A. **Research where the data structures types are applied and give reasons why.**

1. **ARRAY**

Application: Student marksheet

Reason: allows random access using registration numbers for fast retrieval

2. **Stack**

Application: undo functionality in apps

Reason: the behavior matches the need to reverse the most recent actions

3. **Tree**

Application: file system directories

Reason: hierarchical structure is ideal for organizing data into levels

4. **Graph**

Application: Social Network Connections

Reason: Best for representing complex links between many different elements

B. Give examples of applications that are using the data structure type and algorithm. Give reasons why

1. Inventory search

Example: searching 10 raised to 6 items sequentially is too slow is too slow

Using Binary Search algorithm on an organized data structure allows for instant results.

2. Performance Metrics

We measure algorithm success through Time Complexity and Space complexity

C. Research how data structures and algorithms work within systems

- Overcome Processor Limits: Efficient organization handles billion of files that raw processors cannot process alone
- Handle Multiple Requests: They prevent server failure when thousands of users search data simultaneously.
- Abstraction: They provide an interface(ADT)so the system can use data without managing complex internal code

1. Primitive Data Structures

```
#include <stdio.h>
  int main()
  int age = 20;
  float height = 5.9;
  char grade = 'A';

  printf("Primitive Data Structures:\n");
  printf("Age: %d\n", age);
  printf("Height: %.2f\n", height);
  printf("Grade: %c\n", grade);
  printf("Balance: %.2lf\n\n", balance);
}
```

2. Linear Data Structures

(a) Array

```
void arrayExample() {
  int arr[5] = {10, 20, 30, 40, 50};

  printf("Array Elements:\n");
  for(int i = 0; i < 5; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n\n");
}
```

(b) Linked List

```
#include <stdlib.h>

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

void linkedListExample() {
    struct Node *head = NULL, *second = NULL, *third = NULL;

    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 10;
    head->next = second;

    second->data = 20;
    second->next = third;

    third->data = 30;
    third->next = NULL;

    struct Node* temp = head;
    printf("Linked List Elements:\n");
```

```
    while(temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n\n");
}
```

(c) Stack (Using Array)

```
#define SIZE 5
int stack[SIZE];
int top = -1;

void push(int value) {
    if(top == SIZE - 1)
        printf("Stack Overflow\n");
    else {
        stack[++top] = value;
    }
}

void stackExample() {
    push(10);
    push(20);
    push(30);

    printf("Stack Elements:\n");
```

```c
    for(int i = 0; i <= top; i++) {
        printf("%d ", stack[i]);
    }
    printf("\n\n");
}
```

(d) Queue (Using Array)

```c
int queue[SIZE];
int front = 0, rear = -1;

void enqueue(int value) {
    if(rear == SIZE - 1)
        printf("Queue Full\n");
    else {
        queue[++rear] = value;
    }
}

void queueExample() {
    enqueue(5);
    enqueue(15);
    enqueue(25);

    printf("Queue Elements:\n");
    for(int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
```

```
  }
  printf("\n\n");
}
```

3. Non-Linear Data Structures

(a) Binary Tree

```
struct TreeNode {
  int data;
  struct TreeNode *left, *right;
};

struct TreeNode* createNode(int value) {
  struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
  newNode->data = value;
  newNode->left = newNode->right = NULL;
  return newNode;
}

void treeExample() {
  struct TreeNode* root = createNode(1);
  root->left = createNode(2);
  root->right = createNode(3);

  printf("Binary Tree Root: %d\n\n", root->data);
}
```

(b) Graph (Adjacency Matrix)

```
#define V 3

void graphExample() {
    int graph[V][V] = {
        {0, 1, 1},
        {1, 0, 0},
        {1, 0, 0}
    };

    printf("Graph Adjacency Matrix:\n");
    for(int i = 0; i < V; i++) {
        for(int j = 0; j < V; j++) {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
```

4. Main Function

```
int main() {
    primitiveExample();
    arrayExample();
    linkedListExample();
    stackExample();
```

```
    queueExample();
    treeExample();
    graphExample();

    return 0;
}
```