Question Bank

Q1- How does ADT help in programming?

Ans- Abstract Data type are incredibly valuable in programming because they provide a way to define data structures and their operations abstractly, without focusing on the implementation details.

- Organizing data efficiently
- Hiding implementation details
- Improving code reusability
- Enhancing performance

& Example of ADTs.

Q2- Briefly explain the functions of DMA?

Ans- Dynamic Memory Allocation in programming allows you to allocate and manage memory during runtime. Its functions are:

- malloc(): Allocates memory without initialization.
- calloc(): Allocates memory and initializes it to zero.
- realloc(): Changes the size of allocated memory.
- free(): Releases allocated memory.

Q3- Differentiate between Array and Linked List.

Ans-

Array	Linked List
1. Stored in contiguous memory locations.	1. Stored in scattered locations, linked using pointers.
2. Access Speed fast	2. Access speed slow
3. Insertion/Deletion Difficult	3. Insertion/Deletion Easy
4. Can waste memory if unused.	4. Efficient, takes only needed space.
5. Fixed Size	5. Dynamic Size.

Q4- What is Asymptotic Notation?

Ans- Asymptotic Notation describes the efficiency of an algorithm as input size (n) increases. It helps in analyzing performance without running the code.

Common Asymptotic Notations -

- $O(1)$ - Constant time
- $O(n)$ - Linear time
- $O(\log n)$ - Logarithmic time
- $O(n^2)$ - Quadratic time

There are three key notations -

1- Big O (O) - Upper Bound Case - Worst Case.

2- Big Omega (Ω) - Lower Bound Case - Best Case

3- Big Theta (Θ) - Tight Bound Case - exact growth rate.

Q5- Real World Examples of Asymptotic Notation.

Ans -

- $O(1)$ - Constant time → Looking at time on a wristwatch.
- $O(n)$ - Linear time → Searching for a name in unsorted list.
- $O(\log n)$ - Logarithmic time → Finding a word in a dictionary.
- $O(n^2)$ - Quadratic time → Comparing every student's marks with every other student's marks.

Q6- Time Complexity of Linear and Binary Search.

Ans -

Algorithm	Best Case	Worst Case	Average Case.
Linear	$O(1)$ (First element found)	$O(n)$ (Last element or not found)	$O(n)$
Binary	$O(1)$ (Middle Element found)	$O(\log n)$	$O(\log n)$

Q7- What the key condition for applying binary search?

Ans-
- The array must be sorted before applying binary search.
- It works by dividing the array into halves and checking if the middle element matches the search key.
- The data structure should allow direct access to elements.
- The array should have a defined start and end for determining middle element.

Q8- Prove that ADT enhance programming efficiency and development?

Ans- Reusability → The same ADT can be used in multiple applications.

Efficiency → ADTs optimize data handling, reducing execution time.

Flexibility → ADTs can be implemented using different data structures.

Q9- Discuss linear and non-linear data structures with example.

Ans.

Linear	Non-Linear
1. Data is stored sequentially.	Data is stored in a hierachical or interconnected manner.
2. Can be traversed in a single run.	2. Requires multiple runs to traverse completely.
3. Uses Contiguous memory, making it easier to manage	3. Uses non-contiguous memory requiring pointers.
4. Ex- Array, Linked list, etc.	4. Ex- Tree, Graph, etc.

Q10- Describe various form of data structure.

Ans- 1- Primitive data structures- Basic data types used to store single values.

- Usage- Basic data type and manipulation.
- Example- Integer, float, Character, Boolean.

2- Non-Primitive data structures- Complex structures used to store multiple values.

a) Linear data type - These store data in a sequential manner.

eg- Array, Linked List, Stack, Queue

b) Non-Linear data type - Data is ~~stored~~ arranged in a

hierarchical or interconnected manner.
eg- Tree, Graph, etc.

Q11. Define data structure and also describe the difference between primitive and non-primitive data structure.

Ans- The logical or mathematical model of a particular organization of data is called a data structure. It is classified into two -

Primitive	Non-primitive
1. Basic data types provided by a programming lang.	1. Complex data structures derived from primitive type.
2- Built-in data types	2- User-defined data types combining primitive types.
3- Simple Complexity	3- More Complex
4- Fixed size memory requirement.	4- Memory requirement can grow dynamically.
5- Simple operations	5- Complex operations.
6- Ex- int x = 10;	6- Ex- int arr[5] = {1,2,3,4,5};

Q12- List the applications using stack data structure.

Ans- A stack follows LIFO (Last In First Out) principle. Some applications include:

a- Undo/Redo Feature → Used in text editors and design tools.

b- Expression Evaluation → Used to evaluate mathematical expressions

c- Browser History → Stores previous webpages visited.

d- Memory Management → Manages local variables and function calls in the stack memory.

Q13- Explain following mathematical notation:
(i) Infix (ii) Prefix (iii) Postfix

Ans- (i) Infix - The operator is placed between operands.
- Human readable, but not suitable for computer execution without conversion.
- Ex - (A+B) * C

(ii) Prefix - The operator is placed before the operands.
- Used in expression evaluation by computers without paranthesis.
- Ex - *+ABC

(iii) Postfix - The operator is placed after the operands.
- Used in stack-based computations & calculators.
- Ex - AB+C*

Q14- Write an algorithm to convert prefix expression to postfix expression. Carefully state any assumption you make regarding the input.

Ans- STEP 1- START
STEP 2- Reverse the prefix expression
STEP 3- Scan from left to right
STEP 3.1- If an operand, push it onto the stack
STEP 3.2- If an operator, pop two operands, form a postfix expression, and push it back.
STEP 4- Final Stack Value is the postfix expression.

Ex- Prefix : * + A B C
Reversed : C B A + *
Conversion :
- Push C B A in stack
- Encounter + → Pop A, B and form AB+ and push back
- Encounter * → Pop AB+, C and form AB+C*
- Postfix → AB + C *

Q15- Define Priority queue. How does it differ from a regular queue?

Ans-

A Priority Queue is a special type of queue where each element has a priority, and elements with higher priority are dequeued before those with lower priority.

Regular Queue	Priority Queue.
1. Follows FIFO (First In, First Out)	1. Dequeues elements based on priority.
2. Elements are added at the rear.	2. Elements are added based on priority order.
3. Elements are removed from the front.	3. Highest-priority element is removed first.
4. Ex- Ticket booking system, etc.	4. Ex- Emergency room in Hospital, etc.

Q16- Explain the linked list implementations of enque & dequeue operations in queue using C program.

Ans- Enqueue (Insert at Rear)
- create a new node
- If queue is empty, set front & rear to the new node.
- else, link the new node at the end and updated the rear.

Dequeue (Remove from front)
- If queue is empty, return error
- Store the front node's value
- Move the front to the next node.
- free the removed node.

Enqueue

```
#include <stdio.h>
void main ()
{
    int i, n, f=-1, r=-1;
    printf("Enter size of queue \n");
    scanf("%d", &n);
    int Queue[n];
    printf("Enter elements in queue");
    for(i=0; i<n; i++)
    {
        if(f==-1 || r==-1)
        {
            f=r=0;
            scanf("%d", &Queue[i]);
        }else
        {
            r=r+1;
            printf("Enter
            scanf("%d", &Queue[r]);
        }
    }
    for(i=0; i<n; i++)
    {
        printf("%d", Queue[i]);
    }
}
```

Dequeue -

```
#include <stdio.h>
void main()
{
    int f = -1, r = -1, value;
    if (f == -1)
    {
        printf ("Queue is empty");
    }else if (f == r)
    {
        printf ("%d dequeued from queue", Queue[f]);
        for (int i=0; i<r; i++)
        {
            Queue[i] = Queue[i+1];
        }
        r--;
        if (r == -1)
        {
            f = -1;
        }else
        {
            f=0;
        }
    }else
    {
        printf ("Queue : ");
        for (int i=f, i<=r; i++)
        {
            printf ("%d", Queue[i]);
        }
        print ("\n");
    }
```

Q17- List the basic operations that can be performed on a queue?

Ans-
- Enqueue → Insert an element at the rear
- Dequeue → Remove an element from the front
- Front → Get the front element
- Rear → Get the rear element
- IsEmpty → Check if the queue is empty.
- IsFull → Check if the queue is full.

Q18- How do you create a queue in C? Provide an example.

Ans- STEP 1- Define an array to store the queue elements.
STEP 2- Use two pointers, front and rear.
STEP 3- Implement enqueue, dequeue, isEmpty, is full operations.

Program Piche likha hai

Q19. What is a circular queue? How does it differ from a linear queue?

Ans. A circular queue is special type of queue in which the last position is connected to the first, forming a loop, allowing efficient reuse of space when elements are removed.

Linear Queue	Circular Queue
1. Uses simple array or linked list.	1. Uses a circular array
2- Moves only foward	2. Wraps around if space is available.
3. Wastage of Space	3. No wastage of space
4. Ex- Normal queue in ticketing systems	4. Ex- CPU scheduling

Q20. Explain the enqueue and dequeue operations in a circular queue.

Ans- Enqueue.

STEP1- Check if the queue is full
STEP2- If empty, set $f = r = 0$
STEP3- else, set $r = (r+1) \% size$ and insert the element

Dequeue

STEP1- Check if the queue is empty.
STEP2- Retrieve the front element
STEP3- If only one element was left, reset $f = r = -1$.
STEP4- else, set $f = (f+1) \% size$

Q21. Difference between Linear Search and Binary Search algorithm.

Ans.

Linear Search	Binary Search
1. Searches sequentially, one by one.	1. Searches by dividing the array into halves.
2. Found at first index O(1), Best Case	2. Found in the middle, O(1), Best Case
3. Searches all element O(n), worst case	3. Keeps dividing till one element remains O(log n), worst case
4. Slow for large data sets	4. Faster for large datasets
5. Simple and easy	5. Requires more logic

Q22. Most commonly used operations in data structures?

Ans - 1. Insertion
- Add an element to the data structure
- ex - Insert a node in linked list.

2. Deletion:
- Remove an element from the data structure.
- ex - Delete a node from linked list

3. Traversal
- Visit each element in the data structure.
- Ex - Traverse a linked list

4. Searching
- Find an element in a data structure
- Ex - Search for a value in array

5- Sorting • Arrange elements in a specific order.
• Ex - Sort an array using quick sort.

6- Merging • Combine two data structures into one.
• Ex - Merge two sorted arrays.

7- Splitting • Divide a data structure into multiple parts.
• Ex - Split a linked list into two halves.

Q23- What do you mean by worst case time complexity of algorithm. Explain clearly.

Ans- • The worst-case time complexity of an algorithm represents the maximum time it can take to execute for a given input size.

• Ex- Linear Search : O(n) (worst case when the element is at the end or not present)

Binary Search : O(log n) (worst case when the element is in a large dataset and needs full traversal).

Q24. How does a dequeue differ from a normal queue?

Ans-

Queue	Dequeue (Double-ended queue)
1- Insertion only at the rear	1- Insertion at both front & rear
2- Deletion only from front	2- Deletion from both front & rear.
3- Ex - Ticket Counter	3- Ex - Task scheduling.

Q25: What are the advantages and disadvantages of doubly linked list over singly linked list

Ans - Advantages -

1- Can move forward and backward
2- Faster Insertion / Deletion.

Disadvantages

1- Uses more memory
2- More Complex

Question- Bank DSA

Q26- Explain the divide and conquer strategy?

Ans- The divide and conquer strategy is a method used to solve complex problems by breaking them down into smaller, easier to solve subproblems. It follows three main steps -

1- Divide - Split the main problem into smaller subproblems that are similar to the original but simpler.

2- Conquer - Solve each subproblem individually.

3- Combine - Merge the solutions of the subproblems to get the final answer for the original problem.

It is important -

- Reduces Complexity by tackling small instances.
- Often leads to faster algorithms with better time complexity
- Fits naturally with recursive programming.

Ex - Quick Sort, Binary Search, etc.

Q27- Difference between B tree and B+ tree.

Ans- P.T.O →

	B Tree	B+ Tree
1-	Data is stored in both internal and leaf nodes.	Data is stored only in the leaf nodes.
2-	Leaf Nodes are not linked.	Leaf Nodes are Linked.
3-	Traversal is slower for range queries.	Traversal is faster for range queries.
4-	Generally taller in Height.	Generally Shorter in Height
5-	May need to search both internal and leaf nodes.	Always searches at the leaf level for actual data.
6-	No redundancy, each key appears only once.	Redundant keys
7-	Slightly more complex	Easier Accessing
8.	Slightly better memory usage.	Requires more memory

Q28- Discuss the property of AVL tree in the context with binary search trees.

Ans- AVL tree is a special type of Binary Search Tree (BST) that is self-balancing.

NORMAL BST PROBLEM -

- In a regular BST, if you insert elements in sorted order, it can become skewed, leading to O(n) search time.

AVL TREE SOLUTION -

- An AVL Tree automatically keeps itself balanced, ensuring search, insert, and delete operations always take O(log n) time.

→ **Properties of AVL Tree -**

- **Height-Balanced** - for every node, the height difference between its left and right subtree is at most 1.
- **Balanced factor** - Defined as BF = LST Height - RST Height Must be -1, 0, or 1 for every node.
- **Strictly Balanced** - More strictly balanced than Red-Black Trees.
- **Search, Insert, Delete Complexity** - Guranteed O(log n) due to balancing after every operation.
- **BST Property Maintained** - Like any BST, for each node, all keys in the left subtree are smaller, and all keys in the right subtree are larger.

Q29- Explain two traversal strategies used in traversing a graph?

Ans- 1- BREADTH FIRST SEARCH (BFS)

- Explore the graph level by level
- Start at a source node, visit all its neighbours first then their neighbors, and so on.
- Queue data structure used (FIFO)
- Good for finding shortest path.
- Visits nodes in the order they are 'closest' to the starting node.

Ex- If you are looking for a friend at a party, you first talk to people nearby before moving further away.

2- DEPTH FIRST SEARCH (DFS)

- Explore as deep as possible before backtracking
- Start at a source node, go along one path until you can't go further, then backtrack.
- Stack data structure used (LIFO)
- Useful for tasks like detecting cycles, topological sorting, finding connected components.
- Goes deep into one branch before exploring another.

Ex- If you're exploring a maze, you keep going down one corridor until you hit a dead end then backtrack.

Q30- Discuss an unidirected acyclic graph?

Ans- An unidirected acyclic graph is like a web of roads between towns, but no matter which way you travel, you'll never end up circling back to where you started. It is a graph where -

- Edges have no direction
- No cycles exist

Important Points:

→ Undirected: The edges do not have arrows, the connection is mutual.

→ Acyclic: There are no closed loops - you can't circle back to the starting point.

→ Looks like: A collection of connected points without any circles.

→ Special Case: If the graph is connected and acyclic, it becomes a tree.

Q31- Define height and depth of tree?

Ans- Height of a tree - The length of the longest path from the root node down to any leaf node.

- focuses from the root down to the farthest leaf.
- A leaf node has height = 0

~~Depth of a tree~~

Depth of a tree - The distance from the root node to a specific node.

- focuses upward - how many edges

you must travel from the root to reach the node.
• The root itself has depth = 0.

Q32- Define the properties of Graph?

Ans- A graph is a collection of vertices and edges. Its properties define how the graph behaves.

Properties of Graph -

1- A graph consists of a set of vertices and edges.
2- Vertices represent objects; edges represent relationships between them.
*3- For directed graphs, no. of edges coming into a vertex is Indegree, no. of edges going out from a vertex is Outdegree.
4- Edges have no direction in Undirected Graph.
5- A sequence of vertices connected by edges is called path.
6- A graph with no cycles is called acyclic graph.
7- A path that starts and ends at the same vertex without repeating an edge.

Q33- Define Indegree and Outdegree of a graph?

Ans- Indegree - No. of edges coming into the vertex.
eg- If 3 edges point towards a vertex A, then indegree of A = 3.

Outdegree - No. of edges going out from the vertex.
eg- If 2 edges leave a vertex A, then Outdegree of A = 2

Q35

Ans-

Q34. Difference between a Binary Tree and a Binary Search Tree.

Ans-

Binary Tree	Binary Search Tree
1- A tree where each node has atmost two children.	A special type of binary tree where left child < parent < right child
No specific ordering of nodes.	Nodes are ordered based on value
May require checking every node.	Can skip half of the tree each time based on value.
Less efficient for search.	More efficient for search.
Duplicates are allowed freely.	Duplicates are either not allowed or managed carefully.
Mainly used for heirarichal data representation.	Mainly used for efficient searching insertion and deletion.
Eg- Family tree, Charts	Eg- Searching Systems, databases

Q35- What is a circular linked list?

Ans- A circular linked list is a type of linked list where the last node points back to the first node instead of having NULL.
The Nodes form a circle - you can keep traversing the list indefinitely.

- No Null pointer at the end.
- Can be traversed starting from any node
- Useful for applications like circular queues, music playlist, etc.

Q36- List two applications of queues?

Ans- 1- Job Scheduling in Operating System

- Queues manage processes that are waiting to be executed by the CPU.
- Ex - Print queue, where print jobs wait their turn.

2- Handling Requests in Web Servers-

- Web servers use queues to manage incoming client requests in the order they arrive.

Q37- What is a Binary Search Tree?

Ans- A Binary Search Tree is a special type of binary tree where each node follows a specific ordering rule:

- Left Child contains only nodes with values less than the parent node.
- Right Child contains only nodes with values greater than the parent node.

Properties of BST-

→ Every Node has at most two children.
→ Left Subtree has smaller values, right subtree has larger values.
→ No duplicate values.
→ Helps in fast searching, insertion and deletion.

Q38- List any two applications of stacks.

Ans- 1. Expression Evaluation and Syntax Parsing

Stacks are used to evaluate mathematical expressions and check if brackets are balanced.

2. Backtracking Problems

Stacks help in problems like navigating a maze, undoing operations in software.

Q39- Explain Warshall's algorithm for transitive closure of a graph with a step-by-step example.

Ans- Warshall's Algorithm for Transitive Closure -

- Purpose:
To find the transitive closure of a graph, determine if there if a path between any two vertices.
- Transitive Closure:
If there's a path from A to B, we mark a direct connection.

- **Input**
  An adjacency matrix of the graph.

- **Output**
  A matrix that shows reachability between every pair of vertices.

Ex- Suppose you have this graph:

$0 \rightarrow 1 \rightarrow 2$

Adjacency matrix:

	0	1	2
0	0	1	0
1	0	0	1
2	0	0	0

Step 1: $k = 0$
- Check if new paths are made through node 0
- No new paths.

Step 2: $k = 1$
- Now, node 1 can lead to node 2
- Since $0 \rightarrow 1$ and $1 \rightarrow 2$, now $0 \rightarrow 2$ exists!

Step 3: $k = 2$
- Check for paths through node 2
- No new paths.

Final Transitive Closure:

	0	1	2
0	0	1	1
1	0	1	1
2	0	0	0

Q4- Differentiate between a full binary tree and a complete binary tree.

Ans-

	Full Binary Tree	Complete Binary Tree
1-	Every node has 0 or 2 children	All levels are completely filled except possibly the last level, which is filled from left to right
2-	Each node has either 0 or exactly 2 children.	Nodes can have 0, 1, or 2 children, but structure must be left filled.
3-	Looks perfectly balanced at each non-leaf node.	Might not look perfectly balanced, but no gaps allowed before filling left side.
4-	Ex- family tree with each parent having exactly 2 children or none.	Ex- A Heap uses complete binary tree structure.

Q1 Consider Marks[10][5], B = 2000, W = 2, Compute Marks[8][5], assuming column-major order.

Ans- $A[i][j] = B + W * ((j - LC) * M + (i - LR))$

$= 2000 + 2 * (5 * 11 + 8)$

$= 2000 + 2 * (440)$

$= 2000 + 880$

$= 2880.$

Q2 Explain sparse matrix and the ways in which it is represented in the memory?

Ans- A sparse matrix is a matrix where most of the elements are zero. Instead of storing all elements, we store only the non-zero elements to save memory.

$$\begin{bmatrix} 0 & 0 & 5 \\ 0 & 3 & 0 \\ 1 & 0 & 0 \end{bmatrix} =$$

R	3	2	1
C	1	2	3
V	1	3	5

Ways to represent a sparse matrix -

1- Array Representation -

- Stores only non-zero elements along with their row and column indices.
- Ex- R 0 1 2
  C 1 3 2
  V 5 8 6

2- Linked List Representation -

- Each non-zero element is stored as a node with row, column, value, next.
- Efficient for dynamic operations (insertion, deletion)

Q3- Explain doubly linked list. Write the algorithm for insertion operation in doubly linked list.

Ans- A doubly linked list is a linked list where each node contains two pointers -

- prev - Points to previous nodes
- next - Points to next nodes

Algorithm

```
void insertatbeginning(struct node **head, int value)
{
    struct node *newnode;
    newnode=malloc(sizeof(struct node));
    newnode->data = value;
    newnode->prev = NULL;
    newnode->next = *head;
    if(*head != NULL)(*head)->prev = newnode;
    *head = newnode;
}
```

Q4. - Define stack and its implementation in C.

Ans - A stack is a linear data structure that follows the Last in first out (LIFO) principle. This means the last element added to stack is the first one to be removed. It has two main operations:

1- PUSH - Add an element to top of stack
2- POP - Remove the top element from stack

```
#include <stdio.h>
#define SIZE 5
int stack[SIZE], top = -1;
void push (int val)
{
    if (top == SIZE-1)
    {
        printf("Stack Overflow");
    }else{
```

```
        {
            stack[++top] = val;
        }
    }
}
void pop()
{
    if (top == -1)
    {
        printf("Stack underflow");
    }else
    {
        printf("Popped = %d\n", stack[top--]);
    }
}
void display()
{
    if (top == -1)
    {
        printf("Stack is empty");
    }else
    {
        for (int i= top; i>=0; i--)
        {
            printf("%d", stack[i]);
        }
    }
}
int main()
{
    push(10); push(20); push(30);
    display();
```

```
pop();
display;
return 0;
}
```

Q.5 Explain different operations on stack with algorithm & program.

Ans. • PUSH (Insertion)

STEP1- If TOP = SIZE - 1
STEP2- Write Overflow and Exit
STEP3- else
STEP4- stack [++top] = value
STEP5- Exit

```
void push (int value)
{
    if (top == size-1)
    {
        printf("Overflow");
    }else
    {
        stack [++top] = value;
    }
}
```

- **POP (Delution)**

STEP1- if top = = -1
STEP2- write underflow and exit
STEP3- else
STEP4- print Popped value = ___ , stack[top--]
STEP5- exit.

```
void pop()
{
    if (top = = -1)
    {
        printf ("Underflow");
        return;
    }else
    {
        printf ("Popped = %d\n", stack[top--]);
    }
}
```

- **Searching (Peek)**

STEP1- Traverse the stack from top to bottom.
STEP2- If the element is found reverse the position.

```
int search (int key)
{
    for (int i=top; i>=0; i--)
    {
        if (stack[i] = = key)
        {
            return i;
```

006 (((A+B)(C-D))/E)-(F(G+H)/I+J*K)

Symbol	Stack	Expression
(	(	
(	((	
(	(((	
A	(((	A
+	(((+	A
B	(((+	AB
)	((	AB+
(	(((	AB+
C	(((	AB+C
-	(((-	AB+C
D	(((-	AB+CD
)	((	AB+CD-
)	(	AB+CD-
/	(/	AB+CD-
E	(/	AB+CD-E
)	(	AB+CD-E/
-	-	AB+CD-E/
(	-(	AB+CD-E/
F	-(	AB+CD-E/F
(	-((	AB+CD-E/F
G	-((	AB+CD-E/FG
+	-((+	AB+CD-E/FG
H	-((+	AB+CD-E/FGH
)	-(	AB+CD-E/FGH+
/	-(/	AB+CD-E/FGH+
I	-(/	AB+CD-E/FGH+I
+	-(/+	AB+CD-E/FGH+I

J	-(+	AB+CD-E/FGH+I/J
*	-(+*	AB+CD-E/FGH+I/J
K	-(+*	AB+CD-E/FGH+I/JK*
)	-	AB+CD-E/FGH+I/JK*+
		AB+CD-E/FGH+I/JK*+-

Q1- Write a C program to implement insertion of element at end node circular linked list for following functions.

Ans-

```
# include <stdio.h>
# include <stdlib.h>
struct Node {
        int data;
        struct Node * next;
};
void main ()
{
        struct Node * head = NULL;
        struct Node * temp, * newnode;
        int n, i, data;
        printf ("How many nodes you want to insert");
        scanf ("%d", &n);
        for (i=0; i<n; i++)
        {
                printf ("Enter data : ");
                scanf ("%d", &data);
                newnode = malloc (sizeof (struct Node));
                newnode -> data = data;
                newnode -> next = NULL;
                if (head == NULL)
                {
                        head = newnode;
                        newnode -> next = head;
                } else {
```

```
        {
            temp = head;
            while (temp->next != head)
            {
                temp = temp->next;
            }
            temp->next = newnode;
            newnode->next = head;
        }
    }
    if (head == NULL)
    {
        printf("List is empty\n");
    } else
    {
        temp = head;
        printf("Circular Linked List: ");
        while(1) {
            printf("%d", temp->data);
            temp = temp->next;
            if (temp == head)
            {
                break;
            }
        }
    }
}
```

Q 8. Explain with necessary algorithm the representation of stack using linked list and array.

Ans 1- Stack Representation using Array

- Uses a fixed-size array to store elements
- The top variable keeps track of the last inserted element.
- Operations :
  - PUSH → Add element to stack
  - POP → Remove element from stack.

PUSH

STEP1- if top == MAX - 1
STEP2- Write Overflow and exit
STEP3- else
STEP4- stack [++top] = value
STEP5- exit

POP

STEP1- if top == -1
STEP2- write underflow and exit
STEP3- else
STEP4- stack [top--]
STEP5- exit

2) Stack implementation using linked list

- Uses dynamic memory allocation with a linked list.
- Every element is stored in a node containing:
  - Data
  - Pointer to next node.
- Operations:
  - PUSH: Insert element at the top
  - POP: Remove the top element

Algorithm -

PUSH ✓

STEP1- Create a new node

STEP2- newnode[next] = top

STEP3- Set newnode = top

STEP4- Exit

POP

STEP1- If stack is empty

STEP2- write underflow and exit.

STEP3- Store top node in temp.

STEP4- Move top pointer to next node

STEP5- Free temp node

STEP6- Exit

Q9- Algorithm to convert postfix to infix: 752+*415-/-
(i) Find value of expression
(ii) Prefix of expression.

Ans 9- Algorithm to Convert Postfix to Infix.

STEP 1- Initialize an empty stack.

STEP 2- Read the postfix expression from left to right

STEP 3- For each character in postfix expression

STEP 4- If character is an operand, push it onto the stack

STEP 5- If the character is an operator, pop the top two elements from the stack.

STEP 6- After reading the entire postfix expression, the stack will contain one element, which is the infix expression

Ex → 752+*415-/-

(i)
= ((7*(5+2))
= ((7*(5+2))4(1-5)/-
= ((7*(5+2))(4/(1-5)-
= ((7*(5+2))-(4/(1-5))
= (((7*7))-(4/-4)
= ((49)-(-1))
= 49+1
= 50.

(((7*(5+2))-(4/(1-5))

Symbol	Stack	Expression
(	(	
(	((	
(	(((	
7	(((	7
*	(((*	7
(	(((*(	7
5	(((*(	75
+	(((*(+	75
2	(((*(+	752
)	(((*	752+
)	((	752+*
-	((-	752+*
(	((-(	752+*
4	((-(	752+*4
/	((-(/	752+*4
(	((-(/(	752+*4
1	((-(/(	752+*41
-	((-(/(-	752+*41
5	((-(/(-	752+*415
)	((-(/	752+*415-
)	((-	752+*415-/
		752+*415-/-

Q9. $(a+b)(c-d)-(g+j)(h-i)/(m+n)$

Symbol	Stack	Expression
(	(	
a	(	a
+	(+	a
b	(+	ab
)		ab+
(	(	ab+
c	(	ab+c
-	(-	ab+c
d	(-	ab+cd
)		ab+cd-
-	-	ab+cd-
(	-(	ab+cd-
g	-(	ab+cd-g
+	-(+	ab+cd-g
j	-(+	ab+cd-gj
)	-(	ab+cd-gj+
(	-(	ab+cd-gj+
h	-(	ab+cd-gj+h
-	-(-	ab+cd-gj+h
i	-(-	ab+cd-gj+hi
)	-	ab+cd-gj+hi-
/	-/	ab+cd-gj+hi-
(	-(/	ab+cd-gj+hi-
m	-(/	ab+cd-gj+hi-m
+	-(/+	ab+cd-gj+hi-m
n	-(/+	ab+cd-gj+hi-mn
)	-/	ab+cd-gj+hi-mn+
		ab+cd-gj+hi-mn+-

Algorithm

STEP1- Initialize an empty stack
STEP2- Scan the infix expression
STEP3- If an operand appears, add it to expression
STEP4- If an operator appears, push it to stack
STEP5- If an opening parenthesis (appears, push it to stack
STEP6- If a closing parenthesis) appears, ~~push it~~ pop from stack
STEP7- At the end, pop and add all remaining operators from stack
STEP8- exit

Q10- Queue using Linked List

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

#include <stdio.h>
#include <stdlib.h>
void main ()
{
    Struct Node
    {
        int data;
        struct Node * next;
    };
    struct Node * front = NULL, * rear = NULL, * temp,
    * newnode;
    int choice, value;
    while (1)
    {
        printf ("\n 1. Enqueue\n 2. Dequeue\n 3. Display\n
                4. Exit \n");
        printf ("Enter your choice: ");
        scanf ("%d", &choice);
        if (choice == 1)
        {
            newnode = malloc (sizeof (struct Node));
            if (!newnode)
            {
                printf ("Memory allocation failed");
                continue;
```

```
    }
    printf("Enter value: ");
    scanf("%d", &newnode->data);
    if (rear == NULL)
    {
        front = rear = newnode;
    }else
    {
        rear->next = newnode;
        rear = newnode;
    }
}else if (choice == 2){
    if (front == NULL){
        printf("Queue is empty");
    }else{
        temp = front;
        front = front->next;
        printf("Dequeued value: %d\n",
               temp->data);
        free(temp);
        if (front == NULL){
            rear = NULL;
        }
    }
}else if (choice == 3){
    if (front == NULL){
        printf("Queue is empty");
    }else{
        temp = front;
        printf("Queue: ");
        while (temp != NULL){
```

Q11- Priority Queue?

Ans- A Priority Queue is a specialized data structure where each element is associated with a priority. Elements are served based on their priority rather than their order of insertion.

In Regular Queue, elements are dequeued in same order as they are enqueued.
whereas.
In Priority Queue, elements are dequeued based on their priority, not their insertion order.

Q12- Give the algorithm for Binary Search. What is its time complexity? Compare its time complexity with that of linear search?

Ans- Algorithm -

STEP 1- Start with a sorted array
STEP 2- Set low = 0 and high = size - 1
STEP 3- Repeat until low <= high
STEP 4- a: find mid = (low + high) / 2
STEP 5- b: if arr[mid] = key, return index
STEP 6- c: if arr[mid] > key, search left half (high = mid - 1)
STEP 7- d: if arr[mid] < key, search right half (low = mid + 1)
STEP 8- if not found, return -1.
STEP 9- Exit

Aspect	Linear Search	Binary Search
Array type	Unsorted or Sorted	Only Sorted
Best Case	$O(1)$	$O(1)$
Average Case	$O(n)$	$O(\log_2 n)$
Worst Case	$O(n)$	$O(\log_2 n)$
Method	Check each element	Divide & Conquer

Q30- How stacks are used to convert infix to postfix or prefi

Ans. Infix to Postfix $(A+B)*(C-D)/E$

Convert $(A+B) \rightarrow AB+$

Convert $(C-D) \rightarrow CD-$

Convert $(AB+)*(CD-) \rightarrow AB+CD-$

Convert (AB+CD-*)/E → AB+CD-*E/

Infix to prefix (A+B)*(C-D)/E

Convert (A+B) → +AB
Convert (C-D) → -CD
Convert (+AB)*(-CD) → *+AB-CD
Convert (*+AB-CD) /E → /*+AB-CDE

Q14- 6 3 + 2 * 4 /

1- Read 6, push onto stack → stack [6]
2- Read 3, push onto stack → stack [6,3]
3- Read +, pop 3 & 6, compute 6+3=9, push result → stack [9]
4- Read 2, push onto stack → stack [9,2]
5- Read *, pop 9 and 2, compute 9*2, push result → stack [18]
6- Read 4, push onto stack → stack [18,4]
7- Read /, pop 18,4, compute 18/2, push result → stack [4.5]
8- Final result is 4.5.

Q15- Conditions that lead stack overflow and underflow, Provide C snippets to demonstrate PUSH & POP.

Ans- • STACK OVERFLOW - occurs when you try to push an element into a full stack.
• STACK UNDERFLOW - occurs when you try to pop an element from an empty stack.

```
void push (int val)
{
    if (top == size - 1)
    {
        printf ("Stack overflow");
    }else
    {
        stack [++top] = value;
    }
}
void pop ()
{
    if (top == -1)
    {
        printf ("Underflow");
    }else
    {
        printf (" Popped : %d\n", stack [top--]);
    }
}
```

16. C program implementing Push, Pop, Peek and Isempty using an array-based stack.

```
void push (int val)
{
    if (top = size - 1)
    {
        printf ("Overflow");
    }else
    {
        stack [++top] = val;
```

```
void pop()
{
    if (top == -1)
    {
        printf ("Stack Underflow");
    }else
    {
        printf ("Popped : %d\n", stack [top--]);
    }
}
int peek ()
{
    return (top == -1) ? -1 : stack [top];
}
int isEmpty ()
{
    return top == -1;
}
```

Q17. Illustrate different types of arrays with syntax and its operations.

Ans 1D - - - -

2D - - - -

Multi-D - - - -

Basic Operations - Insertion
Deletion
Search
Update

Q.18 Condition of Overflow and Underflow in a Singly linked list.

Ans- Overflow- ~~Happens when~~

```
Node *newnode = (Node *)malloc(sizeof(Node));
if (!newnode) printf("Memory Overflow\n");
```

Underflow -

```
if (head == NULL) printf("List Underflow\n");
```

Q.19 Algorithm to convert infix to postfix.

Ans- Likh Lena

Q.20 Why is Binary Search preferred over Sequential Search for large datasets?

Ans-

1. Faster Search Time:
   - Binary : $O(\log n)$
   - Linear : $O(n)$

2. Efficient for sorted data:
   - Binary search works efficiently if the data is already sorted.

3- Less Comparisons:
   - Instead of checking every element binary search directly eliminates half the dataset at each step.

4- Real-world Applications -
- Used in searching databases, dictionaries, etc.

Q21 Illustrate how searching in an unordered list differs from searching in an ordered list with an example.

Ans- Unordered List -

- The elements are not sorted
- Linear search (O(n)) is the only option.
- Ex- Searching 7 in {3, 8, 1, 7, 5}.

Ordered List

- The elements are sorted
- Binary search (O(log n)) is possible, making it faster.
- Ex- Searching 7 in {1, 2, 3, 4, 5, 6, 7, 8}.

Q22 Explain role of searching. List the applications of searching techniques in data structures.

Ans- Searching is used to find a specific element in a collection of data. It is essential in large datasets to retrieve information efficiently. Searching is widely used in databases, search engines, AI Algorithms, and real-time applications.

Applications -

1- Databases 2) AI & ML 3) Search Engine 4) Real-time Application

Q23 a. To count no. of nodes.
b. To reverse direction of links.
c. To delete alternate nodes that is first, third, fifth and so on.

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

a)
```
int countNodes (struct Node * head)
{
    int count = 0;
    struct Node * current = head;
    while (current != NULL)
    {
        count ++;
        current = current -> next;
    }
    return count;
}
```

b. struct Node * reverselist (struct Node *head)
{
    struct Node * prev = NULL;
    struct Node * next = NULL;
    struct Node *current = head;
    while (current != NULL)
    {
        next = current -> next;
        current -> next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

c. void deletenodes (struct Node * head)
{
    Struct Node * current = head;
    while (current != NULL && current -> next != NULL)
    {
        struct Node *temp = current -> next;
        current -> next = temp -> next;
        free (temp);
        current = current -> next;
    }
}

Q23- Illustrate linear and binary search algorithm with example.

Ans- **Linear Search Algorithm**

STEP 1- Initialise: n, i, data, arr[ ]
STEP 2- Take input for size of array and store in n.
STEP 3- Repeat STEP 4 until i<n, increases by 1 unit.
STEP 4- Store value of i in arr[i].
STEP 5- Print Value to be searched.
STEP 6- Store in data.
STEP 7- Repeat STEPS 8 to 9 until i<n, i increases by 1 unit.
STEP 8- if arr[i] = data
STEP 9- Set f = f+1 and break
STEP 10- if (f = 1)
STEP 11- print Value found at : , i
STEP 12- else, print Not found.
STEP 13- Exit

Ex- Array = [5, 12, 8, 4, 10]
Search for : 4

- Compare 5 with 4 → Not equal
- Compare 12 with 4 → Not equal
- Compare 8 with 4 → Not equal
- Compare 4 with 4 → Match found at position 4.

Output: "Element found at index 3"

Ex- Array = [2, 4, 6, 8, 10, 12, 14]
Search for : 10

- low = 0, high = 6
- mid = (0+6)/2 = 3 → arr[3] = 8
  - 10 > 8 → Search right side (low = 4)
- mid = (4+6)/2 = 5 → arr[5] = 12
  - 10 < 12 → Search left side (high = 4)
- mid = (4+4)/2 = 4 → arr[4] = 10
  - 10 == 10 → found at position 4

Output → "Element found at index 4".

024. Given a sequence of number: 32, 1, 5, 10, 23, 20, 41, 55, 2, 9, 44, 30, 25, 17, 28. Apply Merge Sort to sort the Array.

ns-

32 1 5 10 23 20 41 55 2 9 44 30 25 17 28

32 1 5 10 23 20 41 55 2 9 44 30 25 17 28

32 1 5 10 23 20 41 55 2 9 44 30 25 17 28

1 32 5 10 23 20 41 2 55 9 44 25 30 17 28

1 5 32 10 20 23 41 2 9 44 55 17 25 28 30

1 5 10 20 23 32 41 2 9 17 25 28 30 44 55

1 2 5 9 10 17 20 23 25 28 30 32 41 44 55

Q25 Elaborate fundamental operations of a binary search tree, including insertion, deletion, and searching.

Ans- FUNDAMENTAL OPERATIONS of BST

1- Searching

- Compare key with root:
  - if equal → found
  - if smaller → search left
  - if larger → search right
- Time Complexity: Best/Average Case - O(log n)
  Worst Case - O(n)

2- Insertion

- Compare and move left or right like searching.
- Insert at the first null spot found.
- Time Complexity: Best/Average Case - O(log n)
  Worst Case - O(n)

3- Deletion

- Find the node to delete.
- Three cases → No child: delete Node
  → One child: replace node with child
  → Two children: replace node with inorder successor.
- Time Complexity: Best/Average Case = O(log n) | Worst Case = O(n)

Q26- Explain the quick sort algorithm. Discuss the partitioning process, pivot selection strategies, and apply quick sort the given array: 25, 84, 10, 57, 117, 156, 48, 73, 36, 105, 122, 92, 149, 69, 134.

Ans- It is a highly efficient sorting algorithm that uses a "divide and conquer" approach, partitioning an array around a chosen pivot element and then recursively sorting the sub-arrays.

(Hume pta hai sirf algorithm pucha hai pr program bhi likh rhe hai bs incase puch liya to)

```
#include <stdio.h>
void swap (int *a, int *b)
{
        int temp = *a;
        *a = *b;
        *b = temp;
}
int partition (int arr[], int low, int high)
{
        int pivot = arr[low];
        int i = low + 1;
        int j = high;
        while (i<=j)
        {
                while (i<=high && arr[i] <= pivot)
                {
                        i++;
                }
```

```
        while (j >= low && arr[j] > pivot)
        {
            j--;
        }
        if (i<j)
        {
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[low], &arr[j]);
    return j;
}
void quicksort(int arr[], int low, int high)
{
    if (low < high)
    {
        int a = partition(arr, low, high);
        quicksort(arr, low, a-1);
        quicksort(arr, a+1, high);
    }
}
void printArray(int arr[], int n)
{
    for (int i=0; i<n; i++)
    {
        printf("%d", arr[i]);
    }
}
int main()
{
    int n, i;
```

```
    printf("Enter size of array:");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d values in array", n);
    for(i=0; i<n; i++)
    {
        scanf("%d", &arr[i]);
    }
    printf("Unsorted Array : \n");
    printArray(arr, n);
    quickSort(arr, 0, n-1);
    printf("Sorted Array : \n");
    printArray(arr, n);
    return 0;
}
```

Algorithm for Quick Sort

STEP 1- Create a swap function to replace two values.
STEP 2- Create a partition function that selects first element as the pivot element.
STEP 3- Initialize two pointers i = low+1 (To scan left to right)
j = high (To scan right to left)
STEP 4- Repeat STEPS 5 to 10 while i<j
STEP 5- Repeat STEP 6 while i<= high and arr[i] <= pivot
STEP 6- Set i++;
STEP 7- Repeat STEP 8 while j >= low && arr[j] > pivot
STEP 8- Set j--;
STEP 9- if i<j
STEP 10- call swap (&arr[i], &arr[j]);
STEP 11- call swap (&arr[low], &arr[j]);

STEP 12- return j
STEP 13- Create a recursive function quicksort that calls partition (arr, low, high) to find pivot index 'p'.
STEP 14- Call quicksort (arr, low, p-1)
STEP 15- Call quicksort (arr, p+1, high)
STEP 16- Create printArray function to display array.
STEP 17- Repeat STEP 18 till i<n and increase upto 1 unit
STEP 18- print arr[i]
STEP 19- In main() function
STEP 20- Initialise n, i, arr[n], and take input for size of array and store it n.
STEP 21- Take input for values and store it in arr[].
STEP 22- print Unsorted Array by calling printArray (arr, n)
STEP 23- Call quicksort (arr, 0, n-1)
STEP 24- print sorted array by calling printArray (arr, n)
STEP 25- Exit.

Partitioning Process

- Choose a pivot element from array
- Rearrange elements -
  - Move elements less than pivot to its left.
  - Move elements greater than pivot to its right.
- Return pivot's final position.

"Sort Khud Ko Iesa".

Q27- Write and explain selection sort algorithm. Discuss the strategy used for selecting the key and sort the given array: 47, 82, 15, 63, 29, 54, 91, 38, 72, 26, 59, 10, 42, 77, 93 using selection sort.

Ans-

```
#include <stdio.h>
void main()
{
    int n, i, j, min_idx, temp;
    printf("Enter no. of elements");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements: ", n);
    for(i=0; i<n; i++)
    {
        scanf("%d", &arr[i]);
    }
    for(i=0; i<n-1; i++)
    {
        min_idx = i;
        for(j= i+1; j<n; j++)
        {
            if(arr[j] < arr[min_idx])
            {
                min_idx = j;
            }
        }
    temp = arr[min_idx];
    arr[min_idx] = arr[i];
```

```
    arr[i] = temp;
    }
    printf("Sorted Array ");
    for(i=0; i<n; i++)
    {
        printf("%d", arr[i]);
    }
}
```

Selection Sort is a simple, intuitive comparison-based sorting algorithm. It works by repeatedly selecting the minimum element from the unsorted portion and moving it to the sorted portion.

Algorithm for Selection Sort -

STEP 1- Initialize n, i, j, temp, min_idx
STEP 2- Take input of size of array and store in n.
STEP 3- Initialize arr[n], take input of elements
STEP 4- Repeat STEP 5 till i<n and increases by 1 unit.
STEP 5- store the value of i in arr[i].
STEP 6- Repeat STEP 7 to 11 till i<n-1 and increases by 1 unit
STEP 7- Set min_idx = i
STEP 8- Repeat STEP 8 to 9 till j=i+1, j<n and increases by 1 unit.
STEP 9- if (arr[j] < arr[min_idx])
STEP 10- Set min_idx = j
STEP 11- Set temp = arr[min_idx], arr[min_idx] = arr[i], arr[i] = temp
STEP 12- Print Sorted Array
STEP 13- Repeat STEP 14 till i<n increases by 1 unit
STEP 14- Print arr[i]
STEP 15- Exit

Key Selection Strategy

- At each step, the key is the minimum element in the remaining unsorted array.
- We compare all remaining elements to find this key

47 82 15 63 29 54 91 38 72 26 59 10 42 77 93

Min = 10

10 82 15 63 29 54 91 38 72 26 59 47 42 77 93

Min = 15

10 15 82 63 29 54 91 38 72 26 59 47 42 77 93

Min = 26

10 15 26 63 29 54 91 38 72 82 59 47 42 77 93

Min = 29

10 15 26 29 63 54 91 38 72 82 59 47 42 77 93

Min = 38

10 15 26 29 38 54 91 63 72 82 59 47 42 77 93

Min = 42

10 15 26 29 38 42 91 63 72 82 59 47 54 77 93

Min = 47

10 15 26 29 38 42 47 63 72 82 59 91 54 77 93

Min = 54

10 15 26 29 38 42 47 54 72 82 59 91 63 77 93

Min = 59

10 15 26 29 38 42 47 54 59 82 72 91 63 77 93

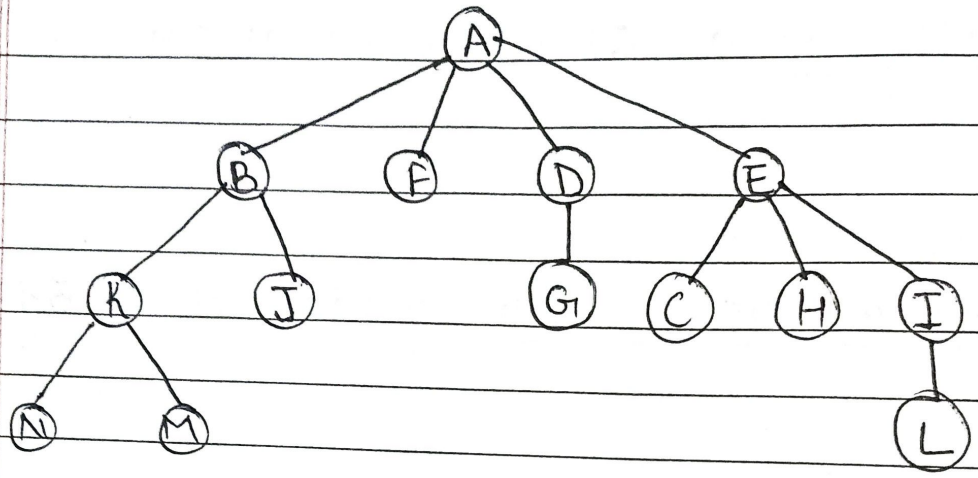
Min = 63

10 15 26 29 38 42 47 54 59 63 72 91 82 77 93

Min = 77

10 15 26 29 38 42 47 54 59 63 72 77 82 91 93

Q28- Perform the following Traversals on the tree - Inorder, Preorder, Postorder.



Ans- 1. In-Order Traversal (Left, Root, Right)

Traverse left subtree → Visit root → Traverse Right subtree

- N K M
- B J
- A F
- G D
- C E H
- L I

N K M B J A F G D C E H L I

Q29.

2- Pre-order Traversal (Root, Left, Right)

visit root → Traverse Left Subtree → Traverse Right Subtree

- A
- B K N M J
- F D G
- E C H I L

A B K N M J F D G E C H I L

3- Postorder Traversal (Left, Right, Root)

Traverse Left Subtree → Traverse Right Subtree → Visit Root

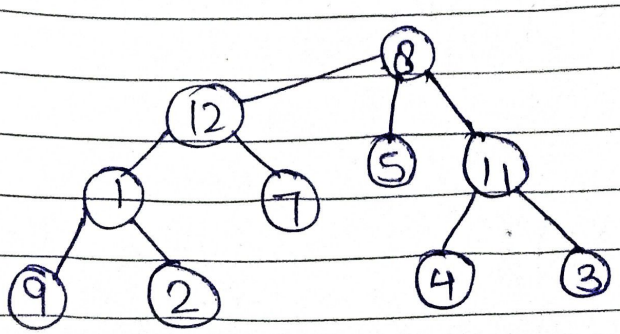
- N M K B J
- F G D
- C H L I E
- A

N M K B J F G D C H L I E A

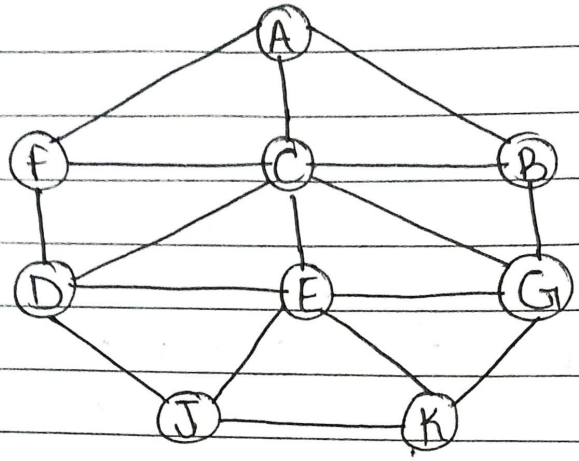
Q29- Construct a binary tree from postorder to inorder

Postorder = 9, 1, 2, 12, 7, 5, 3, 11, 4, 8

Inorder = 9, 5, 1, 7, 2, 12, 8, 4, 3, 11



Q3a- Describe the depth-first search algorithm for traversing graphs. Implement the same for the given graph. Consider J as the source node:



Ans- Depth-first search is a graph traversal algorithm that explores as far as possible along each branch before backtracking. It's typically implemented using a stack.

```
# include <stdio.h>
int stack[5], top = -1;
void push (int value)
{
        stack [++top] = value;
}
int pop()
{
        return stack [top--];
}
int isEmpty()
{
        return top == -1;
}
```

```
void DFS (int adjMatrix [5][5], int visited [5], int start, int n)
{
    push (start);
    while (! isEmpty)
    {
        int vertex = pop();
        if (! visited [vertex])
        {
            printf ("%d", vertex);
            visited [vertex] = 1;
        }
        for (int i = n-1; i >= 0; i--)
        {
            if (adjMatrix [vertex] [i] == 1 && ! visited [i])
            {
                push (i);
            }
        }
    }
}
void main()
{
    int n = 5, visited [5] = {0};
    int adjMatrix [5][5] = {
        {0, 1, 1, 0, 0},
        {1, 0, 0, 1, 1},
        {1, 0, 0, 0, 0},
        {0, 1, 0, 0, 0},
        {0, 1, 0, 0, 0}
    };
    print ("DFS Traversal");
    DFS (adjMatrix, visited, 0, n);
}
```

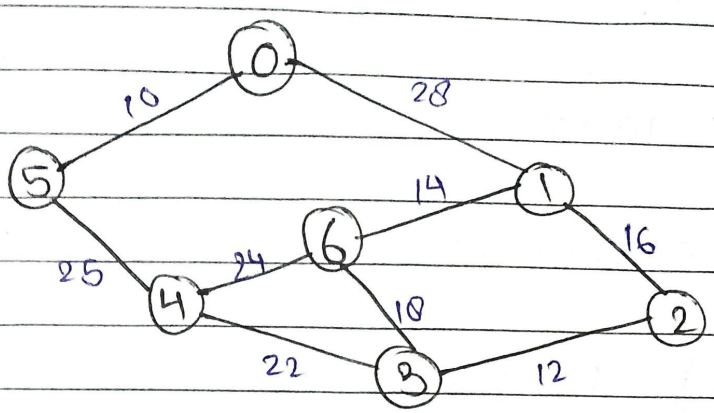
Algorithm -

STEP 1- Initialize stack[5], top = -1
STEP 2- Create a push() function and Set stack[++top] = value
STEP 3- Create a pop() function and return stack[top--]
STEP 4- Create a isEmpty() function and return top == -1
STEP 5- Create a DFS() function consisting adjMatrix[5][5], n, visited[5], start.
STEP 6- push(start)
STEP 7- Repeat while STEPS 8 to 18 not isEmpty.
STEP 8- Set vertex = pop()
STEP 9- if (!visited[vertex])
STEP 10- print vertex, Set visited[vertex] = 1
STEP 11- Repeat STEPS 12 to 13 until i = n-1 & i>=0 decrease by 1
STEP 12- if (adjMatrix[vertex][i] == 1 and !visited[i]
STEP 13- push(i)
STEP 14- Create a main() function
STEP 15- Initialise n=5, visited[5] = {0} also
STEP 16- Initialise adjMatrix[5][5].
STEP 17- print DFS traversal and Set DFS(adjMatrix, visited, 0, n)
STEP 18- Exit

Answer → DFS : J, E, C, A B F G D K

Stack : J
Stack : K D E
Stack : K D G C
Stack : K D G F B A
Stack : K D G F B
Stack : K D G F
Stack : K D G
Stack : K D
Stack : K

Q31- Describe Kruskal's algorithm and find the cost of minimum spanning tree using Prim's algorithm.



Ans- Kruskal's Algorithm is a greedy algorithm used to find the Minimum Spanning Tree of a connected, undirected graph. The goal is to connect all the vertices with the minimum total edge weight without any cycles. The ~~algorithm~~

- Sort all edges in non-decreasing order of weight.
- Add edges to the MST in sorted order, avoiding cycles.
- Stop when you have V-1 edges in MST, where V is no. of vertices.

Prim's Algorithm is another greedy algorithm used to find the MST. It works by growing the MST one edge at a time, starting from an arbitrary vertex.

- Start with an arbitrary vertex and add it to the MST.
- Repeatedly add the smallest edge that connects a vertex in the MST to one outside MST.
- Continue until all vertices are included in MST.

STEP 1- {0}

STEP 2- {0,5} = 10

STEP 3- {0,5,4} = 10+25

STEP 4- {0,5,4,3} = 10+25+22

STEP 5- {0,5,4,3,2} = 10+25+22+12

STEP 6- {0,5,4,3,2,1} = 10+25+22+12+16

STEP 7- {0,5,4,3,2,1,6} = 10+25+22+12+16+14

Total Weight = 99

Q32- Given a sequence of numbers

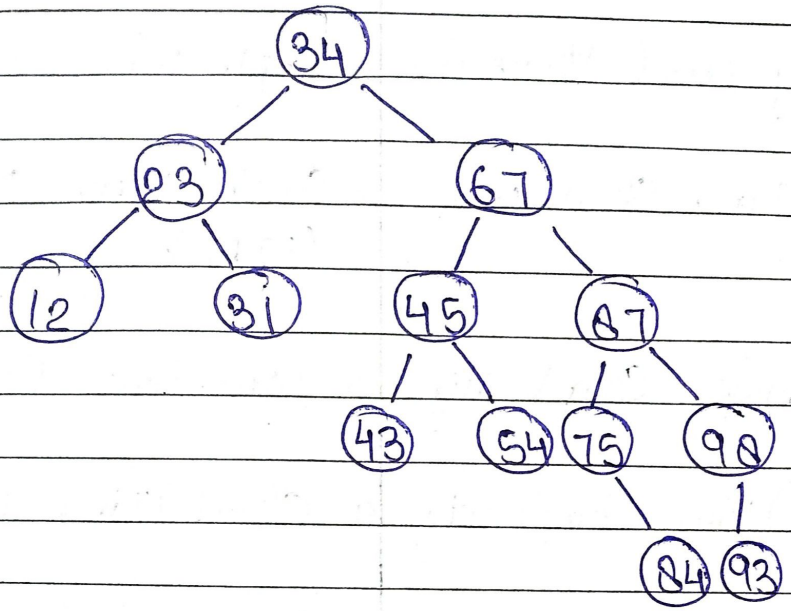
12 3 45 1 56 4 89

Using the Heap sort algorithm, sort the numbers in ascending order.

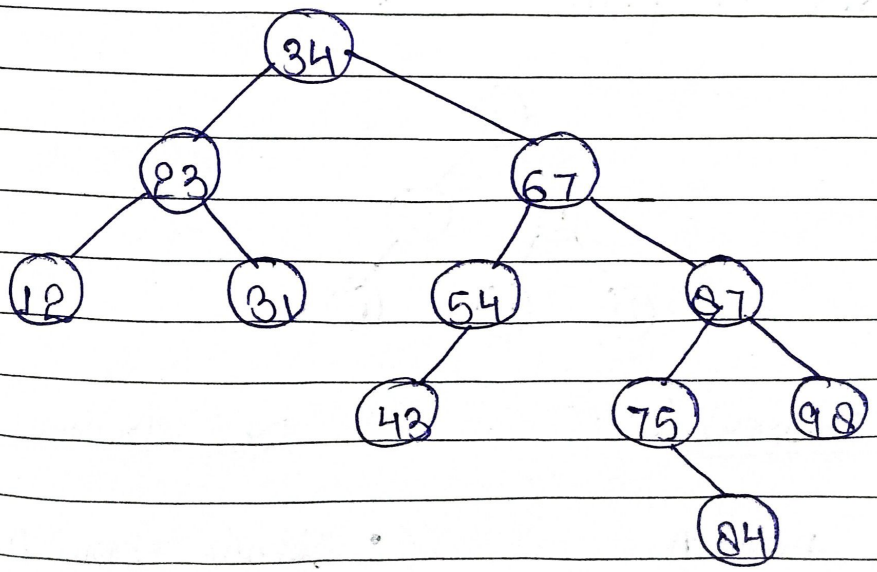
Ans- Syllabus me nhi hai.

Q33- Given a sequence of numbers: 34, 23, 67, 45, 12, 54, 87, 43, 98, 75, 84, 93, 31. Construct Binary Search Tree and delete 45 and 93.

Ans- BST →



After Deleting 45, 93 –

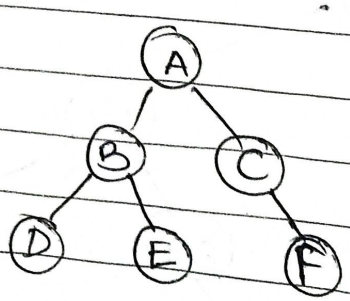


Q34- Describe the difference between DFS and BFS with example.

Ans-

BFS	DFS
1- Level by Level Traversal	1. Depth wise Traversal
2. Queue Data Structure (FIFO)	2. Stack Data Structure (LIFO)
3. High Space Complexity.	3. Low Space Complexity.
4- O(V+E) Time Complexity	4- O(V+E) Time Complexity
5- Shortest path in unweighted graphs.	5- Topological sort, puzzles, cycles.
6- Slightly more complex due to queue	6- Easier Recursion

Ex-



BFS Traversal

- Start from A
- Visit level-by-level

A → B → C → D → E → F

DFS Traversal

- Starts from A
- Go as deep as possible

A → B → D → E → C → F

Q35 Write and explain Bubble sort algorithm. Discuss the strategy used for sort the given array:

47 82 15 63 29 54 91 38 72 26 59 10 42 77 93

using bubble sort.

Ans- Bubble sort is a simple comparison-based sorting algorithm. It works by repeatedly swapping adjacent elements if they are in the wrong order, the largest value to the end of the array in each pass.

```
# include <stdio.h>
void main()
{
    int n, i, j, temp, swapped;
    printf("Enter size of array");
    scanf("%d", &n);
    int arr[n];
    printf("Enter values in array");
    for(i=0; i<n; i++)
    {
        scanf("%d", &arr[i]);
    }
    for(i=0; i<n-1; i++)
    {
        swapped = 0;
        for(j=0; j<n-i-1; j++)
        {
            if(arr[j] > arr[j+1])
```

```
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                swapped = 1;
            }
        }
        if (swapped == 0)
        {
            break;
        }
    }
    printf("Sorted Array");
    for(i=0; i<n; i++)
    {
        printf("%d", arr[i]);
    }
}
```

Algorithm -

STEP 1- Initialise i, j, n, temp, swapped, arr[]
STEP 2- Create an array
STEP 3- Repeat STEP 4 to 10, until i<n-1 increases by 1 unit
STEP 4- Set swapped = 0
STEP 5- Repeat STEP 6 to 10 until j<n-i-1 increases by 1 unit
STEP 6- if arr[j] > arr[j+1]
STEP 7- Set temp = arr[j]
STEP 8- Set arr[j] = arr[j+1]
STEP 9- Set arr[j+1] = temp
STEP 10- Set swapped = 1
STEP 11- if (swapped == 0)

STEP 12 break
STEP 13 print Sorted Array
STEP 14 Repeat STEP 15 until i<n increases by 1 unit
STEP 15 Store value in arr[i].
STEP 16 Exit

Strategy

- Brute force approach
- Does not skip unnecessary comparisons
- Best Case : $O(n)$
- Worst Case : $O(n^2)$

47 82 15 63 29 54 91 38 72 26 59 10 42 77 93

47 82 15 63 29 54 91 38 72 26 59 10 42 77 93

47 15 82 63 29 54 91 38 72 26 59 10 42 77 93

47 15 63 82 29 54 91 38 72 26 59 10 42 77 93

47 15 63 29 82 54 91 38 72 26 59 10 42 77 93

47 15 63 29 54 82 91 38 72 26 59 10 42 77 93

47 15 63 29 54 82 91 38 72 26 59 10 42 77 93

47 15 63 29 54 82 38 91 72 26 59 10 42 77 93

47 15 63 29 54 82 38 72 91 26 59 10 42 77 93

47 15 63 29 54 82 38 72 26 91 59 10 42 77 93

47 15 63 29 54 82 38 72 26 59 91 10 42 77 93

47 15 63 29 54 82 38 72 26 59 10 91 42 77 93

47 15 63 29 54 82 38 72 26 59 10 42 91 77 93

47 15 63 29 54 82 38 72 26 59 10 42 77 91 93

47 15 63 29 54 82 38 72 26 59 10 42 77 91 93

15 47 63 29 54 82 38 72 26 59 10 42 77 91 93

15 47 63 29 54 82 38 72 26 59 10 42 77 91 93

"Ab aage khud kar lena"

Q36- Differentiate between Binary Search and Linear Search ? Apply Binary search to find item 40 in the sorted array : 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99. Also discuss the complexity of binary search.

Ans- Difference Section-A me likha hai

Time Complexity of Binary Search is -

- Best / Avg Case - $O(1)$
- Worst Case - $O(\log_2 n)$

To find 40 in array :

11 22 30 33 40 44 55 60 66 77 80 88 99

$$mid = \frac{low + high}{2}$$

STEP 1- $m = \frac{0 + 12}{2} = 6$

item = 55

STEP 2- 40 < 55, search left

$mid = \frac{0+5}{2} = 2.5$

item = 30

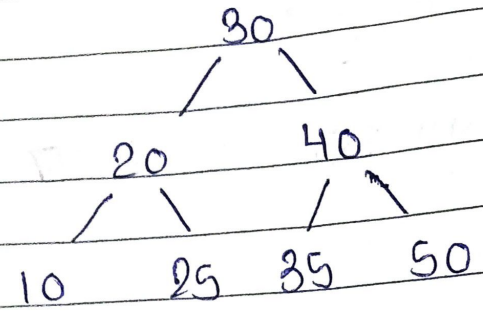
STEP 3- 30 < 40, search right

$mid = \frac{3+5}{2} = 4$

item = 40.

Q37- Construct a BST using the elements : 30, 20, 40, 10, 25, 35, 50. Show in-order, pre-order, post-order traversals.

Ans-



1- In-order traversal (Left, Root, Right)

10 → 20 → 25 → 30 → 35 → 40 → 50

2- Pre-order traversal (Root, Left, Right)

~~30 → 10 → 20 → 25 → 35 → 40 → 50~~

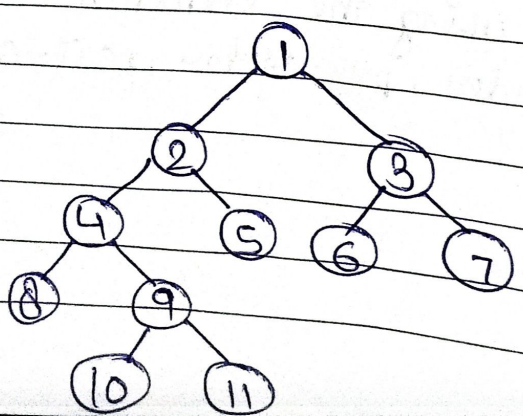
30 → 20 → 10 → 25 → 40 → 35 → 50

3- Post-order traversal (Left, Right, Root)

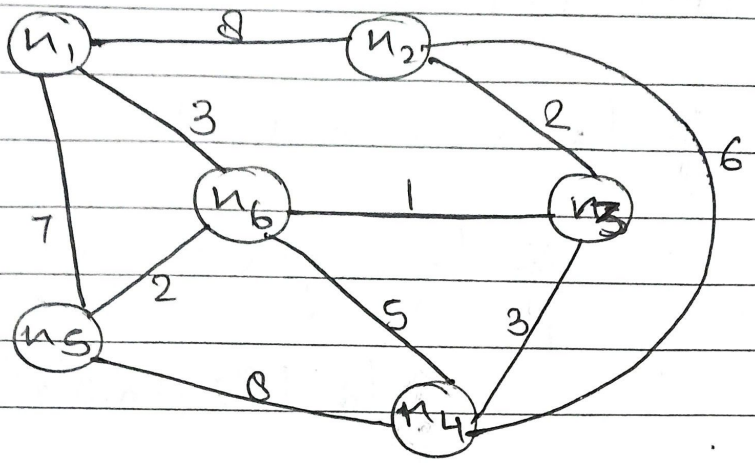
10 → 25 → 20 → 35 → 50 → 40 → 30

Q38- Construct a binary tree using :

Preorder - 1, 2, 4, 8, 9, 10, 11, 5, 3, 6, 7
Inorder - 8, 4, 10, 9, 11, 2, 5, 1, 6, 3, 7



Q9- Write the Floyd Warshall algorithm to compute the all-pair shortest path. Apply the algorithm on following graph.



Ans- The Floyd-Warshall algorithm finds shortest paths between all pairs of vertices in a weighted graph.

```
#include <stdio.h>
void main()
{
    int i, j, k;
    int v = 3;
    int INF = 99999;
    int graph[3][3] = {
        { 0 , 8 , INF},
        {INF, 0 , 2 },
        { 6 , INF, 0 }
    };
    int dist [3][3];
    for (i=0; i<v; i++)
    {
        for (j=0; j<v; j++)
```

```
        }
            dist[i][j] = graph[i][j];
        }
    }
    for (k=0; k<v; k++)
    {
        for (i=0; i<v; i++)
        {
            for (j=0; j<v; j++)
            {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
    printf("Shortest path :");
    for (i=0; i<v; i++)
    {
        for (j=0; j<v; j++)
        {
            if (dist[i][j] == INF)
            {
                printf("%4s", "INF");
            }else
            {
                printf("%4d", dist[i][j]);
            }
        }
    }
}
```

STEP 1
STEP 2
STEP 3
STEP 4
STEP 5
STEP 6
STEP 7
STEP 8
STEP 9
STEP 10
STEP 11
STEP 12
STEP 13
STEP 14
STEP 15
STEP 16
STEP 17

Warshall Algorithm -

STEP 1- Initialise i, j, k, v = 3, INF = 99999, graph[3][3], dist[3][3]
STEP 2- Create a 2-D Matrix and store in graph[3][3].
STEP 3- Repeat STEP 4 until i < v, increases by 1 unit.
STEP 4- Repeat STEP 5 until j < v increases by 1 unit.
STEP 5- Set dist[i][j] = graph[i][j]
STEP 6- Repeat STEP 7 until k < v increases by 1 unit
STEP 7- Repeat STEP 8 until i < v increases by 1 unit
STEP 8- Repeat STEP 9 to 10 until j < v increases by 1 unit
STEP 9- if dist[i][k] + dist[k][j] < dist[i][j]
STEP 10- Set dist[i][j] = dist[i][k] + dist[k][j]
STEP 11- Print shortest path
STEP 12- Repeat STEP 13 until i < v increases by 1 unit
STEP 13- Repeat STEP 14 to 16 until j < v increases by 1 unit
STEP 14- if dist[i][j] == INF
STEP 15- print INF
STEP 16- else, print dist[i][j]
STEP 17- Exit.

	n1	n2	n3	n4	n5	n6
n1	0	8	∞	∞	7	3
n2	8	0	2	6	∞	∞
n3	∞	2	0	3	∞	1
n4	∞	6	3	0	8	5
n5	7	∞	∞	8	0	2
n6	3	∞	1	5	2	0

Q40- Construct a Binary Tree using

Preorder - D, B, E, A, F, C

Inorder - D, E, B, F, C, A