1. Write a program to sort n numbers using Insertion Sort algorithm.

```cpp
#include <bits/stdc++.h>
using namespace std;

void insertionSort(vector<int>& arr) {
    int n = arr.size();
    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 = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main()
{

    vector<int>arr;
    while (true)
    {
        int x;
        cin >> x;
        if (x < 0)
            break;
        arr.push_back(x);
    }

    insertionSort(arr);

    cout << "Sorted array: ";
    for (int i = 0; i < arr.size(); i++)
        cout << arr[i] << " ";
    cout << "\n";

    return 0;
}
```

2. Write a program to sort n numbers using Selection Sort algorithm.

```cpp
#include <bits/stdc++.h>
using namespace std;

void selectionSort(vector<int> &arr)
{
    int n = arr.size();
    for (int i = 0; i < n - 1; ++i)
    {
        int min_idx = i;
        for (int j = i + 1; j < n; ++j)
            if (arr[j] < arr[min_idx])
                min_idx = j;
        swap(arr[min_idx], arr[i]);
    }
}

int main()
{
    vector<int> arr;
    while (true)
    {
        int x;
        cin >> x;
        if (x <0)
            break;
        arr.push_back(x);
    }

    selectionSort(arr);

    cout << "Sorted array: ";
    for (int i = 0; i < arr.size(); i++)
        cout << arr[i] << " ";
    cout << "\n";

    return 0;
}
```

3. Write a program to sort n numbers using Quick Sort algorithm.

```cpp
#include <bits/stdc++.h>
using namespace std;

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

void quickSort(vector<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()
{

    vector<int> arr;
    while (true)
    {
        int x;
        cin >> x;
        if (x < 0)
            break;
        arr.push_back(x);
    }
```

```
    quickSort(arr, 0, arr.size() - 1);

    cout << "Sorted array: ";
    for (int i = 0; i < arr.size(); i++)
        cout << arr[i] << " ";
    cout << "\n";

    return 0;
}
```

4. Write a program to merge two sorted list.

```
#include <bits/stdc++.h>
using namespace std;

vector<int> mergeSortedLists(vector<int> &list1, vector<int> &list2)
{
    vector<int> mergedList;
    int i = 0, j = 0;
    while (i < list1.size() && j < list2.size())
    {
        if (list1[i] < list2[j])
        {
            mergedList.push_back(list1[i]);
            i++;
        }
        else
        {
            mergedList.push_back(list2[j]);
            j++;
        }
    }
    while (i < list1.size())
    {
        mergedList.push_back(list1[i]);
        i++;
    }
    while (j < list2.size())
    {
        mergedList.push_back(list2[j]);
```

```
        j++;
    }
    return mergedList;
}

int main()
{
    vector<int> list1;
    vector<int> list2;
    int x;
    while (true)
    {
        cin >> x;
        if (x < 0)
            break;
        list1.push_back(x);
    }

    while (true)
    {
        cin >> x;
        if (x < 0)
            break;
        list2.push_back(x);
    }
    sort(list1.begin(), list1.end());
    sort(list2.begin(), list2.end());

    vector<int> mergedList = mergeSortedLists(list1, list2);

    cout << "Merged list: ";
    for (int i = 0; i < mergedList.size(); i++)
        cout << mergedList[i] << " ";
    cout << "\n";

    return 0;
}
```

5. Write a program to sort n numbers using Merge Sort algorithm.

```cpp
#include <bits/stdc++.h>
using namespace std;

void merge(vector<int> &arr, int left, int mid, int right)
{
    int n1 = mid - left + 1;
    int n2 = right - mid;

    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
```

```
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(vector<int> &arr, int left, int right)
{
    if (left < right)
    {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

int main()
{

    vector<int> arr;
    int x;

    while (true)
    {
        cin >> x;
        if (x < 0)
            break;
        arr.push_back(x);
    }
    sort(arr.begin(), arr.end());

    mergeSort(arr, 0, arr.size() - 1);

    cout << "Sorted array: ";
    for (int i = 0; i < arr.size(); i++)
```

```
        cout << arr[i] << " ";
    cout << "\n";

    return 0;
}
```

6. Write a program to create a Binary Search Tree of n elements and then display the elements (preorder, inorder and postorder) of the tree.

```
#include <bits/stdc++.h>
using namespace std;

class Node
{
public:
    int val;
    Node *left, *right;
    Node(int val)
    {
        this->val = val;
        this->left = NULL;
        this->right = NULL;
    }
};

Node *insert(Node *root, int val)
{
    if (root == NULL)
        return new Node(val);
    if (val < root->val)
        root->left = insert(root->left, val);
    else
        root->right = insert(root->right, val);
    return root;
}

void preOrder(Node *root)
{
    if (root == NULL)
        return;
    cout << root->val << " ";
```

```
    preOrder(root->left);
    preOrder(root->right);
}

void postOrder(Node *root)
{
    if (root == NULL)
        return;
    postOrder(root->left);
    postOrder(root->right);
    cout << root->val << " ";
}

void InOrder(Node *root)
{
    if (root == NULL)
        return;
    InOrder(root->left);
    cout << root->val << " ";
    InOrder(root->right);
}

int main()
{
    Node *root = NULL;
    int n, x;

    cout << "Number of elements:\n";
    cin >> n;

    cout << "Enter the elements:\n";
    for (int i = 0; i < n; i++)
    {
        cin >> x;
        root = insert(root, x);
    }

    cout << "Preorder traversal: ";
    preOrder(root);
    cout << "\n";

    cout << "Inorder traversal: ";
    InOrder(root);
```

```
    cout << "\n";

    cout << "Postorder traversal: ";
    postOrder(root);
    cout << "\n";

    return 0;
}
```

7. Write a program to create a Binary Search Tree of n elements and then search an element from the tree.

```
#include <bits/stdc++.h>
using namespace std;

class Node
{
public:
    int val;
    Node *left, *right;
    Node(int val)
    {
        this->val = val;
        this->left = NULL;
        this->right = NULL;
    }
};

Node *insert(Node *root, int val)
{
    if (root == NULL)
        return new Node(val);
    if (val < root->val)
        root->left = insert(root->left, val);
    else
        root->right = insert(root->right, val);
    return root;
}

bool search(Node *root, int val)
{
    if (root == NULL)
        return false;
```

```
    if (root->val == val)
        return true;
    if (val < root->val)
        return search(root->left, val);
    else
        return search(root->right, val);
}

int main()
{

    Node *root = NULL;
    int n, x;

    cout << "Number of elements:\n";
    cin >> n;

    cout << "Enter the elements:\n";
    for (int i = 0; i < n; i++)
    {
        cin >> x;
        root = insert(root, x);
    }

    cout << "Element to search:\n";
    cin >> x;

    if (search(root, x))
        cout << "Found\n";
    else
        cout << "Not found\n";

    return 0;
}
```

9. Write a program to create a Maxheap of n elements and then display the elements of the heap.

```
#include <bits/stdc++.h>
using namespace std;

void insert_Max_Heap(vector<int> &v, int x)
{
    v.push_back(x);
    int cur_idx = v.size() - 1;
    while (cur_idx != 0)
    {
        int parent_idx = (cur_idx - 1) / 2;
        if (v[parent_idx] < v[cur_idx])
            swap(v[parent_idx], v[cur_idx]);
        else
            break;
        cur_idx = parent_idx;
    }
}
void print_heap(const vector<int> &v)
{
    for (int u : v)
        cout << u << " ";
    cout << endl;
}
int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    vector<int> maxHeap;
    cout << "Enter the elements:\n";
```

```
    for (int i = 0; i < n; ++i)
    {
        int x;
        cin >> x;
        insert_Max_Heap(maxHeap, x);
    }
    cout << "Maxheap elements: ";
    print_heap(maxHeap);
    return 0;
}
```

10. Write a program to create a Maxheap of n elements and then delete an element from the heap.

```
#include <bits/stdc++.h>
using namespace std;

void insert_Max_Heap(vector<int> &v, int x)
{
    v.push_back(x);
    int cur_idx = v.size() - 1;
    while (cur_idx != 0)
    {
        int parent_idx = (cur_idx - 1) / 2;
        if (v[parent_idx] < v[cur_idx])
            swap(v[parent_idx], v[cur_idx]);
        else
            break;
        cur_idx = parent_idx;
    }
}

void delete_Max_Heap(vector<int> &v)
{
    if (v.empty())
        return;
    v[0] = v.back();
    v.pop_back();
    int cur = 0;
    while (true)
    {
        int left = (cur * 2) + 1;
        int right = (cur * 2) + 2;
```

```
        int last = v.size() - 1;
        int largest = cur;

        if (left <= last && v[left] > v[largest])
            largest = left;
        if (right <= last && v[right] > v[largest])
            largest = right;
        if (largest != cur)
        {
            swap(v[cur], v[largest]);
            cur = largest;
        }
        else
        {
            break;
        }
    }
}

void print_heap(const vector<int> &v)
{
    for (int u : v)
        cout << u << " ";
    cout << endl;
}

int main()
{
    int n;
    cout << "Number of elements: ";
    cin >> n;

    vector<int> maxHeap;
    cout << "Enter the elements:\n";
    for (int i = 0; i < n; ++i)
    {
        int x;
        cin >> x;
        insert_Max_Heap(maxHeap, x);
    }

    cout << "Maxheap elements before deletion: ";
    print_heap(maxHeap);
```

```
    delete_Max_Heap(maxHeap);

    cout << "Maxheap elements after deletion: ";
    print_heap(maxHeap);

    return 0;
}
```

11. Write a program to sort n numbers using Heap sort algorithm.

```
#include <bits/stdc++.h>
using namespace std;

void heap(vector<int> &arr, int n, int i)
{
    int cur = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[cur])
        cur = left;

    if (right < n && arr[right] > arr[cur])
        cur = right;

    if (cur != i)
    {
        swap(arr[i], arr[cur]);
        heap(arr, n, cur);
    }
}

void heapSort(vector<int> &arr, int n)
{

    for (int i = n / 2 - 1; i >= 0; i--)
        heap(arr, n, i);
```

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

void print_heap(const vector<int> &v)

{
    for (int u : v)
        cout << u << " ";
    cout << endl;
}

int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter the elements: ";

    for (int i = 0; i < n; i++)
        cin >> arr[i];

    heapSort(arr, n);
    cout << "Sorted array is: ";
    print_heap(arr);

    return 0;
}
```

12 & 15. Write a program to display the adjacency matrix of a graph.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
```

```
    int n, e;
    cout << "Enter the number of vertices: ";
    cin >> n;
    cout << "Enter the number of edges: ";
    cin >> e;

    int mat[n][n];
    memset(mat, 0, sizeof(mat));

    cout << "Enter the edges : " << endl;
    while (e--)
    {
        int a, b;
        cin >> a >> b;
        mat[a][b] = 1;
        mat[b][a] = 1;
    }

    cout << "The adjacency matrix is:" << endl;
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            cout << mat[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

16. Write a program to traverse a graph using Breadth First Search.

```
#include <bits/stdc++.h>
using namespace std;
vector<int> v[1005];

bool vis[1005];
int level[1005] = {-1};
void bfs(int src)
{
    queue<int> q;
```

```
    q.push(src);
    vis[src] = true;
    level[src] = 0;
    while (!q.empty())
    {
        int par = q.front();
        q.pop();

        cout << par << " ";

        for (auto child : v[par])
        {
            if (vis[child] == false)
            {
                q.push(child);
                vis[child] = true;
                level[child] = level[par] + 1;
            }
        }
    }
}
int main()
{
    int n, e;
    cin >> n >> e;
    while (e--)
    {
        int a, b;
        cin >> a >> b;
        v[a].push_back(b);
        v[b].push_back(a);
    }
    int src;
    cin >> src;

    memset(vis, false, sizeof(vis));

    bfs(src);
    return 0;
}
```

17. Write a program to traverse a graph using Depth First Search.

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
vector<int> v[N];
bool vis[N];
void dfs(int src)
{
    cout << src << " ";
    vis[src] = true;
    for (auto child : v[src])
    {
        if (vis[child] == false)
        {
            dfs(child);
        }
    }
}
int main()
{
    int n, e;
    cin >> n >> e;
    while (e--)
    {
        int a, b;
        cin >> a >> b;
        v[a].push_back(b);
        v[b].push_back(a);
    }

    memset(vis, false, sizeof(vis));
    int src;
    cin >> src;
    dfs(src);
    return 0;
}
```

18. Write a program to implement a hash table using Division method & use linear probing for collision resolution.

```cpp
#include <bits/stdc++.h>
#define SIZE 8
using namespace std;
```

```
int HashTable[SIZE];

int hashFunction(int data)
{
    return data % SIZE;
}

void insert(int data)
{
    int idx = hashFunction(data);
    while (HashTable[idx] != -1)
    {
        idx = (idx + 1) % SIZE;
    }
    HashTable[idx] = data;
}

int search(int data)
{
    int idx = hashFunction(data);
    int startIdx = idx;
    while (HashTable[idx] != -1)
    {
        if (HashTable[idx] == data)
            return idx;
        idx = (idx + 1) % SIZE;
        if (idx == startIdx)
            break;
    }
    return -1;
}

void deleteElement(int data)
{
    int idx = search(data);
    if (idx != -1)
    {
        HashTable[idx] = -1;
    }
}

int main()
```

```
{

    fill(begin(HashTable), end(HashTable), -1);

    int arr[8] = {33, 21, 5, 13, 4, 1, 2, 12};
    int n = 8;

    for (int i = 0; i < n; i++)
    {
        insert(arr[i]);
    }

    for (int i = 0; i < SIZE; i++)
    {
        cout << HashTable[i] << " ";
    }
    cout << endl;

    int s = 11;
    cout << "Search item: " << s << endl;
    if (search(s) == -1)
        cout << "NOT FOUND" << endl;
    else
        cout << "FOUND" << endl;

    int d = 4;
    cout << "Delete element: " << d << endl;
    deleteElement(d);

    for (int i = 0; i < SIZE; i++)
    {
        cout << HashTable[i] << " ";
    }
    cout << endl;

    return 0;
}
```

19. Write a program to implement a hash table using Division method & use double hashing for collision resolution.

```
#include <bits/stdc++.h>
#define SIZE 10
using namespace std;

int HashTable[SIZE];

int hashFunction1(int key)
{
    return key % SIZE;
}

int hashFunction2(int key)
{
    int prime = 7;
    return prime - (key % prime);
}

void insert(int key)
{
    int idx1 = hashFunction1(key);
    int idx2 = hashFunction2(key);

    for (int i = 0; i < SIZE; i++)
    {
        int newIndex = (idx1 + i * idx2) % SIZE;
        if (HashTable[newIndex] == -1)
        {
            HashTable[newIndex] = key;
            return;
        }
    }

    cout << "Hash table is full, cannot insert " << key << endl;
}

bool search(int key)
{
    int idx1 = hashFunction1(key);
    int idx2 = hashFunction2(key);
```

```
    for (int i = 0; i < SIZE; i++)
    {
        int newIndex = (idx1 + i * idx2) % SIZE;
        if (HashTable[newIndex] == key)
        {
            return true;
        }
        if (HashTable[newIndex] == -1)
        {
            return false;
        }
    }
    return false;
}

void deleteElement(int key)
{
    int idx1 = hashFunction1(key);
    int idx2 = hashFunction2(key);

    for (int i = 0; i < SIZE; i++)
    {
        int newIndex = (idx1 + i * idx2) % SIZE;
        if (HashTable[newIndex] == key)
        {
            HashTable[newIndex] = -1;
            return;
        }
        if (HashTable[newIndex] == -1)
        {
            return;
        }
    }
}

int main()
{
    fill(begin(HashTable), end(HashTable), -1);

    vector<int> items = {3, 7, 24, 16, 9, 5, 6};
    for (int item : items)
    {
```

```
        insert(item);
    }

    cout << "Hash table after insertions: ";
    for (int i = 0; i < SIZE; i++)
    {
        if (HashTable[i] != -1)
        {
            cout << HashTable[i] << " ";
        }
        else
        {
            cout << "_ ";
        }
    }
    cout << endl;

    int searchKey = 24;
    if (search(searchKey))
    {
        cout << "Element " << searchKey << " found in the hash table." << endl;
    }
    else
    {
        cout << "Element " << searchKey << " not found in the hash table." <<
endl;
    }

    int deleteKey = 24;
    deleteElement(deleteKey);
    cout << "Hash table after deleting " << deleteKey << ": ";
    for (int i = 0; i < SIZE; i++)
    {
        if (HashTable[i] != -1)
        {
            cout << HashTable[i] << " ";
        }
        else
        {
            cout << "_ ";
        }
    }
    cout << endl;
```

```
    return 0;
}
```