 1, 4, 3, 6, 5, 7
c) 1, 3, 2, 5, 4, 7, 6
d) 2, 3, 4, 5, 6, 7, 1

**Answer: b**

Explanation: The function rearrange() exchanges data of every node with its next node. It starts exchanging data from the first node itself.

5. In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is?

a) log 2 n
b) $n/2$
c) log 2 n – 1
d) n

**Answer: d**

Explanation: In the worst case, the element to be searched has to be compared with all elements of the linked list.

6. Given pointer to a node X in a singly linked list. Only one pointer is given, pointer to head node is not given, can we delete the node X from given linked list?

abc

a) Possible if X is not last node
b) Possible if size of linked list is even
c) Possible if size of linked list is odd
d) Possible if X is not first node
**Answer: a**
Explanation: Following are simple steps.

```
struct node *temp  = X->next;
X->data  = temp->data;
X->next  = temp->next;
free(temp);
```

7. You are given pointers to first and last nodes of a singly linked list, which of the following operations are dependent on the length of the linked list?
a) Delete the first element
b) Insert a new element as a first element
c) Delete the last element of the list
d) Add a new element at the end of the list
**Answer: c**
Explanation: Deletion of the first element of the list is done in O (1) time by deleting memory and changing the first pointer.
Insertion of an element as a first element can be done in O (1) time. We will create a node that holds data and points to the head of the given linked list. The head pointer was changed to a newly created node.
Deletion of the last element requires a pointer to the previous node of last, which can only be obtained by traversing the list. This requires the length of the linked list.
Adding a new element at the end of the list can be done in O (1) by changing the pointer of the last node to the newly created node and last is changed to a newly created node.

8. In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is?
a) log2 n
b) ⁿ⁄₂
c) log2 n – 1
d) n
**Answer: d**
Explanation: The worst-case happens if the required element is at last or the element is absent in the list. For this, we need to compare every element in the linked list. If n elements are there, n comparisons will happen in the worst case.



# Singly Linked List

1. Which of the following is not a disadvantage to the usage of array?
   a) Fixed size
   b) There are chances of wastage of memory space if elements inserted in an

abc

array are lesser than the allocated size
c) Insertion based on position
d) Accessing elements at specified positions
**Answer: d**
Explanation: Array elements can be accessed in two steps. First, multiply the size of the data type with the specified position, second, add this value to the base address. Both of these operations can be done in constant time, hence accessing elements at a given index/position is faster.

2. What is the time complexity of inserting at the end in dynamic arrays?
a) O(1)
b) O(n)
c) O(logn)
d) Either O(1) or O(n)
**Answer: d**
Explanation: Depending on whether the array is full or not, the complexity in dynamic array varies. If you try to insert into an array that is not full, then the element is simply stored at the end, this takes O(1) time. If you try to insert into an array which is full, first you will have to allocate an array with double the size of the current array and then copy all the elements into it and finally insert the new element, this takes O(n) time.

3. What is the time complexity to count the number of elements in the linked list?
a) O(1)
b) O(n)
c) O(logn)
d) $O(n^2)$
**Answer: b**
Explanation: To count the number of elements, you have to traverse through the entire list, hence complexity is O(n).

4. Which of the following performs deletion of the last element in the list? Given below is the Node class.

abc

```
class Node
{
        protected Node next;
        protected Object ele;
        Node(Object e,Node n)
        {
                ele = e;
                next = n;
        }
        public void setNext(Node n)
        {
                next = n;
        }
        public void setEle(Object e)
        {
                ele = e;
        }
        public Node getNext()
        {
                return next;
        }
        public Object getEle()
        {
                return ele;
        }
}
class SLL
{
        Node head;
        int size;
        SLL()
        {
                size = 0;
        }
}
```

b)

a)

```
public Node removeLast()
{
        if(size == 0)
        return null;
        Node cur;
        Node temp;
        cur = head;
        while(cur.getNext() != null)
        {
                temp = cur;
                cur = cur.getNext();
        }
        temp.setNext(null);
        size--;
        return cur;
}
```

```
public void removeLast()
{
        if(size == 0)
        return null;
        Node cur;
        Node temp;
        cur = head;
        while(cur != null)
        {
                temp = cur;
                cur = cur.getNext();
        }
        temp.setNext(null);
        return cur;
}
```

abc

```
public void removeLast()
{
        if(size == 0)
            return null;
        Node cur;
        Node temp;
        cur = head;
        while(cur != null)
        {
                cur = cur.getNext();
                temp = cur;
         }
        temp.setNext(null);
        return cur;
}
```

```
public void removeLast()
{
        if(size == 0)
                return null;
        Node cur;
        Node temp;
        cur = head;
        while(cur.getNext() != null)
        {
                cur = cur.getNext();
                temp = cur;
        }
        temp.setNext(null);
        return cur;
}
```

**Answer: a**

Explanation: Since you have to traverse to the end of the list and delete the last node, you need two reference pointers. 'cur' to traverse all the way and find the last node, and 'temp' is a trailing pointer to 'cur'. Once you reach the end of the list, setNext of 'temp' to null, 'cur' is not being pointed to by any node, and hence it is available for garbage collection.

4. What is the functionality of the following code?

```
public void function(Node node)
{
        if(size == 0)
                head = node;
        else
        {
                Node temp,cur;
                for(cur = head; (temp = cur.getNext())!=null; cur = temp);
                cur.setNext(node);
        }
        size++;
}
```

a) Inserting a node at the beginning of the list
b) Deleting a node at the beginning of the list
c) Inserting a node at the end of the list
d) Deleting a node at the end of the list

**Answer: c**

Explanation: The for loop traverses through the list and then inserts a new node as cur.setNext(node);

6. What is the space complexity for deleting a linked list?

abc

a) O(1)
b) O(n)
c) Either O(1) or O(n)
d) O(logn)

**Answer: a**

Explanation: You need a temp variable to keep track of current node, hence the space complexity is O(1).

7. How would you delete a node in the singly linked list? The position to be deleted is given.

a)

```
public void delete(int pos)
{
        if(pos < 0)
        pos = 0;
        if(pos > size)
        pos = size;
        if( size == 0)
        return;
        if(pos == 0)
        head = head.getNext();
        else
        {
            Node temp = head;
            for(int i=1; i<pos; i++)
            {
                temp = temp.getNext();
            }
            temp.setNext(temp.getNext().getNext());
        }
            size--;
}
```

b)

```
public void delete(int pos)
{
        if(pos < 0)
        pos = 0;
        if(pos > size)
        pos = size;
        if( size == 0)
        return;
        if(pos == 0)
        head = head.getNext();
        else
        {
            Node temp = head;
            for(int i=1; i<pos; i++)
            {
                temp = temp.getNext();
            }
            temp.setNext(temp.getNext());
        }
            size--;
}
```

abc

c)

```
public void delete(int pos)
{
        if(pos < 0)
        pos = 0;
        if(pos > size)
        pos = size;
        if( size == 0)
        return;
        if(pos == 0)
        head = head.getNext();
        else
        {
            Node temp = head;
            for(int i=1; i<pos; i++)
            {
                temp = temp.getNext().getNext();
            }
            temp.setNext(temp.getNext().getNext());
        }
            size--;
}
```

d)

```
public void delete(int pos)
{
        if(pos < 0)
        pos = 0;
        if(pos > size)
        pos = size;
        if( size == 0)
        return;
        if(pos == 0)
        head = head.getNext();
        else
        {
            Node temp = head;
            for(int i=0; i<pos; i++)
            {
                temp = temp.getNext();
            }
            temp.setNext(temp.getNext().getNext());
        }
        size--;
}
```



**Answer: a**

Explanation: Loop through the list to get into position one behind the actual position given. temp.setNext(temp.getNext().getNext()) will delete the specified node.

abc

8. Which of these is not an application of a linked list?
   a) To implement file systems
   b) For separate chaining in hash-tables
   c) To implement non-binary trees
   d) Random Access of elements
   **Answer: d**
   Explanation: To implement file system, for separate chaining in hash-tables and to implement non-binary trees linked lists are used. Elements are accessed sequentially in linked list. Random access of elements is not an applications of linked list.

9. Which of the following piece of code has the functionality of counting the number of elements in the list?
   a)

```
public int length(Node head)
{
        int size = 0;
        Node cur = head;
        while(cur!=null)
        {
            size++;
            cur = cur.getNext();
        }
        return size;
}
```

b)

```
public int length(Node head)
{
        int size = 0;
        Node cur = head;
        while(cur!=null)
        {
            cur = cur.getNext();
            size++;
        }
        return size;
}
```

c)

abc

```
public int length(Node head)
{
        int size = 0;
        Node cur = head;
        while(cur!=null)
        {
            size++;
            cur = cur.getNext();
        }
}
```

d)

```
public int length(Node head)
{
        int size = 0;
        Node cur = head;
        while(cur!=null)
        {
            size++;
            cur = cur.getNext().getNext();
        }
        return size;
}
```

**Answer: a**
Explanation: 'cur' pointer traverses through list and increments the size variable until the end of list is reached.

10. How do you insert an element at the beginning of the list?
    a)

```
public void insertBegin(Node node)
{
        node.setNext(head);
        head = node;
        size++;
}
```

b)

```
public void insertBegin(Node node)
{
        head = node;
        node.setNext(head);
        size++;
}
```

c)

abc

```
public void insertBegin(Node node)
{
        Node temp = head.getNext()
        node.setNext(temp);
        head = node;
        size++;
}
```

d)

```
public void insertBegin(Node node)
{
        Node temp = head.getNext()
        node.setNext(temp);
        node = head;
        size++;
}
```

**Answer: a**
Explanation: Set the 'next' pointer point to the head of the list and then make this new node as the head.

11. What is the functionality of the following piece of code?

```
public int function(int data)
{
        Node temp = head;
        int var = 0;
        while(temp != null)
        {
                if(temp.getData() == data)
                {
                        return var;
                }
                var = var+1;
                temp = temp.getNext();
        }
        return Integer.MIN_VALUE;
}
```

a) Find and delete a given element in the list
b) Find and return the given element in the list
c) Find and return the position of the given element in the list
d) Find and insert a new element in the list

**Answer: c**
Explanation: When temp is equal to data, the position of data is returned.

abc

# Doubly Linked List

1. Which of the following is false about a doubly linked list?
   a) We can navigate in both the directions
   b) It requires more space than a singly linked list
   c) The insertion and deletion of a node take a bit longer
   d) Implementing a doubly linked list is easier than singly linked list
   **Answer: d**
   Explanation: A doubly linked list has two pointers ‘left’ and ‘right’ which enable it to traverse in either direction. Compared to singly liked list which has only a ‘next’ pointer, doubly linked list requires extra space to store this extra pointer. Every insertion and deletion requires manipulation of two pointers, hence it takes a bit longer time. Implementing doubly linked list involves setting both left and right pointers to correct nodes and takes more time than singly linked list.

2. Given the Node class implementation, select one of the following that correctly inserts a node at the tail of the list.

abc

```
public class Node
{
        protected int data;
        protected Node prev;
        protected Node next;
        public Node(int data)
        {
                this.data = data;
                prev = null;
                next = null;
        }
        public Node(int data, Node prev, Node next)
        {
                this.data = data;
                this.prev = prev;
                this.next = next;
        }
        public int getData()
        {
                return data;
        }
        public void setData(int data)
        {
                this.data = data;
        }
        public Node getPrev()
        {
                return prev;
        }
        public void setPrev(Node prev)
        {
                this.prev = prev;
        }
        public Node getNext
        {
                return next;
        }
        public void setNext(Node next)
        {
                this.next = next;
        }
}
public class DLL
{
        protected Node head;
        protected Node tail;
        int length;
        public DLL()
        {
                head = new Node(Integer.MIN_VALUE,null,null);
                tail = new Node(Integer.MIN_VALUE,null,null);
                head.setNext(tail);
                length = 0;
        }
```



a)

abc

```
public void insertRear(int data)
{
        Node node = new Node(data,tail.getPrev(),tail);
        node.getPrev().setNext(node);
        tail.setPrev(node);
        length++;
}
```

b)

```
public void insertRear(int data)
{
        Node node = new Node(data,tail.getPrev(),tail);
        node.getPrev().getPrev().setNext(node);
        tail.setPrev(node);
        length++;
}
```

c)

```
public void insertRear(int data)
{
        Node node = new Node(data,tail.getPrev(),tail);
        node.getPrev().setNext(tail);
        tail.setPrev(node);
        length++;
}
```

d)

```
public void insertRear(int data)
{
        Node node = new Node(data,head,tail);
        node.getPrev().setNext(node);
        tail.setPrev(node);
        length++;
}
```



**Answer: a**

Explanation: First create a new node whose ‘prev’ points to the node pointed to by the ‘prev’ of tail. The ‘next’ of the new node should point to tail. Set the ‘prev’ of tail to point to new node and the ‘prev’ of new node to point to the new node.

abc

3. What is a memory efficient double linked list?
   a) Each node has only one pointer to traverse the list back and forth
   b) The list has breakpoints for faster traversal
   c) An auxiliary singly linked list acts as a helper list to traverse through the doubly linked list
   d) A doubly linked list that uses bitwise AND operator for storing addresses
   **Answer: a**
   Explanation: Memory efficient doubly linked list has only one pointer to traverse the list back and forth. The implementation is based on pointer difference. It uses bitwise XOR operator to store the front and rear pointer addresses. Instead of storing actual memory address, every node store the XOR address of previous and next nodes.

4. Which of the following piece of code removes the node from a given position?
   a)

```
public void remove(int pos)
{
        if(pos<0 || pos>=size)
        {
                System.out.println("Invalid position");
                return;
        }
        else
        {
                if(head == null)
                        return;
                if(pos == 0)
                {
                        head = head.getNext();
                        if(head == null)
                        tail = null;
                }
                else
                {
                        Node temp = head;
                        for(int i=1; i<position; i++)
                        temp = temp.getNext();
                }
                temp.getNext().setPrev(temp.getPrev());
                temp.getPrev().setNext(temp.getNext());
        }
        size--;
}
```



b)

abc

```
public void remove(int pos)
{
        if(pos<0 || pos>=size)
        {
                System.out.println("Invalid position");
                return;
        }
        else
        {
                if(head == null)
                return;
                if(pos == 0)
                {
                        head = head.getNext();
                        if(head == null)
                        tail = null;
                }
                else
                {
                        Node temp = head;
                        for(int i=1; i<position; i++)
                        temp = temp.getNext();
                }
                temp.getNext().setPrev(temp.getNext());
                temp.getPrev().setNext(temp.getPrev());
        }
        size--;
}
```

c)

```
public void remove(int pos)
{
        if(pos<0 || pos>=size)
        {
                System.out.println("Invalid position");
                return;
        }
        else
        {
                if(head == null)
                return;
                if(pos == 0)
                {
                        head = head.getNext();
                        if(head == null)
                        tail = null;
                }
                else
                {
                        Node temp = head;
                        for(int i=1; i<position; i++)
                        temp = temp.getNext().getNext();
                }
                temp.getNext().setPrev(temp.getPrev());
                temp.getPrev().setNext(temp.getNext());
        }
        size--;
}
```

abc

d)

```
public void remove(int pos)
{
        if(pos<0 || pos>=size)
        {
                System.out.println("Invalid position");
                return;
        }
        else
        {
                if(head == null)
                        return;
                if(pos == 0)
                {
                        head = head.getNext();
                        if(head == null)
                        tail = null;
                }
                else
                {
                        Node temp = head;
                        for(int i=1; i<position; i++)
                        temp = temp.getNext().getNext();
                }
                temp.getNext().setPrev(temp.getNext());
                temp.getPrev().setNext(temp.getPrev());
        }
        size--;
}
```

**Answer: a**
Explanation: If the position to be deleted is not the head, advance to the given position and manipulate the previous and next pointers of next and previous nodes respectively.

5. How do you calculate the pointer difference in a memory efficient double linked list?
   a) head xor tail
   b) pointer to previous node xor pointer to next node
   c) pointer to previous node – pointer to next node
   d) pointer to next node – pointer to previous node
   **Answer: b**
   Explanation: The pointer difference is calculated by taking XOR of pointer to previous node and pointer to the next node.

6. What is the worst case time complexity of inserting a node in a doubly linked list?
   a) O(nlogn)
   b) O(logn)
   c) O(n)

abc

d) O(1)

**Answer: c**

Explanation: In the worst case, the position to be inserted maybe at the end of the list, hence you have to traverse through the entire list to get to the correct position, hence O(n).

7. How do you insert a node at the beginning of the list?

a)

```
public class insertFront(int data)
{
  Node node = new Node(data, head, head.getNext());
  node.getNext().setPrev(node);
  head.setNext(node);
  size++;
}
```

b)

```
public class insertFront(int data)
{
        Node node = new Node(data, head, head);
        node.getNext().setPrev(node);
        head.setNext(node);
        size++;
}
```

c)

```
public class insertFront(int data)
{
        Node node = new Node(data, head, head.getNext());
        node.getNext().setPrev(head);
        head.setNext(node);
        size++;
}
```

d)

```
public class insertFront(int data)
{
        Node node = new Node(data, head, head.getNext());
        node.getNext().setPrev(node);
        head.setNext(node.getNext());
        size++;
}
```

abc

**Answer: a**
Explanation: The new node's previous pointer will point to head and next pointer will point to the current next of head.

8. Consider the following doubly linked list: head-1-2-3-4-5-tail. What will be the list after performing the given sequence of operations?

```
Node temp = new Node(6,head,head.getNext());
Node templ = new Node(0,tail.getPrev(),tail);
head.setNext(temp);
temp.getNext().setPrev(temp);
tail.setPrev(templ);
templ.getPrev().setNext(templ);
```

a) head-0-1-2-3-4-5-6-tail
b) head-1-2-3-4-5-6-tail
c) head-6-1-2-3-4-5-0-tail
d) head-0-1-2-3-4-5-tail
**Answer: c**
Explanation: The given sequence of operations performs addition of nodes at the head and tail of the list.

9. What is the functionality of the following piece of code?

```
public int function()
{
        Node temp = tail.getPrev();
        tail.setPrev(temp.getPrev());
        temp.getPrev().setNext(tail);
        size--;
        return temp.getItem();
}
```

a) Return the element at the tail of the list but do not remove it
b) Return the element at the tail of the list and remove it from the list
c) Return the last but one element from the list but do not remove it
d) Return the last but one element at the tail of the list and remove it from the list
**Answer: b**
Explanation: The previous and next pointers of the tail and the last but one element are manipulated, this suggests that the last node is being removed from the list.

10 Consider the following doubly linked list: head-1-2-3-4-5-tail. What will be the list after performing the given sequence of operations?

abc

```
Node temp = new Node(6,head,head.getNext());
head.setNext(temp);
temp.getNext().setPrev(temp);
Node templ = tail.getPrev();
tail.setPrev(templ.getPrev());
templ.getPrev().setNext(tail);
```

a) head-6-1-2-3-4-5-tail
b) head-6-1-2-3-4-tail
c) head-1-2-3-4-5-6-tail
d) head-1-2-3-4-5-tail
**Answer: b**
Explanation: A new node is added to the head of the list and a node is deleted from the tail end of the list.



# Circular Linked List

1. What differentiates a circular linked list from a normal linked list?
   a) You cannot have the 'next' pointer point to null in a circular linked list
   b) It is faster to traverse the circular linked list
   c) You may or may not have the 'next' pointer point to null in a circular linked list
   d) Head node is known in circular linked list
   **Answer: c**
   Explanation: The 'next' pointer points to null only when the list is empty, otherwise it points to the head of the list. Every node in a circular linked list can be a starting point(head).

2. How do you count the number of elements in the circular linked list?
   a)

abc

```
public int length(Node head)
{
        int length = 0;
        if( head == null)
                return 0;
        Node temp = head.getNext();
        while(temp != head)
        {
                temp = temp.getNext();
                length++;
        }
        return length;
}
```

b)

```
public int length(Node head)
{
        int length = 0;
        if( head == null)
                return 0;
        Node temp = head.getNext();
        while(temp != null)
        {
                temp = temp.getNext();
                length++;
        }
        return length;
}
```

c)

```
public int length(Node head)
{
        int length = 0;
        if( head == null)
                return 0;
        Node temp = head.getNext();
        while(temp != head && temp != null)
        {
                temp = head.getNext();
                length++;
        }
        return length;
}
```

d)

```
public int length(Node head)
{
        int length = 0;
        if( head == null)
                return 0;
        Node temp = head.getNext();
        while(temp != head && temp == null)
        {
                temp = head.getNext();
                length++;
        }
        return length;
}
```



**Answer: a**

Explanation: If the head is null, it means that the list is empty. Otherwise, traverse the list until the head of the list is reached.

abc

3. What is the functionality of the following piece of code? Select the most appropriate.

```
public void function(int data)
{
        int flag = 0;
        if( head != null)
        {
                Node temp = head.getNext();
                while((temp != head) && (!(temp.getItem() == data)))
                {
                        temp = temp.getNext();
                        flag = 1;
                        break;
                }
        }
        if(flag)
                System.out.println("success");
        else
                System.out.println("fail");
}
```

a) Print success if a particular element is not found
b) Print fail if a particular element is not found
c) Print success if a particular element is equal to 1
d) Print fail if the list is empty
**Answer: b**
Explanation: The function prints fail if the given element is not found. Note that this option is inclusive of option “Print fail if the list is empty”, the list being empty is one of the cases covered.
4. What is the time complexity of searching for an element in a circular linked list?
a) O(n)
b) O(nlogn)
c) O(1)
d) O($n^2$)
**Answer: a**
Explanation: In the worst case, you have to traverse through the entire list of n elements.

5. Which of the following application makes use of a circular linked list?
a) Undo operation in a text editor
b) Recursive function calls
c) Allocating CPU to resources
d) Implement Hash Tables
**Answer: c**
Explanation: Generally, round robin fashion is employed to allocate CPU time to resources which makes use of the circular linked list data structure. Recursive function calls use stack data structure. Undo Operation in text editor uses doubly linked lists. Hash tables uses singly linked lists.

abc

5. Choose the code snippet which inserts a node to the head of the list?

a)

```
public void insertHead(int data)
{
        Node temp = new Node(data);
        Node cur = head;
        while(cur.getNext() != head)
                cur = cur.getNext()
        if(head == null)
        {
                head = temp;
                head.setNext(head);
        }
        else
        {
                temp.setNext(head);
                head = temp;
                cur.setNext(temp);
        }
        size++;
}
```

b)

```
public void insertHead(int data)
{
        Node temp = new Node(data);
        while(cur != head)
                cur = cur.getNext()
        if(head == null)
        {
                head = temp;
                head.setNext(head);
        }
        else
        {
                temp.setNext(head.getNext());
                cur.setNext(temp);
        }
        size++;
}
```

c)

```
public void insertHead(int data)
{
        Node temp = new Node(data);
        if(head == null)
        {
                head = temp;
                head.setNext(head);
        }
        else
        {
                temp.setNext(head.getNext());
                head = temp;
        }
        size++;
}
```

d)

abc

```
public void insertHead(int data)
{
        Node temp = new Node(data);
        if(head == null)
        {
                head = temp;
                head.setNext(head.getNext());
        }
        else
        {
                temp.setNext(head.getNext());
                head = temp;
        }
        size++;
}
```

**Answer: a**

Explanation: If the list is empty make the new node as ‘head’, otherwise traverse the list to the end and make its ‘next’ pointer point to the new node, set the new node’s next point to the current head and make the new node as the head.

6. What is the functionality of the following code? Choose the most appropriate answer.

```
public int function()
{
        if(head == null)
                return Integer.MIN_VALUE;
        int var;
        Node temp = head;
        while(temp.getNext() != head)
                temp = temp.getNext();
        if(temp == head)
        {
                var = head.getItem();
                head = null;
                return var;
        }
        temp.setNext(head.getNext());
        var = head.getItem();
        head = head.getNext();
        return var;
}
```

a) Return data from the end of the list
b) Returns the data and deletes the node at the end of the list
c) Returns the data from the beginning of the list
d) Returns the data and deletes the node from the beginning of the list

**Answer: d**

Explanation: First traverse through the list to find the end node, then manipulate the ‘next’ pointer such that it points to the current head’s next node, return the data stored in head and make this next node as the head.

7. What is the functionality of the following code? Choose the most appropriate answer.

abc

```
public int function()
{
        if(head == null)
                return Integer.MIN_VALUE;
        int var;
        Node temp = head;
        Node cur;
        while(temp.getNext() != head)
        {
                cur = temp;
                temp = temp.getNext();
        }
        if(temp == head)
        {
                var = head.getItem();
                head = null;
                return var;
        }
        var = temp.getItem();
        cur.setNext(head);
        return var;
}
```

a) Return data from the end of the list
b) Returns the data and deletes the node at the end of the list
c) Returns the data from the beginning of the list
d) Returns the data and deletes the node from the beginning of the list

**Answer: b**
Explanation: First traverse through the list to find the end node, also have a trailing pointer to find the penultimate node, make this trailing pointer's 'next' point to the head and return the data stored in the 'temp' node.

9. Which of the following is false about a circular linked list?
a) Every node has a successor
b) Time complexity of inserting a new node at the head of the list is O(1)
c) Time complexity for deleting the last node is O(n)
d) We can traverse the whole circular linked list by starting from any point
**Answer: b**
Explanation: Time complexity of inserting a new node at the head of the list is O(n) because you have to traverse through the list to find the tail node.

10. Consider a small circular linked list. How to detect the presence of cycles in this list effectively?
a) Keep one node as head and traverse another temp node till the end to check if its 'next points to head
b) Have fast and slow pointers with the fast pointer advancing two nodes at a time and slow pointer advancing by one node at a time
c) Cannot determine, you have to pre-define if the list contains cycles
d) Circular linked list itself represents a cycle. So no new cycles cannot be generated
**Answer: b**

abc

Explanation: Advance the pointers in such a way that the fast pointer advances two nodes at a time and slow pointer advances one node at a time and check to see if at any given instant of time if the fast pointer points to slow pointer or if the fast pointer's 'next' points to the slow pointer. This is applicable for smaller lists.

# Stack using Array



1. Which of the following real world scenarios would you associate with a stack data structure?
a) piling up of chairs one above the other
b) people standing in a line to be serviced at a counter
c) offer services based on the priority of the customer
d) tatkal Ticket Booking in IRCTC
**Answer: a**
Explanation: Stack follows Last In First Out (LIFO) policy. Piling up of chairs one above the other is based on LIFO, people standing in a line is a queue and if the service is based on priority, then it can be associated with a priority queue. Tatkal Ticket Booking Follows First in First Out Policy. People who click the book now first will enter the booking page first.

2. What does the following function check for? (all necessary headers to be included and function is called from main)

```
#define MAX 10

typedef struct stack
{
    int top;
    int item[MAX];
}stack;

int function(stack *s)
{
    if(s->top == -1)
        return 1;
    else return 0;
}
```

a) full stack
b) invalid index
c) empty stack
d) infinite stack
**Answer: c**
Explanation: An empty stack is represented with the top-of-the-stack('top' in this case) to be equal to -1.

3. What does 'stack underflow' refer to?
a) accessing item from an undefined stack

abc

b) adding items to a full stack
c) removing items from an empty stack
d) index out of bounds exception
**Answer: c**
Explanation: Removing items from an empty stack is termed as stack underflow.

4. What is the output of the following program?

```
public class Stack
{
        protected static final int CAPACITY = 100;
        protected int size,top = -1;
        protected Object stk[];

        public Stack()
        {
                stk = new Object[CAPACITY];
        }

        public void push(Object item)
        {
                if(size_of_stack==size)
                {
                        System.out.println("Stack overflow");
                                return;
                }
                else
                {
                        top++;
                        stk[top]=item;
                }
        }
        public Object pop()
        {
                if(top<0)
                {
                        return -999;
                }
                else
                {
                        Object ele=stk[top];
                        top--;
                        size_of_stack--;
                        return ele;
                }
        }
}

public class StackDemo
{
        public static void main(String args[])
        {
                Stack myStack = new Stack();
                myStack.push(10);
                Object element1 = myStack.pop();
                Object element2 = myStack.pop();
                System.out.println(element2);
        }
}
```

abc

a) stack is full
b) 20
c) 0
d) -999
**Answer: d**
Explanation: The first call to pop() returns 10, whereas the second call to pop() would result in stack underflow and the program returns -999.

5. What is the time complexity of pop() operation when the stack is implemented using an array?
a) O(1)
b) O(n)
c) O(logn)
d) O(nlogn)

**Answer: a**
Explanation: pop() accesses only one end of the structure, and hence constant time.

6. Which of the following array position will be occupied by a new element being pushed for a stack of size N elements(capacity of stack > N)?
a) S[N-1]
b) S[N]
c) S[1]
d) S[0]
**Answer: b**
Explanation: Elements are pushed at the end, hence N.

7. What happens when you pop from an empty stack while implementing using the Stack ADT in Java?
a) Undefined error
b) Compiler displays a warning
c) EmptyStackException is thrown
d) NoStackException is thrown
**Answer: c**
Explanation: The Stack ADT throws an EmptyStackException if the stack is empty and a pop() operation is tried on it.

8. What is the functionality of the following piece of Java code?
   Assume: ‘a’ is a non empty array of integers, the Stack class creates an array of specified size and provides a top pointer indicating TOS(top of stack), push and pop have normal meaning.

abc

```
public void some_function(int[] a)
{
        Stack S=new Stack(a.length);
        int[] b=new int[a.length];
        for(int i=0;i<a.length;i++)
        {
                S.push(a[i]);
        }
        for(int i=0;i<a.length;i++)
        {
                b[i]=(int)(S.pop());
        }
        System.out.println("output :");
        for(int i=0;i<b.length;i++)
        {
                System.out.println(b[i]);
        }
}
```

a) print alternate elements of array
b) duplicate the given array
c) parentheses matching
d) reverse the array
**Answer: d**
Explanation: Every element from the given array 'a' is pushed into the stack, and then the elements are popped out into the array 'b'. Stack is a LIFO structure, this results in reversing the given array.

9. Array implementation of Stack is not dynamic, which of the following statements supports this argument?
a) space allocation for array is fixed and cannot be changed during run-time
b) user unable to give the input for stack operations
c) a runtime exception halts execution
d) improper program compilation
**Answer: a**
Explanation: You cannot modify the size of an array once the memory has been allocated, adding fewer elements than the array size would cause wastage of space, and adding more elements than the array size at run time would cause Stack Overflow.

10. Which of the following array element will return the top-of-the-stack-element for a stack of size N elements(capacity of stack > N)?
a) S[N-1]
b) S[N]
c) S[N-2]
d) S[N+1]
**Answer: a**
Explanation: Array indexing start from 0, hence N-1 is the last index.

abc

# Stack using Linked List

1. What is the best case time complexity of deleting a node in a Singly Linked list?
a) O (n)
b) O ($n^2$)
c) O (nlogn)
d) O (1)
**Answer: d**
Explanation: Deletion of the head node in the linked list is taken as the best case. The successor of the head node is changed to head and deletes the predecessor of the newly assigned head node. This process completes in O(1) time.

2. Which of the following statements are not correct with respect to Singly Linked List(SLL) and Doubly Linked List(DLL)?
a) Complexity of Insertion and Deletion at known position is O(n) in SLL and O(1) in DLL
b) SLL uses lesser memory per node than DLL
c) DLL has more searching power than SLL
d) Number of node fields in SLL is more than DLL
**Answer: d**
Explanation: To insert and delete at known positions requires complete traversal of the list in worst case in SLL, SLL consists of an item and a node field, while DLL has an item and two node fields, hence SLL occupies lesser memory, DLL can be traversed both ways(left and right), while SLL can traverse in only one direction, hence more searching power of DLL. Node fields in SLL is 2 (data and address of next node) whereas in DLL is 3(data, address to next node, address to previous node).

3. Given below is the Node class to perform basic list operations and a Stack class with a no arg constructor.
   Select from the options the appropriate pop() operation that can be included in the Stack class. Also ‘first’ is the top-of-the-stack.

abc

```
class Node
{
        protected Node next;
        protected Object ele;
        Node()
        {
                this(null,null);
        }
        Node(Object e,Node n)
        {
                ele=e;
                next=n;
        }
        public void setNext(Node n)
        {
                next=n;
        }
        public void setEle(Object e)
        {
                ele=e;
        }
        public Node getNext()
        {
                return next;
        }
        public Object getEle()
        {
                return ele;
        }
}

class Stack
{
        Node first;
        int size=0;
        Stack()
        {
                first=null;
        }
}
```

a)

```
public Object pop()
{
        if(size == 0)
        System.out.println("underflow");
        else
        {
                Object o = first.getEle();
                first = first.getNext();
                size--;
                return o;
        }
}
```

b)

abc

```
public Object pop()
{
        if(size == 0)
        System.out.println("underflow");
        else
        {
                Object o = first.getEle();
                first = first.getNext().getNext();
                size--;
                return o;
        }
}
```

c)

```
public Object pop()
{
        if(size == 0)
        System.out.println("underflow");
        else
        {
                first = first.getNext();
                Object o = first.getEle();
                size--;
                return o;
        }
}
```

d)

```
public Object pop()
{
        if(size == 0)
        System.out.println("underflow");
        else
        {
                first = first.getNext().getNext();
                Object o = first.getEle();
                size--;
                return o;
        }
}
```

**answer: a**

Explanation: pop() should return the Object pointed to by the node 'first'. The sequence of operations is, first, get the element stored at node 'first' using getEle(), and second, make the node point to the next node using getNext().

4. What does the following function do?

```
public Object some_func()throws emptyStackException
{
        if(isEmpty())
                throw new emptyStackException("underflow");
        return first.getEle();
}
```

a) pop
b) delete the top-of-the-stack element
c) retrieve the top-of-the-stack element
d) push operation

abc

**Answer: c**
Explanation: This code is only retrieving the top element, note that it is not equivalent to pop operation as you are not setting the 'next' pointer point to the next node in sequence.

5. What is the functionality of the following piece of code?

```
public void display()
{
        if(size == 0)
                System.out.println("underflow");
        else
        {
                Node current = first;
                while(current != null)
                {
                        System.out.println(current.getEle());
                        current = current.getNext();
                }
        }
}
```

a) reverse the list
b) display the list
c) display the list excluding top-of-the-stack-element
d) reverse the list excluding top-of-the-stack-element
**Answer: b**
Explanation: An alias of the node 'first' is created which traverses through the list and displays the elements.

6. What does 'stack overflow' refer to?
a) accessing item from an undefined stack
b) adding items to a full stack
c) removing items from an empty stack
d) index out of bounds exception
**Answer: b**
Explanation: Adding items to a full stack is termed as stack underflow.

6. Given below is the Node class to perform basic list operations and a Stack class with a no arg constructor. Select from the options the appropriate push() operation that can be included in the Stack class. Also 'first' is the top-of-the-stack.

abc

```
class Node
{
        protected Node next;
        protected Object ele;
        Node()
        {
                this(null,null);
        }
        Node(Object e,Node n)
        {
                ele=e;
                next=n;
        }
        public void setNext(Node n)
        {
                next=n;
        }
        public void setEle(Object e)
        {
                ele=e;
        }
        public Node getNext()
        {
                return next;
        }
        public Object getEle()
        {
                return ele;
        }
}

class Stack
{
        Node first;
        int size=0;
        Stack()
        {
                first=null;
        }
}
```

a)

```
public void push(Object item)
{
        Node temp = new Node(item,first);
        first = temp;
        size++;
}
```

b)

```
public void push(Object item)
{
        Node temp = new Node(item,first);
        first = temp.getNext();
        size++;
}
```

c)

abc

```
public void push(Object item)
{
        Node temp = new Node();
        first = temp.getNext();
        first.setItem(item);
        size++;
}
```

d)

```
public void push(Object item)
{
        Node temp = new Node();
        first = temp.getNext.getNext();
        first.setItem(item);
        size++;
}
```

**Answer: a**
Explanation: To push an element into the stack, first create a new node with the next pointer point to the current top-of-the-stack node, then make this node as top-of-the-stack by assigning it to 'first'.

7. Consider these functions:
   push() : push an element into the stack
   pop() : pop the top-of-the-stack element
   top() : returns the item stored in top-of-the-stack-node
   What will be the output after performing these sequence of operations

```
push(20);
push(4);
top();
pop();
pop();
pop();
push(5);
top();
```

a) 20
b) 4
c) stack underflow
d) 5
**Answer: d**
Explanation: 20 and 4 which were pushed are popped by the two pop() statements, the recent push() is 5, hence top() returns 5.

9. Which of the following data structures can be used for parentheses matching?
a) n-ary tree
b) queue
c) priority queue

d) stack
**Answer: d**
Explanation: For every opening brace, push it into the stack, and for every closing brace, pop it off the stack. Do not take action for any other character. In the end, if the stack is empty, then the input has balanced parentheses.

10. Minimum number of queues to implement stack is _____
a) 3
b) 4
c) 1
d) 2
**Answer: c**
Explanation: Use one queue and one counter to count the number of elements in the queue.

# Queue using Array

1. Which of the following properties is associated with a queue?
a) First In Last Out
b) First In First Out
c) Last In First Out
d) Last In Last Out
**Answer: b**
Explanation: Queue follows First In First Out structure.

2. In a circular queue, how do you increment the rear end of the queue?
a) rear++
b) (rear+1) % CAPACITY
c) (rear % CAPACITY)+1
d) rear–
**Answer: b**
Explanation: Ensures rear takes the values from 0 to (CAPACITY-1).

3. What is the term for inserting into a full queue known as?
a) overflow
b) underflow
c) null pointer exception
d) program won’t be compiled
**Answer: a**
Explanation: Just as stack, inserting into a full queue is termed overflow.
4. What is the time complexity of enqueue operation?
a) O(logn)
b) O(nlogn)
c) O(n)

abc

d) O(1)
**Answer: d**
Explanation: Enqueue operation is at the rear end, it takes O(1) time to insert a new item into the queue.

5. What does the following Java code do?

```
public Object function()
{
        if(isEmpty())
        return -999;
        else
        {
                Object high;
                high = q[front];
                return high;
        }
}
```

a) Dequeue
b) Enqueue
c) Return the front element
d) Return the last element
**Answer: c**
Explanation: q[front] gives the element at the front of the queue, since we are not moving the 'front' to the next element,
it is not a dequeue operation.

6. What is the need for a circular queue?
a) effective usage of memory
b) easier computations
c) to delete elements based on priority
d) implement LIFO principle in queues
**Answer: a**
Explanation: In a linear queue, dequeue operation causes the starting elements of the array to be empty, and there is no way you can use that space, while in a circular queue, you can effectively use that space. Priority queue is used to delete the elements based on their priority. Higher priority elements will be deleted first whereas lower priority elements will be deleted next. Queue data structure always follows FIFO principle.



6. Which of the following represents a dequeue operation? (count is the number of elements in the queue)
   a)

abc

```
public Object dequeue()
{
        if(count == 0)
        {
                System.out.println("Queue underflow");
                return 0;
        }
        else
        {
                Object ele = q[front];
                q[front] = null;
                front = (front+1)%CAPACITY;
                count--;
                return ele;
        }
}
```

b)

```
public Object dequeue()
{
        if(count == 0)
        {
                System.out.println("Queue underflow");
                return 0;
        }
        else
        {
                Object ele = q[front];
                front = (front+1)%CAPACITY;
                q[front] = null;
                count--;
                return ele;
        }
}
```

c)

```
public Object dequeue()
{
        if(count == 0)
        {
                System.out.println("Queue underflow");
                return 0;
        }
        else
        {
                front = (front+1)%CAPACITY;
                Object ele = q[front];
                q[front] = null;
                count--;
                return ele;
        }
}
```

d)

abc

```
public Object dequeue()
{
        if(count == 0)
        {
                System.out.println("Queue underflow");
                return 0;
        }
        else
        {
                Object ele = q[front];
                q[front] = null;
                front = (front+1)%CAPACITY;
                return ele;
                count--;
        }
}
```

**Answer: a**

Explanation: Dequeue removes the first element from the queue, 'front' points to the front end of the queue and returns the first element.

7. Which of the following best describes the growth of a linear queue at runtime? (Q is the original queue, size() returns the number of elements in the queue)

a)

```
private void expand()
{
        int length = size();
        int[] newQ = new int[length<<1];
        for(int i=front; i<=rear; i++)
        {
                newQ[i-front] = Q[i%CAPACITY];
        }
        Q = newQ;
        front = 0;
        rear = size()-1;
}
```

b)

```
private void expand()
{
        int length = size();
        int[] newQ = new int[length<<1];
        for(int i=front; i<=rear; i++)
        {
                newQ[i-front] = Q[i%CAPACITY];
        }
        Q = newQ;
}
```

c)

abc

```
private void expand()
{
        int length = size();
        int[] newQ = new int[length<<1];
        for(int i=front; i<=rear; i++)
        {
                newQ[i-front] = Q[i];
        }
        Q = newQ;
        front = 0;
        rear = size()-1;
}
```

d)

```
private void expand()
{
        int length = size();
        int[] newQ = new int[length*2];
        for(int i=front; i<=rear; i++)
        {
                newQ[i-front] = Q[i%CAPACITY];
        }
        Q = newQ;
}
```

**Answer: a**
Explanation: A common technique to expand the size of array at run time is simply to double the size. Create a new array of double the previous size and copy all the elements, after copying do not forget to assign front = 0 and rear = size()-1, as these are necessary to maintain the decorum of the queue operations.

9. What is the space complexity of a linear queue having n elements?
a) O(n)
b) O(nlogn)
c) O(logn)
d) O(1)
**Answer: a**
Explanation: Because there are n elements.

9. What is the output of the following Java code?

abc

```
public class CircularQueue
{
        protected static final int CAPACITY = 100;
        protected int size,front,rear;
        protected Object q[];
        int count = 0;

        public CircularQueue()
        {
                this(CAPACITY);
        }
        public CircularQueue (int n)
        {
                size = n;
                front = 0;
                rear = 0;
                q = new Object[size];
        }


        public void enqueue(Object item)
        {
                if(count == size)
                {
                        System.out.println("Queue overflow");
                                return;
                }
                else
                {
                        q[rear] = item;
                        rear = (rear+1)%size;
                        count++;
                }
        }
        public Object dequeue()
        {
                if(count == 0)
                {
                        System.out.println("Queue underflow");
                        return 0;
                }
                else
                {
                        Object ele = q[front];
                        q[front] = null;
                        front = (front+1)%size;
                        count--;
                        return ele;
                }
        }
        public Object frontElement()
        {
                if(count == 0)
                return -999;
                else
```



a) 3 3
b) 3 6
c) 6 6
d) 10 6

**Answer: a**

Explanation: First enqueue 10 and 3 into the queue, followed by a dequeue(removes

abc

10), followed by an enqueue(6), At this point, 3 is at the front end of the queue and 6 at the rear end, hence a call to frontElement() will return 3 which is displayed twice.

# Queue using Linked List

1. In linked list implementation of queue, if only front pointer is maintained, which of the following operation take worst case linear time?
a) Insertion
b) Deletion
c) To empty a queue
d) Both Insertion and To empty a queue
**Answer: d**
Explanation: Since front pointer is used for deletion, so worst time for the other two cases.

2. In linked list implementation of a queue, where does a new element be inserted?
a) At the head of link list
b) At the centre position in the link list
c) At the tail of the link list
d) At any position in the linked list
**Answer: c**
Explanation: Since queue follows FIFO so new element inserted at last.

3. In linked list implementation of a queue, front and rear pointers are tracked. Which of these pointers will change during an insertion into a NONEMPTY queue?
a) Only front pointer
b) Only rear pointer
c) Both front and rear pointer
d) No pointer will be changed
**Answer: b**
Explanation: Since queue follows FIFO so new element inserted at last.

4. In linked list implementation of a queue, front and rear pointers are tracked. Which of these pointers will change during an insertion into EMPTY queue?
a) Only front pointer
b) Only rear pointer
c) Both front and rear pointer
d) No pointer will be changed
**Answer: c**
Explanation: Since its the starting of queue, so both values are changed.

5. In case of insertion into a linked queue, a node borrowed from the _____ list is inserted in the queue.

abc

a) AVAIL
b) FRONT
c) REAR
d) NULL
**Answer: a**
Explanation: All the nodes are collected in AVAIL list.

6. In linked list implementation of a queue, from where is the item deleted?
a) At the head of link list
b) At the centre position in the link list
c) At the tail of the link list
d) Node before the tail
**Answer: a**
Explanation: Since queue follows FIFO so new element deleted from first.

7. In linked list implementation of a queue, the important condition for a queue to be empty is?
a) FRONT is null
b) REAR is null
c) LINK is empty
d) FRONT==REAR-1
**Answer: a**
Explanation: Because front represents the deleted nodes.

8. The essential condition which is checked before insertion in a linked queue is?
a) Underflow
b) Overflow
c) Front value
d) Rear value
**Answer: b**
Explanation: To check whether there is space in the queue or not.

9. The essential condition which is checked before deletion in a linked queue is?
a) Underflow
b) Overflow
c) Front value
d) Rear value
**Answer: a**
Explanation: To check whether there is element in the list or not.

10. Which of the following is true about linked list implementation of queue?
a) In push operation, if new nodes are inserted at the beginning of linked list, then in pop operation, nodes must be removed from end
b) In push operation, if new nodes are inserted at the beginning, then in pop

abc

operation, nodes must be removed from the beginning
c) In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from end
d) In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from beginning
**Answer: a**
Explanation: It can be done by both the methods.

# Priority Queue

1. With what data structure can a priority queue be implemented?
a) Array
b) List
c) Heap
d) Tree

**Answer: d**
Explanation: Priority queue can be implemented using an array, a list, a binary search tree or a heap, although the most efficient one being the heap.

2. Which of the following is not an application of priority queue?
a) Huffman codes
b) Interrupt handling in operating system
c) Undo operation in text editors
d) Bayesian spam filter
**Answer: c**
Explanation: Undo operation is achieved using a stack.
3. Select the appropriate code that inserts elements into the list based on the given key value.
(head and trail are dummy nodes to mark the end and beginning of the list, they do not contain any priority or element)
a)

abc

```
public void insert_key(int key,Object item)
{
        if(key<0)
        {
                Systerm.our.println("invalid");
                System.exit(0);
        }
        else
        {
                Node temp = new Node(key,item,null);
                if(count == 0)
                {
                        head.setNext(temp);
                        temp.setNext(trail);
                }
                else
                {
                        Node dup = head.getNext();
                        Node cur = head;
                        while((key>dup.getKey()) && (dup!=trail))
                        {
                                dup = dup.getNext();
                                cur = cur.getNext();
                        }
                        cur.setNext(temp);
                        temp.setNext(dup);
                }
                count++;
        }
}
```

b)

```
public void insert_key(int key,Object item)
{
        if(key<0)
        {
                Systerm.our.println("invalid");
                System.exit(0);
        }
        else
        {
                Node temp = new Node(key,item,null);
                if(count == 0)
                {
                        head.setNext(temp);
                        temp.setNext(trail);
                }
                else
                {
                        Node dup = head.getNext();
                        Node cur = dup;
                        while((key>dup.getKey()) && (dup!=trail))
                        {
                                dup = dup.getNext();
                                cur = cur.getNext();
                        }
                        cur.setNext(temp);
                        temp.setNext(dup);
                }
                count++;
        }
}
```

abc

c)

```
public void insert_key(int key,Object item)
{
        if(key<0)
        {
                Systerm.our.println("invalid");
                System.exit(0);
        }
        else
        {
                Node temp = new Node(key,item,null);
                if(count == 0)
                {
                        head.setNext(temp);
                        temp.setNext(trail);
                }
                else
                {
                        Node dup = head.getNext();
                        Node cur = head;
                        while((key>dup.getKey()) && (dup!=trail))
                        {
                                dup = dup.getNext();
                                cur = cur.getNext();
                        }
                        cur.setNext(dup);
                        temp.setNext(cur);
                }
                count++;
        }
}
```

d)

abc

```
public void insert_key(int key,Object item)
{
        if(key<0)
        {
                Systerm.our.println("invalid");
                System.exit(0);
        }
        else
        {
                Node temp = new Node(key,item,null);
                if(count == 0)
                {
                        head.setNext(temp);
                        temp.setNext(trail);
                }
                else
                {
                        Node dup = head.getNext();
                        Node cur = head;
                        while((key>dup.getKey()) && (dup!=trail))
                        {
                                dup = cur
                                cur = cur.getNext();
                        }
                        cur.setNext(dup);
                        temp.setNext(cur);
                }
                count++;
        }
}
```

**Answer: a**

Explanation: Have two temporary pointers ‘dup’ and ‘cur’ with ‘cur’ trailing behind ‘dup’. Traverse through the list until the given key is greater than some element with a lesser key, insert the new node ‘temp’ in that position.

4. What is the time complexity to insert a node based on key in a priority queue?

a) O(nlogn)

b) O(logn)

c) O(n)

d) O($n^2$)

**Answer: c**

Explanation: In the worst case, you might have to traverse the entire list.

5. What is the functionality of the following piece of code?

abc

```
public Object delete_key()
{
        if(count == 0)
        {
                System.out.println("Q is empty");
                System.exit(0);
        }
        else
        {
                Node cur = head.getNext();
                Node dup = cur.getNext();
                Object e = cur.getEle();
                head.setNext(dup);
                count--;
                return e;
        }
}
```

a) Delete the second element in the list
b) Return but not delete the second element in the list
c) Delete the first element in the list
d) Return but not delete the first element in the list
**Answer: c**
Explanation: A pointer is made to point at the first element in the list and one more to point to the second element, pointer manipulations are done such that the first element is no longer being pointed by any other pointer, its value is returned.

6. What is not a disadvantage of priority scheduling in operating systems?
a) A low priority process might have to wait indefinitely for the CPU
b) If the system crashes, the low priority systems may be lost permanently
c) Interrupt handling
d) Indefinite blocking
**Answer: c**
Explanation: The lower priority process should wait until the CPU completes the processing higher priority process. Interrupt handling is an advantage as interrupts should be given more priority than tasks at hand so that interrupt can be serviced to produce desired results.

7. Which of the following is not an advantage of a priority queue?
a) Easy to implement
b) Processes with different priority can be efficiently handled
c) Applications with differing requirements
d) Easy to delete elements in any case
**Answer: d**
Explanation: In worst case, the entire queue has to be searched for the element having the highest priority. This will take more time than usual. So deletion of elements is not an advantage.

abc

8. What is the time complexity to insert a node based on position in a priority queue?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O($n^2$)
**Answer: c**
Explanation: In the worst case, you might have to traverse the entire list.

# Double Ended Queue (Dequeue)

1. What is a dequeue?
a) A queue with insert/delete defined for both front and rear ends of the queue
b) A queue implemented with a doubly linked list
c) A queue implemented with both singly and doubly linked lists
d) A queue with insert/delete defined for front side of the queue
**Answer: a**
Explanation: A dequeue or a double ended queue is a queue with insert/delete defined for both front and rear ends of the queue.

2. Select the function which performs insertion at the front end of the dequeue?
a)

```
public void function(Object item)
{
        Node temp = new Node(item,null);
        if(isEmpty())
        {
                temp.setNext(trail);
                head.setNext(temp);
        }
        else
        {
                Node cur = head.getNext();
                temp.setNext(cur);
                head.setNext(temp);
        }
        size++;
}
```

b)

abc

```
public void function(Object item)
{
        Node temp = new Node(item,null);
        if(isEmpty())
        {
                temp.setNext(trail);
                head.setNext(trail);
        }
        else
        {
                Node cur = head.getNext();
                temp.setNext(cur);
                head.setNext(temp);
        }
        size++;
}
```

c)

```
public void function(Object item)
{
        Node temp = new Node(item,null);
        if(isEmpty())
        {
                Node cur = head.getNext();
                temp.setNext(cur);
                head.setNext(temp);
        }
        else
        {
                temp.setNext(trail);
                head.setNext(temp);
        }
        size++;
}
```

d)

```
public void function(Object item)
{
        Node temp = new Node(item,null);
        if(isEmpty())
        {
                Node cur = head.getNext();
                temp.setNext(cur);
                cur.setNext(temp);
        }
        else
        {
                head.setNext(trail);
                trail.setNext(temp);
        }
        size++;
}
```



**Answer: a**

Explanation: Create a new node, if the current list is empty, the ‘head’ points to this node and this new node points to ‘trail’. Otherwise, ‘head’ points to the new node and this in turn points to the current first element(head.getNext()).

abc

3. What is the functionality of the following piece of code?

```
public void function(Object item)
{
        Node temp=new Node(item,trail);
        if(isEmpty())
        {
                head.setNext(temp);
                temp.setNext(trail);
        }
        else
        {
                Node cur=head.getNext();
                while(cur.getNext()!=trail)
                {
                        cur=cur.getNext();
                }
                cur.setNext(temp);
        }
        size++;
}
```

a) Insert at the front end of the dequeue
b) Insert at the rear end of the dequeue
c) Fetch the element at the rear end of the dequeue
d) Fetch the element at the front end of the dequeue
**Answer: b**
Explanation: If the list is empty, this new node will point to ‘trail’ and will be pointed at by ‘head’. Otherwise, traverse till the end of the list and insert the new node there.

4. What are the applications of dequeue?
a) A-Steal job scheduling algorithm
b) Can be used as both stack and queue
c) To find the maximum of all sub arrays of size k
d) To avoid collision in hash tables

**Answer: d**
Explanation: All of the mentioned can be implemented with a dequeue.

5. Which of the following can be used to delete an element from the front end of the queue?
a)

abc

```
public Object deleteFront() throws emptyDEQException
{
        if(isEmpty())
                throw new emptyDEQException("Empty");
        else
        {
                Node temp = head.getNext();
                Node cur = temp;
                Object e = temp.getEle();
                head.setNext(cur);
                size--;
                return e;
        }
}
```

b)

```
public Object deleteFront() throws emptyDEQException
{
        if(isEmpty())
                throw new emptyDEQException("Empty");
        else
        {
                Node temp = head.getNext();
                Node cur = temp.getNext();
                Object e = temp.getEle();
                head.setNext(cur);
                size--;
                return e;
        }
}
```

c)

```
public Object deleteFront() throws emptyDEQException
{
        if(isEmpty())
                throw new emptyDEQException("Empty");
        else
        {
                Node temp = head.getNext();
                Node cur = temp.getNext();
                Object e = temp.getEle();
                head.setNext(temp);
                size--;
                return e;
        }
}
```

d)

abc

```
public Object deleteFront() throws emptyDEQException
{
        if(isEmpty())
                throw new emptyDEQException("Empty");
        else
        {
                Node temp = head.getNext();
                Node cur = temp.getNext();
                Object e = temp.getEle();
                temp.setNext(cur);
                size--;
                return e;
        }
}
```

**Answer: b**

Explanation: Have two pointers, one(temp) pointing to the first element and the other(cur) pointing to the second element. Make the 'head' point to the second element, this removes all reference for 'temp'.

6. Which of the following can be used to delete an element from the rear end of the queue?

a)

```
public Object deleteRear() throws emptyDEQException
{
        if(isEmpty())
                throw new emptyDEQException("Empty");
        else
        {
                Node temp = head.getNext();
                Node cur = temp;
                while(temp.getNext() != trail)
                {
                        temp = temp.getNext();
                        cur = cur.getNext();
                }
                Object e = temp.getEle();
                cur.setNext(trail);
                size--;
                return e;
        }
}
```

b)

abc

```
public Object deleteRear() throws emptyDEQException
{
        if(isEmpty())
                throw new emptyDEQException("Empty");
        else
        {
                Node temp = head.getNext();
                Node cur = head;
                while(temp != trail)
                {
                        temp = temp.getNext();
                        cur = cur.getNext();
                }
                Object e = temp.getEle();
                cur.setNext(trail);
                size--;
                return e;
        }
}
```

c)

```
public Object deleteRear() throws emptyDEQException
{
        if(isEmpty())
        throw new emptyDEQException("Empty");
        else
        {
                Node temp = head.getNext();
                Node cur = head;
                while(temp.getNext()!=trail)
                {
                        temp = temp.getNext();
                        cur = cur.getNext();
                }
                Object e = temp.getEle();
                cur.setNext(trail);
                size--;
                return e;
        }
}
```

d)

abc

```
public Object deleteRear() throws emptyDEQException
{
        if(isEmpty())
        throw new emptyDEQException("Empty");
        else
        {
                Node temp = head.getNext();
                Node cur = head;
                while(temp.getNext()!=trail)
                {
                        temp = temp.getNext();
                        cur = cur.getNext();
                }
                Object e = temp.getEle();
                temp.setNext(trail);
                size--;
                return e;
        }
}
```

**Answer: c**
Explanation: Traverse till the end of the list with a pointer 'temp' and another 'cur' which is trailing behind temp, make 'cur' point to trail, this removes all reference for 'temp'.

7. What is the time complexity of deleting from the rear end of the dequeue implemented with a singly linked list?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O($n^2$)
**Answer: c**
Explanation: Since a singly linked list is used, first you have to traverse till the end, so the complexity is O(n).

8. After performing these set of operations, what does the final list look contain?

```
InsertFront(10);
InsertFront(20);
InsertRear(30);
DeleteFront();
InsertRear(40);
InsertRear(10);
DeleteRear();
InsertRear(15);
display();
```

a) 10 30 10 15
b) 20 30 40 15
c) 20 30 40 10
d) 10 30 40 15
**Answer: d**
Explanation: A careful tracing of the given operation yields the result.

abc

10
20 10
20 10 30
10 30
10 30 40
10 30 40 10
10 30 40
10 30 40 15

# Queue using Stacks

1. A Double-ended queue supports operations such as adding and removing items from both the sides of the queue. They support four operations like addFront(adding item to top of the queue), addRear(adding item to the bottom of the queue), removeFront(removing item from the top of the queue) and removeRear(removing item from the bottom of the queue). You are given only stacks to implement this data structure. You can implement only push and pop operations. What are the total number of stacks required for this operation?(you can reuse the stack)
a) 1
b) 2
c) 3
d) 4
**Answer: b**
Explanation: The addFront and removeFront operations can be performed using one stack itself as push and pop are supported (adding and removing element from top of the stack) but to perform addRear and removeRear you need to pop each element from the current stack and push it into another stack, push or pop the element as per the asked operation from this stack and in the end pop elements from this stack to the first stack.

2. You are asked to perform a queue operation using a stack. Assume the size of the stack is some value ‘n’ and there are ‘m’ number of variables in this stack. The time complexity of performing deQueue operation is (Using only stack operations like push and pop)(Tightly bound).
a) O(m)
b) O(n)
c) O(m*n)
d) Data is insufficient
**Answer: a**
Explanation: To perform deQueue operation you need to pop each element from the first stack and push it into the second stack. In this case you need to pop ‘m’ times and need to perform push operations also ‘m’ times. Then you pop the first element

from this second stack (constant time) and pass all the elements to the first stack (as done in the beginning)('m-1' times). Therfore the time complexity is O(m).

3. Consider you have an array of some random size. You need to perform dequeue operation. You can perform it using stack operation (push and pop) or using queue operations itself (enQueue and Dequeue). The output is guaranteed to be same. Find some differences?
a) They will have different time complexities
b) The memory used will not be different
c) There are chances that output might be different
d) No differences
**Answer: a**
Explanation: To perform operations such as Dequeue using stack operation you need to empty all the elements from the current stack and push it into the next stack, resulting in a O(number of elements) complexity whereas the time complexity of dequeue operation itself is O(1). And there is a need of a extra stack. Therefore more memory is needed.

4. Consider you have a stack whose elements in it are as follows.
5 4 3 2 << top
Where the top element is 2.
You need to get the following stack
6 5 4 3 2 << top
The operations that needed to be performed are (You can perform only push and pop):
a) Push(pop()), push(6), push(pop())
b) Push(pop()), push(6)
c) Push(pop()), push(pop()), push(6)
d) Push(6)
**Answer: a**
Explanation: By performing push(pop()) on all elements on the current stack to the next stack you get 2 3 4 5 << top.Push(6) and perform push(pop()) you'll get back 6 5 4 3 2 << top. You have actually performed enQueue operation using push and pop.

5. A double-ended queue supports operations like adding and removing items from both the sides of the queue. They support four operations like addFront(adding item to top of the queue), addRear(adding item to the bottom of the queue), removeFront(removing item from the top of the queue) and removeRear(removing item from the bottom of the queue). You are given only stacks to implement this data structure. You can implement only push and pop operations. What's the time complexity of performing addFront and addRear? (Assume 'm' to be the size of the stack and 'n' to be the number of elements)
a) O(m) and O(n)
b) O(1) and O(n)

abc

c) O(n) and O(1)
d) O(n) and O(m)
**Answer: b**
Explanation: addFront is just a normal push operation. Push operation is of O(1). Whereas addRear is of O(n) as it requires two push(pop()) operations of all elements of a stack.

6. Why is implementation of stack operations on queues not feasible for a large dataset (Asssume the number of elements in the stack to be n)?
a) Because of its time complexity O(n)
b) Because of its time complexity O(log(n))
c) Extra memory is not required
d) There are no problems
**Answer: a**
Explanation: To perform Queue operations such as enQueue and deQueue there is a need of emptying all the elements of a current stack and pushing elements into the next stack and vice versa. Therfore it has a time complexity of O(n) and the need of extra stack as well, may not be feasible for a large dataset.

7. Consider yourself to be in a planet where the computational power of chips to be slow. You have an array of size 10.You want to perform enqueue some element into this array. But you can perform only push and pop operations .Push and pop operation both take 1 sec respectively. The total time required to perform enQueue operation is?
a) 20
b) 40
c) 42
d) 43
**Answer: d**
Explanation: First you have to empty all the elements of the current stack into the temporary stack, push the required element and empty the elements of the temporary stack into the original stack. Therfore taking 10+10+1+11+11= 43 seconds.

8. You have two jars, one jar which has 10 rings and the other has none. They are placed one above the other. You want to remove the last ring in the jar. And the second jar is weak and cannot be used to store rings for a long time.
a) Empty the first jar by removing it one by one from the first jar and placing it into the second jar
b) Empty the first jar by removing it one by one from the first jar and placing it into the second jar and empty the second jar by placing all the rings into the first jar one by one
c) There exists no possible way to do this
d) Break the jar and remove the last one
**Answer: b**

Explanation: This is similar to performing dequeue operation using push and pop only. Elements in the first jar are taken out and placed in the second jar. After removing the last element from the first jar, remove all the elements in the second jar and place them in the first jar.

9. Given only a single array of size 10 and no other memory is available. Which of the following operation is not feasible to implement (Given only push and pop operation)?
a) Push
b) Pop
c) Enqueue
d) Returntop
**Answer: c**
Explanation: To perform Enqueue using just push and pop operations, there is a need of another array of same size. But as there is no extra available memeory, the given operation is not feasible.

10. Given an array of size n, let's assume an element is 'touched' if and only if some operation is performed on it(for example, for performing a pop operation the top element is 'touched'). Now you need to perform Dequeue operation. Each element in the array is touched atleast?
a) Once
b) Twice
c) Thrice
d) Four times
**Answer: d**
Explanation: First each element from the first stack is popped, then pushed into the second stack, dequeue operation is done on the top of the stack and later the each element of second stack is popped then pushed into the first stack. Therfore each element is touched four times.



# Stack using Queues

1. To implement a stack using queue(with only enqueue and dequeue operations), how many queues will you need?
a) 1
b) 2
c) 3
d) 4
**Answer: b**
Explanation: Either the push or the pop has to be a costly operation, and the costlier operation requires two queues.

abc

2. Making the push operation costly, select the code snippet which implements the same.(let q1 and q2 be two queues)

a)

```
public void push(int x)
{
        if(empty())
        {
            q1.offer(x);
        }
        else{
                if(q1.size()>0)
                {
                    q2.offer(x);
                    int size = q1.size();
                    while(size>0)
                    {
                        q2.offer(q1.poll());
                        size--;
                    }
                }
            else if(q2.size()>0)
            {
                q1.offer(x);
                int size = q2.size();
                while(size>0)
                {
                    q1.offer(q2.poll());
                    size--;
                }
            }
        }
    }
```

b)

abc

```
public void push(int x)
{
        if(empty())
        {
            q1.offer(x);
        }
        else
        {
            if(q1.size()>0)
            {
                q1.offer(x);
                int size = q1.size();
                while(size>0)
                {
                    q2.offer(q1.poll());
                    size--;
                }
            }
            else if(q2.size()>0)
            {
                q2.offer(x);
                int size = q2.size();
                while(size>0)
                {
                    q1.offer(q2.poll());
                    size--;
                }
            }
        }
}
```

c)

```
public void push(int x)
{
        if(empty())
        {
            q1.offer(x);
        }
        else
        {
            if(q1.size()>0)
            {
                q2.offer(x);
                int size = q1.size();
                while(size>0)
                {
                    q1.offer(q2.poll());
                    size--;
                }
            }
            else if(q2.size()>0)
            {
                q1.offer(x);
                int size = q2.size();
                while(size>0)
                {
                    q2.offer(q1.poll());
                    size--;
                }
            }
        }
}
```

d)

abc

```
public void push(int x)
{
        if(empty())
        {
            q1.offer(x);
        }
        else
        {
            if(q1.size()>0)
            {
                q2.offer(x);
                int size = q1.size();
                while(size>0)
                {
                    q2.offer(q2.poll());
                    size--;
                }
            }
            else if(q2.size()>0)
            {
                q1.offer(x);
                int size = q2.size();
                while(size>0)
                {
                    q2.offer(q1.poll());
                    size--;
                }
            }
        }
}
```

**Answer: a**

Explanation: Stack follows LIFO principle, hence a new item added must be the first one to exit, but queue follows FIFO principle, so when a new item is entered into the queue, it will be at the rear end of the queue. If the queue is initially empty, then just add the new element, otherwise add the new element to the second queue and dequeue all the elements from the second queue and enqueue it to the first one, in this way, the new element added will be always in front of the queue. Since two queues are needed to realize this push operation, it is considered to be costlier.

3. Making the push operation costly, select the code snippet which implements the pop operation.

a)

```
public void pop()
{
        if(q1.size()>0)
        {
            q2.poll();
        }
        else if(q2.size()>0)
        {
            q1.poll();
        }
}
```

abc

b)

```
public void pop()
{
        if(q1.size()>0)
        {
            q1.poll();
        }
        else if(q2.size()>0)
        {
            q2.poll();
        }
}
```

c)

```
public void pop()
{
        q1.poll();
        q2.poll();
}
```

d)

```
public void pop()
{
        if(q2.size()>0)
        {
            q1.poll();
        }
        else
        {
            q2.poll();
        }
}
```

**Answer: b**

Explanation: As the push operation is costly, it is evident that the required item is in the front of the queue, so just dequeue the element from the queue.

4. Select the code snippet which returns the top of the stack.

a)

abc

```
public int top()
{
        if(q1.size()>0)
        {
             return q1.poll();
        }
        else if(q2.size()>0)
        {
             return q2.poll();
        }
        return 0;
}
```

b)

```
  public int top()
  {
         if(q1.size()==0)
         {
              return q1.peek();
         }
         else if(q2.size()==0)
         {
              return q2.peek();
         }
          return 0;
       }
```

c)

```
public int top()
{
        if(q1.size()>0)
        {
             return q1.peek();
        }
        else if(q2.size()>0)
        {
             return q2.peek();
        }
        return 0;
}
```

d)

abc

```
public int top()
{
        if(q1.size()>0)
        {
             return q2.peek();
        }
        else if(q2.size()>0)
        {
             return q1.peek();
        }
        return 0;
}
```

**Answer: c**

Explanation: Assuming its a push costly implementation, the top of the stack will be in the front end of the queue, note that peek() just returns the front element, while poll() removes the front element from the queue.

5. Select the code snippet which return true if the stack is empty, false otherwise.

a)

```
public boolean empty()
{
     return q2.isEmpty();
}
```

b)

```
public boolean empty()
{
     return q1.isEmpty() || q2.isEmpty();
}
```

c)

```
public boolean empty()
{
     return q1.isEmpty();
}
```

d)

```
public boolean empty()
{
     return q1.isEmpty() & q2.isEmpty();
}
```

abc

**Answer: b**
Explanation: If both the queues are empty, then the stack also is empty.

6. Making the pop operation costly, select the code snippet which implements the same.
a)

```
public int pop()
{
        int res=-999,count=0;
        if(q1.size()>0)
        {
                count = q1.size();
                while(count>0)
                        q2.offer(q1.poll());
                res = q1.poll();
        }
        if(q2.size()>0)
        {
                count = q2.size();
                while(count>0)
                        q1.offer(q2.poll());
                res = q2.poll();
        }
        return res;
}
```

b)

```
public int pop()
{
        int res=-999,count=0;
        if(q1.size()>0)
        {
                count = q1.size();
                while(count>1)
                        q2.offer(q1.poll());
                res = q2.poll();
        }
        if(q2.size()>0)
        {
                count = q2.size();
                while(count>1)
                        q1.offer(q2.poll());
                res = q1.poll();
        }
        return res;
}
```

c)

abc

```
public int pop()
{
        int res=-999,count=0;
        if(q1.size()>0)
        {
                count = q1.size();
                while(count>1)
                        q2.offer(q1.poll());
                res = q1.poll();
        }
        if(q2.size()>0)
        {
                count = q2.size();
                while(count>1)
                        q1.offer(q2.poll());
                res = q2.poll();
        }
        return res;
}
```

d)

```
public int pop()
{
        int res=-999,count=0;
        if(q1.size()>0)
        {
                count = q2.size();
                while(count>1)
                        q2.offer(q1.poll());
                res = q1.poll();
        }
        if(q2.size()>0)
        {
                count = q1.size();
                while(count>1)
                        q1.offer(q2.poll());
                res = q2.poll();
        }
        return res;
}
```



**Answer: c**

Explanation: Here the pop operation is costly, hence we need two queues, other than the first element, all the the elements from one queue are dequeued and enqueued to the second queue, hence only one element remains in the first queue which is the item we want, so dequeue it and return the result.

7. What is the functionality of the following piece of code?

```
public void fun(int x)
{
        q1.offer(x);
}
```

abc

a) Perform push() with push as the costlier operation
b) Perform push() with pop as the costlier operation
c) Perform pop() with push as the costlier operation
d) Perform pop() with pop as the costlier operation
**Answer: b**
Explanation: offer() suggests that it is a push operation, but we see that it is performed with only one queue, hence the pop operation is costlier.

# Decimal to Binary using Stacks

1. Express -15 as a 6-bit signed binary number.
a) 001111
b) 101111
c) 101110
d) 001110
**Answer: b**
Explanation: The first 4 1s from the right represent the number 15, 2 more bits are padded to make it 6 digits and the leftmost bit is a 1 to represent that it is -15.

2. Which of the following code snippet is used to convert decimal to binary numbers?
a)

```
public void convertBinary(int num)
{
      int bin[] = new int[50];
      int index = 0;
      while(num > 0)
      {
        bin[index++] = num%2;
        num = num/2;
      }
      for(int i = index-1;i >= 0;i--)
      {
        System.out.print(bin[i]);
      }
}
```

b)

abc

```
public void convertBinary(int num)
{
      int bin[] = new int[50];
      int index = 0;
      while(num > 0)
      {
        bin[++index] = num%2;
        num = num/2;
      }
      for(int i = index-1;i >= 0;i--)
      {
        System.out.print(bin[i]);
      }
}
```

c)

```
public void convertBinary(int num)
{
     int bin[] = new int[50];
     int index = 0;
     while(num > 0)
     {
         bin[index++] = num/2;
         num = num%2;
     }
     for(int i = index-1;i >= 0;i--)
     {
         System.out.print(bin[i]);
     }
}
```

d)

```
public void convertBinary(int num)
 {
     int bin[] = new int[50];
     int index = 0;
     while(num > 0)
     {
         bin[++index] = num/2;
         num = num%2;
     }
     for(int i = index-1;i >= 0;i--)
     {
         System.out.print(bin[i]);
     }
 }
```

**Answer: a**

Explanation: Take the modulus by 2 of the number and store in an array while halving the number during each iteration and then display the contents of the array.

abc

3. Which is the predefined method available in Java to convert decimal to binary numbers?
a) toBinaryInteger(int)
b) toBinaryValue(int)
c) toBinaryNumber(int)
d) toBinaryString(int)
**Answer: d**
Explanation: The method toBinaryString() takes an integer argument and is defined in java.lang package. Usage is java.lang.Integer.toBinaryString(int) this returns the string representation of the unsigned integer value.

4. Using stacks, how to obtain the binary representation of the number?
a)

```
public void convertBinary(int num)
{
    Stack<Integer> stack = new Stack<Integer>();
    while (num != 0)
    {
        int digit = num / 2;
        stack.push(digit);
        num = num % 2;
    }
    System.out.print("\nBinary representation is:");
    while (!(stack.isEmpty() ))
    {
        System.out.print(stack.pop());
    }
}
```

b)

```
public void convertBinary(int num)
{
    Stack<Integer> stack = new Stack<Integer>();
    while (num != 0)
    {
        int digit = num % 2;
        stack.push(digit);
    }
    System.out.print("\nBinary representation is:");
    while (!(stack.isEmpty() ))
    {
        System.out.print(stack.pop());
    }
}
```

c)

abc

```
public void convertBinary(int num)
{
    Stack<Integer> stack = new Stack<Integer>();
    while (num != 0)
    {
        int digit = num % 2;
        stack.push(digit);
        num = num / 2;
    }
    System.out.print("\nBinary representation is:");
    while (!(stack.isEmpty() ))
    {
        System.out.print(stack.pop());
    }
 }
```

d)

```
public void convertBinary(int num)
{
    Stack<Integer> stack = new Stack<Integer>();
    while (num != 0)
    {
        int digit = num % 2;
        stack.push(digit%2);
        num = num / 2;
    }
    System.out.print("\nBinary representation is:");
    while (!(stack.isEmpty() ))
    {
        System.out.print(stack.pop());
    }
}
```

**Answer: c**
Explanation: Here instead of adding the digits to an array, you push it into a stack and while printing, pop it from the stack.

5. What is the time complexity for converting decimal to binary numbers?
a) O(1)
b) O(n)
c) O(logn)
d) O(nlogn)
**Answer: c**
Explanation: Since each time you are halving the number, it can be related to that of a binary search algorithm, hence the complexity is O(logn).

6. Write a piece of code which returns true if the string contains balanced parenthesis, false otherwise.
a)

abc

```
public boolean isBalanced(String exp)
{
        int len = exp.length();
        Stack<Integer> stk = new Stack<Integer>();
        for(int i = 0; i < len; i++)
        {
                char ch = exp.charAt(i);
                if (ch == '(')
                stk.push(i);
                else if (ch == ')')
                {
                        if(stk.peek() == null)
                         {
                                return false;
                        }
                        stk.pop();
                }
        }
        return true;
}
```

b)

```
public boolean isBalanced(String exp)
{
        int len = exp.length();
        Stack<Integer> stk = new Stack<Integer>();
        for(int i = 0; i < len; i++)
             {
                char ch = exp.charAt(i);
                if (ch == '(')
                stk.push(i);
                else if (ch == ')')
                {
                        if(stk.peek() != null)
                         {
                                return true;
                        }
                        stk.pop();
                }
        }
        return false;
  }
```

c)

abc

```
public boolean isBalanced(String exp)
{
        int len = exp.length();
        Stack<Integer> stk = new Stack<Integer>();
        for(int i = 0; i < len; i++)
        {
                char ch = exp.charAt(i);
                if (ch == ')')
                stk.push(i);
                else if (ch == '(')
                {
                        if(stk.peek() == null)
                         {
                                return false;
                        }
                        stk.pop();
                }
        }
        return true;
}
```

d)

```
public boolean isBalanced(String exp)
{
        int len = exp.length();
        Stack<Integer> stk = new Stack<Integer>();
        for(int i = 0; i < len; i++)
        {
                char ch = exp.charAt(i);
                 if (ch == '(')
                 stk.push(i);
                 else if (ch == ')')
                 {
                        if(stk.peek() != null)
                         {
                                return false;
                        }
                        stk.pop();
                }
        }
        return true;
}
```



**Answer: a**

Explanation: Whenever a '(' is encountered, push it into the stack, and when a ')' is encountered check the top of the stack to see if there is a matching '(', if not return false, continue this till the entire string is processed and then return true.

7. What is the time complexity of the following code?

abc

```
public boolean isBalanced(String exp)
{
        int len = exp.length();
        Stack<Integer> stk = new Stack<Integer>();
        for(int i = 0; i < len; i++)
        {
                char ch = exp.charAt(i);
                if (ch == '(')
                stk.push(i);
                else if (ch == ')')
                {
                        if(stk.peek() == null)
                         {
                                return false;
                        }
                        stk.pop();
                }
        }
        return true;
}
```

a) O(logn)
b) O(n)
c) O(1)
d) O(nlogn)
**Answer: b**
Explanation: All the characters in the string have to be processed, hence the complexity is O(n).

8. Which of the following program prints the index of every matching parenthesis?
a)

abc

```
public void dispIndex(String exp)
{
    Stack<Integer> stk = new Stack<Integer>();
    for (int i = 0; i < len; i++)
    {
        char ch = exp.charAt(i);
        if (ch == '(')
        stk.push(i);
        else if (ch == ')')
        {
          try
          {
            int p = stk.pop() + 1;
            System.out.println("')' at index "+(i+1)+" matched with ')' at
index "+p);
          }
          catch(Exception e)
          {
             System.out.println("')' at index "+(i+1)+" is unmatched");
          }
        }
    }
    while (!stk.isEmpty() )
    System.out.println("'(' at index "+(stk.pop() +1)+" is unmatched");
}
```

b)

```
public void dispIndex(String exp)
{
    Stack<Integer> stk = new Stack<Integer>();
    for (int i = 0; i < len; i++)
    {
        char ch = exp.charAt(i);
        if (ch == '(')
        stk.push(i);
        else if (ch == ')')
        {
           try
           {
             int p = stk.pop() + 1;
             System.out.println("')' at index "+(i)+" matched with ')' at
index "+p);
           }
           catch(Exception e)
           {
              System.out.println("')' at index "+(i)+" is unmatched");
           }
        }
    }
    while (!stk.isEmpty() )
    System.out.println("'(' at index "+(stk.pop() +1)+" is unmatched");
}
```

c)

abc

```
public void dispIndex(String exp)
{
    Stack<Integer> stk = new Stack<Integer>();
    for (int i = 0; i < len; i++)
    {
        char ch = exp.charAt(i);
        if (ch == ')')
        stk.push(i);
        else if (ch == '(')
        {
          try
          {
            int p = stk.pop() +1;
            System.out.println("')' at index "+(i+1)+" matched with ')' at
index "+p);
          }
          catch(Exception e)
          {
             System.out.println("')' at index "+(i+1)+" is unmatched");
          }
        }
    }
    while (!stk.isEmpty() )
    System.out.println("'(' at index "+(stk.pop() +1)+" is unmatched");
}
```

d)

```
public void dispIndex(String exp)
{
    Stack<Integer> stk = new Stack<Integer>();
    for (int i = 0; i < len; i++)
    {
        char ch = exp.charAt(i);
        if (ch == ')')
        stk.push(i);
        else if (ch == '(')
        {
          try
          {
            int p = stk.pop();
            System.out.println("')' at index "+(i+1)+" matched with ')' at
index "+p);
          }
          catch(Exception e)
          {
            System.out.println("')' at index "+(i+1)+" is unmatched");
          }
        }
    }
    while (!stk.isEmpty() )
    System.out.println("'(' at index "+(stk.pop() +1)+" is unmatched");
}
```

abc

**Answer: a**
Explanation: Whenever a '(' is encountered, push the index of that character into the stack, so that whenever a corresponding ')' is encountered, you can pop and print it.

# Evaluation of an Infix Expression (Not Parenthesized)

1. How many stacks are required for applying evaluation of infix expression algorithm?
a) one
b) two
c) three
d) four
**Answer: b**
Expression: Two stacks are required for evaluation of infix expression – one for operands and one for operators.

2. How many passes does the evaluation of infix expression algorithm makes through the input?
a) One
b) Two
c) Three
d) Four
**Answer: a**
Explanation: Evaluation of infix expression algorithm is linear and makes only one pass through the input.

3. Identify the infix expression from the list of options given below.
a) a/b+(c-d)
b) abc*+d+ab+cd+*ce-f-
c) ab-c-
d) +ab
**Answer: a**
Explanation: a/b+(c-d) is an infix expression since the operators are placed in between the operands.

4. Which of the following statement is incorrect with respect to evaluation of infix expression algorithm?
a) Operand is pushed on to the stack
b) If the precedence of operator is higher, pop two operands and evaluate
c) If the precedence of operator is lower, pop two operands and evaluate
d) The result is pushed on to the operand stack
**Answer: b**

abc

Explanation: If the precedence of the operator is higher than the stack operator, then it is pushed on to the stack operator.

5. Evaluate the following statement using infix evaluation algorithm and choose the correct answer. 1+2*3-2
a) 3
b) 6
c) 5
d) 4
**Answer: c**
Explanation: According to precedence of operators, * is evaluated first. + and – have equal priorities. Hence, 1+6-2= 5.

6. Evaluation of infix expression is done based on precedence of operators.
a) True
b) False
**Answer: a**
Explanation: During evaluation of infix expression, the operators with higher precedence are evaluated first, followed by operators with lower precedence.

7. Of the following choices, which operator has the lowest precedence?
a) ^
b) +
c) /
d) #
**Answer: d**

Explanation: The operator with the lowest precedence is #, preceded by +, / and then ^.

8. The system throws an error if parentheses are encountered in an infix expression evaluation algorithm.
a) True
b) False
**Answer: b**
Explanation: The algorithm holds good for infix expression with parentheses. The system does not throw error.

9. Evaluate the following and choose the correct answer.
a/b+c*d where a=4, b=2, c=2, d=1.
a) 1
b) 4
c) 5
d) 2
**Answer: b**

abc

Explanation: * and / have higher priority. Hence, they are evaluated first. Then, + is evaluated. Hence, 2+2=4.

10. Evaluate the following statement using infix evaluation algorithm and choose the correct answer. 4*2+3-5/5

a) 10
b) 11
c) 16
d) 12

**Answer: a**

Explanation: 4*2 and 5/5 are evaluated first and then, 8+3-1 is evaluated and the result is obtained as 10.

11. Using the evaluation of infix expression, evaluate a^b+c and choose the correct answer. (a=2, b=2, c=2)

a) 12
b) 8
c) 10
d) 6

**Answer: d**

Explanation: ^ has the highest precedence. Hence, 2^2 is evaluated and then 4+2 gives 6.

12. Evaluate the following infix expression using algorithm and choose the correct answer. a+b*c-d/e^f where a=1, b=2, c=3, d=4, e=2, f=2.

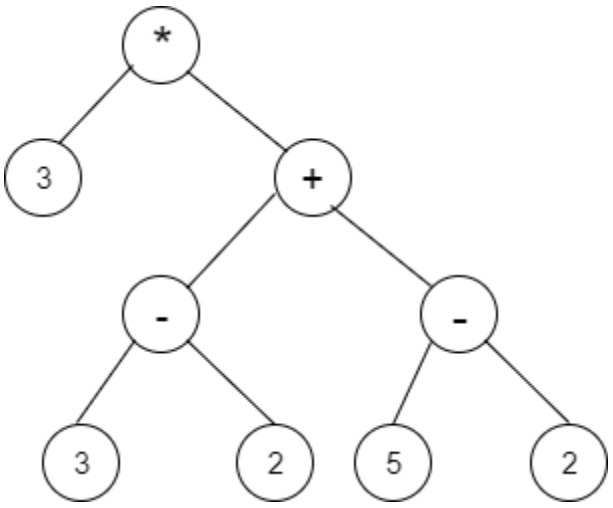
a) 6
b) 8
c) 9
d) 7

**Answer: a**

Explanation: ^ has the highest order of precedence. Hence, 2^2 is evaluated first, and then, 2*3 and 4/4 are evaluated. Therefore, 1+6-1=6.

13. From the given expression tree, identify the infix expression, evaluate it and choose the correct result.

abc




a) 5
b) 10
c) 12
d) 16
**Answer: c**
Explanation: From the given expression tree, the result of the infix expression is evaluated to be 12.

# Evaluation of a Prefix Expression

1. How many stacks are required for evaluation of prefix expression?
a) one
b) two
c) three
d) four
**Answer: b**
Explanation: 2 stacks are required for evaluation of prefix expression, one for integers and one for characters.

2. While evaluating a prefix expression, the string is read from?
a) left to right
b) right to left
c) center to right
d) center to left to right
**Answer: b**
Explanation: The string is read from right to left because a prefix string has operands to its right side.

3. The associativity of an exponentiation operator ^ is right side.
a) True
b) False
**Answer: a**

abc

Explanation: The associativity of ^ is right side while the rest of the operators like +,-,*,/ has its associativity to its left.

4. How many types of input characters are accepted by this algorithm?
a) one
b) two
c) three
d) four
**Answer: c**
Explanation: Three kinds of input are accepted by this algorithm- numbers, operators and new line characters.
5. What determines the order of evaluation of a prefix expression?
a) precedence and associativity
b) precedence only
c) associativity only
d) depends on the parser
**Answer: a**
Explanation: Precedence is a very important factor in determining the order of evaluation. If two operators have the same precedence, associativity comes into action.

6. Find the output of the following prefix expression
*+2-2 1/-4 2+-5 3 1
a) 2
b) 12
c) 10
d) 4
**Answer: a**
Explanation: The given prefix expression is evaluated using two stacks and the value is given by (2+2-1)*(4-2)/(5-3+1)= 2.

7. An error is thrown if the character '\n' is pushed in to the character stack.
a) true
b) false
**Answer: b**
Explanation: The input character '\n' is accepted as a character by the evaluation of prefix expression algorithm.

8. Using the evaluation of prefix algorithm, evaluate +-9 2 7.
a) 10
b) 4
c) 17
d) 14

abc

**Answer: d**
Explanation: Using the evaluation of prefix algorithm, +-9 2 7 is evaluated as 9-2+7=14.

9. If -*+abcd = 11, find a, b, c, d using evaluation of prefix algorithm.
a) a=2, b=3, c=5, d=4
b) a=1, b=2, c=5, d=4
c) a=5, b=4, c=7,d=5
d) a=1, b=2, c=3, d=4

**Answer: b**
Explanation: The given prefix expression is evaluated as ((1+2)*5)-4 =11 while a=1, b=2, c=5, d=4.

10. In the given C snippet, find the statement number that has error.
//C code to push an element into a stack

```
1. void push( struct stack *s, int x)
2. {
3.           if(s->top==MAX-1)
4.          {
5.               printf("stack overflow");
6.          }
7.          else
8.          {
9.                   s->items[++s->top]=x;
10.                  s++;
11.         }
12.}
```

a) 1
b) 9
c) 10
d) 11
**Answer: c**
Explanation: The stack's top position is incremented twice at the same time. So, when the next element is pushed, there is unit gap between this element and the previous element.

abc

# Evaluation of a Postfix Expression

1. What is the other name for a postfix expression?
a) Normal polish Notation
b) Reverse polish Notation
c) Warsaw notation
d) Infix notation
**Answer: b**
Explanation: Reverse polish Notation is the other name for a postfix expression whereas Polish Notation, Warsaw notation are the other names for a prefix expression.

2. Which of the following is an example for a postfix expression?
a) a*b(c+d)
b) abc*+de-+
c) +ab
d) a+b-c
**Answer: b**
Explanation: abc*+de-+ is a postfix expression. +ab is a prefix expression and others are infix expressions.

3. Reverse Polish Notation is the reverse of a Polish Notation
a) True
b) False
**Answer: b**
Explanation: Reverse Polish Notation is not the reverse of a polish notation. Though both NPN and RPN read the expression from left to right, they follow different strategies.

4. What is the time complexity of evaluation of postfix expression algorithm?
a) O (N)
b) O (N log N)
c) O ($N^2$)
d) O (M log N)
**Answer: a**
Explanation: The time complexity of evaluation of infix, prefix and postfix expressions is O (N).

5. In Postfix expressions, the operators come after the operands.
a) True
b) False
**Answer: a**

abc

Explanation: In postfix expressions, the operators follow operands. In prefix expressions,

6. Which of these operators have the highest order of precedence?
a) '(' and ')'
b) '*' and '/'
c) '~' and '^'
d) '+' and '-'
**Answer: c**
Explanation: The highest order of precedence is ~ and ^ followed by '*' ,' /', '+',-' and then braces '(' ')'.

7. Which of the following is not an application of stack?
a) evaluation of postfix expression
b) conversion of infix to postfix expression
c) balancing symbols
d) line at ticket counter
**Answer: d**
Explanation: Line at ticket counter is an application of queue whereas conversion of infix to postfix expression, balancing symbols, line at ticket counter are stack applications.

8. While evaluating a postfix expression, when an operator is encountered, what is the correct operation to be performed?
a) push it directly on to the stack
b) pop 2 operands, evaluate them and push the result on to the stack
c) pop the entire stack
d) ignore the operator
**Answer: b**
Explanation: When an operator is encountered, the first two operands are popped from the stack, they are evaluated and the result is pushed into the stack.

9. Which of the following statement is incorrect?
a) Postfix operators use value to their right
b) Postfix operators use value to their left
c) Prefix operators use value to their right
d) In postfix expression, operands are followed by operators
**Answer: a**
Explanation: All prefix operators use values to their right and all postfix operators use values to their left.

10. What is the result of the given postfix expression? abc*+ where a=1, b=2, c=3.
a) 4
b) 5
c) 6

abc

d) 7
**Answer: d**
Explanation: The infix expression is a+b*c. Evaluating it, we get 1+2*3=7.

11. What is the result of the following postfix expression?
ab*cd*+ where a=2,b=2,c=3,d=4.
a) 16
b) 12
c) 14
d) 10
**Answer: a**
Explanation: The infix expression is a*b+c*d. Evaluating it, we get, 2*2+3*4=16.

12. Consider the stack
| 5 |
| 4 |
| 3 |
| 2 |.
At this point, '*' is encountered. What has to be done?
a) 5*4=20 is pushed into the stack
b) * is pushed into the stack
c) 2*3=6 is pushed into the stack
d) * is ignored
**Answer: a**
Explanation: When an operator is encountered, the first two operands of the stack are popped, evaluated and the result is pushed into the stack.

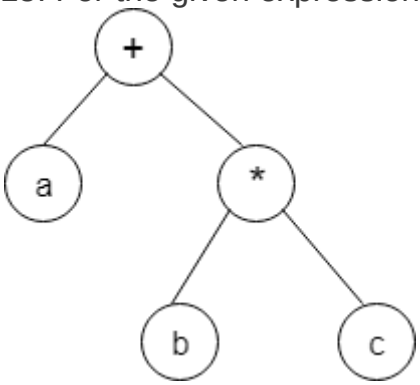
13. Evaluate the postfix expression ab + cd/- where a=5, b=4, c=9, d=3.
a) 23
b) 15
c) 6
d) 10
**Answer: c**
Explanation: The infix expression is (a+b)-c/d. Evaluating it, (5+4)-9/3 gives 6.

14. Evaluate and write the result for the following postfix expression
abc*+de*f+g*+ where a=1, b=2, c=3, d=4, e=5, f=6, g=2.
a) 61
b) 59

abc

c) 60
d) 55
**Answer: b**
Explanation: The infix expression is a+b*c+(d*e+f)*g. Evaluating it, 1+2*3+(4*5+6)*2 gives 59.

15. For the given expression tree, write the correct postfix expression.




a) abc*+
b) abc+*
c) ab+c*
d) a+bc*
**Answer: a**
Explanation: Evaluating the given expression tree gives the infix expression a+b*c. Converting it to postfix, we get, abc*+.



# Infix to Prefix Conversion

1. What data structure is used when converting an infix notation to prefix notation?
a) Stack
b) Queue
c) B-Trees
d) Linked-list
**Answer: a**
Explanation: First you reverse the given equation and carry out the algorithm of infix to postfix expression. Here, the data structure used is stacks.

2. What would be the Prefix notation for the given equation?

```
A+(B*C)
```

a) +A*CB
b) *B+AC
c) +A*BC

abc

d) *A+CB
**Answer: c**
Explanation: Reverse the equation or scan the equation from right to left. Apply the infix-postfix algorithm. The equation inside the bracket evaluates to CB* and outside the bracket evaluates to A+ therefore getting CB*A+. Reversing this and we get +A*BC.

3. What would be the Prefix notation for the given equation?

```
(A*B)+(C*D)
```

a) +*AB*CD
b) *+AB*CD
c) **AB+CD
d) +*BA*CD
**Answer: a**
Explanation: Reverse the equation or scan the equation from right to left. Apply the infix-postfix algorithm. The equation inside the brackets evaluate to DC* and BA* respectively giving us DC*BA*+ in the end. Reversing this we get the +*AB*CD.

4. What would be the Prefix notation for the given equation?

```
A+B*C^D
```

a) +A*B^CD
b) +A^B*CD
c) *A+B^CD
d) ^A*B+CD
**Answer: a**
Explanation: Reverse the equation or scan the equation from right to left. Apply the infix-prefix algorithm. The preference order in ascending order are as follows +*^. Operators are pushed into the stack and popped if its preference is greater than the one which is getting pushed. In the end all operators are popped. The equation evaluates to DC^B*A+. Reversing this we get our following answer.

5. Out of the following operators (^, *, +, &, $), the one having highest priority is _____
a) +
b) $
c) ^
d) &
**Answer: c**
Explanation: According to the algorithm (infix-prefix), it follows that the exponentiation will have the highest priority.

abc

6. Out of the following operators (|, *, +, &, $), the one having lowest priority is _____

a) +
b) $
c) |
d) &

**Answer: c**

Explanation: According to the algorithm (infix-prefix), it follows that the logical OR will have the lowest priority.

7. What would be the Prefix notation for the given equation?

```
A^B^C^D
```

a) ^^^ABCD
b) ^A^B^CD
c) ABCD^^^
d) AB^C^D

**Answer: a**

Explanation: Reverse the equation or scan the equation from right to left. Apply the infix-prefix algorithm. Here we have to remember that the exponentiation has order of associativity from right to left. Therefore the stack goes on pushing ^. Therefore resulting in ^^^ABCD.

8. What would be the Prefix notation for the given equation?

```
a+b-c/d&e|f
```

a) |&-+ab/cdef
b) &|-+ab/cdef
c) |&-ab+/cdef
d) |&-+/abcdef

**Answer: a**

Explanation: Reverse the equation or scan the equation from right to left. Apply the infix-prefix algorithm. The preference order in ascending order are as follows |&+*/.

9. What would be the Prefix notation for the given equation?

```
(a+(b/c)*(d^e)-f)
```

a) -+a*/^bcdef
b) -+a*/bc^def
c) -+a*b/c^def
d) -a+*/bc^def

**Answer: b**

Explanation: Reverse the equation or scan the equation from right to left. Apply the

abc

infix-prefix algorithm. The preference order in ascending order are as follows +*/^. Brackets have the highest priority. The equations inside the brackets are solved first.

10. What would be the Prefix notation and Postfix notation for the given equation?

```
A+B+C
```

a) ++ABC and AB+C+
b) AB+C+ and ++ABC
c) ABC++ and AB+C+
d) ABC+ and ABC+
**Answer: a**
Explanation: For prefix notation there is a need of reversing the giving equation and solving it as a normal infix-postfix question. We see that it doesn't result as same as normal infix-postfix conversion.

11. What would be the Prefix notation for the given equation?

```
a|b&c
```

a) a|&bc
b) &|abc
c) |a&bc
d) ab&|c
**Answer: c**
Explanation: The order of preference of operators is as follows (descending): & |.
The equation **a|b&c** will be parenthesized as **(a|(b&c))** for evaluation.
Therefore the equation for prefix notation evaluates to **|a&bc**.



# Infix to Postfix Conversion Multiple Choice Questions and Answers (MCQs)

1. When an operand is read, which of the following is done?
a) It is placed on to the output
b) It is placed in operator stack
c) It is ignored
d) Operator stack is emptied
**Answer: a**
Explanation: While converting an infix expression to a postfix expression, when an operand is read, it is placed on to the output. When an operator is read, it is placed in the operator stack.

abc

2. What should be done when a left parenthesis ‘(‘ is encountered?
a) It is ignored
b) It is placed in the output
c) It is placed in the operator stack
d) The contents of the operator stack is emptied
**Answer: c**
Explanation: When a left parenthesis is encountered, it is placed on to the operator stack. When the corresponding right parenthesis is encountered, the stack is popped until the left parenthesis and remove both the parenthesis.

3. Which of the following is an infix expression?
a) (a+b)*(c+d)
b) ab+c*
c) +ab
d) abc+*
**Answer: a**
Explanation: (a+b)*(c+d) is an infix expression. +ab is a prefix expression and ab+c* is a postfix expression.

4. What is the time complexity of an infix to postfix conversion algorithm?
a) O(N log N)
b) O(N)
c) $O(N^2)$
d) O(M log N)
**Answer: b**
Explanation: The time complexity of an infix to postfix expression conversion algorithm is mathematically found to be O(N).

5.What is the postfix expression for the corresponding infix expression?

```
a+b*c+(d*e)
```

a) abc*+de*+
b) abc+*de*+
c) a+bc*de+*
d) abc*+(de)*+
View Answer
Answer: a
Explanation: Using the infix to postfix expression conversion algorithm, the corresponding postfix expression is found to be abc*+de*+.

abc

6. Parentheses are simply ignored in the conversion of infix to postfix expression.
a) True
b) False
**Answer: b**
Explanation: When a parenthesis is encountered, it is placed on the operator stack. When the corresponding parenthesis is encountered, the stack is popped until the other parenthesis is reached and they are discarded.

7. It is easier for a computer to process a postfix expression than an infix expression.
a) True
b) False
**Answer: a**
Explanation: Computers can easily process a postfix expression because a postfix expression keeps track of precedence of operators.

8. What is the postfix expression for the infix expression?

```
a-b-c
```

a) -ab-c
b) ab – c –
c) – -abc
d) -ab-c
**Answer: b**
Explanation: The corresponding postfix expression for the given infix expression is found to be ab-c- and not abc--.

9. What is the postfix expression for the following infix expression?

```
a/b^c-d
```

a) abc^/d-
b) ab/cd^-
c) ab/^cd-
d) abcd^/-
**Answer: a**
Explanation: Using the infix to postfix conversion algorithm, the corresponding postfix expression for the infix expression is found to be abc^/d-.

10. Which of the following statement is incorrect with respect to infix to postfix conversion algorithm?

abc

a) operand is always placed in the output
b) operator is placed in the stack when the stack operator has lower precedence
c) parenthesis are included in the output
d) higher and equal priority operators follow the same condition
**Answer: c**
Explanation: Parentheses are not included in the output. They are placed in the operator stack and then discarded.

11. In infix to postfix conversion algorithm, the operators are associated from?
a) right to left
b) left to right
c) centre to left
d) centre to right
**Answer: b**
Explanation: In infix, prefix and postfix expressions, the operators are associated from left to right and not right to left.

12. What is the corresponding postfix expression for the given infix expression?

```
a*(b+c)/d
```

a) ab*+cd/
b) ab+*cd/
c) abc*+/d
d) abc+*d/
**Answer: d**
Explanation: Using the infix to postfix conversion algorithm, the corresponding postfix expression is obtained as abc+*d/.

13. What is the corresponding postfix expression for the given infix expression?

```
a+(b*c(d/e^f)*g)*h)
```

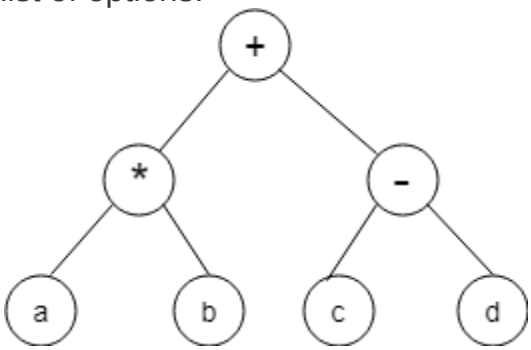
a) ab*cdef/^*g-h+
b) abcdef^/*g*h*+
c) abcd*^ed/g*-h*+
d) abc*de^fg/*-*h+
**Answer: b**
Explanation: Using the infix to postfix expression conversion algorithm using stack, the corresponding postfix expression is found to be abcdef^/*g*h*+.

14. What is the correct postfix expression for the following expression?

abc

```
a+b*(c^d-e)^(f+g*h)-i
```

a) abc^de-fg+*^*+i-
b) abcde^-fg*+*^h*+i-
c) abcd^e-fgh*+^*+i-
d) ab^-dc*+ef^gh*+i-
**Answer: c**
Explanation: The postfix expression for the given infix expression is found to be abcd^e-fgh*+^*+i- when we use infix to postfix conversion algorithm.
15. From the given Expression tree, identify the correct postfix expression from the list of options.




a) ab*cd*+
b) ab*cd-+
c) abcd-*+
d) ab*+cd-
**Answer: b**
Explanation: From the given expression tree, the infix expression is found to be (a*b)+(c-d). Converting it to postfix, we get, ab*cd-+.



# Prefix to Infix Conversion Multiple Choice Questions and Answers (MCQs)

1. What would be the solution to the given prefix notation?

```
- + 5 / 10 5 5
```

a) 2
b) 5
c) 10
d) 7
**Answer: a**
Explanation: The infix notation of the given prefix notation is 5+10/5-5 which gives us 2 as our answer.

abc

2. What would be the solution to the given prefix notation?

```
/ / / / 16 4 2 1
```

a) 1
b) 4
c) 2
d) 8
**Answer: a**
Explanation: The infix notation to the given prefix notation is 16/4/2/1 which gives us 1 as our answer. The infix notation is got from the prefix notation by traversing the equation from the right.

3. What would be the solution to the given prefix notation?

```
+ 9 * 3 / 8 4
```

a) 14
b) 15
c) 18
d) 12
**Answer: b**
Explanation: The infix notation for the given prefix notation is (9+(3*(8/4))) which solves to 15. So 15 is correct answer.
4. What would be the solution to the given prefix notation?

```
- + 1 2 * 3 / 6 2
```

a) 6
b) -6
c) 3
d) -3
View Answer
Answer: b
Explanation: The infix notation for the given prefix notation is (1+2)-3*(6/2). The result of the given equation is -6.

5. What would be the solution to the given prefix notation?

abc

```
- * 1 5 / * / 6 3 6 2
```

a) 1
b) 0
c) -1
d) -2
**Answer: c**
Explanation: The infix notation for the given prefix notation is (1*5)-(6/3)*6/2. The result of the equation is -1.

6. What would be the solution to the given prefix notation?

```
* / + 1 2 / 4 2 + 3 5
```

a) 12
b) 7.5
c) 9
d) 13.5
**Answer: a**
Explanation: The infix notation of the given prefix notation is ((1+2)/(4/2))*(3+5) which solves to (3/2)*8 which by solving gives us 12.

7. Given a prefix and a postfix notation what are the difference between them?
a) The postfix equation is solved starting from the left whereas the prefix notation is solved from the right
b) The postfix equation is solved starting from the right whereas the prefix notation is solved from the left
c) Both equations are solved starting from the same side(right)
d) Both equations are solved starting from the same side(left)
**Answer: a**
Explanation: The postfix notation is solved starting from left but whereas the prefix notation is reversed after creating them, therefore it's solved starting from right.

8. When converting the prefix notation into an infix notation, the first step to be followed is _____
a) Reverse the equation
b) Push the equation to the stack
c) Push the equation onto the queue
d) Push the equation to the stack or queue
**Answer: a**
Explanation: The steps that are followed are: the equation is reversed, pushed onto a stack, popped one by one and solved. Therefore the first step is reversing the equation.

9. The time complexity of converting a prefix notation to infix notation is _____
a) O(n) where n is the length of the equation
b) O(n) where n is number of operands
c) O(1)

abc

d) O(logn) where n is length of the equation
**Answer: a**

Explanation: The processes that are involved are reversing the equation (O(n)), pushing them all onto the stack(O(n)), and popping them one by one and solving them (O(n)). Hence the answer is O(n) where n is the length of the equation.

10. Given two processes (conversion of postfix equation to infix notation and conversion of prefix notation to infix notation), which of the following is easier to implement?
a) Both are easy to implement
b) Conversion of postfix equation to infix equation is harder than converting a prefix notation to infix notation
c) Conversion of postfix equation to infix equation is easier than converting a prefix notation to infix notation
d) Insufficient data
**Answer: c**
Explanation: As the conversion of prefix notation to infix notation involves reversing the equation, the latter is harder to implement than postfix to infix process.

# Postfix to Infix Conversion Multiple Choice Questions and Answers (MCQs)

1. Which of the following data structure is used to convert postfix expression to infix expression?
a) Stack
b) Queue
c) Linked List
d) Heap
**Answer: a**
Explanation: To convert the postfix expression into infix expression we need stack. We need stack to maintain the intermediate infix expressions. We use stack to hold operands.

2. The postfix expression abc+de/*- is equivalent to which of the following infix expression?
a) abc+-de*/
b) (a+b)-d/e*c
c) a-(b+c)*(d/e)
d) abc+*-(d/e)
**Answer: c**
Explanation: Given postfix expression : abc+de/*-
infix ⇒ a(b+c)(d/e)*-
⇒ a(b+c)*(d/e)-
⇒ a-(b+c)*(d/e)
Hence, correct choice is a-(b+c)*(d/e).

3. The equivalent infix expression and value for the postfix form 1 2 + 3 * 4 5 * – will be

_____
a) 1 + 2 * 3 – 4 * 5 and -13
b) (2 + 1) * (3 – 4) * 5 and 13

abc

c) 1 + 2 * (3 – 4) * 5 and -11
**Answer: d**
Explanation: Given postfix expression : 1 2 + 3 * 4 5 * –
⇒ (1 + 2) 3 * 4 5 * –
⇒ ((1 + 2) * 3) 4 5 * –
⇒ ((1 + 2) * 3) (4 * 5) –
⇒ ((1 + 2) * 3) – (4 * 5)
So, the equivalent infix expression is (1 + 2) * 3 – (4 * 5) and it's value is -11.

4. What is the value of the postfix expression 2 3 + 4 5 6 – – *
a) 19
b) 21
c) -4
d) 25
**Answer: d**
Explanation: Given postfix expression : 2 3 + 4 5 6 – – *
infix ⇒ (2 + 3)4 (5 – 6) – *
⇒ (2 + 3)*4 – (5 – 6)
Hence, value = (2 + 3) * (4 – (5 – 6)) = 5 *(4 – (-1)) = 5*5 = 25.

5. The prefix expression of the postfix expression AB+CD-* is _____
a) (A+B)*(C-D)
b) +AB*-CD
c) A+*BCD-
d) *+AB-CD
**Answer: d**
Explanation: To convert from postfix to prefix, we first convert it to infix and then to prefix.
postfix : AB+CD-*
infix ⇒ (A+B) * (C-D)
So, prefix ⇒ +AB*-CD,
⇒ *+AB-CD.
Therefore, correct choice is *+AB-CD.

6. Consider the postfix expression 4 5 6 a b 7 8 a c, where a, b, c are operators. Operator a has higher precedence over operators b and c. Operators b and c are right associative. Then, equivalent infix expression is
a) 4 a 5 6 b 7 8 a c
b) 4 a 5 c 6 b 7 a 8
c) 4 b 5 a 6 c 7 a 8
d) 4 a 5 b 6 c 7 a 8
**Answer: c**
Explanation: Given postfix expression: 4 5 6 a b 7 8 a c
infix ⇒ 4 (5 a 6) b (7 a 8) c
⇒ (4 b (5 a 6)) (7 a 8) c
⇒ (4 b (5 a 6)) c (7 a 8)
So, the required infix expression is 4 b 5 a 6 c 7 a 8.

abc

7. To convert the postfix expression into the infix expression we use stack and scan the postfix expression from left to right.
a) True
b) False
**Answer: a**
Explanation: Stack is used to postfix expression to infix expression. And to convert we follow the following steps: (i) Scan the expression from left to right. (ii) If operand is found, push it on stack.(iii) If operator is found, the two operands are popped and the combined infix expression is formed and pushed onto the stack.

8. Which of the following is valid reverse polish expression?
a) a op b
b) op a b
c) a b op
d) both op a b and a b op
**Answer: c**
Explanation: The postfix expression is also known as the reverse polish expression. In postfix expressions, the operators come after the operands. So, the correct expression is a b op and hence a b op is correct.

9. The result of the postfix expression 5 3 * 9 + 6 / 8 4 / + is _____
a) 8
b) 6
c) 10
d) 9
**Answer: b**
Explanation: Given postfix expression: 5 3 * 9 + 6 / 8 4 / +
Result = 5 3 * 9 + 6 / 8 4 / +
= (5 * 3) 9 + 6 / (8 / 4) +
= ((5 * 3) + 9) / 6 + ( 8 / 4) = ( 24 / 6) + 2 = 4 + 2 = 6.



# Sparse Matrix

1. Which matrix has most of the elements (not all) as Zero?
a) Identity Matrix
b) Unit Matrix
c) Sparse Matrix
d) Zero Matrix
**Answer: c**
Explanation: Sparse Matrix is a matrix in which most of the elements are Zero. Identity Matrix is a matrix in which all principle diagonal elements are 1 and rest of the elements are Zero. Unit Matrix is also called Identity Matrix. Zero Matrix is a matrix in which all the elements are Zero.

abc

2. What is the relation between Sparsity and Density of a matrix?
a) Sparsity = 1 – Density
b) Sparsity = 1 + Density
c) Sparsity = Density*Total number of elements
d) Sparsity = Density/Total number of elements
**Answer: a**
Explanation: Sparsity of a matrix is equal to 1 minus Density of the matrix. The Sparsity of matrix is defined as the total number of Zero Valued elements divided total number of elements.

3. Who coined the term Sparse Matrix?
a) Harry Markowitz
b) James Sylvester
c) Chris Messina
d) Arthur Cayley
**Answer: a**
Explanation: Harry Markowitz coined the term Sparse Matrix. James Sylvester coined the term Matrix. Chris Messina coined the term Hashtag and Arthur Cayley developed the algebraic aspects of a matrix.

4. Is O(n) the Worst case Time Complexity for addition of two Sparse Matrix?
a) True
b) False
**Answer: a**
Explanation: In Addition, the matrix is traversed linearly, hence it has the time complexity of O(n) where n is the number of non-zero elements in the largest matrix amongst two.

5. The matrix contains m rows and n columns. The matrix is called Sparse Matrix if
_____
a) Total number of Zero elements > (m*n)/2
b) Total number of Zero elements = m + n
c) Total number of Zero elements = m/n
d) Total number of Zero elements = m-n
**Answer: a**
Explanation: For matrix to be Sparse Matrix, it should contain Zero elements more than the non-zero elements. Total elements of the given matrix is m*n. So if Total number of Zero elements > (m*n)/2, then the matrix is called Sparse Matrix.



6. Which of the following is not the method to represent Sparse Matrix?
a) Dictionary of Keys
b) Linked List
c) Array
**d) Heap**
**Answer: d**
Explanation: Heap is not used to represent Sparse Matrix while in Dictionary, rows and column numbers are used as Keys and values as Matrix entries, Linked List is used with

each node of Four fields (Row, Column, Value, Next Node) (2D array is used to represent the Sparse Matrix with three fields (Row, Column, Value).

7. Is Sparse Matrix also known as Dense Matrix?
a) True
b) False
**Answer: b**
Explanation: Sparse Matrix is a matrix with most of the elements as Zero elements while Dense Matrix is a matrix with most of the elements as Non-Zero element.

8. Which one of the following is a Special Sparse Matrix?
a) Band Matrix
b) Skew Matrix
c) Null matrix
d) Unit matrix
**Answer: a**
Explanation: A band matrix is a sparse matrix whose non zero elements are bounded to a diagonal band, comprising the main diagonal and zero or more diagonals on either side.

9. In what way the Symmetry Sparse Matrix can be stored efficiently?
a) Heap
b) Binary tree
c) Hash table
d) Adjacency List
**Answer: b**
Explanation: Since Symmetry Sparse Matrix arises as the adjacency matrix of the undirected graph. Hence it can be stored efficiently as an adjacency list.



# Binary Trees using Array

1. How many children does a binary tree have?
a) 2
b) any number of children
c) 0 or 1 or 2
d) 0 or 1

**Answer: c**
Explanation: Can have atmost 2 nodes.

2. What is/are the disadvantages of implementing tree using normal arrays?
a) difficulty in knowing children nodes of a node
b) difficult in finding the parent of a node
c) have to know the maximum number of nodes possible before creation of trees
d) difficult to implement

abc

**Answer: c**
Explanation: The size of array is fixed in normal arrays. We need to know the number of nodes in the tree before array declaration. It is the main disadvantage of using arrays to represent binary trees.

3. What must be the ideal size of array if the height of tree is 'l'?
a) $2^l$-1
b) l-1
c) l
d) 2l

**Answer: a**
Explanation: Maximum elements in a tree (complete binary tree in worst case) of height 'L' is $2^L$-1. Hence size of array is taken as $2^L$-1.

4. What are the children for node 'w' of a complete-binary tree in an array representation?
a) 2w and 2w+1
b) 2+w and 2-w
c) w+1/2 and w/2
d) w-1/2 and w+1/2

**Answer: a**
Explanation: The left child is generally taken as 2*w whereas the right child will be taken as 2*w+1 because root node is present at index 0 in the array and to access every index position in the array.

5. What is the parent for a node 'w' of a complete binary tree in an array representation when w is not 0?
a) floor(w-1/2)
b) ceil(w-1/2)
c) w-1/2
d) w/2
**Answer: a**
Explanation: Floor of w-1/2 because we can't miss a node.



6. If the tree is not a complete binary tree then what changes can be made for easy access of children of a node in the array?
a) every node stores data saying which of its children exist in the array
b) no need of any changes continue with 2w and 2w+1, if node is at i
c) keep a seperate table telling children of a node
d) use another array parallel to the array with tree
**Answer: a**
Explanation: Array cannot represent arbitrary shaped trees. It can only be used in case of complete trees. If every node stores data saying that which of its children exists in the array then elements can be accessed easily.

abc

7. What must be the missing logic in place of missing lines for finding sum of nodes of binary tree in alternate levels?

```
  //e.g:-consider -complete binary tree:-height-3, [1,2,3,4,5,6,7]-answer
must be 23
  n=power(2,height)-1; //assume input is height and a[i] contains tree
elements
  for(i=1;i<=n;)
  {
        for(j=1;j<=pow(2,currentlevel-1);j++) //present level is initialized
to 1 and sum is initialized to  0
        {
           sum=sum+a[i];
           i=i+1;
        }
     //missing logic
  }
```

a)

```
i=i+pow(2,currentlevel);
currentlevel=currentlevel+2;
j=1;
```

b)

```
i=i+pow(2,currentlevel);
currentlevel=currentlevel+2;
j=0;
```

c)

```
i=i-pow(2,currentlevel);
currentlevel=currentlevel+2;
j=1;
```

d)

```
i=i+pow(2,currentlevel);
currentlevel=currentlevel+1;
j=1;
```

**Answer: a**

Explanation: The i value must skip through all nodes in the next level and current level must be one+next level.

abc

8. Consider a situation of writing a binary tree into a file with memory storage efficiency in mind, is array representation of tree is good?
a) yes because we are overcoming the need of pointers and so space efficiency
b) yes because array values are indexable
c) No it is not efficient in case of sparse trees and remaning cases it is fine
d) No linked list representation of tree is only fine
**Answer: c**
Explanation: In case of sparse trees (where one node per level in worst cases), the array size $(2^h)$-1 where h is height but only h indexes will be filled and $(2^h)$-1-h nodes will be left unused leading to space wastage.

9. Why is heap implemented using array representations than tree(linked list) representations though both tree representations and heaps have same complexities?

```
for binary heap
-insert: O(log n)
-delete min: O(log n)

for a tree
-insert: O(log n)
-delete: O(log n)
```

Then why go with array representation when both are having same values ?
a) arrays can store trees which are complete and heaps are not complete
b) lists representation takes more memory hence memory efficiency is less and go with arrays and arrays have better caching
c) lists have better caching
d) In lists insertion and deletion is difficult
**Answer: b**
Explanation: In memory the pointer address for next node may not be adjacent or nearer to each other and also array have wonderful caching power from os and manipulating pointers is a overhead. Heap data structure is always a complete binary tree.



10. Can a tree stored in an array using either one of inorder or post order or pre order traversals be again reformed?
a) Yes just traverse through the array and form the tree
b) No we need one more traversal to form a tree
c) No in case of sparse trees
d) Yes by using both inorder and array elements
**Answer: b**
Explanation: We need any two traversals for tree formation but if some additional stuff or

techniques are used while storing a tree in an array then one traversal can facilitate like also storing null values of a node in array.

# Binary Trees using Linked Lists

1. Advantages of linked list representation of binary trees over arrays?
a) dynamic size
b) ease of insertion/deletion
c) ease in randomly accessing a node
d) both dynamic size and ease in insertion/deletion

**Answer: d**
Explanation: It has both dynamic size and ease in insertion and deletion as advantages.

2. Disadvantages of linked list representation of binary trees over arrays?
a) Randomly accessing is not possible
b) Extra memory for a pointer is needed with every element in the list
c) Difficulty in deletion
d) Random access is not possible and extra memory with every element

**Answer: d**
Explanation: Random access is not possible with linked lists.

3. Which of the following traversing algorithm is not used to traverse in a tree?
a) Post order
b) Pre order
c) Post order
d) Randomized
**Answer: d**
Explanation: Generally, all nodes in a tree are visited by using preorder, inorder and postorder traversing algorithms.

4. Level order traversal of a tree is formed with the help of
a) breadth first search
b) depth first search
c) dijkstra's algorithm
d) prims algorithm
**Answer: a**
Explanation: Level order is similar to bfs.

5. Identify the reason which doesn't play a key role to use threaded binary trees?
a) The storage required by stack and queue is more
b) The pointers in most of nodes of a binary tree are NULL
c) It is Difficult to find a successor node
d) They occupy less size
**Answer: d**
Explanation: Threaded binary trees are introduced to make the Inorder traversal faster without using any stack or recursion. Stack and Queue require more space and pointers in the majority of binary trees are null and difficulties are raised while finding successor nodes.

abc

Size constraints are not taken on threaded binary trees, but they occupy less space than a stack/queue.

6. The following lines talks about deleting a node in a binary tree.(the tree property must not be violated after deletion)
i) from root search for the node to be deleted
ii)
iii) delete the node at
what must be statement ii) and fill up statement iii)
a) ii)-find random node,replace with node to be deleted. iii)- delete the node
b) ii)-find node to be deleted. iii)- delete the node at found location
c) ii)-find deepest node,replace with node to be deleted. iii)- delete a node
d) ii)-find deepest node,replace with node to be deleted. iii)- delete the deepest node
**Answer: d**
Explanation: We just replace a to be deleted node with last leaf node of a tree. this must not be done in case of BST or heaps.

7. What may be the psuedo code for finding the size of a tree?
a) find_size(root_node–>left_node) + 1 + find_size(root_node–>right_node)
b) find_size(root_node–>left_node) + find_size(root_node–>right_node)
c) find_size(root_node–>right_node) – 1
d) find_size(root_node–>left_node + 1
**Answer: a**
Explanation: Draw a tree and analyze the expression. we are always taking size of left subtree and right subtree and adding root value(1) to it and finally printing size.

8. What is missing in this logic of finding a path in the tree for a given sum (i.e checking whether there will be a path from roots to leaf nodes with given sum)?

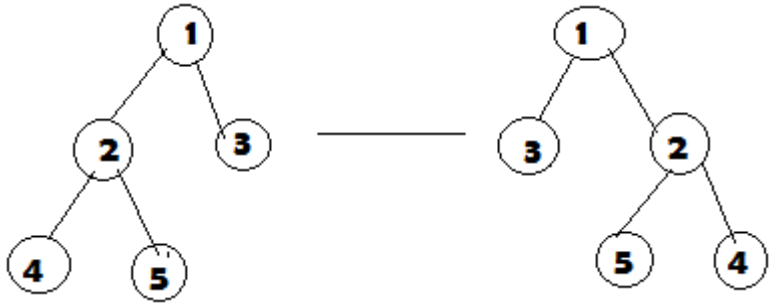
```
checkSum(struct bin-treenode *root , int sum) :
  if(root==null)
    return sum as 0
  else :
     leftover_sum=sum-root_node-->value
     //missing
```

a) code for having recursive calls to either only left tree or right trees or to both subtrees depending on their existence
b) code for having recursive calls to either only left tree or right trees
c) code for having recursive calls to either only left tree
d) code for having recursive calls to either only right trees

**Answer: a**
Explanation: if(left subtree and right subtree) then move to both subtrees
else if only left subtree then move to left subtree carrying leftover_sum parameter
else if only right subtree then move to right subtree carrying leftover_sum parameter.

abc

9. What must be the missing logic below so as to print mirror of a tree as below as an example?




```
if(rootnode):
  mirror(rootnode-->left)
  mirror(rootnode-->right)

  //missing

end
```

a) swapping of left and right nodes is missing
b) swapping of left with root nodes is missing
c) swapping of right with root nodes is missing
d) nothing is missing
**Answer: a**
Explanation: Mirror is another tree with left and right children of nodes are interchanged as shown in the figure.

10. What is the code below trying to print?

```
void print(tree *root,tree *node)
{
  if(root ==null) return 0
  if(root-->left==node || root-->right==node || print(root-
>left,node)||printf(root->right,node)
  {
     print(root->data)
  }
}
```

a) just printing all nodes
b) not a valid logic to do any task
c) printing ancestors of a node passed as argument
d) printing nodes from leaf node to a node passed as argument
**Answer: c**
Explanation: We are checking if left or right node is what the argument sent or else if not the case then move to left node or right node and print all nodes while searching for the argument node.

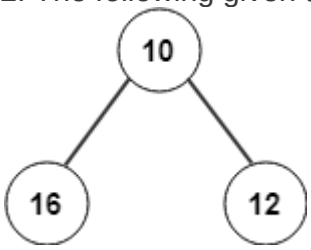
abc

# Binary Tree Operations

1. What is the maximum number of children that a binary tree node can have?
a) 0
b) 1
c) 2
d) 3

**Answer: c**
Explanation: In a binary tree, a node can have atmost 2 nodes (i.e.) 0,1 or 2 left and right child.

2. The following given tree is an example for?




a) Binary tree
b) Binary search tree
c) Fibonacci tree
d) AVL tree
**Answer: a**
Explanation: The given tree is an example for binary tree since has got two children and the left and right children do not satisfy binary search tree's property, Fibonacci and AVL tree.

3. A binary tree is a rooted tree but not an ordered tree.
a) true
b) false
**Answer: b**
Explanation: A binary tree is a rooted tree and also an ordered tree (i.e) every node in a binary tree has at most two children.

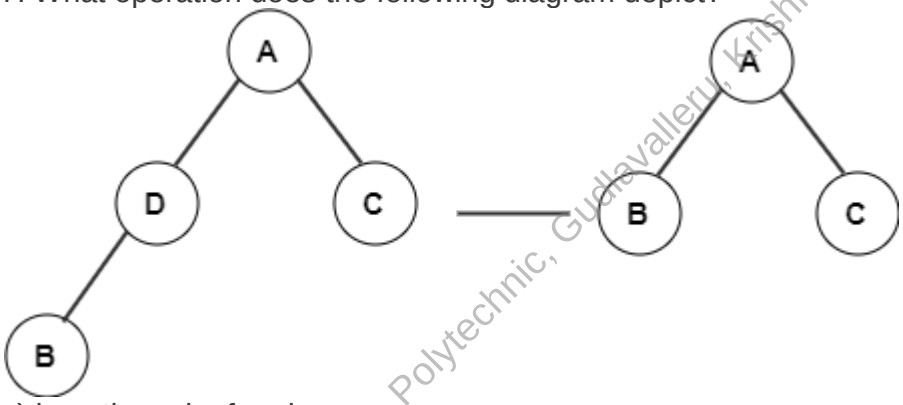
4. How many common operations are performed in a binary tree?
a) 1
b) 2
c) 3
d) 4
**Answer: c**
Explanation: Three common operations are performed in a binary tree- they are insertion, deletion and traversal.

abc

5. What is the traversal strategy used in the binary tree?
a) depth-first traversal
b) breadth-first traversal
c) random traversal
d) Priority traversal
**Answer: b**
Explanation: Breadth first traversal, also known as level order traversal is the traversal strategy used in a binary tree. It involves visiting all the nodes at a given level.

6. How many types of insertion are performed in a binary tree?
a) 1
b) 2
c) 3
d) 4
**Answer: b**
Explanation: Two kinds of insertion operation is performed in a binary tree- inserting a leaf node and inserting an internal node.

7. What operation does the following diagram depict?




a) inserting a leaf node
b) inserting an internal node
c) deleting a node with 0 or 1 child
d) deleting a node with 2 children
**Answer: c**
Explanation: The above diagram is a depiction of deleting a node with 0 or 1 child since the node D which has 1 child is deleted.

8. General ordered tree can be encoded into binary trees.
a) true
b) false
**Answer: a**
Explanation: General ordered tree can be mapped into binary tree by representing them in a left-child-right-sibling way.

abc

9. How many bits would a succinct binary tree occupy?
a) n+O(n)
b) 2n+O(n)
c) n/2
d) n
**Answer: b**
Explanation: A succinct binary tree occupies close to minimum possible space established by lower bounds. A succinct binary tree would occupy 2n+O(n) bits.

10. The average depth of a binary tree is given as?
a) O(N)
b) O(√N)
c) O($N^2$)
d) O(log N)
**Answer: d**
Explanation: The average depth of a binary tree is given as O(√N). In case of a binary search tree, it is O(log N).

11. How many orders of traversal are applicable to a binary tree (In General)?
a) 1
b) 4
c) 2
d) 3

**Answer: d**
Explanation: The three orders of traversal that can be applied to a binary tree are in-order, pre-order and post order traversal.

12. If binary trees are represented in arrays, what formula can be used to locate a left child, if the node has an index i?
a) 2i+1
b) 2i+2
c) 2i
d) 4i
**Answer: a**
Explanation: If binary trees are represented in arrays, left children are located at indices 2i+1 and right children at 2i+2.

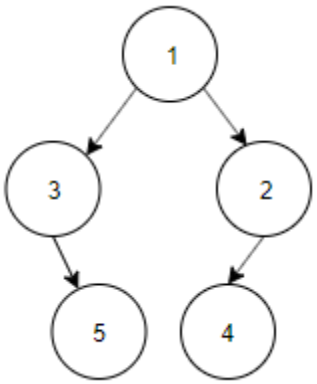
13. Using what formula can a parent node be located in an array?
a) (i+1)/2
b) (i-1)/2
c) i/2
d) 2i/2
**Answer: b**
Explanation: If a binary tree is represented in an array, parent nodes are found at indices (i-1)/2.

abc

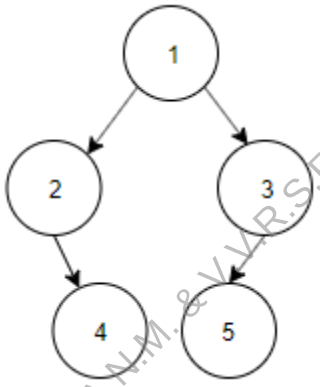
14. Which of the following properties are obeyed by all three tree – traversals?
a) Left subtrees are visited before right subtrees
b) Right subtrees are visited before left subtrees
c) Root node is visited before left subtree
d) Root node is visited before right subtree
**Answer: a**
Explanation: In preorder, inorder and postorder traversal the left subtrees are visited before the right subtrees. In Inorder traversal, the Left subtree is visited first then the Root node then the Right subtree. In postorder traversal, the Left subtree is visited first, then Right subtree and then the Root node is visited.

15. Construct a binary tree using the following data.
The preorder traversal of a binary tree is 1, 2, 5, 3, 4. The inorder traversal of the same binary tree is 2, 5, 1, 4, 3.

a)

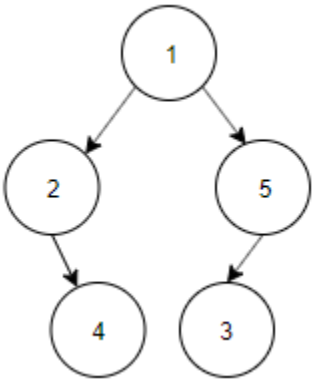



b)

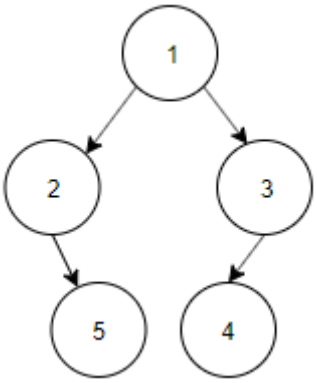


abc

c)

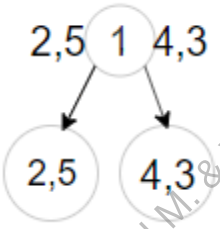



d)




**Answer: d**

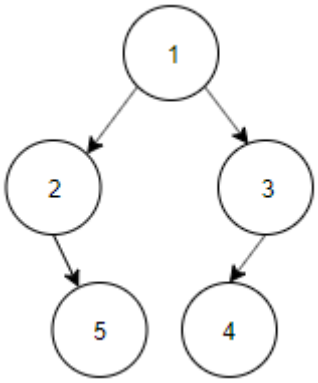
Explanation: Here,
Preorder Traversal is 1, 2, 5, 3, 4
Inorder Traversal is 2, 5, 1, 4, 3
Root node of binary tree is the first node in Preorder traversal.
The rough sketch of tree is:




Second node in preorder traversal is 2. This makes 5 as right child to node 2. The fourth

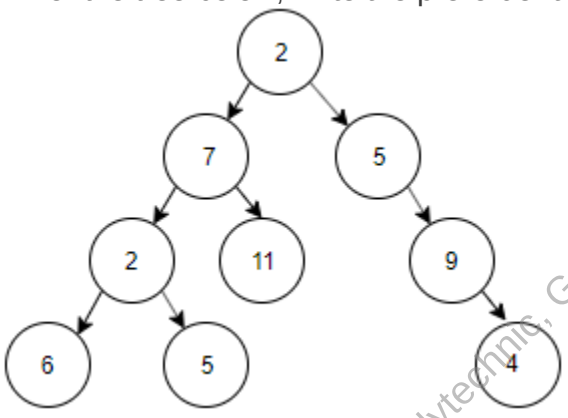
abc

node in preorder traversal is 3. This makes 4 as right child to node 3. Thus the final tree is:




## Preorder Traversal

1. For the tree below, write the pre-order traversal.



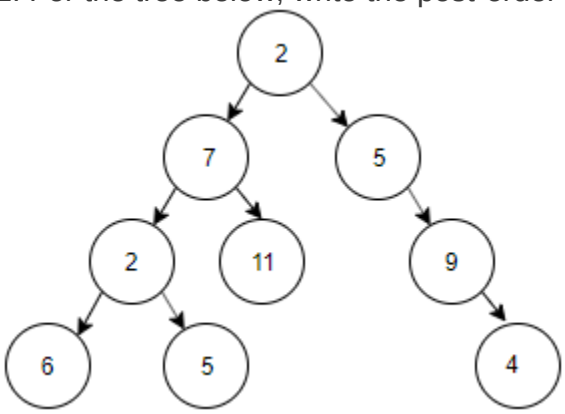

a) 2, 7, 2, 6, 5, 11, 5, 9, 4
b) 2, 7, 5, 2, 6, 9, 5, 11, 4
c) 2, 5, 11, 6, 7, 4, 9, 5, 2
d) 2, 7, 5, 6, 11, 2, 5, 4, 9

**Answer: a**

Explanation: Pre order traversal follows NLR(Node-Left-Right).

abc

2. For the tree below, write the post-order traversal.




a) 2, 7, 2, 6, 5, 11, 5, 9, 4
b) 2, 7, 5, 2, 6, 9, 5, 11, 4
c) 2, 5, 11, 6, 7, 4, 9, 5, 2
d) 2, 7, 5, 6, 11, 2, 5, 4, 9

**Answer: c**
Explanation: Post order traversal follows LRN(Left-Right-Node).

3. Select the code snippet which performs pre-order traversal.
a)

```
public void preorder(Tree root)
{
        System.out.println(root.data);
        preorder(root.left);
        preorder(root.right);
}
```

b)

```
public void preorder(Tree root)
{
        preorder(root.left);
        System.out.println(root.data);
        preorder(root.right);
}
```

c)

abc

```
public void preorder(Tree root)
{
        System.out.println(root.data);
        preorder(root.right);
        preorder(root.left);
}
```

d)

```
public void preorder(Tree root)
{
        preorder(root.right);
        preorder(root.left);
        System.out.println(root.data);
}
```

**Answer: a**
Explanation: Pre-order traversal follows NLR(Node-Left-Right).

4. Select the code snippet which performs post-order traversal.
a)

```
public void postorder(Tree root)
{
        System.out.println(root.data);
        postorder(root.left);
        postorder(root.right);
}
```

b)

```
public void postorder(Tree root)
{
        postorder(root.left);
        postorder(root.right);
        System.out.println(root.data);
}
```

c)

abc

```
public void postorder(Tree root)
{
        System.out.println(root.data);
        postorder(root.right);
        postorder(root.left);
}
```

d)

```
public void postorder(Tree root)
{
        postorder(root.right);
        System.out.println(root.data);
        postorder(root.left);
}
```

**Answer: b**
Explanation: Post order traversal follows NLR(Left-Right-Node).

5. Select the code snippet that performs pre-order traversal iteratively.
a)

```
public void preOrder(Tree root)
{
        if (root == null) return;
        Stack<Tree> stk = new Stack<Tree>();
        st.add(root);
        while (!stk.empty())
        {
            Tree node = stk.pop();
            System.out.print(node.data + " ");
                    if (node.left != null) stk.push(node.left);
            if (node.right != null) stk.push(node.right);
        }
}
```

b)

abc

```
public void preOrder(Tree root)
{
        if (root == null) return;
                Stack<Tree> stk = new Stack<Tree>();
        while (!stk.empty())
        {
            Tree node = stk.pop();
            System.out.print(node.data + " ");
            if (node.right != null) stk.push(node.right);
            if (node.left != null) stk.push(node.left);
        }
}
```

c)

```
public void preOrder(Tree root)
{
        if (root == null) return;
        Stack<Tree> stk = new Stack<Tree>();
        st.add(root);
        while (!stk.empty())
        {
            Tree node = stk.pop();
            System.out.print(node.data + " ");
            if (node.right != null) stk.push(node.right);
            if (node.left != null) stk.push(node.left);
        }
}
```

d)

```
public void preOrder(Tree root)
{
        if (root == null) return;
        Stack<Tree> stk = new Stack<Tree>();
        st.add(root);
        while (!stk.empty())
        {
            Tree node = stk.pop();
            System.out.print(node.data + " ");
            if (node.right != null) stk.push(node.left);
            if (node.left != null) stk.push(node.right);
        }
}
```



**Answer: c**

Explanation: Add the root to the stack first, then continously check for the right and left children of the top-of-the-stack.

abc

6. Select the code snippet that performs post-order traversal iteratively.

a)

```
public void postorder(Tree root)
{
        if (root == null)
        return;
        Stack<Tree> stk = new Stack<Tree>();
        Tree node = root;
        while (!stk.isEmpty() || node != null)
        {
            while (node != null)
            {
                if (node.right != null)
                    stk.add(node.left);
                stk.add(node);
                node = node.right;
            }
            node = stk.pop();
            if (node.right != null && !stk.isEmpty() && node.right ==
stk.peek())
            {
                Trees nodeRight = stk.pop();
                stk.push(node);
                node = nodeRight;
            } else
            {
                System.out.print(node.data+" ");
                node = null;
            }
        }
}
```

b)

abc

```
public void postorder(Tree root)
{
        if (root == null)
        return;
        Stack<Tree> stk = new Stack<Tree>();
        Tree node = root;
        while (!stk.isEmpty() || node != null)
        {
            while (node != null)
            {
                if (node.right != null)
                    stk.add(node.right);
                stk.add(node);
                node = node.left;
            }
            node = stk.pop();
            if (node.right != null && !stk.isEmpty() && node.right ==
stk.peek())
            {
                Trees nodeRight = stk.pop();
                stk.push(node);
                node = nodeRight;
            } else
            {
                System.out.print(node.data + " ");
                node = null;
            }
        }
}
```

c)

```
public void postorder(Tree root)
{
        if (root == null)
            return;
        Stack<Tree> stk = new Stack<Tree>();
        Tree node = root;
        while (!stk.isEmpty() || node != null)
        {
            while (node != null)
            {
                if (node.right != null)
                    stk.add(node.right);
                stk.add(node);
                node = node.left;
            }
            node = stk.pop();
            if (node.right != null)
            {
                Trees nodeRight = stk.pop();
                stk.push(node);
                node = nodeRight;
            } else
            {
                System.out.print(node.data + " ");
                node = null;
            }
        }
}
```

abc

d)

```
public void postorder(Tree root)
{
        if (root == null)
            return;
        Stack<Tree> stk = new Stack<Tree>();
        Tree node = root;
        while (!stk.isEmpty() || node != null)
        {
            while (node != null)
            {
                if (node.right != null)
                    stk.add(node.left);
                stk.add(node);
                node = node.left;
            }
            node = stk.pop();
            if (node.right != null)
            {
                Trees nodeRight = stk.pop();
                stk.push(node);
                node = nodeLeft;
            } else
            {
                System.out.print(node.data + " ");
                node = null;
            }
        }
}
```

**Answer: b**
Explanation: The left and right children are added first to the stack, followed by the node, which is then popped. Post-order follows LRN policy.

7. What is the time complexity of pre-order traversal in the iterative fashion?
a) O(1)
b) O(n)
c) O(logn)
d) O(nlogn)
**Answer: b**
Explanation: Since you have to go through all the nodes, the complexity becomes O(n).

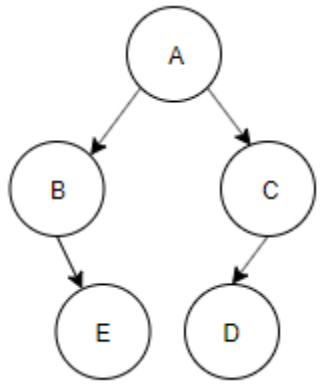
8. What is the space complexity of the post-order traversal in the recursive fashion? (d is the tree depth and n is the number of nodes)
a) O(1)
b) O(nlogd)
c) O(logd)
d) O(d)

**Answer: d**
Explanation: In the worst case we have d stack frames in the recursive call, hence the complexity is O(d).

abc

9. To obtain a prefix expression, which of the tree traversals is used?
a) Level-order traversal
b) Pre-order traversal
c) Post-order traversal
d) In-order traversal
**Answer: b**
Explanation: As the name itself suggests, pre-order traversal can be used.

10. Consider the following data. The pre order traversal of a binary tree is A, B, E, C, D. The in order traversal of the same binary tree is B, E, A, D, C. The level order sequence for the binary tree is _____
a) A, C, D, B, E
b) A, B, C, D, E
c) A, B, C, E, D
d) D, B, E, A, C
**Answer: b**
Explanation: The inorder sequence is B, E, A, D, C and Preorder sequence is A, B, E, C, D. The tree constructed with the inorder and preorder sequence is




The levelorder traversal (BFS traversal) is A, B, C, E, D.



11. Consider the following data and specify which one is Preorder Traversal Sequence, Inorder and Postorder sequences.
S1: N, M, P, O, Q
S2: N, P, Q, O, M
S3: M, N, O, P, Q
a) S1 is preorder, S2 is inorder and S3 is postorder
b) S1 is inorder, S2 is preorder and S3 is postorder
c) S1 is inorder, S2 is postorder and S3 is preorder
d) S1 is postorder, S2 is inorder and S3 is preorder
**Answer: c**
Explanation: Preorder traversal starts from the root node and postorder and inorder starts from the left child node of the left subtree. The first node of S3 is different and for S1 and S2 it's the same. Thus, S3 is preorder traversal and the root node is M. Postorder traversal visits the root node at last. S2 has the root node(M) at last that implies S2 is postorder

abc

traversal. S1 is inorder traversal as S2 is postorder traversal and S3 is preorder traversal. Therefore, S1 is inorder traversal, S2 is postorder traversal and S3 is preorder traversal.

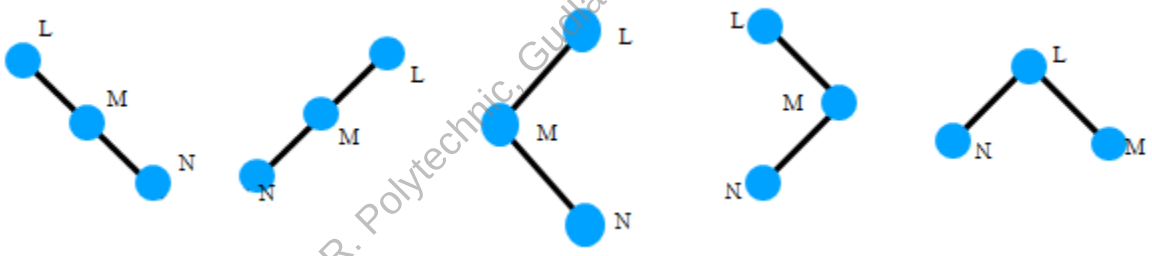
# Postorder Traversal

1. In postorder traversal of binary tree right subtree is traversed before visiting root.
a) True
b) False

**Answer: a**
Explanation: Post-order method of traversing involves – i) Traverse left subtree in post-order, ii) Traverse right subtree in post-order, iii) visit the root.

2. What is the possible number of binary trees that can be created with 3 nodes, giving the sequence N, M, L when traversed in post-order.
a) 15
b) 3
c) 5
d) 8

**Answer: c**
Explanation: 5 binary trees are possible and they are,



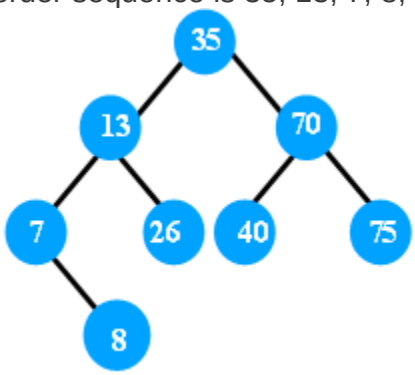



3. The post-order traversal of a binary tree is O P Q R S T. Then possible pre-order traversal will be _____
a) T Q R S O P
b) T O Q R P S
c) T Q O P S R
d) T Q O S P R

**Answer: c**
Explanation: The last, second last nodes visited in post-order traversal are root and it's right child respectively. Option T Q R S O P can't be a pre-order traversal, because nodes O, P are visited after the nodes Q, R, S. Option T O Q R P S, can't be valid, because the pre-order sequence given in option T O Q R P S and given post-order traversal creates a tree with node T as root and node O as left subtree. Option T Q O P S R is valid. Option T Q O S P R is not valid as node P is visited after visiting node S.

abc

4. A binary search tree contains values 7, 8, 13, 26, 35, 40, 70, 75. Which one of the following is a valid post-order sequence of the tree provided the pre-order sequence as 35, 13, 7, 8, 26, 70, 40 and 75?
a) 7, 8, 26, 13, 75, 40, 70, 35
b) 26, 13, 7, 8, 70, 75, 40, 35
c) 7, 8, 13, 26, 35, 40, 70, 75
d) 8, 7, 26, 13, 40, 75, 70, 35
**Answer: d**
Explanation: The binary tree contains values 7, 8, 13, 26, 35, 40, 70, 75. The given pre-order sequence is 35, 13, 7, 8, 26, 70, 40 and 75. So, the binary search tree formed is




Thus post-order sequence for the tree is 8, 7, 26, 13, 40, 75, 70 and 35.

5. Which of the following pair's traversals on a binary tree can build the tree uniquely?
a) post-order and pre-order
b) post-order and in-order
c) post-order and level order
d) level order and preorder

**Answer: b**
Explanation: A binary tree can uniquely be created by post-order and in-order traversals.

6. A full binary tree can be generated using _____
a) post-order and pre-order traversal
b) pre-order traversal
c) post-order traversal
d) in-order traversal
**Answer: a**
Explanation: Every node in a full binary tree has either 0 or 2 children. A binary tree can be generated by two traversals if one of them is in-order. But, we can generate a full binary tree using post-order and pre-order traversals.

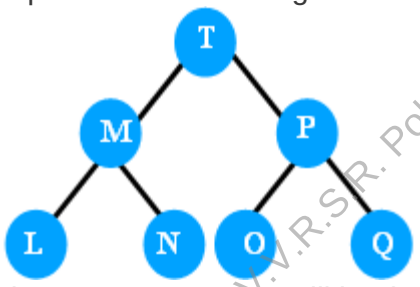
abc

7. The maximum number of nodes in a tree for which post-order and pre-order traversals may be equal is _____
a) 3
b) 1
c) 2
d) any number
**Answer: b**
Explanation: The tree with only one node has post-order and pre-order traversals equal.

8. The steps for finding post-order traversal are traverse the right subtree, traverse the left subtree or visit the current node.
a) True
b) False

**Answer: b**
Explanation: Left subtree is traversed first in post-order traversal, then the right subtree is traversed and then the output current node.

9. The pre-order and in-order are traversals of a binary tree are T M L N P O Q and L M N T O P Q. Which of following is post-order traversal of the tree?
a) L N M O Q P T
b) N M O P O L T
c) L M N O P Q T
d) O P L M N Q T
**Answer: a**
Explanation: The tree generated by using given pre-order and in-order traversal is

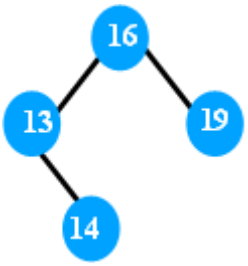



Thus, L N M O Q P T will be the post-order traversal.

10. For a binary tree the first node visited in in-order and post-order traversal is same.
a) True
b) False

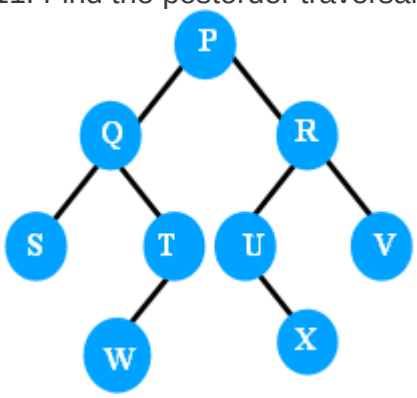
**Answer: b**
Explanation: Consider a binary tree,

abc




Its in-order traversal – 13 14 16 19
Its post-order traversal- 14 13 19 16. Here the first node visited is not same.

11. Find the postorder traversal of the binary tree shown below.

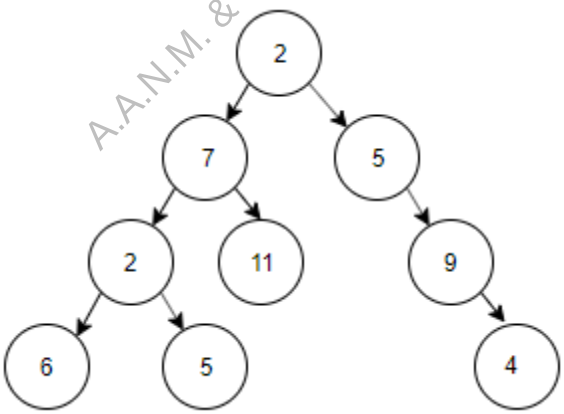



a) P Q R S T U V W X
b) W R S Q P V T U X
c) S W T Q X U V R P
d) S T W U X V Q R P
**Answer: c**
Explanation: In postorder traversal the left subtree is traversed first and then the right subtree and then the current node. So, the posturer traversal of the tree is, S W T Q X U V R P.



# Inorder Traversal

1. For the tree below, write the in-order traversal.

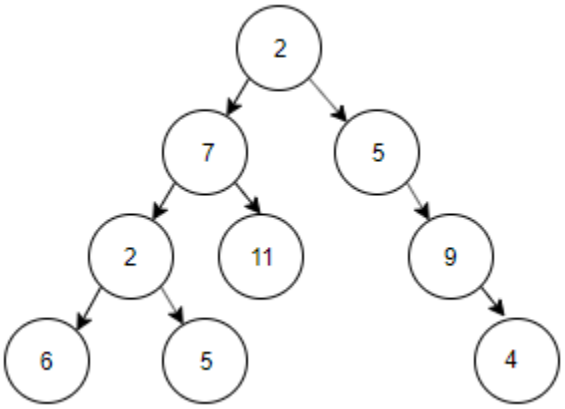



a) 6, 2, 5, 7, 11, 2, 5, 9, 4

abc

b) 6, 5, 2, 11, 7, 4, 9, 5, 2
c) 2, 7, 2, 6, 5, 11, 5, 9, 4
d) 2, 7, 6, 5, 11, 2, 9, 5, 4
**Answer: a**
Explanation: In-order traversal follows LNR(Left-Node-Right).

2. For the tree below, write the level-order traversal.




a) 2, 7, 2, 6, 5, 11, 5, 9, 4
b) 2, 7, 5, 2, 11, 9, 6, 5, 4
c) 2, 5, 11, 6, 7, 4, 9, 5, 2
d) 2, 7, 5, 6, 11, 2, 5, 4, 9
**Answer: b**
Explanation: Level order traversal follows a breadth first search approach.

3. Select the code snippet which performs in-order traversal.
a)

```
public void inorder(Tree root)
{
        System.out.println(root.data);
        inorder(root.left);
        inorder(root.right);
}
```

b)

```
public void inorder(Tree root)
{
        inorder(root.left);
        System.out.println(root.data);
        inorder(root.right);
}
```

c)

abc

```
public void inorder(Tree root)
{
        System.out.println(root.data);
        inorder(root.right);
        inorder(root.left);
}
```

d)

```
public void inorder(Tree root)
{
        inorder(root.right);
        inorder(root.left);
        System.out.println(root.data);
}
```

**Answer: b**
Explanation: In-order traversal follows LNR(Left-Node-Right).

4. Select the code snippet which performs level-order traversal.
a)

```
public static void levelOrder(Tree root)
{
    Queue<Node> queue=new LinkedList<Node>();
    queue.add(root);
    while(!queue.isEmpty())
    {
        Node tempNode=queue.poll();
        System.out.println("%d ",tempNode.data);
        if(tempNode.left!=null)
        queue.add(tempNode.left);
        if(tempNode.right!=null)
        queue.add(tempNode.right);
    }
}
```

b)

abc

```
public static void levelOrder(Tree root)
{
    Queue<Node> queue=new LinkedList<Node>();
    queue.add(root);
    while(!queue.isEmpty())
    {
        Node tempNode=queue.poll();
        System.out.println("%d ",tempNode.data);
        if(tempNode.left!=null)
        queue.add(tempNode.right);
        if(tempNode.right!=null)
        queue.add(tempNode.left);
    }
}
```

c)

```
public static void levelOrder(Tree root)
{
    Queue<Node> queue=new LinkedList<Node>();
    queue.add(root);
    while(!queue.isEmpty())
    {
        Node tempNode=queue.poll();
        System.out.println("%d ",tempNode.data);
        if(tempNode.right!=null)
        queue.add(tempNode.left);
        if(tempNode.left!=null)
        queue.add(tempNode.right);
    }
}
```

d)

```
public static void levelOrder(Tree root)
{
    Queue<Node> queue=new LinkedList<Node>();
    queue.add(root);
    while(!queue.isEmpty())
    {
        Node tempNode=queue.poll();
        System.out.println("%d ",tempNode.data);
        if(tempNode.right!=null)
        queue.add(tempNode.left.left);
        if(tempNode.left!=null)
        queue.add(tempNode.right.right);
    }
}
```

**Answer: a**

Explanation: Firstly add the root node to the queue. Then for all the remaining nodes, pop the front end of the queue and print it, add the left and right children of the popped node to the queue.

abc

5. What is the space complexity of the in-order traversal in the recursive fashion? (d is the tree depth and n is the number of nodes)
a) O(1)
b) O(nlogd)
c) O(logd)
d) O(d)

**Answer: d**
Explanation: In the worst case we have d stack frames in the recursive call, hence the complexity is O(d).

6. What is the time complexity of level order traversal?
a) O(1)
b) O(n)
c) O(logn)
d) O(nlogn)

**Answer: b**
Explanation: Since you have to go through all the nodes, the complexity becomes O(n).

7. Which of the following graph traversals closely imitates level order traversal of a binary tree?
a) Depth First Search
b) Breadth First Search
c) Depth & Breadth First Search
d) Binary Search

**Answer: b**
Explanation: Both level order tree traversal and breadth first graph traversal follow the principle that visit your neighbors first and then move on to further nodes.

8. In a binary search tree, which of the following traversals would print the numbers in the ascending order?
a) Level-order traversal
b) Pre-order traversal
c) Post-order traversal
d) In-order traversal

**Answer: d**
Explanation: In a binary search tree, a node's left child is always lesser than the node and right child is greater than the node, hence an in-order traversal would print them in a non decreasing fashion.

abc

# Binary Tree Properties

1. The number of edges from the root to the node is called _____ of the tree.
a) Height
b) Depth
c) Length
d) Width
**Answer: b**
Explanation: The number of edges from the root to the node is called depth of the tree.

2. The number of edges from the node to the deepest leaf is called _____ of the tree.
a) Height
b) Depth
c) Length
d) Width
**Answer: a**
Explanation: The number of edges from the node to the deepest leaf is called height of the tree.

3. What is a full binary tree?
a) Each node has exactly zero or two children
b) Each node has exactly two children
c) All the leaves are at the same level
d) Each node has exactly one or two children

**Answer: a**
Explanation: A full binary tree is a tree in which each node has exactly 0 or 2 children.

4. What is a complete binary tree?
a) Each node has exactly zero or two children
b) A binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from right to left
c) A binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right
d) A tree In which all nodes have degree 2
**Answer: c**
Explanation: A binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right is called complete binary tree. A Tree in which each node has exactly zero or two children is called full binary tree. A Tree in which the degree of each node is 2 except leaf nodes is called perfect binary tree.

5. What is the average case time complexity for finding the height of the binary tree?
a) h = O(loglogn)
b) h = O(nlogn)
c) h = O(n)

abc

d) h = O(log n)

**Answer: d**
Explanation: The nodes are either a part of left sub tree or the right sub tree, so we don't have to traverse all the nodes, this means the complexity is lesser than n, in the average case, assuming the nodes are spread evenly, the time complexity becomes O(logn).

6. Which of the following is not an advantage of trees?
a) Hierarchical structure
b) Faster search
c) Router algorithms
d) Undo/Redo operations in a notepad
**Answer: d**
Explanation: Undo/Redo operations in a notepad is an application of stack. Hierarchical structure, Faster search, Router algorithms are advantages of trees.

7. In a full binary tree if number of internal nodes is I, then number of leaves L are?
a) L = 2*I
b) L = I + 1
c) L = I – 1
d) L = 2*I – 1
**Answer: b**
Explanation: Number of Leaf nodes in full binary tree is equal to 1 + Number of Internal Nodes i.e L = I + 1

8. In a full binary tree if number of internal nodes is I, then number of nodes N are?
a) N = 2*I
b) N = I + 1
c) N = I – 1
d) N = 2*I + 1
**Answer: d**
Explanation: Relation between number of internal nodes(I) and nodes(N) is N = 2*I+1.

9. In a full binary tree if there are L leaves, then total number of nodes N are?
a) N = 2*L
b) N = L + 1
c) N = L – 1
d) N = 2*L – 1
**Answer: d**
Explanation: The relation between number of nodes(N) and leaves(L) is N=2*L-1.

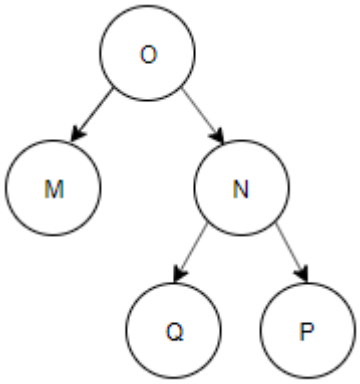
abc

10. Which of the following is incorrect with respect to binary trees?
a) Let T be a binary tree. For every k ≥ 0, there are no more than 2k nodes in level k
b) Let T be a binary tree with λ levels. Then T has no more than $2^{\lambda-1}$ nodes
c) Let T be a binary tree with N nodes. Then the number of levels is at least ceil(log (N + 1))
d) Let T be a binary tree with N nodes. Then the number of levels is at least floor(log (N + 1))

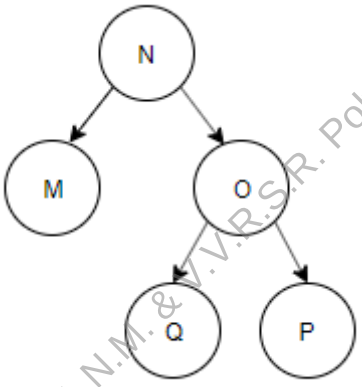
**Answer: d**
Explanation: In a binary tree, there are atmost 2k nodes in level k and $2^{k-1}$ total number of nodes. Number of levels is at least ceil(log(N+1)).

11. Construct a binary tree by using postorder and inorder sequences given below.
Inorder: N, M, P, O, Q
Postorder: N, P, Q, O, M

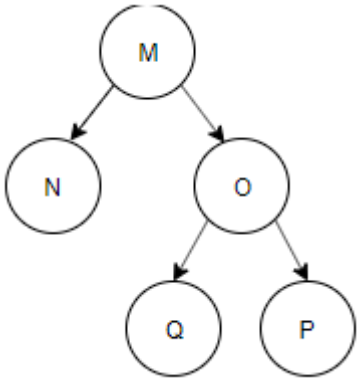
a)



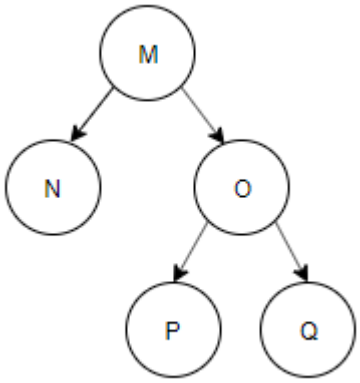

b)



abc



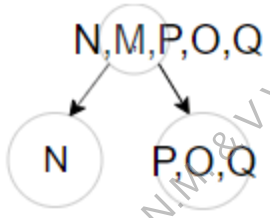

c)




d)

**Answer: d**

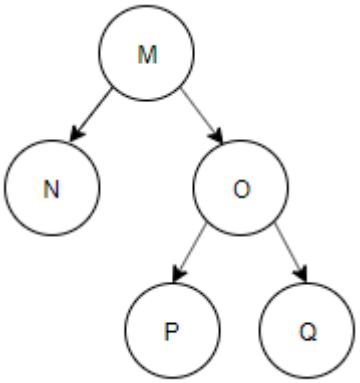
Explanation: Here,
Postorder Traversal: N, P, Q, O, M
Inorder Traversal: N, M, P, O, Q
Root node of tree is the last visiting node in Postorder traversal. Thus, Root Node = 'M'.
The partial tree constructed is:




The second last node in postorder traversal is O. Thus, node P becomes left child of node

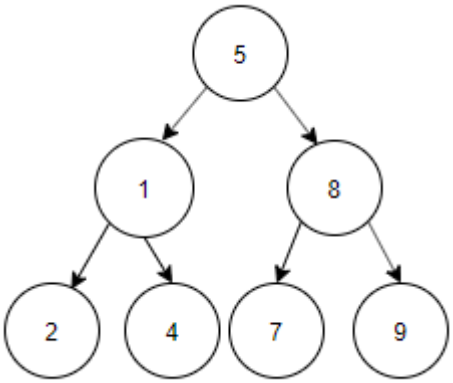
abc

O and node Q becomes right child of node Q. Thus, the final tree is:

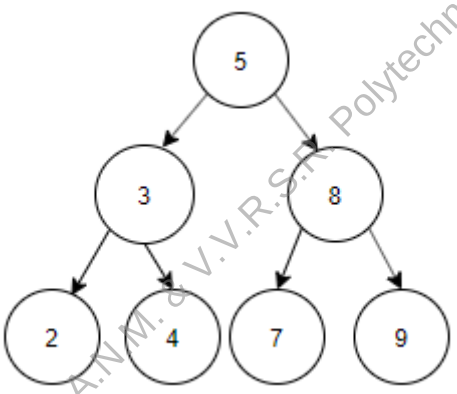



12. Construct a binary search tree by using postorder sequence given below.
Postorder: 2, 4, 3, 7, 9, 8, 5.

a)

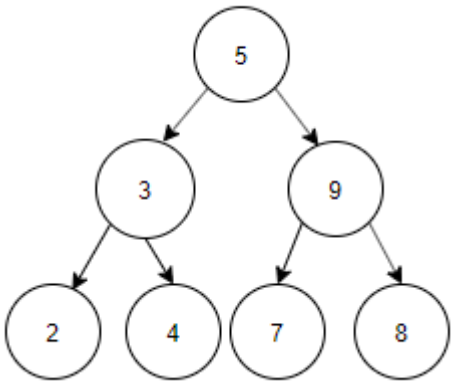



b)

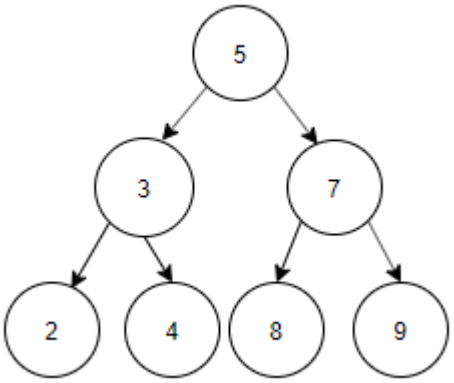


abc

c)




d)

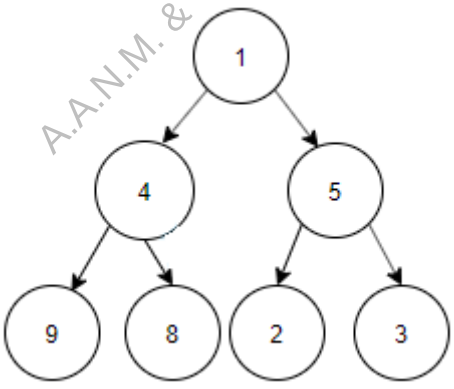



**Answer: b**

Explanation: Postorder sequence is 2, 4, 3, 7, 9, 8, 5.
Inorder sequence is the ascending order of nodes in Binary search tree. Thus, Inorder sequence is 2, 3, 4, 5, 7, 8, 9. The tree constructed using Postorder and Inorder sequence is

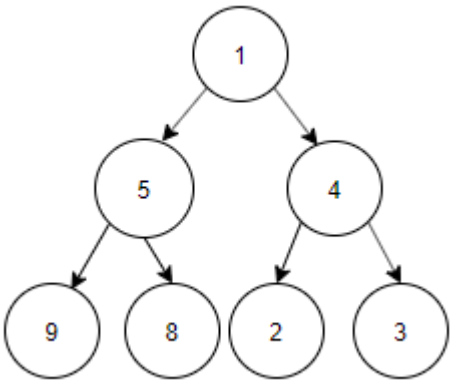
13. Construct a binary tree using inorder and level order traversal given below.
Inorder Traversal: 3, 4, 2, 1, 5, 8, 9
Level Order Traversal: 1, 4, 5, 9, 8, 2, 3

a)

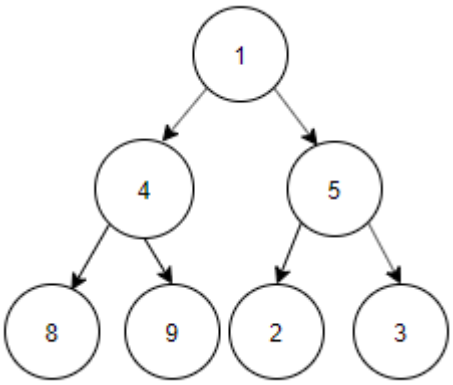


abc

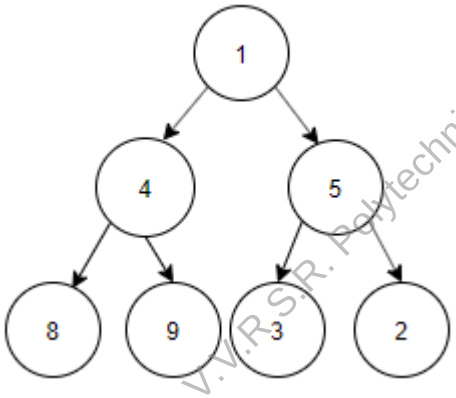
b)




c)



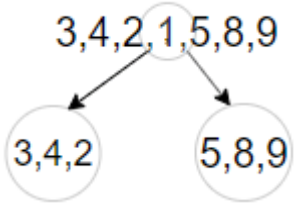

d)






**Answer: a**

Explanation: Inorder Traversal: 3, 4, 2, 1, 5, 8, 9
Level Order Traversal: 1, 4, 5, 9, 8, 2, 3
In level order traversal first node is the root node of the binary tree.
Thus the partially formed tree is:




In level order traversal, the second node is 4. Then, node 3 becomes left child of node 4

abc

and node 2 becomes right child of node 4. Third node of level order traversal is 8. Then, node 5 becomes left child of node 8 and node 9 becomes right child of node 8. Thus, the final tree is: