1. Differentiate between stack and queue Ans.

Stack	Queue
1. Stack is a data structure in which insertion and deletion operations are performed at **same end**.	1. Queue is a data structure in which insertion and deletion operations are performed at **different ends**.
2. In stack an element inserted last is deleted first so it is called **Last In First Out list.**	2. In Queue an element inserted first is deleted first so it is called **First In First Out list.**
3.In stack only one pointer is used called as **stack top**	3.In Queue two pointers are used called as **front** and **rear**
4. **Example**: Stack of books	4. **Example**: Students standing in a line at fees counter
5.**Application**: • Recursion • Polish notation	5. **Application**: • In computer system for organizing processes. • In mobile device for sending receiving messages.

2. Convert infix expression into prefix expression or postfix. Ans.

Infix expression	Read Character	Stack contents	Prefix expression
(A+B)*(C/G)+F	F	-	F
(A+B)*(C/G)+	+	+	F
(A+B)*(C/G)	)	+)	F
(A+B)*(C/G	G	+)	GF
(A+B)*(C/	/	+)/	GF
(A+B)*(C	C	+)/	CGF
(A+B)*(	(	+	/CGF
(A+B)*	*	+*	/CGF
(A+B)	)	+*)	/CGF
(A+B	B	+*)	B/CGF
(A+	+	+*)+	B/CGF
(A	A	+*)+	AB/CGF
(	(	+*	+AB/CGF
			*+AB/CGF
			+*+AB/CGF

3. Describe working of linear search with an example.

Ans.

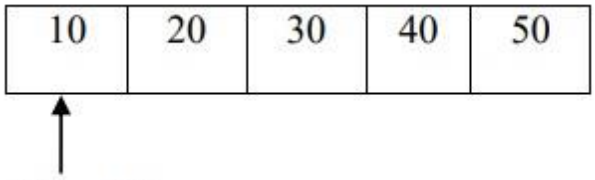
In linear search, search element is compared with each element from the list in a sequence. Comparison starts with first element from the list and continues till number is found or comparison reaches to the last element of the list.

As each element is checked with search element, the process of searching requires more time. Time complexity of linear search is O (n) where n indicates number of elements in list.

Linear search on sorted array:-On sorted array search takes place till element is found or comparison reaches to an element greater than search element.

Example:- Using array representation Input list 10, 20, 30, 40, 50 and Search element 30, Index =0

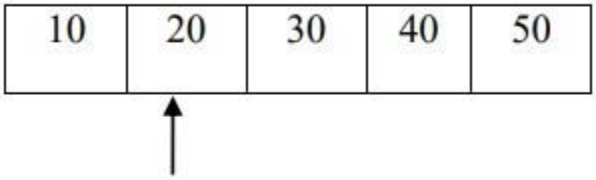
Iteration 1



10 ! = 30

Index = Index + 1

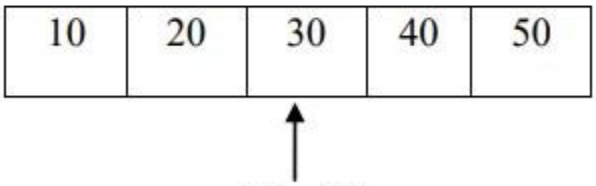
Iteration 2



20 ! = 30

Index = Index + 1

Iteration 3



30 = 30

Number found

4. Find the position of element 29 using binary search method in an array 'A' given below. Show each step.
A={11,5,21,3,29,17,2,43} Ans.

An array which is given A[]= {11,5,21,3,29,17,2,43} is not in sorted manner, first we need to sort them in order; So an array will be A[ ]={2,3,5,11,17,21,29,43} and the value to be searched is VAL = 29. The binary search algorithm will proceed in the following manner.

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
2	3	5	11	17	21	29	43

Iteration 1:

BEG = 0, END = 7, MID = (0 + 7)/2 = 3

Now, VAL = 29 and A[MID] = A[3] =11

A[3] is less than VAL, therefore, we now search for the value in the second half of the array.

So, we change the values of BEG and MID.

Iteration 2:

Now, BEG = MID + 1 = 4, END = 7, MID = (4 + 7)/2 =11/2 = 5; VAL = 29 and A [MID] = A [5] = 21

A[5] is less than VAL, therefore, we now search for the value in the second half of the segment.

So, again we change the values of BEG and MID.

Iteration 3:

Now, BEG = MID + 1 = 6, END = 7, MID = (6 + 7)/2 = 6 Now, VAL = 29 and A [MID] = A [6]=29

So, Element 29 is found at 6th location in given array A[]={2,3,5,11,17,21,29,43}.

5.Explain Double Ended Queue with example
Ans.

Deque (or double-ended queue)

The deque stands for Double Ended Queue. Deque is a linear data structure where the insertion and deletion operations are performed from both ends. We can say that deque is a generalized version of the queue.

Though the insertion and deletion in a deque can be performed on both ends, it does not follow the FIFO rule. The representation of a deque is given as follows -

Representation of deque

Types of deque

There are two types of deque -

- Input restricted queue
- Output restricted queue

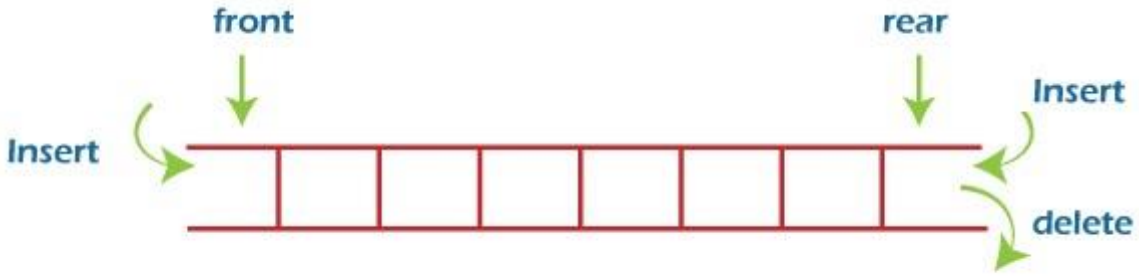
Input restricted Queue

In input restricted queue, insertion operation can be performed at only one end, while deletion can be performed from both ends.

input restricted double ended queue

Output restricted Queue

In output restricted queue, deletion operation can be performed at only one end, while insertion can be performed from both ends.



Output restricted double ended queue

Operations performed on deque

There are the following operations that can be applied on a deque -

- Insertion at front ○ Insertion at rear ○ Deletion at front ○ Deletion at rear

We can also perform peek operations in the deque along with the operations listed above. Through peek operation, we can get the deque's front and rear elements of the deque. So, in addition to the above operations, following operations are also supported in deque -

- Get the front item from the deque ○ Get the rear item from the deque ○ Check whether the deque is full or not ○ Checks whether the deque is empty or not

6. Describe working of bubble sort , insertion , selection with example. Ans.

1. **Bubble Sort Working**:

- Bubble sort compares adjacent elements in an array and swaps them if they are in the wrong order.
- This process is repeated for each element, and after each complete pass, the largest unsorted element "bubbles" up to its correct position.
- The algorithm continues until no more swaps are needed, indicating that the array is sorted.

Steps:

1. Compare the first two elements. If the first is greater than the second, swap them.
2. Move to the next pair and repeat until you reach the end of the array.
3. After each full pass through the array, the largest element gets placed in its correct position.
4. Repeat the process for the remaining unsorted part of the array until the entire array is sorted.

Example (Sorting the array [5, 3, 8, 4, 2]):

1. **First pass**:
   - Compare 5 and 3 → Swap them → [3, 5, 8, 4, 2]
   - Compare 5 and 8 → No swap → [3, 5, 8, 4, 2] ○ Compare 8 and 4 → Swap them → [3, 5, 4, 8, 2]
   - Compare 8 and 2 → Swap them → [3, 5, 4, 2, 8]
   - After the first pass, 8 is in its correct position.
2. **Second pass**:
   - Compare 3 and 5 → No swap → [3, 5, 4, 2, 8] ○ Compare 5 and 4 → Swap them → [3, 4, 5, 2, 8]
   - Compare 5 and 2 → Swap them → [3, 4, 2, 5, 8]
   - After the second pass, 5 is in its correct position.
3. **Third pass**:
   - Compare 3 and 4 → No swap → [3, 4, 2, 5, 8] ○ Compare 4 and 2 → Swap them → [3, 2, 4, 5, 8] ○ After the third pass, 4 is in its correct position.

4. **Fourth pass**:
    - Compare 3 and 2 → Swap them → [2, 3, 4, 5, 8]
    - Now the array is sorted.

Time Complexity:

- Best case: O(n)O(n)O(n) (if the array is already sorted)
- Average and worst case: O(n2)O(n^2)O(n2)

---

2. Insertion Sort
Working:

- Insertion sort builds the final sorted array one element at a time.
- It works by picking elements from the unsorted part of the array and inserting them into the correct position in the sorted part of the array.
- It shifts elements in the sorted part to the right to make space for the new element.

Steps:

1. Start with the second element (since the first is considered sorted).
2. Compare the current element with the elements in the sorted part.
3. Move elements that are greater than the current element to one position ahead.
4. Insert the current element in the correct position.
5. Repeat until the entire array is sorted.

Example (Sorting the array [5, 3, 8, 4, 2]):

1. **Insert 3**: Compare with 5 → Move 5 to the right → [3, 5, 8, 4, 2]
2. **Insert 8**: No change as 8 is greater than 5 → [3, 5, 8, 4, 2]
3. **Insert 4**: Compare with 8 → Move 8 → Compare with 5 → Move 5 → Insert 4 → [3, 4, 5, 8, 2]
4. **Insert 2**: Compare with 8, 5, 4, and 3 → Move all elements → Insert 2 → [2, 3, 4, 5, 8]

Time Complexity:

- Best case: O(n)O(n)O(n) (if the array is already sorted)
- Average and worst case: O(n2)O(n^2)O(n2)

---

3. Selection Sort Working:

- Selection sort works by selecting the smallest (or largest) element from the unsorted part of the array and swapping it with the first unsorted element.
- It repeats this process until the entire array is sorted.

Steps:

1. Find the smallest (or largest) element in the unsorted part of the array.
2. Swap it with the first unsorted element.
3. Move the boundary of the sorted part of the array to the right and repeat until the array is sorted.

Example (Sorting the array [5, 3, 8, 4, 2]):

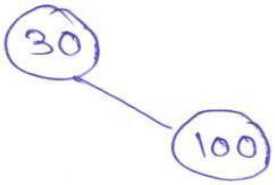
1. **First pass**: Find the smallest element (2) and swap it with the first element → [2, 3, 8, 4, 5]
2. **Second pass**: Find the smallest element in the remaining unsorted part ([3, 8, 4, 5]), which is 3 → No swap → [2, 3, 8, 4, 5]
3. **Third pass**: Find the smallest element in the remaining unsorted part ([8, 4, 5]), which is 4 → Swap it with 8 → [2, 3, 4, 8, 5]
4. **Fourth pass**: Find the smallest element in the remaining unsorted part ([8, 5]), which is 5 → Swap it with 8 → [2, 3, 4, 5, 8]
5. Now the array is sorted.

7. Construct a binary search tree for following elements: 30,100,90,15,2,25,36,72,78,10 show each step of construction of BST Ans.

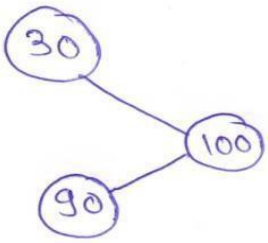
Stepwise construction of Binary search tree for following elements: 30,100,90,15,2,25,36,72,78,10 is as follows:

1)

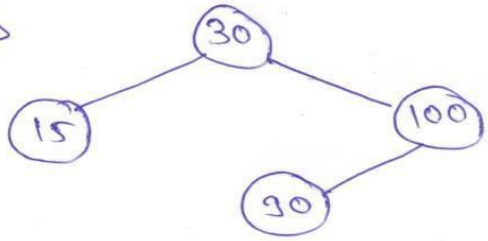
2)



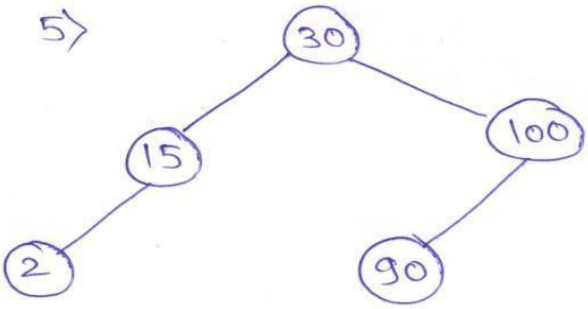
3)



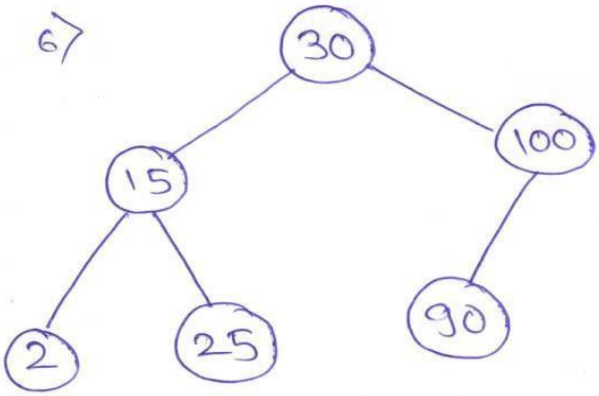
4)



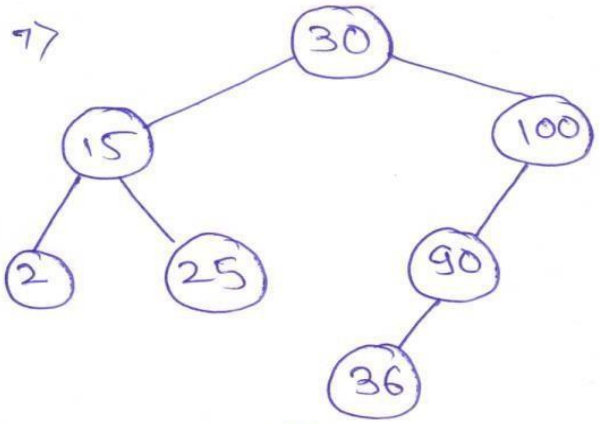
5)



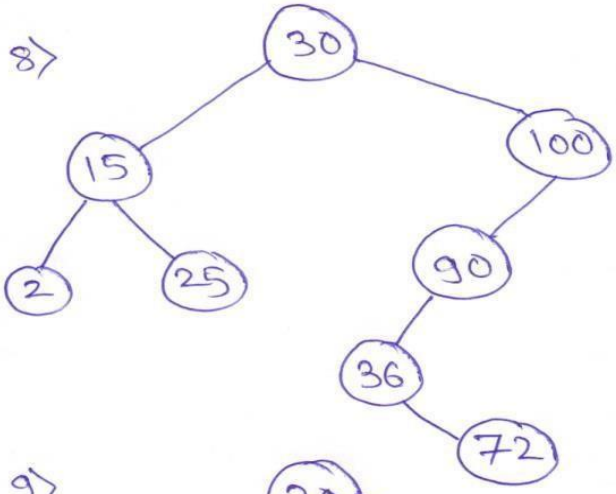
6)



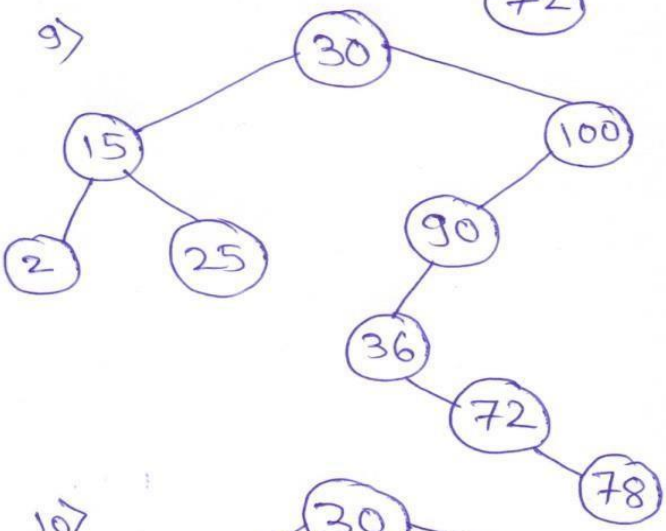
7)



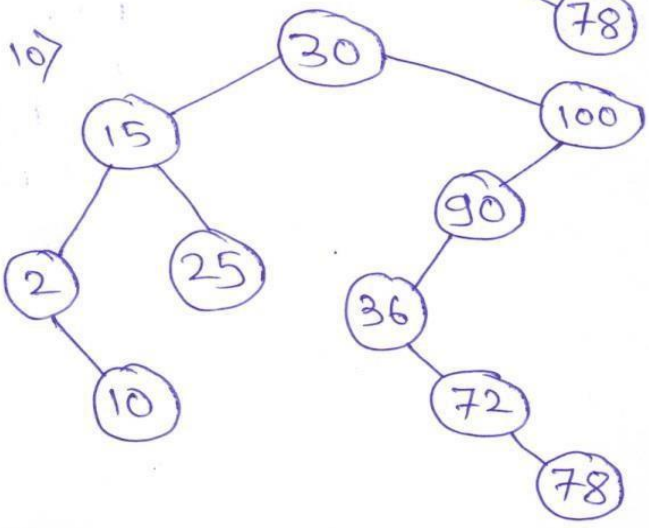
8)



9)



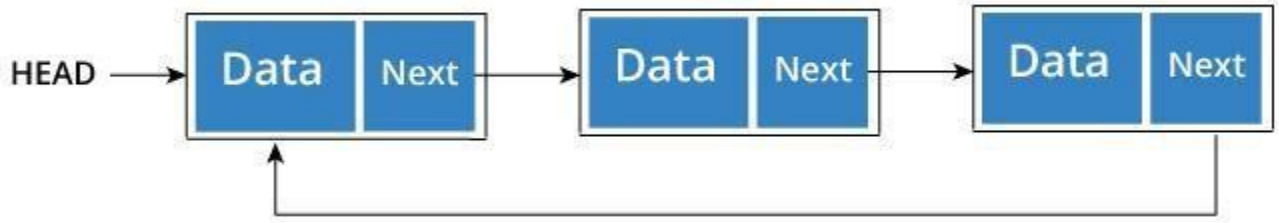
10)



8. Describe circular linked list with suitable diagram. Also state advantage of circular linked list over linear linked list.

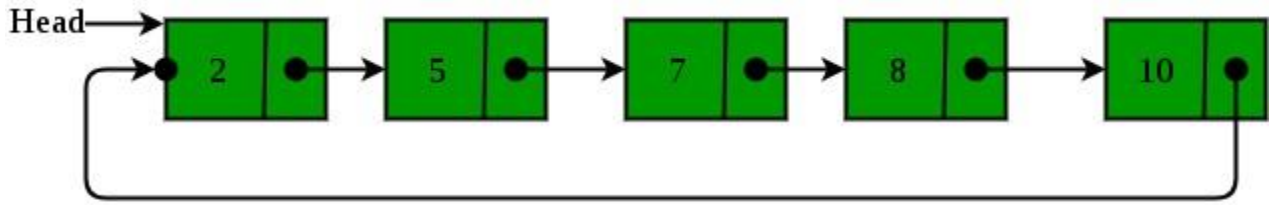
Ans.

Circular Linked List A circular linked list is a variation of linked list in which the last element is linked to the first element. This forms a circular loop.



A circular linked list can be either singly linked or doubly linked. for singly linked list, next pointer of last item points to the first item In doubly linked list, prev pointer of first item points to last item as well. We declare the structure for the circular linked list in the same way as follows:

```
Struct node
{
int data;
Struct node *next;
};
Typedef struct node *Node;
Node *start = null;
Node *last = null;  For
example:
```



Advantages of Circular Linked Lists:

1) Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.

2) Useful for implementation of queue. Unlike this implementation, we don't need to maintain two pointers for front and rear if we use circular linked list. We can maintain a pointer to the last inserted node and front can always be obtained as next of last.

3) Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.

4) Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.

9. Write a program to traverse a linked list.

Ans.

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct Node {
int data;
    struct Node* next;
};

struct Node* createNode(int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
new_node->data = new_data;     new_node->next = NULL;     return
new_node;
}

// Function to traverse and print the singly linked list void
traverseList(struct Node* start) {

    while (start != NULL) {

        printf("%d ", start->data);

        start = start->next;
    }
    printf("\n");
}

int main() {

    struct Node* start = createNode(10);     start-
>next = createNode(20);     start->next->next =
createNode(30);     start->next->next->next =
createNode(40);
    traverseList(head);

    return 0;
}
```

10. Write C program for performing following operations on array: insertion, display.
Ans.

```
#include<stdio.h> #include<conio.h>
void main()
{
inta[10],x,i,n,pos;
clrscr();
printf(“Enter the number of array element\n”); scanf(“%d”,&n);
printf(“Enter the array with %d element\n”, n);
for(i=0;i<n;i++) scanf(“%d”,&a[i]);
printf(“Enter the key value and its position\n”); scanf(“%d%d”
,&x,&pos);
for(i=n; i >= pos; i--)
{
a[i]=a[i-1];
}
```

```
a[pos-1]=x; printf("Array
element\n ");
for(i=0;i<n+1;i++)
printf("%d\t",a[i]); getch();
}
```

11. Differentiate between binary search and sequential search.

Ans.

Sr. No.	Binary Search	Sequential search (linear search)
1	Input data needs to be sorted in Binary Search	Input data need not to be sorted in Linear Search.
2	In contrast, binary search compares key value with the middle element of an array and if comparison is unsuccessful then cuts down search to half.	A linear search scans one item at a time, without jumping to any item.
3	Binary search implements divide and conquer approach.	Linear search uses sequential approach.
4	In binary search the worst case complexity is O(log n) comparisons.	In linear search, the worst case complexity is O(n), comparisons.
5	Binary search is efficient for the larger array.	Linear search is efficient for the smaller array.

12. Evaluate the following prefix expression: - * + 4 3 2 5 and 5, 6, 2, +, *, 12, 4, /, - show diagrammatically
Ans.

Scanned Symbol	Operand 1	Operand 2	Value	Stack Content
5				5
2				5,2
3				5,2,3
4				5,2,3,4
+	4	3	12	5,2,12
*	12	2	24	5,24
-	24	5	19	19

Result of above prefix expression evaluation - 19

Scanned Symbol	Operand 1	Operand 2	Value	Stack Content
5				5
6				5, 6
2				5, 6, 2
+	6	2	8	5, 8
*	5	8	40	40
12				40, 12
4				40, 12, 4
/	12	4	3	40, 3
-	40	3	37	37

Result of above prefix expression evaluation – 37

13. Sort the given number in ascending order using Radix sort: 348, 14, 641, 3851, 74. Ans.

Pass 1:

	0	1	2	3	4	5	6	7	8	9
0348									0348	
0014					0014					
0641		0641								
3851		3851								
0074					0074					

0641,3851,0014,0074,0348

Pass 2:

	0	1	2	3	4	5	6	7	8	9
0641					0641					
3851						3851				
0014		0014								
0074								0074		
0348					0348					

0014,0641,0348,3851,0074

Pass 3:

	0	1	2	3	4	5	6	7	8	9
0014	0014									
0641							0641			
0348				0348						
3851									3851	
0074	0074									

0014,0074,0348,0641,3851

Pass 4:

	0	1	2	3	4	5	6	7	8	9
0014	0014									
0074	0074									
0348	0348									
0641	0641									
3851					3851					

Sorted Elements are: 14, 74, 348, 641, 3851

14. Write an algorithm to delete and insert a node from the beginning and end of a circular linked list.
Ans.
Delete at Beginning (Circular Queue):

Step 1:
IF FRONT = -1
◦ Write "UNDERFLOW"
◦ Goto Step 5
◦ [END of IF]

Step 2:
SET VAL = QUEUE[FRONT]

Step 3:
IF FRONT == REAR
◦ SET FRONT = REAR = -1
◦ ELSE
◦ IF FRONT = MAX - 1
◦ SET FRONT = 0
◦ ELSE
◦ SET FRONT = FRONT + 1
◦ [END of IF]
◦ [END of IF]

Step 4:
Write "Deleted Element = VAL"

Step 5:
EXIT

Delete at End (Circular Queue):

Step 1:
IF FRONT = -1
◦ Write "UNDERFLOW"
◦ Goto Step 5
◦ [END of IF]

Step 2:
SET VAL = QUEUE[REAR]

Step 3:
IF FRONT == REAR
◦ SET FRONT = REAR = -1
◦ ELSE
◦ IF REAR = 0
◦ SET REAR = MAX - 1
◦ ELSE
◦ SET REAR = REAR - 1

◦ [END of IF]
◦ [END of IF]

Step 4:
Write "Deleted Element = VAL"

Step 5:
EXIT

Insert at Beginning (Circular Queue):
Step 1:
IF (FRONT == 0 AND REAR == MAX - 1) OR (FRONT = REAR -1)
◦ Write "OVERFLOW"
◦ Goto Step 6
◦ [END of IF]

Step 2:
IF FRONT = -1
◦ SET FRONT = REAR = 0
◦ Goto Step 5
◦ [END of IF]

Step 3:
IF FRONT = 0
◦ SET FRONT = MAX - 1
◦ ELSE
◦ SET FRONT = FRONT - 1
◦ [END of IF]

Step 4:
SET QUEUE[FRONT] = VAL

Step 5:
Write "Inserted Element = VAL"

Step 6:
EXIT

Insert at End (Circular Queue):

Step 1:
IF (FRONT == 0 AND REAR == MAX - 1) OR (FRONT = REAR + 1)
◦ Write "OVERFLOW"
◦ Goto Step 6
◦ [END of IF]

Step 2:
IF FRONT = -1
◦ SET FRONT = REAR = 0
◦ Goto Step 5

◦ [END of IF]

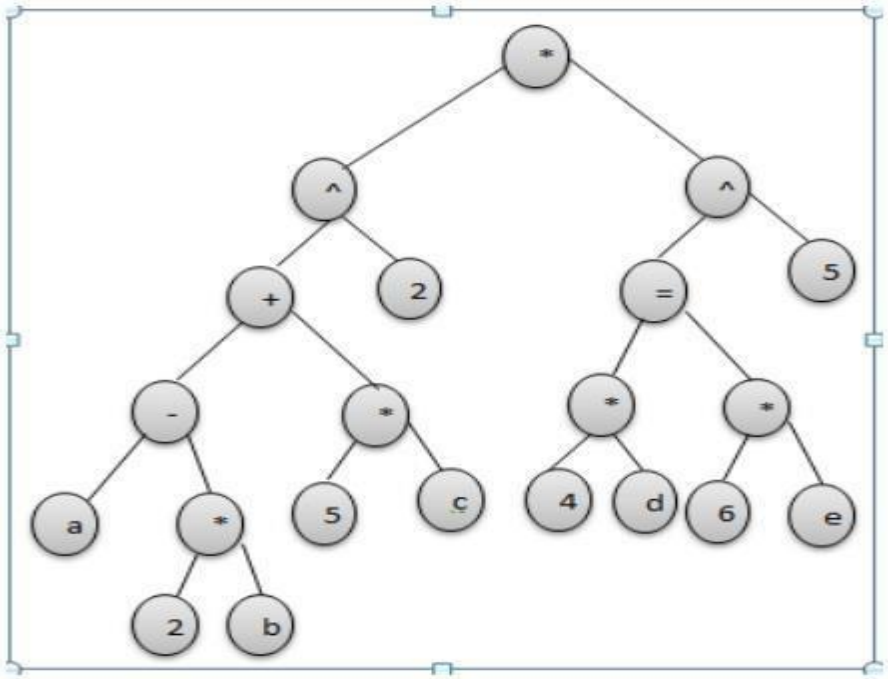
Step 3:
IF REAR = MAX - 1
◦ SET REAR = 0
◦ ELSE
◦ SET REAR = REAR + 1
◦ [END of IF]

Step 4:
SET QUEUE[REAR] = VAL

Step 5:
Write "Inserted Element = VAL"

Step 6:
EXIT

15. Draw an expression tree for the following expression: (a-2b+5e) 2 * (4d=6e) 5
Ans.



16. Write an algorithm to insert an element at the beginning and end of linked list
Ans.

Inserting a Node at the Beginning of a Linked List

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 7

[END OF IF]

Step 2: SET NEW_NODE = AVAIL

Step 3: SET NEW_NODE->DATA = VAL

Step 4: SET NEW_NODE-> NEXT = START

Step 5: SET START = NEW_NODE

Step 6: EXIT

Inserting a Node at the End of a Linked List

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 7

[END OF IF]

Step 2: SET NEW_NODE = AVAIL

Step 3: SET NEW_NODE->DATA = VAL

SET NEW_NODE->NEXT = NULL

Step 4: PTR=START

Step 5: Repeat Step 6 while PTR->NEXT != NULL

Step 6: SET PTR = PTR->NEXT

[END OF LOOP]

Step 5: SET PTR->NEXT = NEW_NODE

Step 6: EXIT

17. Write an algorithm for performing push and pop operations on stack.

Ans.

Push algorithm:

Step 1: IF TOP = MAX-1

PRINT OVERFLOW

Goto step 4

[END OF IF]

Step 2: SET TOP = TOP + 1

Step 3: SET STACK[TOP] = VALUE

Step 4: END

Pop algorithm:
Step 1: IF TOP = NULL PRINT
UNDERFLOW
Go to step 4
[END OF IF]
Step 2: SET VAL = STACK[TOP]
Step 3: SET TOP = TOP - 1
Step 4: END

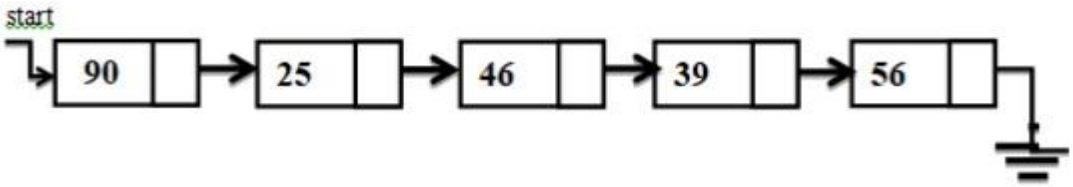
18. Write an algorithm to search a particular node in the given linked list.
Ans.
Step 1: SET PTR = HEAD
Step 2: Set I = 0
STEP 3: IF PTR = NULL
WRITE "EMPTY LIST"
GOTO STEP 8
END OF IF

STEP 4: REPEAT STEP 5 TO 7 UNTIL PTR != NULL
STEP 5: if ptr → data = item write i+1
End of IF

STEP 6: I = I + 1
STEP 7: PTR = PTR → NEXT
[END OF LOOP]

STEP 8: EXIT

19. Create a singly linked list using data fields 90, 25, 46, 39, 56. Search a node 40 from the SLL and show the procedure step-by-step with the help of a diagram from start to end Ans.
To Search a data field in singly linked list, need to start searching the data field from first node of singly linked list. ORIGINAL LIST:



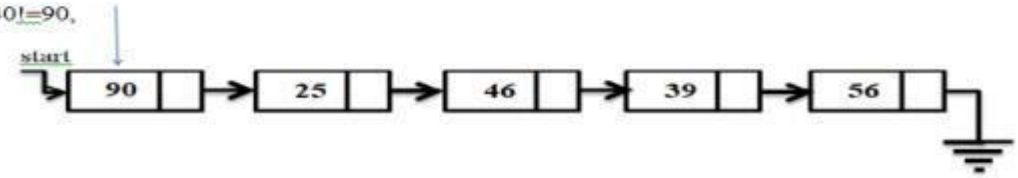
SEARCHING A NODE STEP 1:
Compare 40 with 90
40!=90

SEARCHING A NODE

STEP 1:

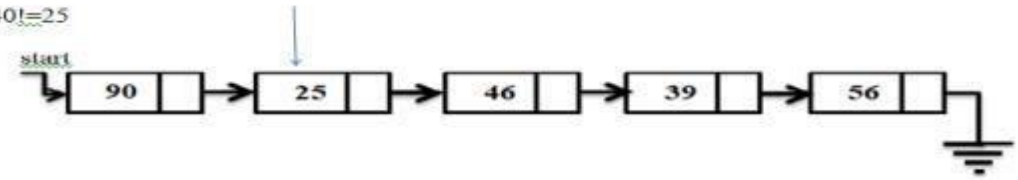
Compare 40 with 90

40!=90,



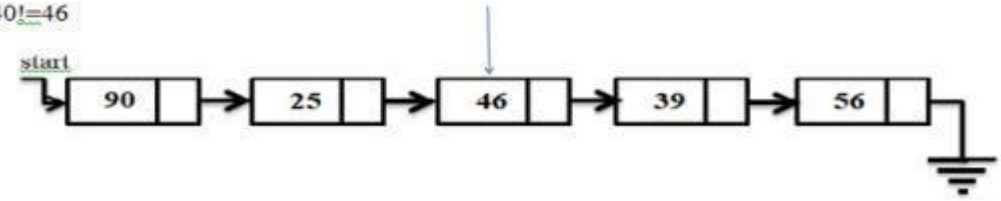
STEP 2:

40!=25



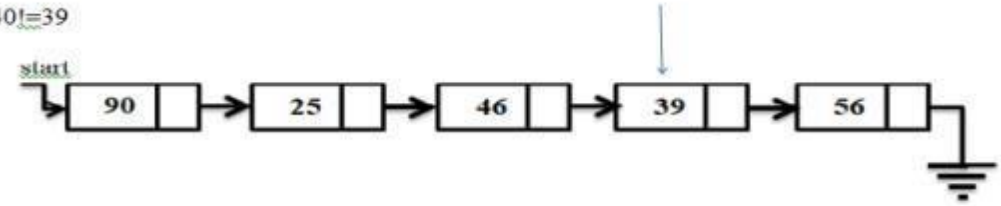
STEP 3:

40!=46



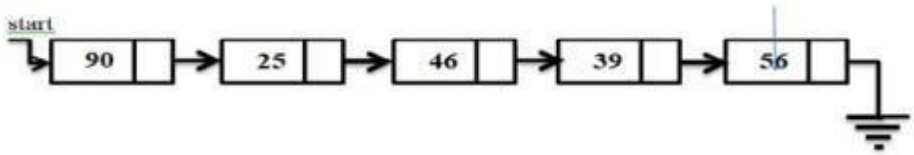
STEP 4:

40!=39



Step 5:

40!=56



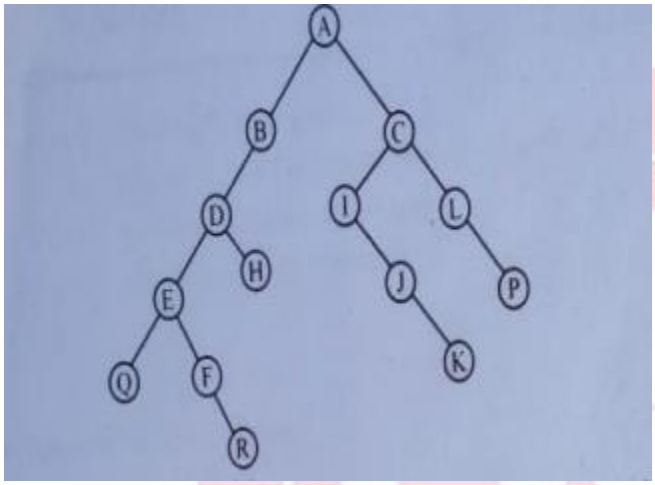
Node not found. Search unsuccessful

20. Write an algorithm to count the number of nodes in a singly linked list Ans.

0. Set Count =0
1. Initialize a node pointer, ptr = start.
2. Do following while ptr is not NULL
   a. ptr = ptr -> next
   b. Increment count by 1.
3. Return count.

21. Traverse a tree in inorder preorder and post order

Ans.



Inorder Traversal: Q,E,F,R,D,H,B,A,I,J,K,C,L,P

Preorder Traversal: A,B,D,E,Q,F,R,H,C,I,J,K,L,P

Postorder Traversal: Q,R,F,E,H,D,B,K,J,I,P,L,C,A

22. Compare Linked List and Array (any 4 points) Ans.

Criteria	**Use Arrays When...**	**Use Linked Lists When...**
Memory Allocation	Memory needs to be allocated in a contiguous block.	Memory can be allocated dynamically and non-contiguously.
Access Speed	Fast access to elements by index is required (O(1) access).	Access speed is not the primary concern, and traversal is acceptable.
Size of Data Structure	The size of the data structure is fixed or known in advance.	The size of the data structure is dynamic or frequently changing.
Insertion/Deletion	Insertions and deletions are infrequent and mostly at the end.	Frequent insertions and deletions are needed, especially at the beginning or middle.

Memory Efficiency	Memory efficiency is important, and overhead from pointers should be minimized.	Slightly higher memory usage is acceptable due to the overhead of pointers.
Complexity of Implementation	Simplicity and ease of implementation are priorities.	Flexibility and dynamic memory management are priorities.

23. Explain circular Queue with it advantage and need

Ans.

A **Circular Queue** is a linear data structure that follows the **FIFO (First In, First Out)** principle but differs from a regular queue in its way of handling the overflow condition. In a regular queue, when the queue is full, no more elements can be added. However, in a circular queue, the last position of the queue is connected back to the first position, forming a circle. This allows the queue to efficiently use space by reusing the vacant positions created when elements are dequeued.

Advantages of Circular Queue:

1. **Efficient Space Utilization**: Unlike a regular queue, a circular queue allows the reuse of space that becomes available once elements are dequeued, leading to better space utilization.
2. **No Wasted Space**: It ensures that there is no unused space, even if there are empty spots in the queue.

Need of Circular Queue:

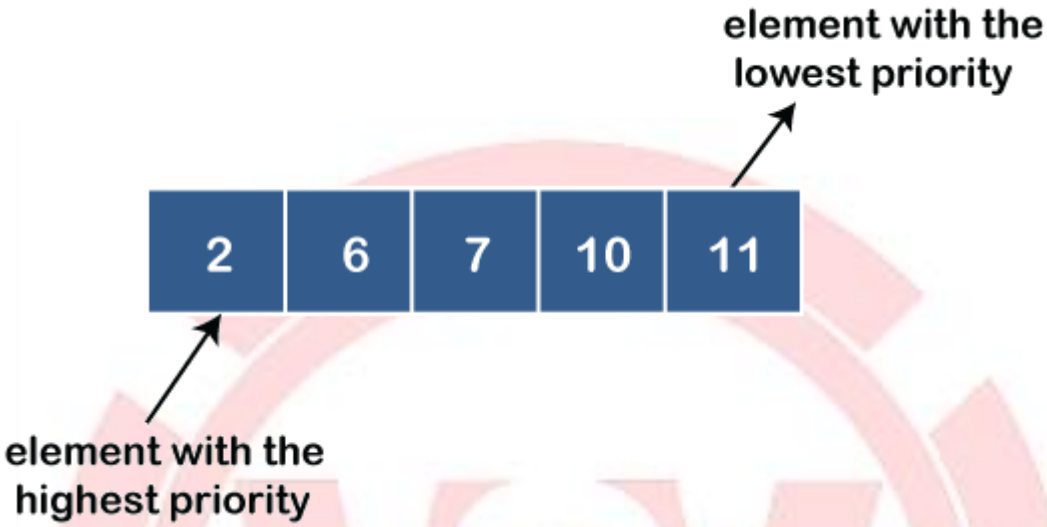
1. **Fixed-Size Buffer**: It is especially useful in scenarios like CPU scheduling or memory management where a fixed-size queue is required, and elements are processed in a continuous loop.
2. **Queue Overflow Prevention**: It prevents overflow from happening prematurely as long as there is available space in the queue.

24. Explain Priority queue with example Ans.

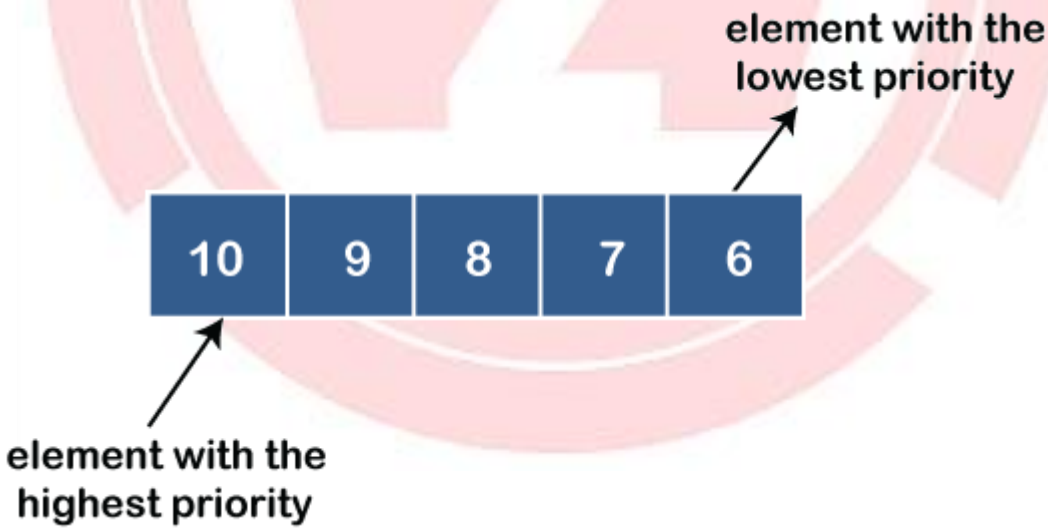
A priority queue is an abstract data type that behaves similarly to the normal queue except that each element has some priority, i.e., the element with the highest priority would come first in a priority queue. The priority of the elements in a priority queue will determine the order in which elements are removed from the priority queue. Types of Priority Queue

There are two types of priority queue:

- **Ascending order priority queue:** In ascending order priority queue, a lower priority number is given as a higher priority in a priority. For example, we take the numbers from 1 to 5 arranged in an ascending order like 1,2,3,4,5; therefore, the smallest number, i.e., 1 is given as the highest priority in a priority queue.



- **Descending order priority queue:** In descending order priority queue, a higher priority number is given as a higher priority in a priority. For example, we take the numbers from 1 to 5 arranged in descending order like 5, 4, 3, 2, 1; therefore, the largest number, i.e., 5 is given as the highest priority in a priority queue.



Representation of priority queue

Now, we will see how to represent the priority queue through a one-way list.

We will create the priority queue by using the list given below in which **INFO** list contains the data elements, **PRN** list contains the priority numbers of each data element available in the **INFO** list, and LINK basically contains the address of the next node.

25. Explain Recursion with Factorial Program Ans.

Recursion is a programming technique where a function calls itself to solve a problem by breaking it down into smaller sub-problems. Each recursive function has:

1. **Base Case**: A condition to stop the recursion.
2. **Recursive Case**: The part where the function calls itself with a smaller or simpler version of the problem.

Factorial Program Using Recursion:

The **factorial** of a number n (denoted as n!) is the product of all positive integers less than or equal to n. It is defined as:

- n! = n * (n-1) * (n-2) * ... * 1
- Base case: 0! = 1 **Recursive Formula:**
- For n > 0, n! = n * (n-1)!
- Base case: 0! = 1

C Program to Compute Factorial Using Recursion:

```
#include <stdio.h>

// Function to calculate factorial using recursion int
factorial(int n) {
   if (n == 0) return 1;  // Base case: 0! = 1     return n *
factorial(n - 1);  // Recursive case: n! = n * (n-1)!
}

int main() {     int number;
printf("Enter a number: ");
scanf("%d", &number);
   printf("Factorial of %d is %d\n", number, factorial(number));
return 0; }
```

Explanation:

- **Base Case**: When n == 0, the function returns 1 (since 0! = 1).
- **Recursive Case**: For n > 0, the function calls itself with n-1 and multiplies it by n.

Example: For factorial(5), the function will compute 5 * 4 * 3 * 2 * 1.

Example Output:
mathematica
Copy code
Enter a number: 5
Factorial of 5 is 120

26. Define and Explain term
    - Leaf Node
    - Root Node
    - Path
    - Ancestor
    - Descendants
    - Level of Node
    - Application of Queue
    - Algorithm
    - Queue Operation
    - Stack Operation
    - Link List Operation
    - Array Operation
    - Data Structure Operation
    - Overflow of Stack
    - Underflow of Queue Ans.

1. **Leaf Node**: A leaf node is a node in a tree that has no children.
2. **Root Node**: The root node is the topmost node in a tree, from which all other nodes originate.
3. **Path**: A path is a sequence of nodes connected by edges in a tree or graph.
4. **Ancestor**: An ancestor of a node is any node on the path from the root to that node.
5. **Descendants**: Descendants are all the nodes that can be reached by traversing down from a given node.
6. **Level of Node**: The level of a node is the number of edges from the root to that node.
7. **Application of Queue**: A queue is used in scenarios like task scheduling, BFS in graphs, and request handling.
8. **Algorithm**: An algorithm is a set of step-by-step instructions designed to solve a problem or perform a task.
9. **Queue Operation**: Queue operations include enqueue (insertion), dequeue (removal), and peek (viewing front element).
10. **Stack Operation**: Stack operations include push (insertion), pop (removal), and peek (viewing top element).
11. **Linked List Operation**: Linked list operations include insertion, deletion, traversal, and search of nodes.
12. **Array Operation**: Array operations include insertion, deletion, traversal, and search of elements.
13. **Data Structure Operation**: Data structure operations involve manipulating the elements of structures like arrays, stacks, queues, etc.
14. **Overflow of Stack**: Stack overflow occurs when an attempt is made to push an element onto a full stack.
15. **Underflow of Queue**: Queue underflow occurs when an attempt is made to dequeue from an empty queue.