

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

#include <bits/stdc++.h>

using namespace std;

void insertsort(vector<int>&ar, int n)

{

    for (int i = 1; i < n; i++)

    {

        int mil = ar[i], j = i - 1;

        while (j >= 0 && ar[j] > mil)

        {

            ar[j + 1] = ar[j--];

        }

        ar[j + 1] = mil;

    }

}

int main()

{

    int a;

    cout << "Input Size : ";

    cin >> a;

    vector<int> ar(a);

    cout << "Input Elements : ";

    for (int i = 0; i < a; i++) cin >> ar[i];

    insertsort(ar,a);

    cout << "Array after Sort : ";

    for (int i = 0; i < a; i++) cout << ar[i] << " ";

}

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

#include <bits/stdc++.h>

using namespace std;

void selectsort(vector<int>&ar, int n)

{

    for (int i = 0; i < n - 1; i++)

    {

        int idx = i;

        for (int j = i + 1; j < n; j++)

        {

            if (ar[j] < ar[idx])

            {

                idx = j;

            }

        }

        swap(ar[i], ar[idx]);

    }

}

int main()

{

    int a;

    cout << "Input Size : ";

    cin >> a;

    vector<int> ar(a);

    cout << "Input Elements : ";

    for (int i = 0; i < a; i++) cin >> ar[i];

    selectsort(ar, a);

    cout << "Array after Sort : ";

    for (int i = 0; i < a; i++) cout << ar[i] << " ";

}

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

#include <bits/stdc++.h>

using namespace std;

int partition(vector<int> &ar, int l, int h)

{

    int pivot = ar[h];

    int i = (l - 1);

    for (int j = l; j < h; j++)

    {

        if (ar[j] <= pivot)

        {

            i++;

            swap(ar[i], ar[j]);

        }

    }

    swap(ar[i + 1], ar[h]);

    return (i + 1);

}

void quicksort(vector<int> &ar, int l, int h)

{

    if (l < h)

    {

        int pi = partition(ar, l, h);

        quicksort(ar, l, pi - 1);

        quicksort(ar, pi + 1, h);

    }

}

int main()

{

    int a;

    cout << "Input Size : ";

    cin >> a;

    vector<int> ar(a);

    cout << "Input Elements : ";

    for (int i = 0; i < a; i++) cin >> ar[i];

    quicksort(ar, 0, a-1);

    cout << "Array after Sort : ";

    for (int i = 0; i < a; i++) cout << ar[i] << " ";

}

**4. Write a program to merge two sorted list.**

#include <bits/stdc++.h>

using namespace std;

void mergearray(int A[], int R, int B[], int S, int C[])

{

    int NA = 1, NB = 1, PTR = 1;

    while (NA <= R && NB <= S)

    {

        if (A[NA] < B[NB])

        {

            C[PTR] = A[NA];

            PTR++;

            NA++;

        }

        else

        {

            C[PTR] = B[NB];

            PTR++;

            NB++;

        }

    }

    if (NA > R)

    {

        for (int K = 0; K <= S - NB; K++)

        {

            C[PTR + K] = B[NB + K];

        }

    }

    else

    {

        for (int K = 0; K <= R - NA; K++)

        {

            C[PTR + K] = A[NA + K];

        }

    }

}

int main()

{

    int R, S;

    cout << "Input Size of Ar1&Ar2 : ";

    cin >> R >> S;

    int ar1[R + 1], ar2[S + 1], C[R + S + 1];

    cout << "Input elements of Ar1 : ";

    for (int i = 1; i <= R; i++)

    {

        cin >> ar1[i];

    }

    cout << "Input elements of Ar2 : ";

    for (int i = 1; i <= S; i++)

    {

        cin >> ar2[i];

    }

    mergearray(ar1, R, ar2, S, C);

    cout << "Merged array : ";

    for (int i = 1; i <= R + S; i++)

    {

        cout << C[i] << " ";

    }

    cout << endl;

    return 0;

}

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

#include <iostream>

using namespace std;

void partition(int ar[], int l, int m, int r)

{

    int n1 = m - l + 1, n2 = r - m;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)

    {

        L[i] = ar[l + i];

    }

    for (int i = 0; i < n2; i++)

    {

        R[i] = ar[m + 1 + i];

    }

    int i = 0, j = 0, k = l;

    while (i < n1 && j < n2)

    {

        if (L[i] <= R[j])

        {

            ar[k] = L[i];

            i++;

        }

        else

        {

            ar[k] = R[j];

            j++;

        }

        k++;

    }

    while (i < n1)

    {

        ar[k] = L[i];

        i++;

        k++;

    }

    while (j < n2)

    {

        ar[k] = R[j];

        j++;

        k++;

    }

}

void mergesort(int ar[], int l, int r)

{

    if (l < r)

    {

        int m = l + (r - l) / 2;

        mergesort(ar, l, m);

        mergesort(ar, m + 1, r);

        partition(ar, l, m, r);

    }

}

int main()

{

    int n;

    cout << "Input Size : ";

    cin >> n;

    int ar[n + 1];

    cout << "Input Elements : ";

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

    mergesort(ar, 0, n - 1);

    cout << "Array after Sort : ";

    for (int i = 0; i < n; i++) cout << ar[i] << " ";

}

**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;

struct Node

{

    int data;

    Node \*left, \*right;

    Node(int val) : data(val), left(NULL), right(NULL) {}

};

Node \*insert(Node \*root, int val)

{

    if (!root)

    {

        return new Node(val);

    }

    if (val < root->data)

    {

        root->left = insert(root->left, val);

    }

    else

    {

        root->right = insert(root->right, val);

    }

    return root;

}

void inorder(Node \*root)

{

    if (!root) return;

    inorder(root->left);

    cout << root->data << " ";

    inorder(root->right);

}

void preorder(Node \*root)

{

    if (!root) return;

    cout << root->data << " ";

    preorder(root->left);

    preorder(root->right);

}

void postorder(Node \*root)

{

    if (!root) return;

    postorder(root->left);

    postorder(root->right);

    cout << root->data << " ";

}

int main()

{

    Node \*root = NULL;

    int n, val;

    cout << "Input Size : ";

    cin >> n;

    cout << "Input Value : ";

    for (int i = 0; i < n; i++)

    {

        cin >> val;

        root = insert(root, val);

    }

    cout << "Inorder : ";

    inorder(root);

    cout << "\n";

    cout << "Preorder : ";

    preorder(root);

    cout << "\n";

    cout << "Postorder : ";

    postorder(root);

    cout << "\n";

}

**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;

struct Node

{

    int data;

    Node \*left, \*right;

    Node(int val) : data(val), left(NULL), right(NULL) {}

};

Node \*insert(Node \*root, int val)

{

    if (!root) return new Node(val);

    if (val < root->data) root->left = insert(root->left, val);

    else root->right = insert(root->right, val);

    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()

{

    Node \*root = NULL;

    int n, val;

    cout << "Input size : ";

    cin >> n;

    cout << "Input elements : ";

    for (int i = 0; i < n; i++)

    {

        cin >> val;

        root = insert(root, val);

    }

    int key;

    cout << "Input value to find : ";

    cin >> key;

    cout << (search(root, key) ? "Elememnt Found" : "Element Not Found");

}

**8. Write a program to create a Binary Search Tree of n elements and then delete an element from the tree**

#include <bits/stdc++.h>

using namespace std;

struct Node

{

    int data;

    Node \*left, \*right;

    Node(int val) : data(val), left(NULL), right(NULL) {}

};

Node \*insert(Node \*root, int val)

{

    if (!root) return new Node(val);

    if (val < root->data) root->left = insert(root->left, val);

    else root->right = insert(root->right, val);

    return root;

}

Node \*minValueNode(Node \*node)

{

    Node \*current = node;

    while (current && current->left)

    {

        current = current->left;

    }

    return current;

}

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) return root->right;

        else if (!root->right) return root->left;

        Node \*temp = minValueNode(root->right);

        root->data = temp->data;

        root->right = deleteNode(root->right, temp->data);

    }

    return root;

}

void inorder(Node \*root)

{

    if (!root) return;

    inorder(root->left);

    cout << root->data << " ";

    inorder(root->right);

}

int main()

{

    Node \*root = NULL;

    int n, val;

    cout << "Input size : ";

    cin >> n;

    cout << "Input elements : ";

    for (int i = 0; i < n; i++)

    {

        cin >> val;

        root = insert(root, val);

    }

    int del;

    cout << "Input value to delete : ";

    cin >> del;

    root = deleteNode(root, del);

    inorder(root);

}

**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 heap(int arr[], int n, int i)

{

    int largest = i, l = 2 \* i + 1, r = 2 \* i + 2;

    if (l < n && arr[l] > arr[largest]) largest = l;

    if (r < n && arr[r] > arr[largest]) largest = r;

    if (largest != i)

    {

        swap(arr[i], arr[largest]);

        heap(arr, n, largest);

    }

}

void maxheap(int arr[], int n)

{

    for (int i = n / 2 - 1; i >= 0; i--)

    {

        heap(arr, n, i);

    }

}

int main()

{

    int arr[100], n;

    cout << "Input size : ";

    cin >> n;

    cout << "Input elements : ";

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    maxheap(arr, n);

    cout << "Max Heap : ";

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

}

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

#include <iostream>

using namespace std;

void heap(int arr[], int n, int i)

{

    int largest = i, l = 2 \* i + 1, r = 2 \* i + 2;

    if (l < n && arr[l] > arr[largest])

        largest = l;

    if (r < n && arr[r] > arr[largest])

        largest = r;

    if (largest != i)

    {

        swap(arr[i], arr[largest]);

        heap(arr, n, largest);

    }

}

int deleteroot(int arr[], int &n)

{

    if (n <= 0)

        return -1;

    int root = arr[0];

    arr[0] = arr[n - 1];

    n--;

    heap(arr, n, 0);

    return root;

}

int main()

{

    int arr[100], n;

    cout << "Input size : ";

    cin >> n;

    cout << "Input elements : ";

    for (int i = 0; i < n; i++)

        cin >> arr[i];

    for (int i = n / 2 - 1; i >= 0; i--)

        heap(arr, n, i);

    deleteroot(arr, n);

    cout << "After Deletion : ";

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

}

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

#include <bits/stdc++.h>

using namespace std;

void heap(int arr[], int n, int i)

{

    int largest = i, l = 2 \* i + 1, r = 2 \* i + 2;

    if (l < n && arr[l] > arr[largest]) largest = l;

    if (r < n && arr[r] > arr[largest]) largest = r;

    if (largest != i)

    {

        swap(arr[i], arr[largest]);

        heap(arr, n, largest);

    }

}

void heapsort(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);

    }

}

int main()

{

    int n, arr[100];

    cout << "Input size : ";

    cin >> n;

    cout << "Input elements : ";

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    heapsort(arr, n);

    cout << "After Sort : ";

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

}

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

#include <bits/stdc++.H>

using namespace std;

int main()

{

    int a, b;

    cout << "Input number of nodes and edges : ";

    cin >> a >> b;

    int adj[100][100] = {0};

    cout << "Input " << b << " edges (a b) : ";

    for (int i = 0; i < b; i++)

    {

        int x, y;

        if(i>0) cout << "                      ";

        cin >> x >> y;

        adj[x][y] = 1;

        adj[y][x] = 1;

    }

    cout << "Adjacency Matrix : ";

    for (int i = 0; i < a; i++)

    {

        if(i>0) cout << "                   ";

        for (int j = 0; j < a; j++)

        {

            cout << adj[i][j] << " ";

        }

        cout << endl;

    }

}

**13. Write a program to display the path matrix of a graph from an adjacency matrix.**

#include <bits/stdc++.h>

using namespace std;

const int MAX = 100;

int adj[MAX][MAX];

int path[MAX][MAX];

int n;

void dfs(int src, int visited[])

{

    for (int v = 0; v < n; v++)

    {

        if (adj[src][v] && !visited[v])

        {

            visited[v] = 1;

            path[src][v] = 1;

            dfs(v, visited);

        }

    }

}

int main()

{

    cout << "Input number of nodes : ";

    cin >> n;

    cout << "Input the adjacency matrix : ";

    for (int i = 0; i < n; i++)

    {

        if(i>0) cout << "                             ";

        for (int j = 0; j < n; j++)

        {

            cin >> adj[i][j];

            path[i][j] = 0;

        }

    }

    for (int i = 0; i < n; i++)

    {

        int visited[MAX] = {0};

        visited[i] = 1;

        dfs(i, visited);

    }

    cout << "Path Matrix : ";

    for (int i = 0; i < n; i++)

    {

        if(i>0) cout << "              ";

        for (int j = 0; j < n; j++)

        {

            cout << path[i][j] << " ";

        }

        cout << endl;

    }

}

**14. Write a program to display the path matrix of a graph using Warshall’s**

**algorithm.**

#include <bits/stdc++.h>

using namespace std;

int main()

{

    int n;

    cout << "Input number of nodes : ";

    cin >> n;

    int adj[100][100];

    cout << "Input the adjacency matrix : ";

    for (int i = 0; i < n; i++)

    {

        if(i>0) cout << "                             ";

        for (int j = 0; j < n; j++)

        {

            cin >> adj[i][j];

        }

    }

    int path[100][100];

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            path[i][j] = adj[i][j];

        }

    }

    for (int k = 0; k < n; k++)

    {

        for (int i = 0; i < n; i++)

        {

            for (int j = 0; j < n; j++)

            {

                if (path[i][k] && path[k][j])

                {

                    path[i][j] = 1;

                }

            }

        }

    }

    cout << "Path Matrix : ";

    for (int i = 0; i < n; i++)

    {

        if(i>0) cout << "              ";

        for (int j = 0; j < n; j++)

        {

            cout << path[i][j] << " ";

        }

        cout << endl;

    }

}

**15. Write a program to display the adjacency list of a graph.**

#include <bits/stdc++.h>

using namespace std;

int main()

{

    int a, b;

    cout << "Input number of nodes and edges : ";

    cin >> a >> b;

    vector<int> adj[100];

    cout << "Input " << b << " edges (a b) : ";

    for (int i = 0; i < b; i++)

    {

        if(i>0) cout << "                      ";

        int u, v;

        cin >> u >> v;

        adj[u].push\_back(v);

        adj[v].push\_back(u);

    }

    cout << "Adjacency List = ";

    for (int i = 0; i < a; i++)

    {

        if(i>0) cout << "                 ";

        cout << i << " : ";

        for (int j : adj[i])

        {

            cout << j << " ";

        }

        cout << endl;

    }

}

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

#include <bits/stdc++.h>

using namespace std;

int main()

{

    int nodes, edges;

    cout << "Input number of nodes and edges : ";

    cin >> nodes >> edges;

    vector<int> adj[100];

    cout << "Input " << edges << " edges (u v) : ";

    for (int i = 0; i < edges; i++)

    {

        int u, v;

        if(i>0) cout << "                      ";

        cin >> u >> v;

        adj[u].push\_back(v);

        adj[v].push\_back(u);

    }

    bool visited[100] = {false};

    int start;

    cout << "Input the starting node for BFS : ";

    cin >> start;

    queue<int> q;

    q.push(start);

    visited[start] = true;

    cout << "BFS Traversal : ";

    while (!q.empty())

    {

        int node = q.front();

        q.pop();

        cout << node << " ";

        for (int neigh : adj[node])

        {

            if (!visited[neigh])

            {

                visited[neigh] = true;

                q.push(neigh);

            }

        }

    }

    cout << endl;

}

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

#include <bits/stdc++.h>

using namespace std;

void dfs(int node, vector<int> adj[], bool visited[])

{

    visited[node] = true;

    cout << node << " ";

    for (int neigh : adj[node])

    {

        if (!visited[neigh])

        {

            dfs(neigh, adj, visited);

        }

    }

}

int main()

{

    int a, b;

    cout << "Input number of nodes and edges : ";

    cin >> a >> b;

    vector<int> adj[100];

    cout << "Input " << b << " edges (a b) : ";

    for (int i = 0; i < b; i++)

    {

        if(i>0) cout << "                      ";

        int u, v;

        cin >> u >> v;

        adj[u].push\_back(v);

        adj[v].push\_back(u);

    }

    bool visited[100] = {false};

    int start;

    cout << "Input starting node : ";

    cin >> start;

    cout << "DFS Traversal : ";

    dfs(start, adj, visited);

    cout << endl;

}

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

#include <iostream>

using namespace std;

const int SIZE = 10;

int hashTable[SIZE];

void insert(int key)

{

    int idx = key % SIZE;

    int cnt = 0;

    while (hashTable[idx] != -1 && cnt < SIZE)

    {

        idx = (idx + 1) % SIZE;

        cnt++;

    }

    if (cnt == SIZE)

    {

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

        return;

    }

    hashTable[idx] = key;

}

int main()

{

    fill(hashTable, hashTable + SIZE, -1);

    int n, key;

    cout << "Input size : ";

    cin >> n;

    cout << "Input " << n << " keys : ";

    for (int i = 0; i < n; i++)

    {

        cin >> key;

        insert(key);

    }

    cout << "Final Hash Table : ";

    for (int i = 0; i < SIZE; i++)

    {

        if(i>0) cout << "                   ";

        cout << i << " : " << (hashTable[i] == -1 ? -1 : hashTable[i]) << endl;

    }

}

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

#include <bits/stdc++.h>

using namespace std;

const int SIZE = 10;

int hashTable[SIZE];

int h2(int key)

{

    return 7 - (key % 7);

}

void insert(int key)

{

    int i = 0;

    int idx = (key % SIZE + i \* h2(key)) % SIZE;

    while (hashTable[idx] != -1 && i < SIZE)

    {

        i++;

        idx = (key % SIZE + i \* h2(key)) % SIZE;

    }

    if (i == SIZE)

    {

        cout << "Hash table is full! Cannot insert key " << key << endl;

        return;

    }

    hashTable[idx] = key;

}

int main()

{

    fill(hashTable, hashTable + SIZE, -1);

    int n, key;

    cout << "Input number of elements : ";

    cin >> n;

    cout << "Input " << n << " keys : ";

    for (int i = 0; i < n; i++)

    {

        cin >> key;

        insert(key);

    }

    cout << "Hash Table of Double Hashing : ";

    for (int i = 0; i < SIZE; i++)

    {

        if(i>0) cout << "                               ";

        cout << i << " : " << (hashTable[i] == -1 ? -1 : hashTable[i]) << endl;

    }

}

**20. Write a program to implement a hash table using chaining method for collision resolution.**

#include <bits/stdc++.h>

using namespace std;

const int SIZE = 10;

vector<int> hashTable[SIZE];

void insert(int key)

{

    int idx = key % SIZE;

    hashTable[idx].push\_back(key);

}

int main()

{

    int n, key;

    cout << "Input number of elements : ";

    cin >> n;

    cout << "Input " << n << " keys : ";

    for (int i = 0; i < n; i++)

    {

        cin >> key;

        insert(key);

    }

    cout << "Hash Table of chaining : ";

    for (int i = 0; i < SIZE; i++)

    {

        if(i>0) cout << "                         ";

        cout << i << " : ";

        for (int val : hashTable[i])

        {

            cout << val << " ";

        }

        cout << endl;

    }

    return 0;

}