1. Consider a student database of SEIT class (at least 10 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of structure) a. Design a roll call list, arrange list of students according to roll numbers in ascending order (Use Bubble Sort) b. Arrange list of students alphabetically. (Use Insertion sort)

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

// Structure to store student details
struct Student {
    int rollNo;       // Roll Number
    string name;      // Name of the student
    float sgpa;       // SGPA
};

// Bubble Sort to sort by Roll Number
void bubbleSortByRollNo(Student students[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (students[j].rollNo > students[j + 1].rollNo) {
                // Swap if current roll number is greater
                swap(students[j], students[j + 1]);
            }
        }
    }
}

// Insertion Sort to sort by Name
void insertionSortByName(Student students[], int n) {
    for (int i = 1; i < n; i++) {
        Student key = students[i]; // Take the current student as key
        int j = i - 1;
        // Move students with names greater than key.name forward
        while (j >= 0 && students[j].name > key.name) {
            students[j + 1] = students[j];
            j--;
        }
        students[j + 1] = key; // Insert key at the correct position
    }
}

// Function to display the student database
void displayStudents(Student students[], int n) {
    for (int i = 0; i < n; i++) {
        cout << "Roll No: " << students[i].rollNo
             << ", Name: " << students[i].name
             << ", SGPA: " << students[i].sgpa << endl;
```

```
    }
}

int main() {
    // Create an array of students (SEIT class with 10 records)
    Student students[10] = {
        {3, "Alice", 8.7}, {1, "Bob", 9.1}, {10, "Charlie", 7.8}, {5, "David", 8.3},
        {2, "Eve", 9.0}, {8, "Frank", 7.5}, {6, "Grace", 8.0}, {4, "Hannah", 8.2},
        {9, "Ivy", 7.9}, {7, "Jack", 8.6}
    };

    int n = 10; // Number of students

    // a. Sort students by Roll Number
    cout << "Sorting by Roll Number (Ascending Order):" << endl;
    bubbleSortByRollNo(students, n);
    displayStudents(students, n);

    // b. Sort students by Name
    cout << "\nSorting by Name (Alphabetical Order):" << endl;
    insertionSortByName(students, n);
    displayStudents(students, n);

    return 0;
}
```

2. Consider a student database of SEIT class (at least 15 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of structure) a. Arrange list of students to find out first ten toppers from a class. (Use Quick sort) b. Search students according to SGPA. If more than one student having same SGPA, then print list of all students having same SGPA.

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

// Define a structure to represent a Student with roll number, name, and SGPA
struct Student {
    int rollNo;       // Roll Number
    string name;      // Name of the student
    float sgpa;       // SGPA (Semester Grade Point Average)
};

// Partition function for Quick Sort: organizes elements around a pivot
int partition(Student s[], int low, int high) {
    float pivot = s[high].sgpa; // Select the last element's SGPA as the pivot
    int i = low - 1;            // Index for smaller elements
```

```
    for (int j = low; j < high; j++) {
        // If current element's SGPA is greater than pivot (for descending order)
        if (s[j].sgpa > pivot)
            swap(s[++i], s[j]); // Move it to the left side
    }
    swap(s[i + 1], s[high]);    // Place pivot in its correct position
    return i + 1;               // Return the partition index
}

// Quick Sort function to sort the array by SGPA in descending order
void quickSort(Student s[], int low, int high) {
    if (low < high) {
        int pi = partition(s, low, high); // Partition the array

        // Recursively sort the sub-arrays on either side of the pivot
        quickSort(s, low, pi - 1); // Sort elements to the left of pivot
        quickSort(s, pi + 1, high); // Sort elements to the right of pivot
    }
}

// Function to display details of students in the array
void display(Student s[], int n) {
    for (int i = 0; i < n; i++) {
        // Print Roll Number, Name, and SGPA of each student
        cout << "Roll No: " << s[i].rollNo
             << ", Name: " << s[i].name
             << ", SGPA: " << s[i].sgpa << endl;
    }
}

// Function to search for students with a specific SGPA
void searchSGPA(Student s[], int n, float sgpa) {
    bool found = false; // To track if any students match the given SGPA
    for (int i = 0; i < n; i++) {
        // If a student's SGPA matches the search value
        if (s[i].sgpa == sgpa) {
            found = true; // Mark as found
            // Print their details
            cout << "Roll No: " << s[i].rollNo
                 << ", Name: " << s[i].name << endl;
        }
    }
    if (!found) {
        // If no student matches the SGPA
        cout << "No students found with SGPA " << sgpa << endl;
    }
}
```

```
int main() {
    // Array of 15 students with roll number, name, and SGPA
    Student s[15] = {
        {1, "Alice", 9.2}, {2, "Bob", 8.8}, {3, "Charlie", 8.7}, {4, "David", 9.0},
        {5, "Eve", 9.5}, {6, "Frank", 7.8}, {7, "Grace", 8.0}, {8, "Hannah", 9.3},
        {9, "Ivy", 9.1}, {10, "Jack", 8.5}, {11, "Kevin", 9.5}, {12, "Luna", 8.9},
        {13, "Mike", 8.6}, {14, "Nina", 8.7}, {15, "Oscar", 8.4}
    };
    int n = 15; // Total number of students

    // a. Find top 10 toppers
    quickSort(s, 0, n - 1); // Sort the students by SGPA in descending order
    cout << "Top 10 Toppers:" << endl;
    display(s, 10); // Display the top 10 students

    // b. Search for students with a specific SGPA
    float sgpa; // SGPA to search for
    cout << "\nEnter SGPA to search: ";
    cin >> sgpa; // Input SGPA from the user
    searchSGPA(s, n, sgpa); // Search and display students with the entered SGPA

    return 0; // Exit the program
}
```

3. Consider a student database of SEIT class (at least 15 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of structure) a. Search a particular student according to name using binary search without recursion. (all the student records having the presence of search key should be displayed)

```
#include <iostream>
#include <algorithm> // For sort()
#include <string>   // For string
using namespace std;

// Structure to hold student details
struct Student {
    int rollNo;         // Roll Number of the student
    string name;        // Name of the student
    float sgpa;         // SGPA of the student
};

// Function to display student details
void displayStudent(const Student& student) {
    cout << "Roll No: " << student.rollNo << ", Name: " << student.name << ", SGPA: " <<
student.sgpa << endl;
}
```

```
// Binary Search Function to find students by name
void binarySearchByName(Student students[], int n, string key) {
    int left = 0, right = n - 1; // Initialize search range
    bool found = false;          // To check if any student is found

    while (left <= right) { // Loop until search range is valid
        int mid = (left + right) / 2; // Find the middle index

        if (students[mid].name == key) { // Check if the middle student matches the key
            found = true;

            // Display all matching students
            cout << "\nStudents with name '" << key << "':\n";

            // Check students with the same name on the left
            int i = mid;
            while (i >= 0 && students[i].name == key) {
                displayStudent(students[i]);
                i--;
            }

            // Check students with the same name on the right
            i = mid + 1;
            while (i < n && students[i].name == key) {
                displayStudent(students[i]);
                i++;
            }
            break;
        } else if (students[mid].name < key) {
            left = mid + 1; // Search the right half
        } else {
            right = mid - 1; // Search the left half
        }
    }

    if (!found) {
        cout << "\nNo student found with the name '" << key << "'.\n";
    }
}

int main() {
    // Array of 15 student records
    Student students[15] = {
        {1, "Alice", 8.5}, {2, "Bob", 9.0}, {3, "Charlie", 7.8}, {4, "Alice", 8.2},
        {5, "Eve", 9.1}, {6, "David", 7.5}, {7, "Alice", 8.3}, {8, "Frank", 6.9},
        {9, "Grace", 9.2}, {10, "Heidi", 7.6}, {11, "Ivan", 8.0}, {12, "Judy", 8.4},
        {13, "Mallory", 7.9}, {14, "Oscar", 8.6}, {15, "Peggy", 9.3}
```

```
    };

    // Sort the array of students by name (required for binary search)
    sort(students, students + 15, [](const Student& a, const Student& b) {
        return a.name < b.name;
    });

    // Input the name to search
    string searchName;
    cout << "Enter the name of the student to search: ";
    cin >> searchName;

    // Call the binary search function
    binarySearchByName(students, 15, searchName);

    return 0;
}
```

4. Implement stack as an abstract data type using singly linked list and use this ADT for conversion of infix expression to postfix and evaluation of postfix expression.

```
#include <iostream>
#include <stack>
#include <string>
#include <cmath> // For pow function
using namespace std;

// Function to check precedence
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 3;
    return 0;
}

// Function to convert infix to postfix
string infixToPostfix(const string& infix) {
    stack<char> s;
    string postfix;

    for (char ch : infix) {
        if (isalnum(ch)) {
            postfix += ch; // Add operand to postfix
        } else if (ch == '(') {
            s.push(ch); // Push '(' onto stack
        } else if (ch == ')') {
            while (!s.empty() && s.top() != '(') {
```

```
            postfix += s.top(); s.pop();
        }
        s.pop(); // Remove '(' from stack
    } else {
        while (!s.empty() && precedence(s.top()) >= precedence(ch)) {
            postfix += s.top(); s.pop();
        }
        s.push(ch); // Push current operator
    }
}

while (!s.empty()) {
    postfix += s.top(); s.pop();
}

return postfix;
}

// Function to evaluate postfix expression
int evaluatePostfix(const string& postfix) {
    stack<int> s;

    for (char ch : postfix) {
        if (isdigit(ch)) {
            s.push(ch - '0'); // Push operand
        } else {
            int b = s.top(); s.pop();
            int a = s.top(); s.pop();

            switch (ch) {
                case '+': s.push(a + b); break;
                case '-': s.push(a - b); break;
                case '*': s.push(a * b); break;
                case '/': s.push(a / b); break;
                case '^': s.push(pow(a, b)); break;
            }
        }
    }

    return s.top(); // Final result
}

int main() {
    string infix;
    cout << "Enter infix expression: ";
    cin >> infix;

    string postfix = infixToPostfix(infix);
```

```
    cout << "Postfix: " << postfix << endl;

    int result = evaluatePostfix(postfix);
    cout << "Result: " << result << endl;

    return 0;
}
```

5. Implement stack as an abstract data type using singly linked list and use this ADT for conversion of infix expression to prefix and evaluate prefix expression.

```
#include <iostream>
#include <stack>
#include <string>
#include <algorithm> // For reverse and swap
using namespace std;

// Function to check precedence
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}

// Function to convert infix to prefix
string infixToPrefix(string infix) {
    stack<char> s;
    string prefix = "";
    reverse(infix.begin(), infix.end());

    // Swap '(' and ')' in the reversed expression
    for (char& c : infix) {
        if (c == '(') c = ')';
        else if (c == ')') c = '(';
    }

    for (char c : infix) {
        if (isalnum(c)) {
            prefix = c + prefix; // Add operand to prefix
        } else if (c == '(') {
            s.push(c); // Push '(' onto stack
        } else if (c == ')') {
            while (!s.empty() && s.top() != '(') {
                prefix = s.top() + prefix; // Append operator to prefix
                s.pop();
```

```
            }
            s.pop(); // Remove '(' from stack
        } else {
            while (!s.empty() && precedence(s.top()) > precedence(c)) {
                prefix = s.top() + prefix; // Append operator to prefix
                s.pop();
            }
            s.push(c); // Push current operator
        }
    }

    while (!s.empty()) {
        prefix = s.top() + prefix; // Append remaining operators
        s.pop();
    }

    return prefix;
}

// Function to evaluate prefix expression
int evaluatePrefix(string prefix) {
    stack<int> s;
    for (int i = prefix.size() - 1; i >= 0; --i) {
        if (isdigit(prefix[i])) {
            s.push(prefix[i] - '0'); // Push operand
        } else {
            int op1 = s.top(); s.pop();
            int op2 = s.top(); s.pop();

            switch (prefix[i]) {
                case '+': s.push(op1 + op2); break;
                case '-': s.push(op1 - op2); break;
                case '*': s.push(op1 * op2); break;
                case '/': s.push(op1 / op2); break;
            }
        }
    }

    return s.top(); // Final result
}

int main() {
    string infix;
    cout << "Enter infix expression: ";
    cin >> infix;

    string prefix = infixToPrefix(infix);
    cout << "Prefix expression: " << prefix << endl;
```

```
    int result = evaluatePrefix(prefix);
    cout << "Evaluation result: " << result << endl;

    return 0;
}
```

6. Implement Circular Queue using Array. Perform following operations on it. a) Insertion (Enqueue) b) Deletion (Dequeue) c) Display

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

#define SIZE 5

class CircularQueue {
    int arr[SIZE], front = -1, rear = -1;

public:
    void enqueue(int value) {
        if ((front == 0 && rear == SIZE - 1) || (rear == (front - 1) % (SIZE - 1))) {
            cout << "Queue Full!\n";
            return;
        }
        rear = (rear + 1) % SIZE;
        if (front == -1) front = 0;
        arr[rear] = value;
    }

    int dequeue() {
        if (front == -1) {
            cout << "Queue Empty!\n";
            return -1;
        }
        int data = arr[front];
        if (front == rear) front = rear = -1;
        else front = (front + 1) % SIZE;
        return data;
    }

    void display() {
        if (front == -1) {
            cout << "Queue Empty!\n";
            return;
        }
```

```
        for (int i = front; i != rear; i = (i + 1) % SIZE)
            cout << arr[i] << " ";
        cout << arr[rear] << endl;
    }
};

int main() {
    CircularQueue cq;
    cq.enqueue(10);
    cq.enqueue(20);
    cq.enqueue(30);
    cq.display();
    cout << "Dequeued: " << cq.dequeue() << endl;
    cq.display();
    return 0;
}
```

7. Construct an Expression Tree from postfix expression and Perform recursive and non-recursive In-order, pre-order and post-order traversals.

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

struct Node {
    char data;
    Node *left, *right;
    Node(char val) : data(val), left(nullptr), right(nullptr) {}
};

Node* buildTree(string postfix) {
    stack<Node*> s;
    for (char c : postfix) {
        Node* node = new Node(c);
        if (!isalnum(c)) {
            node->right = s.top(); s.pop();
            node->left = s.top(); s.pop();
        }
        s.push(node);
    }
    return s.top();
}

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() {
    string postfix;
    cout << "Enter postfix expression: ";
    cin >> postfix;

    Node* root = buildTree(postfix);

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

    return 0;
}
```

8. Construct an Expression Tree from prefix expression and Perform recursive and non-recursive In-order, pre-order and post-order traversals.

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

// Node structure for Expression Tree
struct Node {
    char data;
    Node* left;
    Node* right;
};

// Create a new node
Node* createNode(char data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = nullptr;
    newNode->right = nullptr;
    return newNode;
}

// Construct Expression Tree from prefix expression
Node* constructTree(string prefix) {
```

```
    stack<Node*> s;
    for (int i = prefix.size() - 1; i >= 0; i--) {
        if (isalpha(prefix[i])) {
            s.push(createNode(prefix[i]));
        } else {
            Node* node = createNode(prefix[i]);
            node->left = s.top(); s.pop();
            node->right = s.top(); s.pop();
            s.push(node);
        }
    }
    return s.top();
}

// In-order traversal (Recursive)
void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

// Pre-order traversal (Recursive)
void preorder(Node* root) {
    if (root) {
        cout << root->data << " ";
        preorder(root->left);
        preorder(root->right);
    }
}

// Post-order traversal (Recursive)
void postorder(Node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        cout << root->data << " ";
    }
}

int main() {
    string prefix;  // Example prefix expression
    cout << "Enter prefix expression: ";
    cin >> prefix;
    Node* root = constructTree(prefix);

    cout << "In-order: ";
```

```
    inorder(root);
    cout << endl;

    cout << "Pre-order: ";
    preorder(root);
    cout << endl;

    cout << "Post-order: ";
    postorder(root);
    cout << endl;

    return 0;
}
```

9. Implement binary search tree and perform following operations: a) Insert (Handle insertion of duplicate entry) b) Delete c) Search d) Display Tree e) Display - Depth of tree

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

// Node structure for BST
struct Node {
    int data;
    Node* left;
    Node* right;
};

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = nullptr;
    newNode->right = nullptr;
    return newNode;
}

// Function to insert into BST (Handles duplicates)
Node* insert(Node* root, int data) {
    if (!root) return createNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);
```

```
    return root;
}

// Function to delete a node from BST
Node* deleteNode(Node* root, int data) {
    if (!root) return nullptr;
    if (data < root->data)
        root->left = deleteNode(root->left, data);
    else if (data > root->data)
        root->right = deleteNode(root->right, data);
    else {
        if (!root->left) return root->right;
        else if (!root->right) return root->left;

        Node* successor = root->right;
        while (successor->left)
            successor = successor->left;
        root->data = successor->data;
        root->right = deleteNode(root->right, successor->data);
    }
    return root;
}

// Function to search in BST
bool search(Node* root, int data) {
    if (!root) return false;
    if (root->data == data) return true;
    if (data < root->data) return search(root->left, data);
    else return search(root->right, data);
}

// Function to display BST (In-order traversal)
void display(Node* root) {
    if (root) {
        display(root->left);
        cout << root->data << " ";
        display(root->right);
    }
}

// Function to get the depth of the tree
int getDepth(Node* root) {
    if (!root) return 0;
    return max(getDepth(root->left), getDepth(root->right)) + 1;
}

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

```
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

    cout << "Tree (In-order): ";
    display(root);
    cout << endl;

    cout << "Search 40: " << (search(root, 40) ? "Found" : "Not Found") << endl;
    cout << "Tree Depth: " << getDepth(root) << endl;

    root = deleteNode(root, 50);
    cout << "After deleting 50 (In-order): ";
    display(root);
    cout << endl;

    return 0;
}
```

10. Implement binary search tree and perform following operations: a) Insert (Handle insertion of duplicate entry) b) Delete c) Search d) Display Tree F) Display - Mirror image

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

// Node structure for BST
struct Node {
    int data;
    Node* left;
    Node* right;
};

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = nullptr;
    newNode->right = nullptr;
    return newNode;
}
```

```
// Function to insert into BST (Handles duplicates)
Node* insert(Node* root, int data) {
    if (!root) return createNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);
    return root;
}

// Function to delete a node from BST
Node* deleteNode(Node* root, int data) {
    if (!root) return nullptr;
    if (data < root->data)
        root->left = deleteNode(root->left, data);
    else if (data > root->data)
        root->right = deleteNode(root->right, data);
    else {
        if (!root->left) return root->right;
        if (!root->right) return root->left;

        Node* successor = root->right;
        while (successor->left)
            successor = successor->left;
        root->data = successor->data;
        root->right = deleteNode(root->right, successor->data);
    }
    return root;
}

// Function to search in BST
bool search(Node* root, int data) {
    if (!root) return false;
    if (root->data == data) return true;
    if (data < root->data) return search(root->left, data);
    return search(root->right, data);
}

// Function to display BST (In-order traversal)
void display(Node* root) {
    if (root) {
        display(root->left);
        cout << root->data << " ";
        display(root->right);
    }
}
```

```
// Function to display the mirror image of the tree
void displayMirror(Node* root) {
    if (root) {
        displayMirror(root->right);  // Mirror image by swapping left/right
        cout << root->data << " ";
        displayMirror(root->left);
    }
}

int main() {
    Node* root = nullptr;

    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

    cout << "Tree (In-order): ";
    display(root);
    cout << endl;

    cout << "Search 40: " << (search(root, 40) ? "Found" : "Not Found") << endl;

    cout << "Tree Mirror (In-order): ";
    displayMirror(root);
    cout << endl;

    root = deleteNode(root, 50);
    cout << "After deleting 50 (In-order): ";
    display(root);
    cout << endl;

    return 0;
}
```

{not correct code}

10. Implement In-order Threaded Binary Tree and traverse it in In-order and Pre-order.

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

```
// Node structure for Threaded Binary Tree
struct Node {
    int data;
    Node* left;
    Node* right;
    bool isThread;  // Flag to indicate if it's a thread
};

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = nullptr;
    newNode->right = nullptr;
    newNode->isThread = false;
    return newNode;
}

// Function to create In-order Threads
void threadInOrder(Node* root, Node*& prev) {
    if (root == nullptr) return;

    threadInOrder(root->left, prev);

    if (prev) {
        prev->right = root;
        prev->isThread = true;
    }
    prev = root;

    threadInOrder(root->right, prev);
}

// Function for In-order Traversal using threads
void inOrderTraversal(Node* root) {
    Node* current = root;
    while (current) {
        while (!current->isThread && current->left) {
            current = current->left;
        }
        cout << current->data << " ";
        while (current->isThread) {
            current = current->right;
            cout << current->data << " ";
        }
        current = current->right;
    }
    cout << endl;
```

```
}

// Function for Pre-order Traversal
void preOrderTraversal(Node* root) {
    if (root) {
        cout << root->data << " ";
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}

int main() {
    Node* root = createNode(10);
    root->left = createNode(5);
    root->right = createNode(15);
    root->left->left = createNode(2);
    root->left->right = createNode(7);
    root->right->right = createNode(20);

    Node* prev = nullptr;
    threadInOrder(root, prev);

    cout << "In-order Traversal: ";
    inOrderTraversal(root);

    cout << "Pre-order Traversal: ";
    preOrderTraversal(root);
    cout << endl;

    return 0;
}
```

11. Represent a graph of your college campus using adjacency list /adjacency matrix. Nodes should represent the various departments/institutes and links should represent the distance between them. Find minimum spanning tree Using Kruskal's algorithm.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// Edge structure for graph
struct Edge {
    int src, dest, weight;
};
```

```
// Disjoint Set (Union-Find) for cycle detection
class DisjointSet {
    vector<int> parent, rank;

public:
    DisjointSet(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; ++i) parent[i] = i;
    }

    int find(int u) {
        if (parent[u] != u)
            parent[u] = find(parent[u]);  // Path compression
        return parent[u];
    }

    void unionSets(int u, int v) {
        int rootU = find(u);
        int rootV = find(v);

        if (rootU != rootV) {
            if (rank[rootU] > rank[rootV])
                parent[rootV] = rootU;
            else if (rank[rootU] < rank[rootV])
                parent[rootU] = rootV;
            else {
                parent[rootV] = rootU;
                rank[rootU]++;
            }
        }
    }
};

// Kruskal's Algorithm to find MST
vector<Edge> kruskalMST(int n, vector<Edge>& edges) {
    vector<Edge> result;
    sort(edges.begin(), edges.end(), [](Edge a, Edge b) { return a.weight < b.weight; });

    DisjointSet ds(n);

    for (auto& edge : edges) {
        int u = edge.src;
        int v = edge.dest;
        if (ds.find(u) != ds.find(v)) {
            ds.unionSets(u, v);
            result.push_back(edge);
```

```
        }
    }

    return result;
}

int main() {
    int n = 5;  // Number of departments
    vector<Edge> edges = {
        {0, 1, 2}, {0, 3, 6}, {0, 4, 5},
        {1, 2, 3}, {1, 3, 8},
        {2, 3, 7}, {3, 4, 9}
    };

    vector<Edge> mst = kruskalMST(n, edges);

    cout << "Minimum Spanning Tree (MST) Edges:\n";
    for (auto& edge : mst) {
        cout << edge.src << " - " << edge.dest << " : " << edge.weight << endl;
    }

    return 0;
}
```

12. Represent a graph of your college campus using adjacency list /adjacency matrix. Nodes should represent the various departments/institutes and links should represent the distance between them. Find minimum spanning tree Using Prim's algorithm.

```
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

// Prim's Algorithm to find MST
void primMST(vector<vector<int>>& graph, int n) {
    vector<bool> inMST(n, false);  // To track included nodes in MST
    vector<int> key(n, INT_MAX);    // Minimum edge weight to include each vertex
    vector<int> parent(n, -1);      // To store MST
    key[0] = 0;

    for (int i = 0; i < n - 1; ++i) {
        int minKey = INT_MAX, u;
        for (int v = 0; v < n; ++v) {
            if (!inMST[v] && key[v] < minKey) {
                minKey = key[v];
```

```
                u = v;
            }
        }

        inMST[u] = true;

        for (int v = 0; v < n; ++v) {
            if (graph[u][v] && !inMST[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    // Output MST
    cout << "Minimum Spanning Tree (MST) Edges:\n";
    for (int i = 1; i < n; ++i) {
        cout << parent[i] << " - " << i << " : " << graph[i][parent[i]] << endl;
    }
}

int main() {
    int n = 5;  // Number of departments
    vector<vector<int>> graph = {
        {0, 2, INT_MAX, 6, 5},
        {2, 0, 3, 8, INT_MAX},
        {INT_MAX, 3, 0, 7, INT_MAX},
        {6, 8, 7, 0, 9},
        {5, INT_MAX, INT_MAX, 9, 0}
    };

    primMST(graph, n);

    return 0;
}
```

13. Represent a graph of city using adjacency matrix /adjacency list. Nodes should represent the various landmarks and links should represent the distance between them. Find the shortest path using Dijkstra's algorithm from single source to all destination.

```
#include <iostream>
#include <vector>
#include <climits>
#include <queue>

using namespace std;
```

```
// Graph representation using adjacency matrix
const int INF = INT_MAX;

// Dijkstra's Algorithm to find shortest path
void dijkstra(vector<vector<int>>& graph, int n, int source) {
    vector<int> dist(n, INF);      // Distance from source to each node
    vector<bool> visited(n, false);  // To track visited nodes
    dist[source] = 0;

    for (int i = 0; i < n - 1; ++i) {
        // Select the minimum distance node
        int u = -1;
        int minDist = INF;
        for (int v = 0; v < n; ++v) {
            if (!visited[v] && dist[v] < minDist) {
                minDist = dist[v];
                u = v;
            }
        }

        visited[u] = true;

        for (int v = 0; v < n; ++v) {
            if (graph[u][v] && !visited[v] && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    // Output shortest distances
    cout << "Shortest paths from source " << source << ":\n";
    for (int i = 0; i < n; ++i) {
        cout << "Distance to " << i << " : " << dist[i] << endl;
    }
}

int main() {
    int n = 5;  // Number of landmarks
    vector<vector<int>> graph = {
        {0, 10, INF, 30, 100},
        {10, 0, 50, INF, INF},
        {INF, 50, 0, 20, 10},
        {30, INF, 20, 0, 60},
        {100, INF, 10, 60, 0}
    };

    int source = 0;  // Starting landmark
```

```
    dijkstra(graph, n, source);

    return 0;
}
```

14. Implement Heap sort to sort given set of values using max or min heap.

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

class MaxHeap {
    vector<int> heap;

public:
    void insert(int val) {
        heap.push_back(val);
        int i = heap.size() - 1;
        while (i > 0 && heap[i] > heap[(i - 1) / 2]) {
            swap(heap[i], heap[(i - 1) / 2]);
            i = (i - 1) / 2;
        }
    }

    void heapify(int n, int i) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;

        if (left < n && heap[left] > heap[largest])
            largest = left;

        if (right < n && heap[right] > heap[largest])
            largest = right;

        if (largest != i) {
            swap(heap[i], heap[largest]);
            heapify(n, largest);
        }
    }

    void heapSort() {
        int n = heap.size();
        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(n, i);
        }
```

```
        for (int i = n - 1; i > 0; i--) {
            swap(heap[0], heap[i]);
            heapify(i, 0);
        }
    }

    void printHeap() {
        for (int val : heap)
            cout << val << " ";
        cout << endl;
    }
};

int main() {
    MaxHeap maxHeap;
    vector<int> arr = {5, 1, 10, 3, 7, 8, 2};

    for (int val : arr) {
        maxHeap.insert(val);
    }

    maxHeap.heapSort();
    maxHeap.printHeap();

    return 0;
}
```

{not correct code}

15. Department maintains student's database. The file contains roll number, name, division and address. Write a program to create a sequential file to store and maintain student data. It should allow the user to add, delete information of student. Display information of particular student. If record of student does not exist an appropriate message is displayed. If student record is found it should display the student details.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

using namespace std;

// Structure to store student information
struct Student {
    int rollNumber;
    string name;
    char division;
```

```
    string address;
};

// Function to add a student record
void addStudent() {
    ofstream outFile("students.dat", ios::app | ios::binary);
    if (!outFile) {
        cerr << "File could not be opened!" << endl;
        return;
    }

    Student s;
    cout << "Enter Roll Number: ";
    cin >> s.rollNumber;
    cin.ignore();
    cout << "Enter Name: ";
    getline(cin, s.name);
    cout << "Enter Division: ";
    cin >> s.division;
    cin.ignore();
    cout << "Enter Address: ";
    getline(cin, s.address);

    outFile.write(reinterpret_cast<char*>(&s), sizeof(Student));
    outFile.close();
    cout << "Student added successfully!" << endl;
}

// Function to display a student record
void displayStudent(int rollNumber) {
    ifstream inFile("students.dat", ios::binary);
    if (!inFile) {
        cerr << "File could not be opened!" << endl;
        return;
    }

    Student s;
    bool found = false;
    while (inFile.read(reinterpret_cast<char*>(&s), sizeof(Student))) {
        if (s.rollNumber == rollNumber) {
            found = true;
            cout << "Roll Number: " << s.rollNumber << endl;
            cout << "Name: " << s.name << endl;
            cout << "Division: " << s.division << endl;
            cout << "Address: " << s.address << endl;
            break;
        }
    }
```

```
    inFile.close();

    if (!found) {
        cout << "Student record not found!" << endl;
    }
}

int main() {
    int choice, rollNumber;
    do {
        cout << "1. Add Student" << endl;
        cout << "2. Display Student" << endl;
        cout << "3. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            addStudent();
            break;
        case 2:
            cout << "Enter Roll Number to display: ";
            cin >> rollNumber;
            displayStudent(rollNumber);
            break;
        case 3:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice!" << endl;
        }
    } while (choice != 3);

    return 0;
}
```