SECTION -A

Q 1: 20 friends put their wallets in a row. The first wallet contains 20 dollars, the second has 30 dollars, the third has 40 dollars, and so on, with each wallet having 10 dollars more than the previous one. Since the data is already sorted in ascending order, no sorting is required. But if you are given a chance to sort the wallets, which sorting technique would be best to apply? Write a C++ program to implement your chosen sorting approach.

Q 2: In a park, 10 friends were discussing a game based on sorting. They placed their wallets in a row. The maximum money in any wallet is **$6**. Among them, **3 wallets contain exactly $2**, **2 wallets contain exactly $3**, **2 wallets are empty ($0)**, **1 wallet contains $1**, and **1 wallet contains $4**. Which sorting technique would you apply to sort the wallets on the basis of the money they contain? Write a program to implement your chosen sorting technique.

Q 3: During a college fest, 12 students participated in a gaming competition. Each student's score was recorded as follows: 45, 12, 78, 34, 23, 89, 67, 11, 90, 54, 32, 76 The organizers want to arrange the scores in **ascending order** to decide the ranking of the players. Since the data set is unsorted and contains numbers spread across a wide range, the most efficient technique to apply here is **Quick Sort**. Write a C++ program to implement **Quick Sort** to arrange the scores in ascending order.

Q 4: A software company is tracking project deadlines (in days remaining to submit). The deadlines are: 25, 12, 45, 7, 30, 18, 40, 22, 10, 35. The manager wants to arrange the deadlines in **ascending order** to prioritize the projects with the least remaining time. For efficiency, the project manager hints to the team to apply a **divide-and-conquer technique that divides the array into unequal parts**.

Write a C++ program to sort the project deadlines using the above sorting technique.

Q 5: Suppose there is a square named **SQ-1**. By connecting the midpoints of SQ-1, we create another square named **SQ-2**. Repeating this process, we create a total of **50 squares** {SQ-1, SQ-2, …, SQ-50}. The areas of these squares are stored in an array. Your task is to **search whether a given area is present in the array or not**.What would be the best searching approach? Write a C++ program to implement this approach.

Q 6: Before a match, the chief guest wants to meet all the players. The head coach introduces the first player, then that player introduces the next player, and so on, until all players are introduced. The chief guest moves forward with each introduction, meeting the players one at a time. How would you implement the above activity using a Linked List? Write a C++ program to implement the logic.

Q 7: A college bus travels from stop A → stop B → stop C → stop D and then returns in reverse order D → C → B → A. Model this journey using a **doubly linked list**. Write a program to:

- Store bus stops in a doubly linked list.
- Traverse forward to show the onward journey.
- Traverse backward to show the return journey.

Q 8: There are two teams named Dalta Gang and Malta Gang. Dalta Gang has 4 members, and each member has 2 Gullaks (piggy banks) with some money stored in them. Malta Gang has 2 members, and each member has 3 Gullaks. Both gangs store their Gullak money values in a 2D array. Write a C++ program to:

- Display the stored data in matrix form.

- To multiply Dalta Gang matrix with Malta Gang Matrix

SECTION -B

Q 1: To store the names of family members, an expert suggests organizing the data in a way that allows efficient searching, traversal, and insertion of new members. For this purpose, use a **Binary Search Tree (BST)** to store the names of family members, starting with the letters:

<Q, S, R, T, M, A, B, P, N>

Write a C++ program to Create a **Binary Search Tree (BST)** using the given names and find and display the **successor** of the family member whose name starts with **M**.

Q 2: Implement the In-Order, Pre- Order and Post-Order traversal of Binary search tree with help of C++ Program.

Q 3: Write a C++ program to search an element in a given binary search Tree.

Q 4: In a university, the roll numbers of newly admitted students are: 45, 12, 78, 34, 23, 89, 67, 11, 90, 54

The administration wants to store these roll numbers in a way that allows **fast searching, insertion, and retrieval in ascending order**. For efficiency, they decide to apply a **Binary Search Tree (BST)**.

Write a C++ program to construct a **Binary Search Tree** using the above roll numbers and perform an **in-order traversal** to display them in ascending order.

Q 5: In a university database, student roll numbers are stored using a **Binary Search Tree (BST)** to allow efficient searching, insertion, and deletion. The roll numbers are: 50, 30, 70, 20, 40, 60, 80. The administrator now wants to **delete a student record** from the BST. Write a C++

program to delete a node (student roll number) entered by theuser.

Q 6: Design and implement a **family tree hierarchy** using a **Binary Search Tree (BST)**. The family tree should allow efficient storage, retrieval, and manipulation of information related to individuals and their relationships within the family.

Write a C++ program to:

1. Insert family members into the BST (based on their names).
2. Perform in-order, pre-order, and post-order traversals to display the hierarchy.
3. Search for a particular family member by name.

ANSWERS

✓SECTION - A

Q1: Best Sorting for Already Sorted Data → Insertion Sort

```
#include <iostream>
using namespace std;

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = key;
    }
}

int main() {
    int wallets[20];
    int money = 20;
    for (int i = 0; i < 20; i++) {
        wallets[i] = money;
        money += 10;
```

```
    }

    insertionSort(wallets, 20);

    cout << "Sorted Wallets: ";
    for (int i = 0; i < 20; i++) cout << wallets[i] << " ";
    return 0;
}
```

Q2: Best Sorting for Small Range & Repeated Values → Counting Sort

```
#include <iostream>
using namespace std;

void countingSort(int arr[], int n, int maxVal) {
    int count[maxVal+1] = {0};
    for (int i = 0; i < n; i++) count[arr[i]]++;

    int index = 0;
    for (int i = 0; i <= maxVal; i++) {
        while (count[i]--) arr[index++] = i;
    }
}

int main() {
    int wallets[10] = {2,2,2,3,3,0,0,1,4,6};

    countingSort(wallets, 10, 6);

    cout << "Sorted Wallets: ";
    for (int i = 0; i < 10; i++) cout << wallets[i] << " ";
    return 0;
}
```

Q3: Quick Sort for Scores

```
#include <iostream>
using namespace std;

int partition(int arr[], int low, int high) {
    int pivot = arr[high], i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i+1], arr[high]);
    return i+1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi-1);
        quickSort(arr, pi+1, high);
    }
```

```
}

int main() {
    int scores[12] = {45,12,78,34,23,89,67,11,90,54,32,76};
    int n = 12;

    quickSort(scores, 0, n-1);

    cout << "Sorted Scores: ";
    for (int i = 0; i < n; i++) cout << scores[i] << " ";
    return 0;
}
```

Q4: Quick Sort for Deadlines

(Same as Q3 but with deadlines array)

```
#include <iostream>
using namespace std;

int partition(int arr[], int low, int high) {
    int pivot = arr[high], i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i+1], arr[high]);
    return i+1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi-1);
        quickSort(arr, pi+1, high);
    }
}

int main() {
    int deadlines[10] = {25,12,45,7,30,18,40,22,10,35};
    int n = 10;

    quickSort(deadlines, 0, n-1);

    cout << "Sorted Deadlines: ";
    for (int i = 0; i < n; i++) cout << deadlines[i] << " ";
    return 0;
}
```

Q5: Searching in 50 Squares → Binary Search

```
#include <iostream>
#include <cmath>
using namespace std;

bool binarySearch(double arr[], int n, double key) {
    int low = 0, high = n-1;
```

```
    while (low <= high) {
        int mid = (low+high)/2;
        if (fabs(arr[mid] - key) < 1e-6) return true;
        else if (arr[mid] < key) low = mid+1;
        else high = mid-1;
    }
    return false;
}

int main() {
    double areas[50];
    double side = 10;
    for (int i = 0; i < 50; i++) {
        areas[i] = side * side;
        side /= sqrt(2);
    }

    double searchArea;
    cout << "Enter area to search: ";
    cin >> searchArea;

    if (binarySearch(areas, 50, searchArea))
        cout << "Area Found!";
    else
        cout << "Area Not Found!";
    return 0;
}
```

Q6: Players Introduction → Singly Linked List

```
#include <iostream>
using namespace std;

struct Node {
    string player;
    Node* next;
};

void insert(Node*& head, string name) {
    Node* newNode = new Node{name, nullptr};
    if (!head) head = newNode;
    else {
        Node* temp = head;
        while (temp->next) temp = temp->next;
        temp->next = newNode;
    }
}

void display(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << "Meeting " << temp->player << endl;
        temp = temp->next;
    }
}

int main() {
    Node* head = nullptr;
    insert(head, "Player1");
    insert(head, "Player2");
```

```
    insert(head, "Player3");
    insert(head, "Player4");

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

Q7: Bus Journey → Doubly Linked List

```
#include <iostream>
using namespace std;

struct Node {
    string stop;
    Node *next, *prev;
};

void insert(Node*& head, string stop) {
    Node* newNode = new Node{stop, nullptr, nullptr};
    if (!head) head = newNode;
    else {
        Node* temp = head;
        while (temp->next) temp = temp->next;
        temp->next = newNode;
        newNode->prev = temp;
    }
}

void forward(Node* head) {
    Node* temp = head;
    cout << "Onward Journey: ";
    while (temp) {
        cout << temp->stop << " ";
        temp = temp->next;
    }
    cout << endl;
}

void backward(Node* head) {
    Node* temp = head;
    while (temp->next) temp = temp->next;
    cout << "Return Journey: ";
    while (temp) {
        cout << temp->stop << " ";
        temp = temp->prev;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    insert(head, "A");
    insert(head, "B");
    insert(head, "C");
    insert(head, "D");

    forward(head);
    backward(head);
    return 0;
}
```

Q8: Matrix Multiplication (Gang Gullaks)

```
#include <iostream>
using namespace std;

int main() {
    int dalta[4][2] = {{1,2},{3,4},{5,6},{7,8}};
    int malta[2][3] = {{1,2,3},{4,5,6}};
    int result[4][3] = {0};

    // Multiplication
    for (int i=0;i<4;i++) {
        for (int j=0;j<3;j++) {
            for (int k=0;k<2;k++) {
                result[i][j] += dalta[i][k]*malta[k][j];
            }
        }
    }

    cout << "Dalta Gang Matrix:\n";
    for(int i=0;i<4;i++){ for(int j=0;j<2;j++) cout<<dalta[i][j]<<" ";
cout<<"\n";}

    cout << "Malta Gang Matrix:\n";
    for(int i=0;i<2;i++){ for(int j=0;j<3;j++) cout<<malta[i][j]<<" ";
cout<<"\n";}

    cout << "Result Matrix:\n";
    for(int i=0;i<4;i++){ for(int j=0;j<3;j++) cout<<result[i][j]<<" ";
cout<<"\n";}
}
```

✓SECTION - B

Q1: Create BST with Names and Find Successor of “M”

```
#include <iostream>
#include <string>
using namespace std;

struct Node {
    string data;
    Node *left, *right;
};

Node* newNode(string key) {
    Node* node = new Node{key, nullptr, nullptr};
    return node;
}

Node* insert(Node* root, string key) {
    if (!root) return newNode(key);
    if (key < root->data) root->left = insert(root->left, key);
    else if (key > root->data) root->right = insert(root->right, key);
    return root;
}
```

```
Node* minValueNode(Node* node) {
    Node* current = node;
    while (current && current->left) current = current->left;
    return current;
}

Node* findSuccessor(Node* root, Node* target) {
    if (target->right) return minValueNode(target->right);
    Node* succ = nullptr;
    while (root) {
        if (target->data < root->data) {
            succ = root;
            root = root->left;
        } else if (target->data > root->data) {
            root = root->right;
        } else break;
    }
    return succ;
}

Node* search(Node* root, string key) {
    if (!root || root->data == key) return root;
    if (key < root->data) return search(root->left, key);
    return search(root->right, key);
}

int main() {
    string names[] = {"Q","S","R","T","M","A","B","P","N"};
    Node* root = nullptr;
    for (string n : names) root = insert(root, n);

    Node* target = search(root, "M");
    if (target) {
        Node* succ = findSuccessor(root, target);
        if (succ) cout << "Successor of M is: " << succ->data;
        else cout << "No successor exists.";
    } else cout << "M not found!";
    return 0;
}
```

Q2: BST Traversals (In-Order, Pre-Order, Post-Order)

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node *left, *right;
};

Node* newNode(int key) {
    return new Node{key, nullptr, nullptr};
}

Node* insert(Node* root, int key) {
    if (!root) return newNode(key);
    if (key < root->data) root->left = insert(root->left, key);
    else root->right = insert(root->right, key);
    return root;
}
```

```
void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

void preorder(Node* root) {
    if (root) {
        cout << root->data << " ";
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(Node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        cout << root->data << " ";
    }
}

int main() {
    int arr[] = {50,30,20,40,70,60,80};
    Node* root = nullptr;
    for (int x : arr) root = insert(root, x);

    cout << "Inorder: "; inorder(root); cout << endl;
    cout << "Preorder: "; preorder(root); cout << endl;
    cout << "Postorder: "; postorder(root); cout << endl;
}
```

Q3: Search Element in BST

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node *left, *right;
};

Node* newNode(int key) {
    return new Node{key, nullptr, nullptr};
}

Node* insert(Node* root, int key) {
    if (!root) return newNode(key);
    if (key < root->data) root->left = insert(root->left, key);
    else root->right = insert(root->right, key);
    return root;
}

bool search(Node* root, int key) {
    if (!root) return false;
    if (root->data == key) return true;
    if (key < root->data) return search(root->left, key);
```

```
    return search(root->right, key);
}

int main() {
    int arr[] = {50,30,20,40,70,60,80};
    Node* root = nullptr;
    for (int x : arr) root = insert(root, x);

    int key;
    cout << "Enter element to search: ";
    cin >> key;

    if (search(root, key)) cout << "Element found!";
    else cout << "Element not found!";
}
```

Q4: BST for Roll Numbers & In-Order Traversal

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node *left, *right;
};

Node* newNode(int key) {
    return new Node{key, nullptr, nullptr};
}

Node* insert(Node* root, int key) {
    if (!root) return newNode(key);
    if (key < root->data) root->left = insert(root->left, key);
    else root->right = insert(root->right, key);
    return root;
}

void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

int main() {
    int arr[] = {45,12,78,34,23,89,67,11,90,54};
    Node* root = nullptr;
    for (int x : arr) root = insert(root, x);

    cout << "Roll Numbers in Ascending Order: ";
    inorder(root);
}
```

Q5: Delete a Node in BST

```
#include <iostream>
using namespace std;

struct Node {
```

```
    int data;
    Node *left, *right;
};

Node* newNode(int key) {
    return new Node{key, nullptr, nullptr};
}

Node* insert(Node* root, int key) {
    if (!root) return newNode(key);
    if (key < root->data) root->left = insert(root->left, key);
    else root->right = insert(root->right, key);
    return root;
}

Node* minValueNode(Node* node) {
    while (node && node->left) node = node->left;
    return node;
}

Node* deleteNode(Node* root, int key) {
    if (!root) return root;
    if (key < root->data) root->left = deleteNode(root->left, key);
    else if (key > root->data) root->right = deleteNode(root->right, key);
    else {
        if (!root->left) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (!root->right) {
            Node* temp = root->left;
            delete root;
            return temp;
        }
        Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

int main() {
    int arr[] = {50,30,70,20,40,60,80};
    Node* root = nullptr;
    for (int x : arr) root = insert(root, x);

    cout << "Original Tree (Inorder): ";
    inorder(root); cout << endl;

    int key;
    cout << "Enter roll number to delete: ";
    cin >> key;
```

```
    root = deleteNode(root, key);

    cout << "After Deletion (Inorder): ";
    inorder(root);
}
```

Q6: Family Tree using BST

```
#include <iostream>
using namespace std;

struct Node {
    string name;
    Node *left, *right;
};

Node* newNode(string key) {
    return new Node{key, nullptr, nullptr};
}

Node* insert(Node* root, string key) {
    if (!root) return newNode(key);
    if (key < root->name) root->left = insert(root->left, key);
    else root->right = insert(root->right, key);
    return root;
}

void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->name << " ";
        inorder(root->right);
    }
}

void preorder(Node* root) {
    if (root) {
        cout << root->name << " ";
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(Node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        cout << root->name << " ";
    }
}

bool search(Node* root, string key) {
    if (!root) return false;
    if (root->name == key) return true;
    if (key < root->name) return search(root->left, key);
    return search(root->right, key);
}

int main() {
    Node* root = nullptr;
```

```
    root = insert(root, "Raj");
    root = insert(root, "Anita");
    root = insert(root, "Vikram");
    root = insert(root, "Meena");
    root = insert(root, "Kiran");

    cout << "Inorder (Alphabetical): "; inorder(root); cout << endl;
    cout << "Preorder: "; preorder(root); cout << endl;
    cout << "Postorder: "; postorder(root); cout << endl;

    string name;
    cout << "Enter family member to search: ";
    cin >> name;
    if (search(root, name)) cout << name << " found in family tree.";
    else cout << name << " not found!";
}
```
