Q.1 (a) What is best-case, average-case, and worst-case time complexity analysis? (3 Marks)

- **Best-Case Time Complexity**
  - It is the time taken by an algorithm for the **most favorable input**.
  - The algorithm finishes in the **minimum possible time**.
- **Average-Case Time Complexity**
  - It is the time taken for all possible inputs, calculated as the **expected time**.
  - It represents the **typical behavior** of the algorithm.
- **Worst-Case Time Complexity**
  - It is the time taken by the algorithm for the **most unfavorable input**.
  - Shows the **maximum time required**, which is useful for guaranteeing performance.

Q.1 (b) Row-major and Column-major order representation of 2-D array (4 Marks)

*Row-Major Order **

The entire **first row** is stored first, then the second row, and so on.

- Elements are stored **row by row** in contiguous memory locations.
- It is the **default representation** in languages like C/C++.
- **Formula to find address of element A[i][j]:**

  $$Loc(A[i][j]) = Base(A) + [i \times \text{No. of columns} + j] \times size$$

*Column-Major Order **

The entire **first column** is stored first, then the second column, and so on.

- Elements are stored **column by column** in contiguous memory locations.
- It is used in languages like Fortran and MATLAB.
- **Formula to find address of element A[i][j]:**

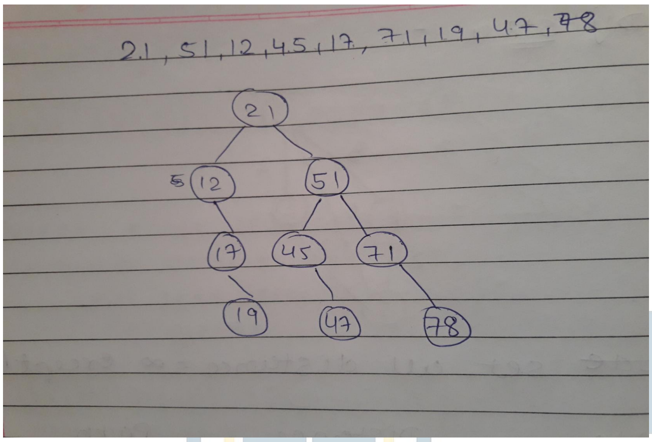
  $$Loc(A[i][j]) = Base(A) + [j \times \text{No. of rows} + i] \times size$$

Q.1 (c) Construction of Binary Search Tree (BST) write preorder inorder and postorder of constructed bst.

Insert the elements in given order: 21, 51, 12, 45, 17, 71, 19, 47, 78

*Step-by-Step Insertion **
21 → root

- **51** >21 → right of 21
- **12** <21 → left of 21
- **45** <51 → left of 51
- **17** >12 → right of 12
- **71** >51 → right of 51
- **19** >17 → right of 17
- **47** >45 → right of 45
- **78** >71 → right of 71



Tree Traversals
1. Pre-order Traversal (Root → Left → Right):
21, 12, 17, 19, 51, 45, 47, 71, 78

2. **In-order Traversal (Left → Root → Right):**

   12,17,19,21,45,47,51,71,78 (This will always give ascending order for a BST)

3. **Post-order Traversal (Left → Right → Root):**

   19,17,12,47,45,78,71,51,21

Q.2 (a) Define the following terms (3 Marks)

1. **Full Binary Tree:** A binary tree in which every node has either **0 or 2 children**.
   - No node in a full binary tree has only one child.

2. **Complete Binary Tree:** A binary tree where **all levels are completely filled**, except possibly the last level.
    - In the last level, all nodes are filled **from left to right** without gaps.
3. **Skewed Binary Tree:** A binary tree in which all nodes are arranged like a **linked list**.
    - It can be **left-skewed** (all left children) or **right-skewed** (all right children).

Q.2 (b) Importance of asymptotic analysis + is O(nlogn2) faster than O(n2)? Justify your answer with help of example (4 Marks)

Importance of Asymptotic Analysis

- It helps to evaluate the performance of an algorithm **independent of hardware**.
- It compares algorithms based on **growth rate** as input size increases.
- It focuses on **order of magnitude**, not exact execution time.
- Helps to choose the **most efficient algorithm** for large inputs.

Is O(nlogn2) faster than O(n2)? – Yes

- **Yes**, O(nlogn) is faster than O(n2) because it grows slower as n increases.
- **Example for n=1000:**
    - nlogn=1000×10=10,000
    - n2=(1000)2=10,00,000
- As n becomes large, the difference in execution time becomes even bigger.
- Hence, algorithms like **Merge Sort** (O(nlogn)) are faster than **Bubble Sort** (O(n2)) for large input sizes.

Q.2 (c) Convert the following infix expression to postfix:

Expression:

(a + b)^((c * d) / (e – f))

Symbol	Action	Stack	Postfix
(	push	(	
a	output	(	a
+	push	(+	a
b	output	(+	ab
)	pop until (	—	ab+
^	push	^	ab+
(	push	^(	ab+
(	push	^((	ab+
c	o/p	^((	ab+c
*	push	^((*	ab+c
d	o/p	^((*	ab+cd
)	pop	^(	ab+cd*
/	push	^(/	ab+cd*
(	push	^(/(	ab+cd*
e	o/p	^(/(	ab+cd*e
-	push	^(/-	ab+cd*e
f	o/p	^(/-	ab+cd*ef
)	pop	^(/	ab+cd*ef-
)	pop	^	ab+cd*ef-/
)	pop	—	ab+cd*ef-/^

final postfix: ab+cd*ef-/^

Q.2 (c) Algorithm to Convert Infix Expression to Postfix (7 Marks)

- The algorithm involves reading the expression from left to right.

1. If an **operand** (A-Z, 0-9) is encountered, add it **directly to the output**.
2. If an **operator** is encountered:
   - Pop operators from the stack to the output if they have **higher or equal precedence** than the current operator.
   - Push the current operator onto the stack.
3. If an **opening parenthesis '('** is found, push it onto the stack.
4. If a **closing parenthesis ')'** is found, pop operators from the stack to the output until an **opening parenthesis** is encountered. Discard both parentheses.
5. After scanning the entire expression, pop any **remaining operators** from the stack to the output.

Q.3 (a) Illustrate how stack is used in recursion. (3 Marks)

- Recursion uses the **system stack** to keep track of function calls.
- Each time a recursive function is called, a new **activation record (stack frame)** is pushed onto the stack.
- The stack frame stores **parameters, local variables, return address**, etc. for that specific function call.
- When the function calls itself again, a new frame is created and **added on top** of the stack.
- When a function finishes execution, its stack frame is **popped** from the stack.
- Recursion continues until the **base condition** is reached, after which the stack unwinds backward, and values are returned.

Q.3 (b) Describe Threaded Binary Tree with example. (4 Marks)

*Definition **

A **Threaded Binary Tree** is a binary tree where **NULL pointers** are replaced with pointers called **threads**.

- A thread points to the **inorder predecessor** or **inorder successor** of the node.

Types

- **Single Threaded:** Each node stores only one thread (either left or right).
- **Double Threaded:** Each node stores both left and right threads.

Purpose

- Threads help in **faster traversal** (e.g., inorder) without using an auxiliary stack or recursion.
- It **improves memory utilization** by converting NULL links into useful references.

Q.3 (c) C Program for Circular Queue Operations (Insert, Delete, Display) – 7 Marks

```
#include <stdio.h>

#deine SIZE 5
```

```
int cq[SIZE];
int front = -1, rear = -1;
//INSERT
void insert(int item){
	if((rear+1)%SIZE==front){
		printf("queue is full\n");
		return;
	}if(front==-1)
		front=rear=0;
	else
		rear=(rear+1)%SIZE;
	cq[rear]=item;
	printf("inserted:%d\n", item);
}
//delete
void delete(){
	if(Front==-1){
		printf("queue empty");
		return;
	}
	printf("deleted: %d\n", cq[front]);
	if(front=rear)
		front=rear=-1;
	else
		front=(front+1)%SIZE;
}
//display
```

```
void display(){
    if (front=-1){
    printf("queue empty\n");
    return;
    }
    int i=front;
    printf("queue:");
    while(1){
        printf("%d", cq[i]);
        if(i==rear) break;
        i=(i+1)%SIZE;
    }
    printf("\n");
}
int main(){
    insert(10);
    insert(20);
    insert(30);
    display();
    delete();
    display();
    return 0;
}
```

OR

Q.3 (a) Write a recursive solution for Tower of Hanoi problem. (3 Marks)

Recursive Logic

- Tower of Hanoi is a classic **recursive problem**.

- The goal is to move n disks from **Source (A) to Destination (C)** using **Auxiliary (B)**.
- **Base Case:** If n=1, simply move the disk from A → C.
- **Recursive Case:** 1. Move n–1 disks from **A → B** (Source to Auxiliary). 2. Move the nth disk from **A → C** (Source to Destination). 3. Move n–1 disks from **B → C** (Auxiliary to Destination).
- Recursion divides the big problem into smaller subproblems.
- Total number of moves = 2n–1.

C Code (Recursive Function)

```
void TOH(int n, char A, char B, char C) {
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", A, C);
        return;
    }
    // Step 1: Move n-1 disks from A to B (using C as auxiliary)
    TOH(n-1, A, C, B);

    // Step 2: Move the nth disk from A to C
    printf("Move disk %d from %c to %c\n", n, A, C); [cite: 24]

    // Step 3: Move n-1 disks from B to C (using A as auxiliary)
    TOH(n-1, B, A, C); [cite: 24]
}
```

Q.3 (b) Explain DFS traversal of a graph with example. (4 Marks)

Definition

- **DFS (Depth First Search)** is a graph traversal method that explores **as far as possible along a branch** before backtracking.
- It uses a **stack** (or recursion) to keep track of unvisited nodes.

Procedure

1. Start from any node and **visit it first**.
2. Move to its **unvisited adjacent node** and continue deeper into that path.
3. If a node has **no unvisited neighbours**, backtrack to the previous node.
4. Continue the process until **all nodes are visited**.
5. DFS helps in **path finding, cycle detection**, and **topological sorting**.

Q.3 (c) Algorithm to sort a singly linked list in ascending order (by info field). (7 Marks)

This algorithm uses a **Bubble Sort** approach to sort the list by swapping data fields.

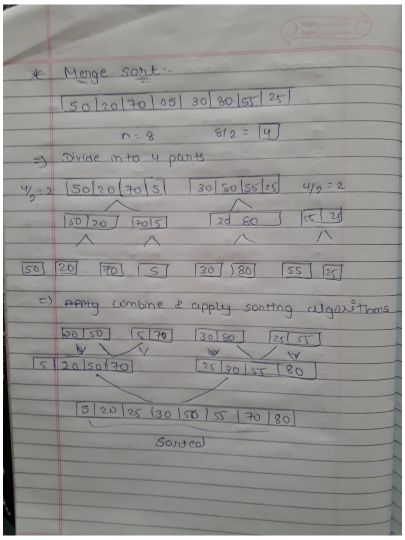
1. **Start** with the head of the linked list.
2. Check for base cases: If the list is empty or has only one node, **return** (already sorted).
3. Use two pointers: **ptr1** to traverse the list (outer loop) and **ptr2** to point to the next node of ptr1 (inner loop).
4. Repeat the outer loop until ptr1 reaches the **second-last node**.
5. For each ptr1, start the inner loop with ptr2 and compare the data.
6. If ptr1's data is **greater** than ptr2's data (ptr1 $\rightarrow$ info > ptr2 $\rightarrow$ info), then **swap the information fields** (info) of the two nodes.
7. Move ptr2 to the next node (ptr2 = ptr2 $\rightarrow$ next) and **continue comparing** within the inner loop.
8. Once the inner loop completes (meaning the largest unsorted element has bubbled to its correct position), move ptr1 to the next node (ptr1 = ptr1 $\rightarrow$ next).
9. Continue steps 4-8 until the **entire list becomes sorted**.
10. **Stop** – the linked list is now sorted in ascending order.

Q.4 (a) Define the following terms: Field, Record, File (3 Marks)

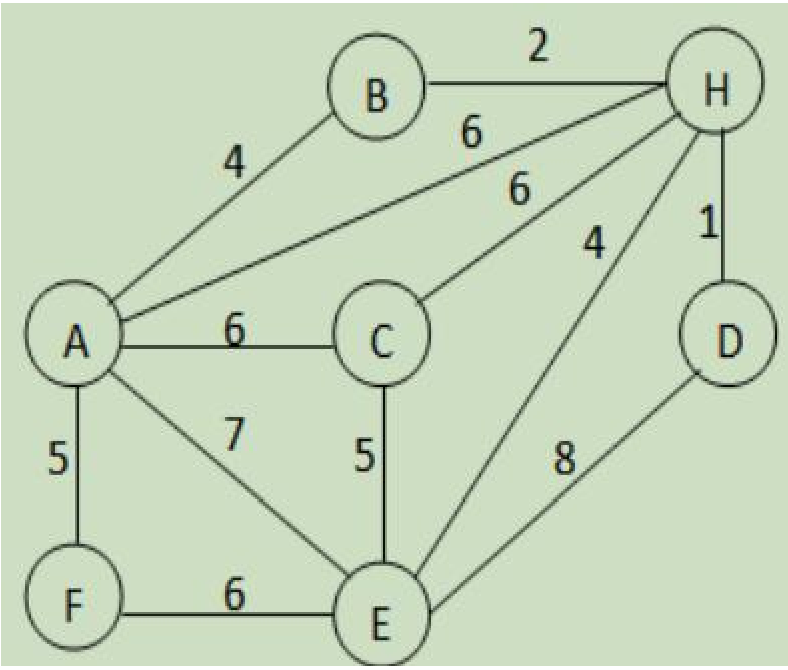
1. **Field**
   - A field is the **smallest unit of data** in a database.
   - It represents a **single attribute** such as Name, Age, or Roll No.
2. **Record**
   - A record is a **collection of related fields**.
   - It represents **one complete entry** (e.g., one student's full details).
3. **File**
   - A file is a **collection of related records**.
   - A student file may contain many student records.

Q.4 (b) Sort the following data using Merge Sort (4 Marks)

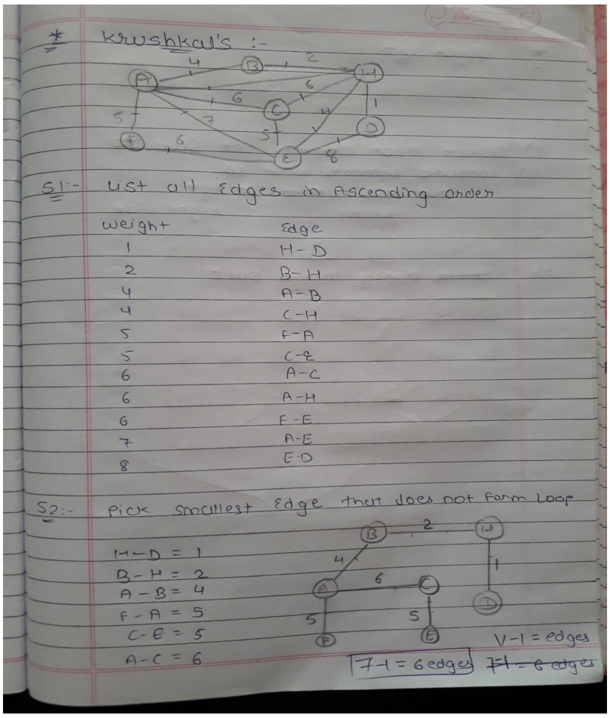
Data:
50, 20, 70, 05, 30, 80, 55, 25



Q.4 (c) Find MST of the following graph using krushkal's algorithm (7 Marks)



- Use to find shortest path from source to destination
- Each vertex must be visited only once.
- No loop should be form.
- No.of edges in krushkal must be: |V-1|=edges
- Eg: if there are 6 vertex in graph so answer must consist v-1 edges means 6-1=5 edges.



OR

Q.4 (a) What is hashing? Properties of a good hash function (3 Marks)

Hashing

- Hashing is a technique to **map keys to positions** (indices) in a hash table.
- A **hash function** converts a key into a hash index for **fast searching**.

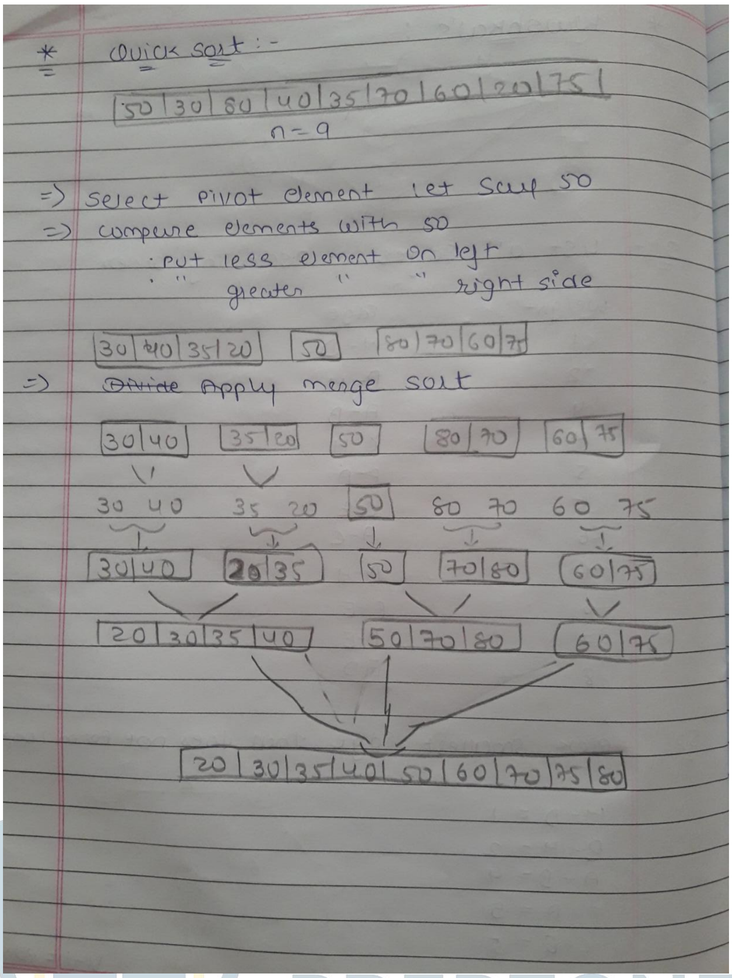
Properties of a Good Hash Function

- Should be **simple and fast** to compute.
- Should **distribute keys uniformly** across the hash table.
- Should **minimize collisions** (multiple keys mapping to the same index).
- Should use the **full range of table indices** effectively.

Q.4 (b) sort the following data using Quick Sort (4 Marks)

Data: **50, 30, 80, 40, 35, 70, 60, 20, 75**

Choose **first element as pivot = 50**



* Quick sort :-

|50|30|80|40|35|70|60|20|75|

n = 9

=> Select pivot element let say 50

=> compare elements with 50

put less element on left

" " greater " " right side

|30|40|35|20| |50| |80|70|60|75|

=> ~~Divide~~ Apply merge sort

|30|40| |35|20| |50| |80|70| |60|75|

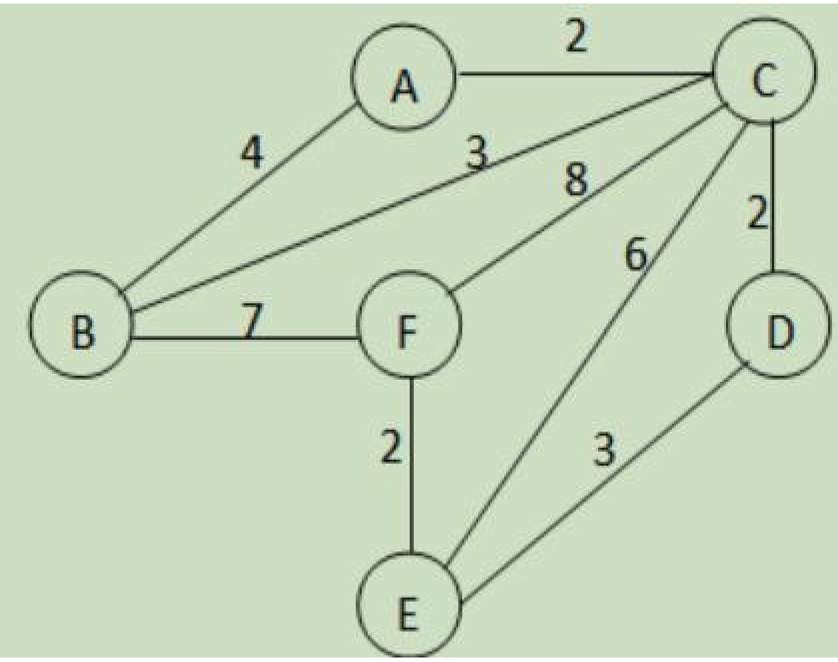
30 40 35 20 50 80 70 60 75

|30|40| |20|35| |50| |70|80| |60|75|

|20|30|35|40| |50|70|80| |60|75|

|20|30|35|40|50|60|70|75|80|

Q.4 (c) Find shortest path from A to F using Dijkstra (7 Marks)



- Dijkstra algorithm is used to find shortest path between source to destination.
- Aim to choose path that contains minimum weight/cost.
- First step is to set all distance to infinite except source vertex which would be set to 0.
- Now compute cost from each vertex to find shortest path from source to destination.
- In this above graph we have to reach from source=A to destination= F with min cost.
- If we choose path from A to b and b to f it would take cost=4+7=11
- If we choose alternate path from A to C then C to D, D to E and then E to F it would cost= 2+2+3+2=9
- As we can see there is no other path which is less than 9 cost.
- So shortest path from A to F is= A--C--D--E--F=total cost=9.

Q.5 (a) Compare linear search and binary search in terms of their time complexity. (3 Marks)

Feature	Linear Search	Binary Search
Data Requirement	Works on **unsorted** data.	Works only on **sorted** data.
Search Mechanism	Checks each element **one by one** from start to end.	Repeatedly **divides the list into two halves**.
Best Case	O(1) (element found at first position).	O(1) (middle element matches).
Worst Case	O(n) (element at last or not present).	O(logn) (repeated halving).
Average Case	O(n).	O(logn).
Efficiency	Simple but **slow** for large data.	**Faster** than linear search for large datasets.

Q.5 (b) Write a C program for bubble sort

```
#include <stdio.h>
int main() {
    int a[50], n, i, j, temp;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements: ");
    for(i = 0; i < n; i++)
        scanf("%d", &a[i]);

    // Bubble Sort
    for(i = 0; i < n-1; i++) { [cite: 33]
        for(j = 0; j < n-i-1; j++) { [cite: 33]
            if(a[j] > a[j+1]) { // Compare adjacent elements
                // Swap a[j] and a[j+1]
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }

    printf("Sorted List: ");
    for(i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Q.5 (c) What is hash collision? Explain collision resolution techniques. (7 Marks)

Hash Collision

A hash collision occurs when **two or more keys generate the same hash index** in the hash table.

- Since two values cannot be stored at the same index, we need collision resolution techniques.

Collision Resolution Techniques

1. **Open Addressing (Closed Hashing)**
    - If a collision occurs, the algorithm **searches for another empty position** in the table.
    - **(a) Linear Probing:** Checks the **next index sequentially**: h(key),h(key)+1,h(key)+2... Simple but creates **primary clustering**.
    - **(b) Quadratic Probing:** Jumps by **quadratic distance**: h(key)+12,h(key)+22,h(key)+32... Reduces clustering.
    - **(c) Double Hashing:** Uses a second hash function: h1 (key)+i×h2 (key). Very effective and results in fewer clusters.
2. **Rehashing**
    - When the table becomes too full, a **bigger table is created**.
    - All keys are **reinserted** into the new, larger table using a new hash function.
    - This reduces the frequency of collisions.
3. **Coalesced Hashing**
    - It is a **combination** of open addressing and chaining.
    - An **overflow area** is used to store collided elements.

OR

Q.5 (a) Does a pivot selection method affect the time complexity of quick sort? Justify your answer. (3 Marks)

- **Yes**, pivot selection **directly affects** the time complexity of Quick Sort.
- **Optimal Case:** If the pivot divides the array into **two equal halves**, Quick Sort works fastest.
- In the optimal case, the time complexity is O(nlogn).
- **Worst Case:** If the pivot is chosen poorly (e.g., always the smallest or largest element), the partitions become **highly unbalanced**.
- In the worst case, the time complexity of Quick Sort becomes O(n2).
- Therefore, good pivot strategies like **median-of-3, random pivot**, or choosing the middle element are used to improve average performance.

Q.5 (b) Write a C program for selection sort

```
#include <stdio.h>
int main() {
    int a[50], n, i, j, min, temp;
```

```
    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter elements: ");
    for(i = 0; i < n; i++)
        scanf("%d", &a[i]);

    // Selection Sort
    for(i = 0; i < n-1; i++) {
        // Outer loop
        min = i; // Assume current element is the minimum
        for(j = i+1; j < n; j++) {
       // Inner loop to find the minimum element
            if(a[j] < a[min])
                min = j; // Update index of minimum element
        }

        // Swap the found minimum element with the element at position i
        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }

    printf("Sorted List: ");
    for(i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Q.5 (c) List various file organizations and explain one in detail. (7 Marks)

File Organizations (List)

1. Sequential File Organization
2. Direct / Hash File Organization
3. Indexed Sequential File Organization (ISAM)
4. Heap (Unordered) File Organization
5. Clustered File Organization

Explain One in Detail – Sequential File Organization

1. **Meaning:** Records are stored one after another in a **fixed order**, usually based on a key field.
    - *Example:* Students sorted by roll number.
2. **Features:**
    - Simple to create and maintain.
    - Data is stored in sorted or unsorted sequence.
3. **Advantages:**
    - Best suited for **batch processing**.
    - **Fast for sequential access**.
    - Easy to implement.
4. **Disadvantages:**
    - **Slow for random access** (requires reading sequentially).
    - Inserting and deleting a record requires **shifting multiple records**.
    - Not suitable for large dynamic files.
5. **Applications:**
    - Payroll systems
    - Monthly billing
    - Library book issue registers
