**LAB MANNUAL PCS-409**

**Week 1:**

1. Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case.

**Time Complexity: O(n)**

**LINEAR SEARCH ALGORITHIM**

#include<stdio.h>

#include<time.h> //for clock\_t, clock(), CLOCKS\_PER\_SEC

int main()

{

int t;

printf("Enter the number of test cases: \n");

scanf("%d", &t);

for(int i=0;i<t;i++)

{

int n, i, key, flag=0, count=0;

printf("Enter the size of the array: \n");

scanf("%d", &n);

int arr[n];

printf("Random Array Elements Genrated: \n");

for(i=0;i<n;i++)

{

arr[i]=(int)rand()%999999;

}

for(i=0;i<n;i++)

{

printf("[%d -> %d]\t",i+1,arr[i]);

}

printf("\nEnter the key element: \n");

scanf("%d", &key);

clock\_t start, end;

start = clock();

for(i=0; i<n; i++)

{

count++;

if(arr[i]==key)

{

flag=1;

break;

}

}

end = clock();

double time\_taken = (double)(end-start)/CLOCKS\_PER\_SEC;

if(flag==1)

{

printf("\nKey element found at position %d\t\n", i+1);

}

else

{

printf("Key element not found\n");

}

printf("Total number of comparisons: %d\t\n", count);

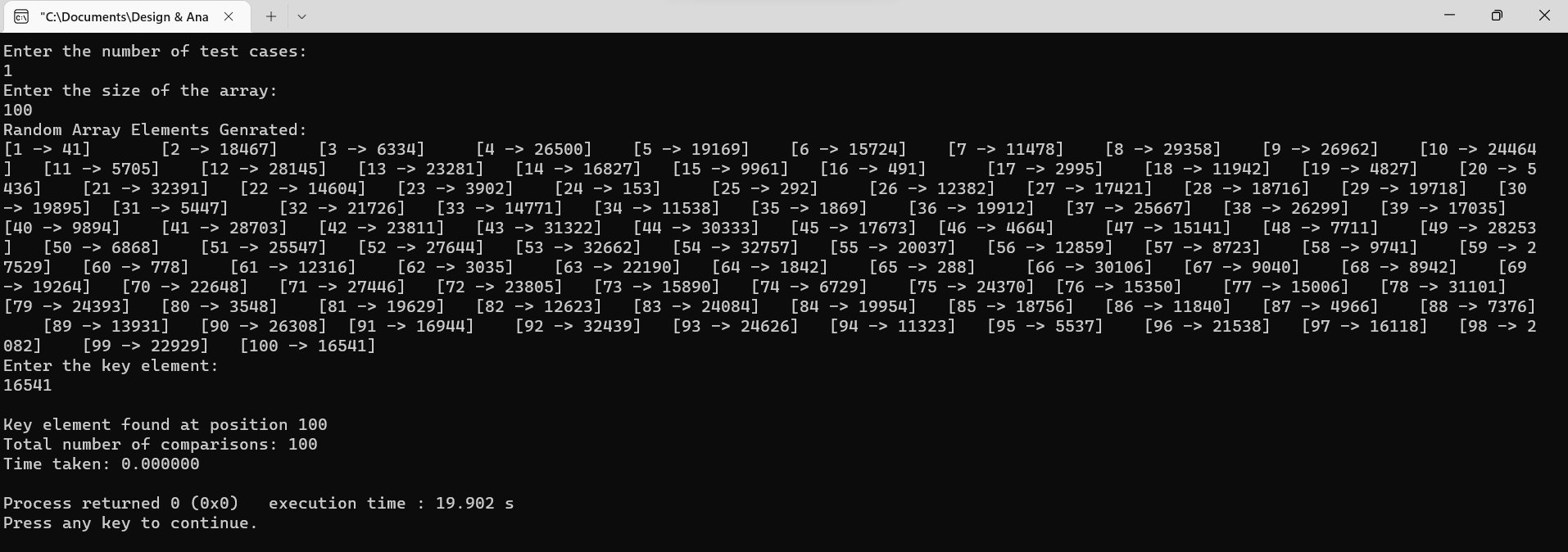
printf("Time taken: %f\t\n", time\_taken);

}

return 0;

}

Output:



Chart, line chart

Description automatically generated

1. Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case. (Time Complexity = O(logn), where n is the size of input).

**Time Complexity: O(log n)**

**BINARY SEARCH ALGORITHIM**

#include<stdio.h>

#include<time.h>

int main()

{

printf("\nARRAY MUST BE SORTED IN ASCENDING ORDER\n");

int t;

printf("\nEnter the number of test cases: \n");

scanf("%d", &t);

for(int i=0;i<t;i++)

{

int n, i, key, flag=0, count=0,l=1;

printf("Enter the size of the array: \n");

scanf("%d", &n);

int arr[n];

printf("Enter the elements of the array: \n");

for(i=0;i<n;i++)

{

arr[i]=l;

l+=5;

}

for(i=0;i<n;i++)

{

printf("[%d -> %d]\t",i+1,arr[i]);

}

printf("Enter the key element: \n");

scanf("%d", &key);

clock\_t start, end;

start = clock();

int low=0, high=n-1, mid;

while(low<=high)

{

mid = (low+high)/2;

count++;

if(arr[mid]==key)

{

flag=1;

break;

}

else if(arr[mid]<key)

{

low = mid+1;

}

else

{

high = mid-1;

}

}

end = clock();

double time\_taken = (double)(end-start)/CLOCKS\_PER\_SEC;

if(flag==1)

{

printf("Key element found at position %d\t\n", mid+1);

}

else

{

printf("Key element not found\n");

}

printf("Total number of comparisons: %d\t\n", count);

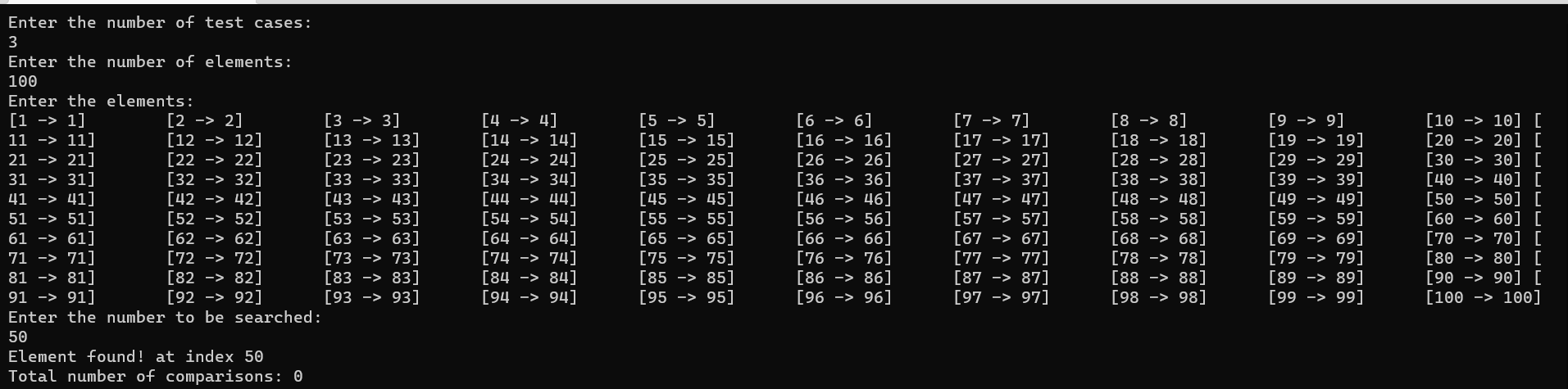
printf("Time taken: %f\t\n", time\_taken);

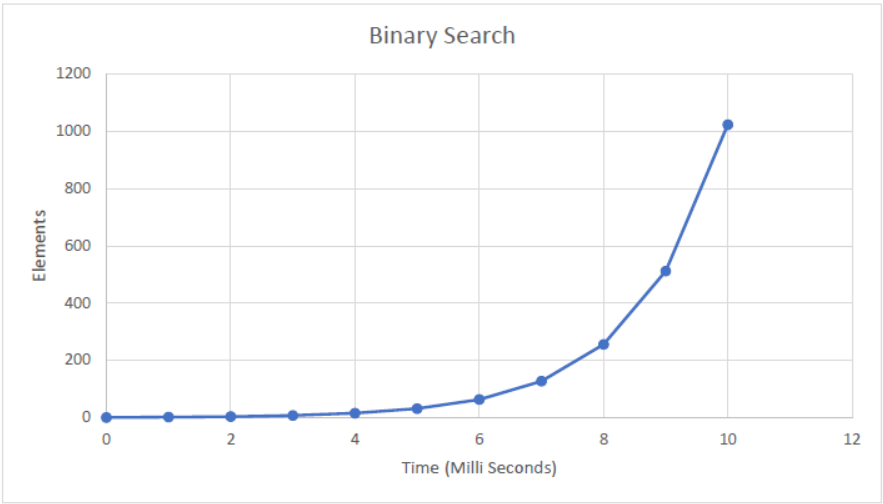
}

return 0;

}

Output :





1. Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array arr[n], search at the indexes arr[0], arr[2], arr[4],. ,arr[2k] and so on. Once the interval (arr[2k] < key < arr[ 2k+1] ) is found, perform a linear search operation from the index 2k to find the element key.

**Time Complexity: O (log n)**

**EXPONENTIAL SEARCH – JUMP SEARCH ALGORITHM**

#include<stdio.h>

#include<time.h>

#include<math.h>

int main()

{

printf("\nARRAY MUST BE SORTED IN ASCENDING ORDER\n");

int t;

printf("Enter the number of test cases: \n");

scanf("%d", &t);

for(int i=0;i<t;i++)

{

int n, i, key, flag=0, count=0;

printf("Enter the size of the array: \n");

scanf("%d", &n);

int arr[n];

printf("Enter the elements of the array: \n");

for(i=0;i<n;i++)

{

arr[i]=(int)rand()%999999;

}

printf("\nEnter the key element: \n");

scanf("%d", &key);

clock\_t start, end;

start = clock();

int low=0, high=n-1, mid;

if(arr[0]==key)

{

flag=1;

mid=0;

}

else

{

i=1;

while(i<n && arr[i]<=key)

{

i=i\*2;

}

low = i/2;

high = i;

while(low<=high)

{

mid = (low+high)/2;

count++;

if(arr[mid]==key)

{

flag=1;

break;

}

else if(arr[mid]<key)

{

low = mid+1;

}

else

{

high = mid-1;

}

}

}

end = clock();

double time\_taken = (double)(end-start)/CLOCKS\_PER\_SEC;

if(flag==1)

{

printf("Key element found at position %d\t\n", mid+1);

}

else

{

printf("Key element not found in the array \n");

}

printf("Total number of comparisons: %d\t\n", count);

printf("Time taken: %f\t\n", time\_taken);

}

return 0;

}

Output:

Background pattern

Description automatically generated

Chart, line chart

Description automatically generated

**Week 2:**

1. Given a sorted array of positive integers containing few duplicate elements, design an algorithm   
   and implement it using a program to find whether the given key element is present in the array or   
   not. If present, then also find the number of copies of given key.

**Time Complexity = O(log n)**

**ARRAY DUPLICACY**

#include<stdio.h>

int main()

{

printf("Enter Test Cases: ");

int t;

scanf("%d",&t);

for(int k=0;k<t;k++)

{

int i,n,key,count=0;

printf("Enter the number of elements in the array: ");

scanf("%d",&n);

int a[n];

printf("Enter the elements of the array:\n");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

printf("Enter the element whose frequency is to be found: ");

scanf("%d",&key);

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

{

if(key == a[i])

{

count++;

}

}

printf("%d -> %d",key,count);

}

return 0;

}

1. Given a sorted array of positive integers, design an algorithm and implement it using a program to   
   find three indices i, j, k such that arr[i] + arr[j] = arr[k].

**Time Complexity : O(n^3)**

**ARRAY ELEMENTS SUM**

#include<stdio.h>

int main()

{

printf("Enter Test Cases: ");

int t;

scanf("%d",&t);

for(int k=0;k<t;k++)

{

int i,n;

printf("\nEnter the number of elements in the array: ");

scanf("%d",&n);

int a[n];

printf("\nEnter the elements of the array:\n");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

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

{

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

{

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

{

if(a[i]+a[j]==a[k])

{

printf("The indices are %d, %d, %d",i,j,k);

return 0;

}

}

}

}

printf("No such indices exist");

}

return 0;

}

1. Given an array of non-negative integers, design an algorithm and a program to count the number   
   of pairs of integers such that their difference is equal to a given key, K.

**Time Complexity : O(n^2)**

**ARRAY ELEMENTS DIFFRENCE**

#include<stdio.h>

int main()

{

int t,n,k,count=0;

printf("Enter the number of test cases: ");

scanf("%d",&t);

for(int i=0;i<t;i++)

{

printf("Enter the number of elements in the array: ");

scanf("%d",&n);

printf("Enter the difference between the elements: ");

scanf("%d",&k);

int a[n];

printf("Enter the elements of the array: ");

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

{

scanf("%d",&a[j]);

}

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

{

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

{

if(a[j]-a[l]==k || a[l]-a[j]==k)

{

count++;

}

}

}

printf("%d"count);

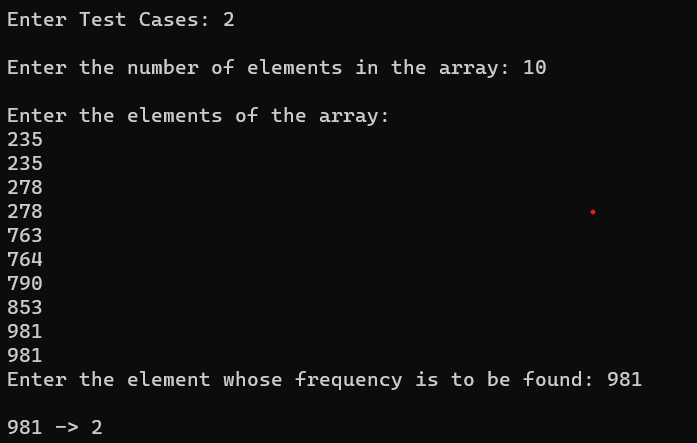
}

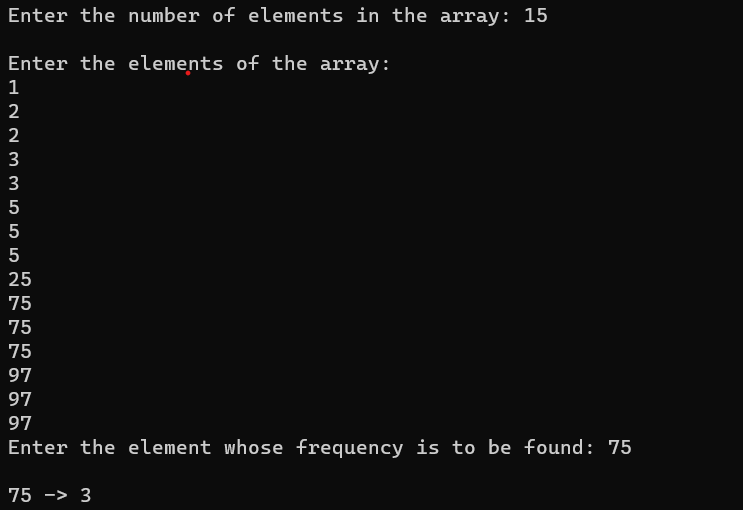
return 0;

}

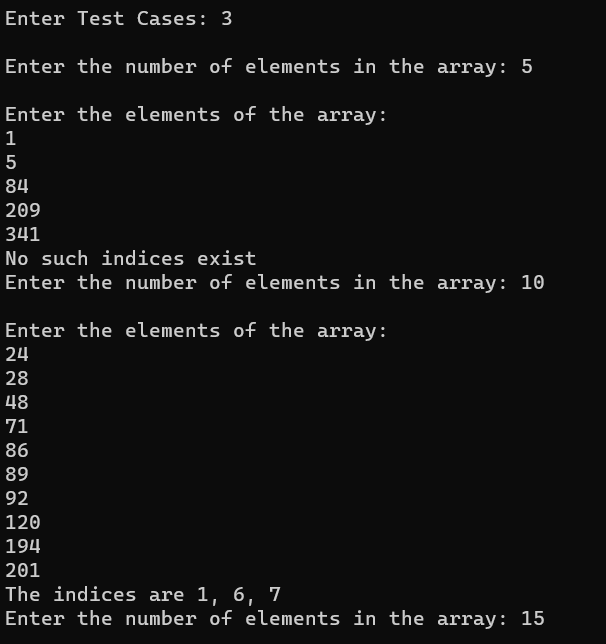
Output:

Array Duplicate :





Output Array Sum :

Text

Description automatically generated

Array Difference :

Text

Description automatically generated

**Week 3 :**

1. Given an unsorted array of integers, design an algorithm and a program to sort the array using   
   insertion sort. Your program should be able to find number of comparisons and shifts ( shifts -   
   total number of times the array elements are shifted from their place) required for sorting the array.

**Time Complexity : O(n^2)**

**INSERTION SORT ALGORITHM**

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

int main()

{

int t;

printf("Enter No. of testcases \n");

scanf("%d",&t);

for(int m=0;m<t;m++){

int n;

printf("Enter No. of elements \n");

scanf("%d",&n);

int a[n];

printf("Enter elements \n");

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

{

a[i]=(int)rand()%999;

}

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

{

printf("[%d -> %d]\t",i+1,a[i]);

}

int comp=0,shift=0;

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

int temp=a[i];

int j=i-1;

while(j>=0 && a[j]>temp){

a[j+1]=a[j];

j--;

comp++;

shift++;

}

a[j+1]=temp;

shift++;

}

printf("Sorted Array is \n");

for(int i=0;i<n;i++){

printf("[%d -> %d]\t",i+1,a[i]);

}

printf("\n");

printf("No. of comparisons = %d \n",comp);

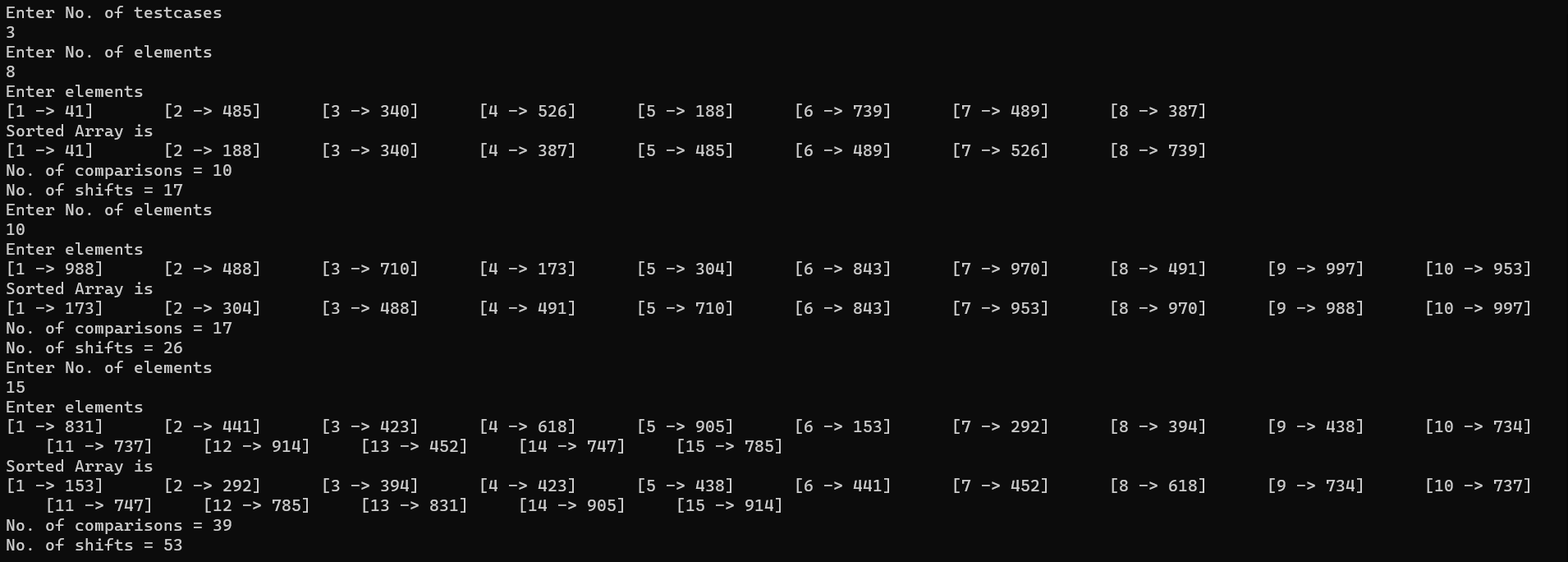
printf("No. of shifts = %d \n",shift);

}

return 0;

}

Output :



1. Given an unsorted array of integers, design an algorithm and implement a program to sort this   
   array using selection sort. Your program should also find number of comparisons and number of   
   swaps required.

**Time Complexity : O(N^2)**

**Selection Sort Algorithm**

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

int main()

{

int t;

printf("Enter No. of testcases \n");

scanf("%d",&t);

for(int m=0;m<t;m++){

int n;

printf("Enter No. of elements \n");

scanf("%d",&n);

int a[n],comp=0,swap=0;

printf("Enter elements \n");

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

{

a[i]=(int)rand()%999;

}

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

{

printf("[%d -> %d]\t",i+1,a[i]);

}

for(int i=0;i<n-1;i++){

int min=i;

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

if(a[j]<a[min]){

min=j;

}

comp++;

}

int temp=a[i];

a[i]=a[min];

a[min]=temp;

swap++;

}

printf("\nSorted Array is \n");

for(int i=0;i<n;i++){

printf("[%d -> %d]\t",i+1,a[i]);

}

printf("\n");

printf("No. of comparisons = %d \n",comp);

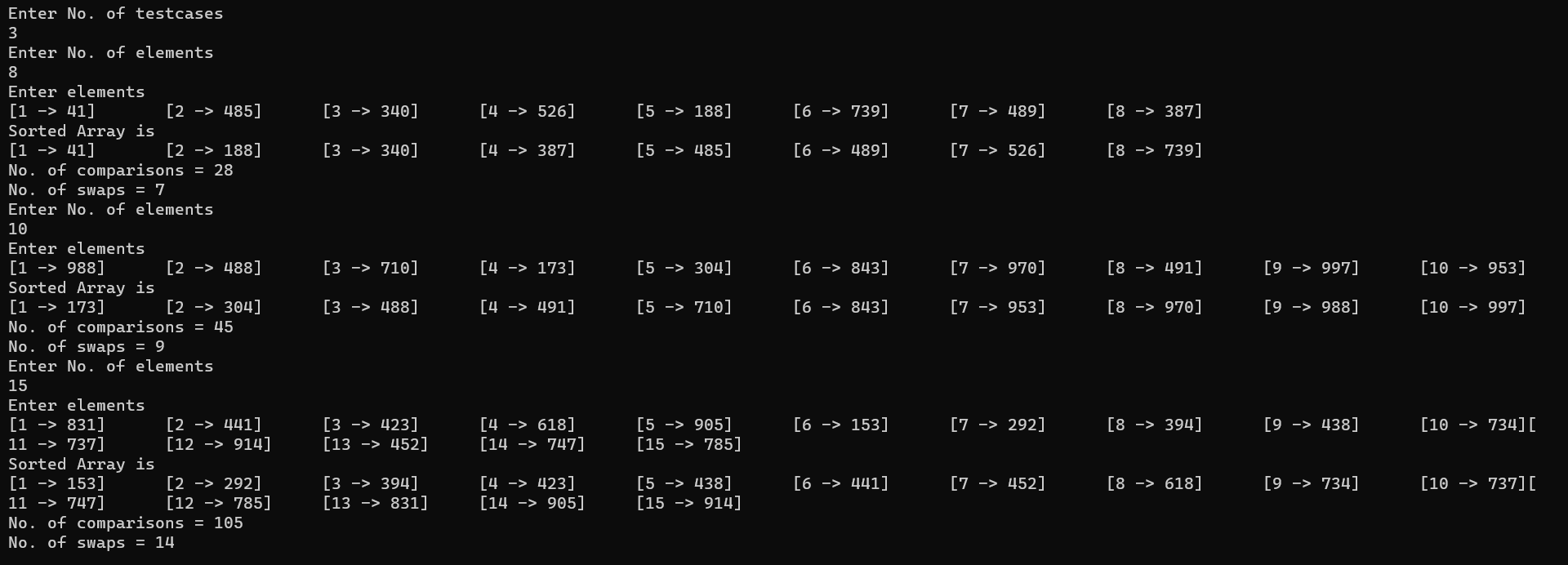
printf("No. of swaps = %d \n",swap);

}

return 0;

}

Output:



1. Given an unsorted array of positive integers, design an algorithm and implement it using a program   
   to find whether there are any duplicate elements in the array or not. (use sorting) (Time Complexity   
   = O(n log n)).

**No Initial Sorting**

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

int main()

{

int t;

printf("Enter No. of testcases \n");

scanf("%d",&t);

for(int m=0;m<t;m++){

int n;

printf("Enter No. of elements \n");

scanf("%d",&n);

int a[n];

printf("Enter elements \n");

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

{

a[i]=(int)rand()%99;

}

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

{

printf("[%d -> %d]\t",i+1,a[i]);

}

int comp=0,shift=0;

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

int temp=a[i];

int j=i-1;

while(j>=0 && a[j]>temp){

a[j+1]=a[j];

j--;

comp++;

shift++;

}

a[j+1]=temp;

shift++;

}

printf("\nSorted Array is \n");

for(int i=0;i<n;i++){

printf("[%d -> %d]\t",i+1,a[i]);

}

printf("\n");

printf("No. of comparisons = %d \n",comp);

printf("No. of shifts = %d \n",shift);

int flag=0;

for(int i=0;i<n-1;i++){

if(a[i]==a[i+1]){

flag=1;

break;

}

}

if(flag==1){

printf("Duplicate elements exist \n");

}

else{

printf("Duplicate elements do not exist \n");

}

}

return 0;

}

Output :

Text

Description automatically generated

**Week – 4 :**

1. Given an unsorted array of integers, design an algorithm and implement it using a program to sort   
   an array of elements by dividing the array into two subarrays and combining these subarrays after   
   sorting each one of them. Your program should also find number of comparisons and inversions   
   during sorting the array.

**Time Complexity : 2T(n/2) + θ(n)**

**Merge Sort Algorithm**

#include <bits/stdc++.h>

using namespace std;

int c = 0;

void mergeArray(int arr[], int l, int mid, int h) {

int n1 = mid - l + 1;

int n2 = h - mid;

int a[n1], b[n2];

for(int i = 0; i < n1; i++) {

a[i] = arr[l + i];

}

for(int i = 0; i < n2; i++) {

b[i] = arr[mid + 1 + i];

}

int i = 0, j = 0, k = l;

while(i < n1 && j < n2) {

c++;

if(a[i] <= b[j]) {

arr[k] = a[i];

i++;

}

else {

arr[k] = b[j];

j++;

}

k++;

}

while(i < n1) {

arr[k] = a[i];

i++; k++;

}

while(j < n2) {

arr[k] = b[j];

j++; k++;

}

}

void mergeSort(int arr[], int l, int h) {

if(l < h) {

int mid = l + (h - l) / 2;

mergeSort(arr, l, mid);

mergeSort(arr, mid + 1, h);

mergeArray(arr, l, mid, h);

}

}

void display(int arr[], int n) {

for(int i = 0; i < n; i++) {

cout<<arr[i]<<" ";

}

cout<<endl;

}

int main() {

int t;

cin>>t;

while(t--) {

int n;

cin>>n;

int \*arr = new int[n];

for(int i = 0; i < n; i++) {

cin>>arr[i];

}

mergeSort(arr, 0, n - 1);

display(arr, n);

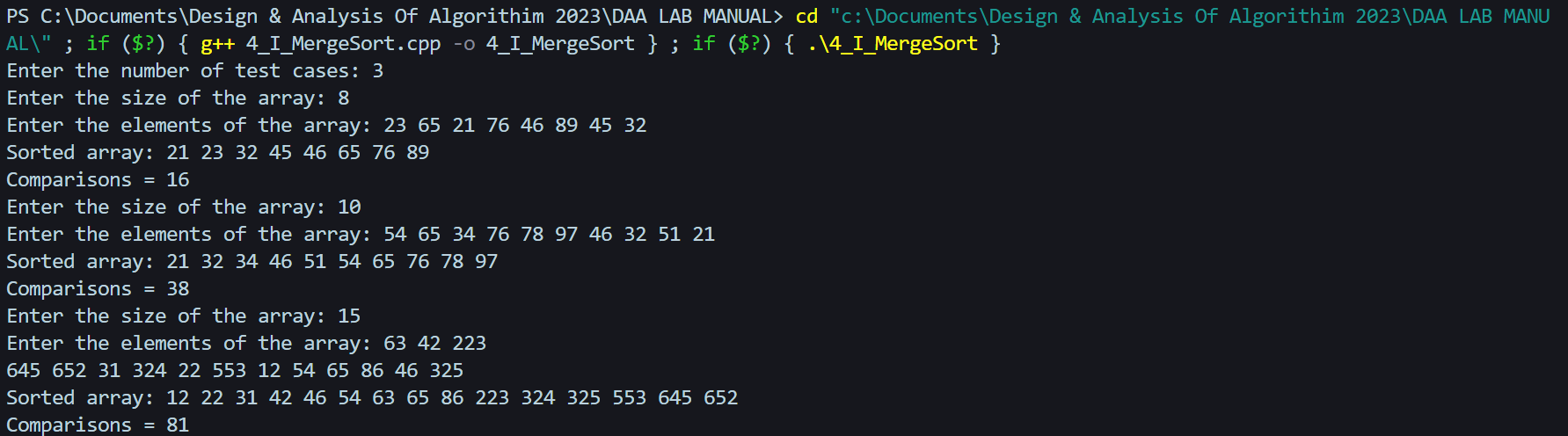
cout<<"comparisons = "<<c<<endl;

delete []arr;

}

}

Output :



1. Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by partitioning the array into two subarrays based on a pivot element such that one of the sub array holds values smaller than the pivot element while another sub array holds values greater than the pivot element. Pivot element should be selected randomly from the array. Your program should also find number of comparisons and swaps required for sorting the array.

**Time Complexity =** *T(N) = T(N-1) +*  ϴ *(N)*

**Quick Sort Algorithm**

#include <iostream>

using namespace std;

int c = 0, s = 0;

int partition(int arr[], int l, int h) {

int x = rand() % (h - l + 1) + l;

if(h != x) {

s++;

swap(arr[x], arr[h]);

}

int pivot = arr[h];

int i = l - 1;

for(int j = l; j <= h - 1; j++) {

if(arr[j] < pivot) {

i++;

s++;

swap(arr[i], arr[j]);

}

}

s++;

swap(arr[i + 1], arr[h]);

return i + 1;

}

void quickSort(int arr[], int l, int h) {

if(l < h) {

int pivot = partition(arr, l, h);

quickSort(arr, l, pivot - 1);

quickSort(arr, pivot + 1, h);

}

}

void display(int arr[], int n) {

for(int i = 0; i < n; i++) {

cout<<arr[i]<<" ";

}

cout<<endl;

}

int main() {

int t;

cout<<"Enter the number of test cases: ";

cin>>t;

while(t--) {

int n;

cout<<"Enter the size of the array: ";

cin>>n;

int \*arr = new int[n];

cout<<"Enter the elements of the array: ";

for(int i = 0; i < n; i++) {

cin>>arr[i];

}

quickSort(arr, 0, n - 1);

cout<<"Sorted array: ";

display(arr, n);

cout<<"Number of Swaps = "<<s<<endl;

}

}

**Output :**

Text

Description automatically generated

1. Given an unsorted array of integers, design an algorithm and implement it using a program to find Kth smallest or largest element in the array. (Worst case Time Complexity = O(n))

**Time Complexity = O(n)**

**Count Sort**

#include<bits/stdc++.h>

using namespace std;

int main() {

int t;

cout<<"Enter the number of test cases: ";

cin>>t;

while(t--) {

int n;

cout<<"Enter the size of the array: ";

cin>>n;

int \*arr = new int[n];

cout<<"Enter the elements of the array: ";

for(int i = 0; i < n; i++) {

cin>>arr[i];

}

int k;

cout<<"Enter the value of k: ";

cin>>k;

priority\_queue<int> pq;

for(int i = 0; i < k; i++) {

pq.push(arr[i]);

}

for(int i = k; i < n; i++) {

pq.push(arr[i]);

if(pq.size() > k) pq.pop();

}

if(pq.empty()) cout<<"Not present"<<endl;

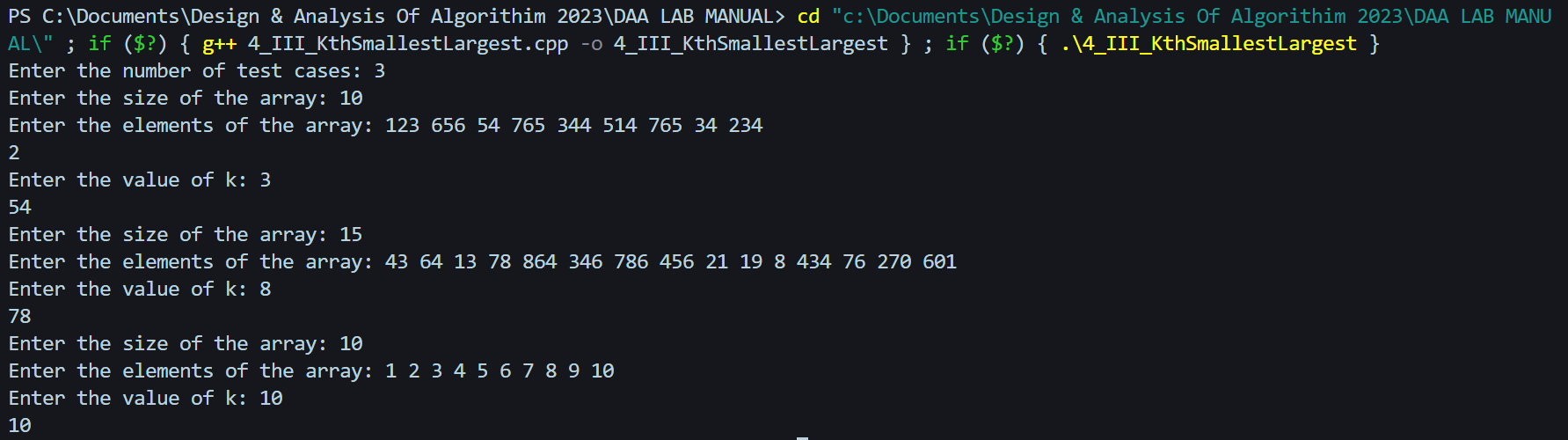
else cout<<pq.top()<<endl;

delete []arr;

}

}

**Output :**



**Week – 5 :**

1. Given an unsorted array of alphabets containing duplicate elements. Design an algorithm and implement it using a program to find which alphabet has maximum number of occurrences and

print it.

**Time Complexity = O(n)**

**Counting Sort Algorithm**

#include <iostream>

#include <vector>

using namespace std;

int main() {

int t;

cout<<"Enter the number of test cases: ";

cin>>t;

while(t--) {

int n;

cout<<"Enter the size of the array: ";

cin>>n;

vector<char> arr(n);

cout<<"Enter the elements of the array: ";

for(int i = 0; i < n; i++) {

cin>>arr[i];

}

vector<int> res(26, 0);

for(int i = 0; i < n; i++) {

res[arr[i] - 'a'] += 1;

}

int max = 1;

char c = '\0';

for(int i = 0; i < 26; i++) {

if(res[i] > max) {

max = res[i];

c = i + 'a';

}

}

if(max != 1) {

cout<<c<<" - "<<max<<endl;

}

else {

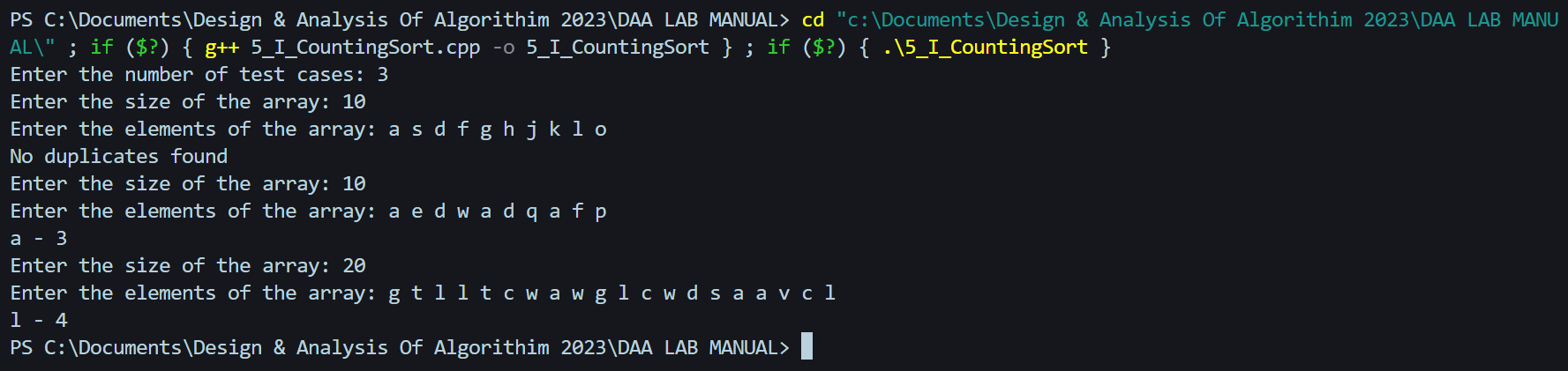
cout<<"No duplicates found"<<endl;

}

}

}

**Output :**



1. Given an unsorted array of integers, design an algorithm and implement it using a program to find whether two elements exist such that their sum is equal to the given key element.

**Time Complexity = O(n log n))**

#include <iostream>

#include<vector>

using namespace std;

int main() {

cout<<"Enter The Test Cases"<<endl;

int t,p=1;

cin>>t;

while(t--) {

cout<<"Test Case #"<<p++<<":"<<endl;

int n;

cout<<"Enter The Size Of The Array"<<endl;

cin>>n;

int arr[n];

cout<<"Enter The Elements Of The Array"<<endl;

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

cin>>arr[i];

int key;

cout<<"Enter The Key"<<endl;

cin>>key;

sort(arr, arr + n);

vector<int> sol;

int i = 0, j = n - 1;

while(i < j) {

if(arr[i] + arr[j] == key) {

sol.push\_back(arr[i]);

sol.push\_back(arr[j]);

break;

}

if(arr[i] + arr[j] > key) j--;

else i++;

}

if(sol.empty())

cout<<"No such element exist";

else {

for(auto it : sol)

cout<<it<<" ";

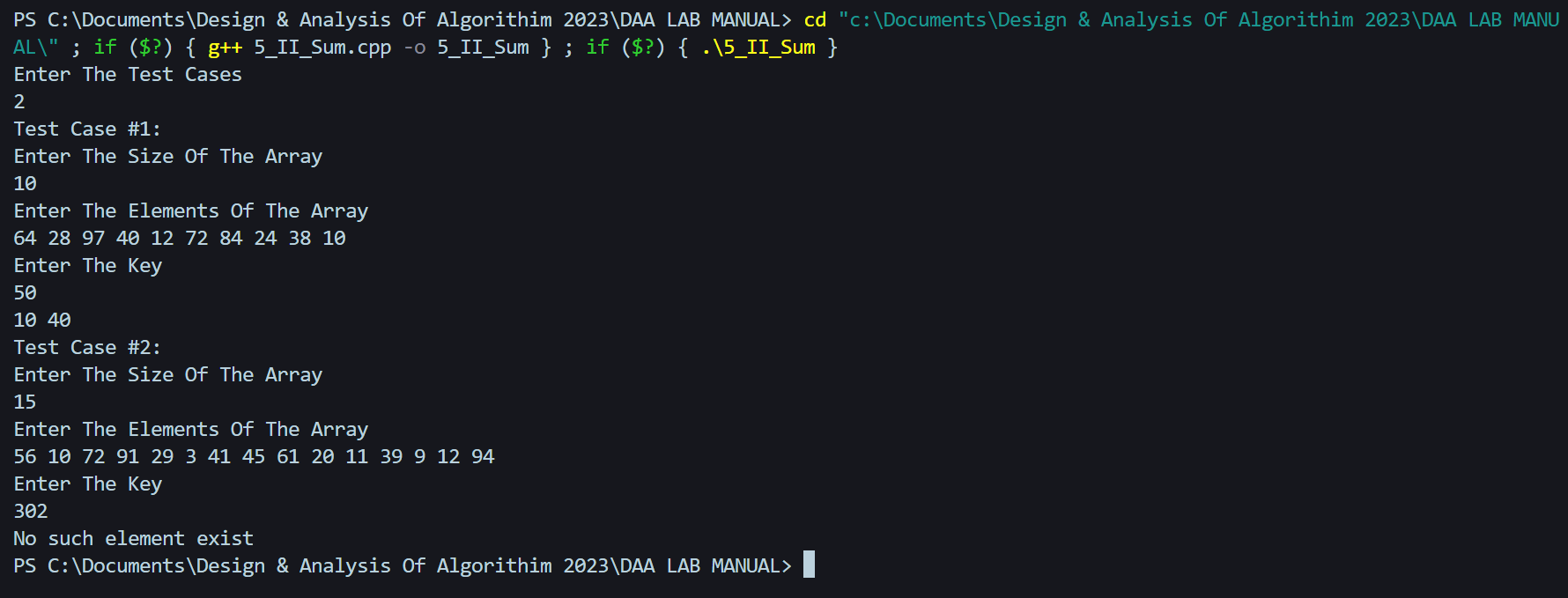
}

cout<<endl;

}

}

**Output :**



1. You have been given two sorted integer arrays of size m and n. Design an algorithm and implement it using a program to find list of elements which are common to both.

**Time Complexity = O(m+n)**

#include <iostream>

#include<vector>

using namespace std;

int main()

{

int m,n;

cout<<"Enter The Size Of The First Array"<<endl;

cin>>m;

int a1[m];

cout<<"Enter The Elements Of The First Array"<<endl;

for(int i = 0; i < m; i++)

cin>>a1[i];

cout<<"Enter The Size Of The Second Array"<<endl;

cin>>n;

int a2[n];

cout<<"Enter The Elements Of The Second Array"<<endl;

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

cin>>a2[i];

int i = 0, j = 0;

while(i < m && j < n) {

if(a1[i] == a2[j]) {

cout<<a1[i]<<" ";

i++;

j++;

}

else if(a1[i] < a2[j]) i++;

else j++;

}

for(int i = 0; i < m; i++){

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

if(a1[i] == a2[j]){

cout<<a1[i]<<" ";

}

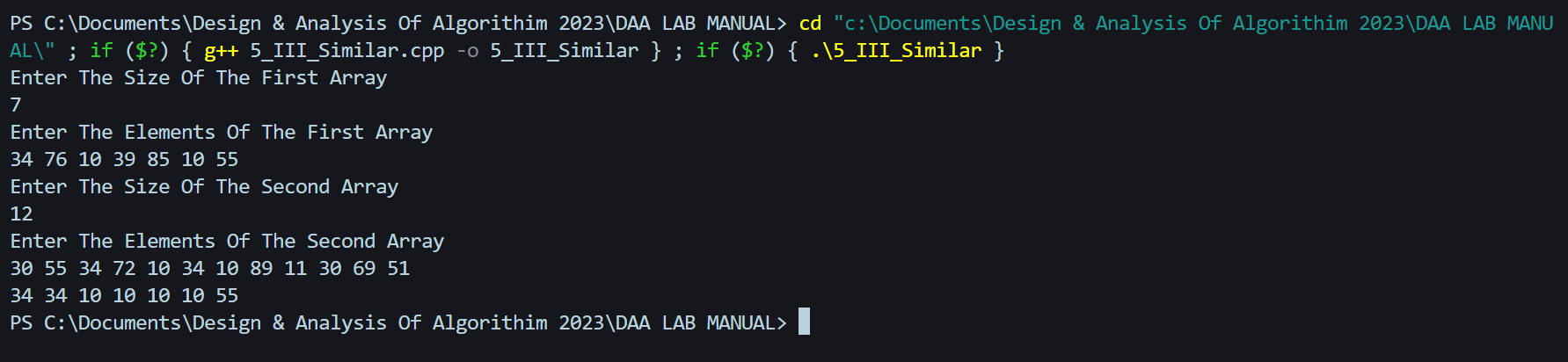
}

}

return 0;

}

**Output :**



**Week – 6:**

1. Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not.

**Depth First Search Algorithm**

#include <iostream>

using namespace std;

class Graph {

int numVertices;

int\*\* adjacencyMatrix;

public:

Graph(int vertices) {

numVertices = vertices;

adjacencyMatrix = new int\*[numVertices];

for (int i = 0; i < numVertices; i++) {

adjacencyMatrix[i] = new int[numVertices];

// Initialize all elements to 0

for (int j = 0; j < numVertices; j++) {

adjacencyMatrix[i][j] = 0;

}

}

}

void addEdge(int src, int dest)

{

adjacencyMatrix[src][dest] = 1;

}

bool DFS(int start, int end, bool\* visited)

{

if (start == end)

{

return true;

}

visited[start] = true;

for (int i = 0; i < numVertices; i++) {

if (adjacencyMatrix[start][i] == 1 && !visited[i]) {

if (DFS(i, end, visited)) {

return true; // Found a path

}

}

}

return false;

}

bool isPathExists(int start, int end) {

bool\* visited = new bool[numVertices];

for (int i = 0; i < numVertices; i++) {

visited[i] = false;

}

bool result = DFS(start, end, visited);

delete[] visited;

return result;

}

};

int main()

{

int numVertices, numEdges;

cout << "Enter the number of vertices: ";

cin >> numVertices;

cout << "Enter the number of edges: ";

cin >> numEdges;

Graph graph(numVertices);

cout << "Enter the edges (source destination):\n";

for (int i = 0; i < numEdges; i++) {

int src, dest;

cin >> src >> dest;

graph.addEdge(src, dest);

}

int start, end;

cout << "Enter the start vertex: ";

cin >> start;

cout << "Enter the end vertex: ";

cin >> end;

if (graph.isPathExists(start, end))

{

cout << "Path exists between " << start << " and " << end << endl;

} else {

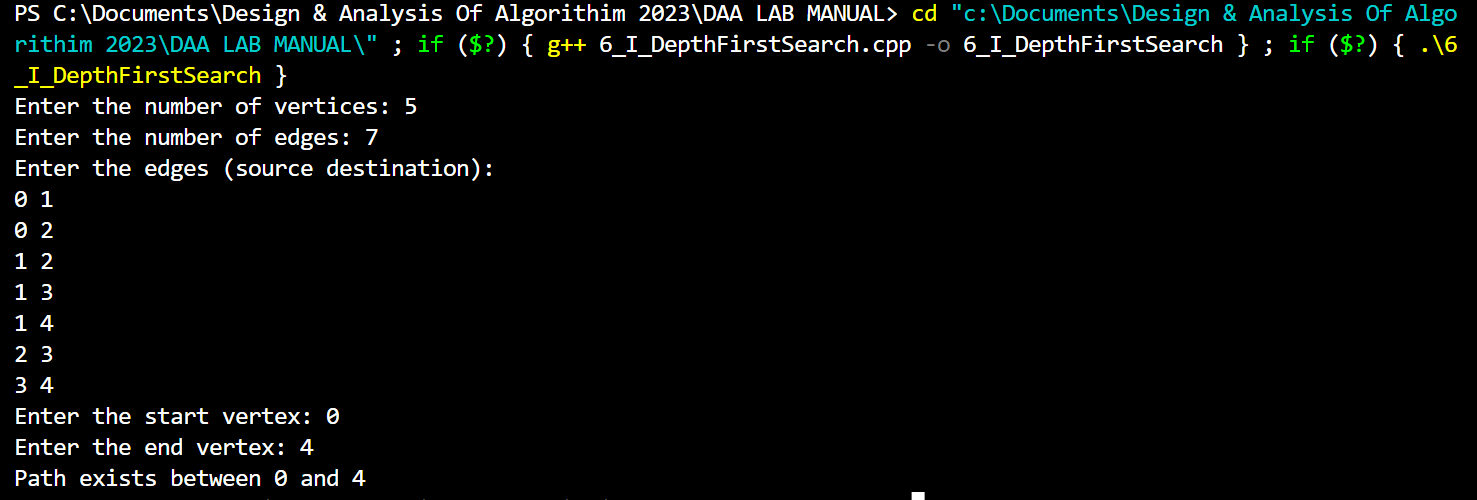
cout << "Path does not exist between " << start << " and " << end << endl;

}

return 0;

}

**Output :**

****

1. Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not.

**Breadth First Search Algorithm**

#include <iostream>

using namespace std;

class Graph

{

int numVertices;

int\*\* adjacencyMatrix;

public:

Graph(int vertices)

{

numVertices = vertices;

adjacencyMatrix = new int\*[numVertices];

for (int i = 0; i < numVertices; i++) {

adjacencyMatrix[i] = new int[numVertices];

for (int j = 0; j < numVertices; j++) {

adjacencyMatrix[i][j] = 0;

}

}

}

void addEdge(int src, int dest) {

adjacencyMatrix[src][dest] = 1;

adjacencyMatrix[dest][src] = 1;

}

bool isBipartite() {

int\* colors = new int[numVertices];

for (int i = 0; i < numVertices; i++) {

colors[i] = -1;

}

colors[0] = 0;

int queue[numVertices];

int front = 0;

int rear = 0;

queue[rear++] = 0;

while (front < rear) {

int currentVertex = queue[front++];

for (int i = 0; i < numVertices; i++) {

if (adjacencyMatrix[currentVertex][i] == 1 && colors[i] == -1) {

colors[i] = 1 - colors[currentVertex];

queue[rear++] = i;

} else if (adjacencyMatrix[currentVertex][i] == 1 && colors[i] == colors[currentVertex]) {

delete[] colors;

return false;

}

}

}

delete[] colors;

return true;

}

};

int main()

{

int numVertices, numEdges;

cout << "Enter the number of vertices: ";

cin >> numVertices;

cout << "Enter the number of edges: ";

cin >> numEdges;

Graph graph(numVertices);

cout << "Enter the edges (source destination):\n";

for (int i = 0; i < numEdges; i++) {

int src, dest;

cin >> src >> dest;

graph.addEdge(src, dest);

}

if (graph.isBipartite())

{

cout << "Graph is bipartite." << endl;

} else {

cout << "Graph is not bipartite." << endl;

}

return 0;

}

**Output :**

****

1. Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.

**Depth First Search Algorithm with Adjacency List Representation**

#include <iostream>

using namespace std;

class Graph

{

int numVertices;

int\*\* adjacencyMatrix;

public:

Graph(int vertices) {

numVertices = vertices;

adjacencyMatrix = new int\*[numVertices];

for (int i = 0; i < numVertices; i++) {

adjacencyMatrix[i] = new int[numVertices];

for (int j = 0; j < numVertices; j++) {

adjacencyMatrix[i][j] = 0;

}

}

}

void addEdge(int src, int dest)

{

adjacencyMatrix[src][dest] = 1;

}

bool DFS(int vertex, bool\* visited, bool\* recStack)

{

visited[vertex] = true;

recStack[vertex] = true;

for (int adjVertex = 0; adjVertex < numVertices; adjVertex++) {

if (adjacencyMatrix[vertex][adjVertex] == 1) {

if (!visited[adjVertex]) {

if (DFS(adjVertex, visited, recStack)) {

return true;

}

} else if (recStack[adjVertex]) {

return true;

}

}

}

recStack[vertex] = false;

return false;

}

bool isCycleExists()

{

bool\* visited = new bool[numVertices];

bool\* recStack = new bool[numVertices];

for (int i = 0; i < numVertices; i++) {

visited[i] = false;

recStack[i] = false;

}

for (int i = 0; i < numVertices; i++) {

if (!visited[i]) {

if (DFS(i, visited, recStack)) {

delete[] visited;

delete[] recStack;

return true;

}

}

}

delete[] visited;

delete[] recStack;

return false;

}

};

int main()

{

int numVertices, numEdges;

cout << "Enter the number of vertices: ";

cin >> numVertices;

cout << "Enter the number of edges: ";

cin >> numEdges;

Graph graph(numVertices);

cout << "Enter the edges (source destination):\n";

for (int i = 0; i < numEdges; i++) {

int src, dest;

cin >> src >> dest;

graph.addEdge(src, dest);

}

if (graph.isCycleExists())

{

cout << "Cycle exists in the graph." << endl;

} else {

cout << "No cycle exists in the graph." << endl;

}

return 0;

}

**Output :**

A picture containing text, screenshot, software, multimedia software

Description automatically generated

**Week – 7 :**

1. After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay’s house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends’ location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend’s location to his house and then whatsapp them   
   this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra’s algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay’s results?

**Dijkstra’s Algorithm**

#include <iostream>

#include <climits>

using namespace std;

int minDistance(int\* distances, bool\* visited, int numVertices)

{

int minDistance = INT\_MAX;

int minVertex = -1;

for (int v = 0; v < numVertices; v++) {

if (!visited[v] && distances[v] <= minDistance) {

minDistance = distances[v];

minVertex = v;

}

}

return minVertex;

}

void printPath(int\* parent, int destination)

{

if (parent[destination] == -1) {

cout << destination;

return;

}

printPath(parent, parent[destination]);

cout << " -> " << destination;

}

void dijkstra(int\*\* graph, int numVertices, int source)

{

int\* distances = new int[numVertices];

bool\* visited = new bool[numVertices];

int\* parent = new int[numVertices];

for (int i = 0; i < numVertices; i++) {

distances[i] = INT\_MAX;

visited[i] = false;

parent[i] = -1;

}

distances[source] = 0;

for (int count = 0; count < numVertices - 1; count++) {

int u = minDistance(distances, visited, numVertices);

visited[u] = true;

for (int v = 0; v < numVertices; v++) {

if (!visited[v] && graph[u][v] != 0 && distances[u] != INT\_MAX && distances[u] + graph[u][v] < distances[v]) {

distances[v] = distances[u] + graph[u][v];

parent[v] = u;

}

}

}

for (int i = 0; i < numVertices; i++) {

if (i != source) {

cout << "Shortest distance from Friend " << i << " to Akshay's house: " << distances[i] << endl;

cout << "Path: ";

printPath(parent, i);

cout << endl;

}

}

delete[] distances;

delete[] visited;

delete[] parent;

}

int main() {

int numVertices;

cout << "Enter the number of vertices (including Akshay's house): ";

cin >> numVertices;

int\*\* graph = new int\*[numVertices];

for (int i = 0; i < numVertices; i++) {

graph[i] = new int[numVertices];

}

cout << "Enter the adjacency matrix (0 for no edge, positive value for edge weight):\n";

for (int i = 0; i < numVertices; i++) {

for (int j = 0; j < numVertices; j++) {

cin >> graph[i][j];

}

}

int source = 0;

dijkstra(graph, numVertices, source);

for (int i = 0; i < numVertices; i++) {

delete[] graph[i];

}

delete[] graph;

return 0;

}

**Output :**

**A screen shot of a computer

Description automatically generated with medium confidence**

1. After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay’s house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends’ location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend’s location to his house and then whatsapp them   
   this path so that they can reach his house in minimum time.

**Bellman- Ford's Shortest Path Algorithm**

#include <iostream>

#include <climits>

using namespace std;

struct Edge

{

int src;

int dest;

int weight;

};

void printPath(int\* parent, int destination)

{

if (parent[destination] == -1) {

cout << destination;

return;

}

printPath(parent, parent[destination]);

cout << " -> " << destination;

}

void bellmanFord(Edge\* edges, int numEdges, int numVertices, int source)

{

int\* distances = new int[numVertices];

int\* parent = new int[numVertices];

for (int i = 0; i < numVertices; i++) {

distances[i] = INT\_MAX;

parent[i] = -1;

}

distances[source] = 0;

for (int i = 1; i <= numVertices - 1; i++) {

for (int j = 0; j < numEdges; j++) {

int u = edges[j].src;

int v = edges[j].dest;

int weight = edges[j].weight;

if (distances[u] != INT\_MAX && distances[u] + weight < distances[v]) {

distances[v] = distances[u] + weight;

parent[v] = u;

}

}

}

for (int i = 0; i < numEdges; i++) {

int u = edges[i].src;

int v = edges[i].dest;

int weight = edges[i].weight;

if (distances[u] != INT\_MAX && distances[u] + weight < distances[v])

{

cout << "Negative-weight cycle detected. No solution exists." << endl;

delete[] distances;

delete[] parent;

return;

}

}

for (int i = 0; i < numVertices; i++) {

if (i != source) {

cout << "Shortest distance from Friend " << i << " to Akshay's house: " << distances[i] << endl;

cout << "Path: ";

printPath(parent, i);

cout << endl;

}

}

delete[] distances;

delete[] parent;

}

int main() {

int numVertices, numEdges;

cout << "Enter the number of vertices (including Akshay's house): ";

cin >> numVertices;

cout << "Enter the number of edges: ";

cin >> numEdges;

Edge\* edges = new Edge[numEdges];

cout << "Enter the edges (source destination weight):\n";

for (int i = 0; i < numEdges; i++) {

cin >> edges[i].src >> edges[i].dest >> edges[i].weight;

}

int source = 0;

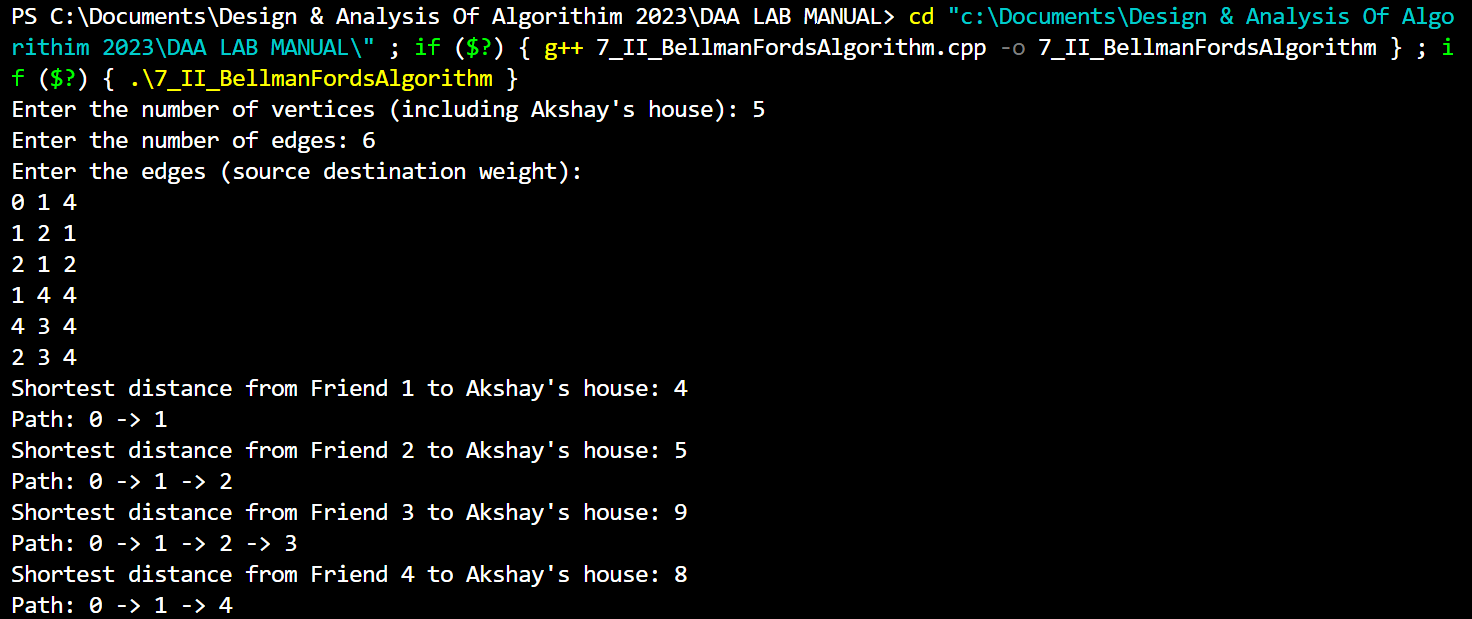
bellmanFord(edges, numEdges, numVertices, source);

delete[] edges;

return 0;

}

**Output :**

****

1. Given a directed graph with two vertices ( source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.

**Dynamic Programming 3D Matrix For Intermediate Result**

#include <iostream>

#include <climits>

using namespace std;

struct Edge

{

int src;

int dest;

int weight;

};

int shortestPathWithKEdges(Edge\* edges, int numEdges, int numVertices, int source, int destination, int k)

{

int\*\*\* dp = new int\*\*[numVertices];

for (int i = 0; i < numVertices; i++) {

dp[i] = new int\*[numVertices];

for (int j = 0; j < numVertices; j++) {

dp[i][j] = new int[k + 1];

}

}

for (int i = 0; i < numVertices; i++) {

for (int j = 0; j < numVertices; j++) {

for (int m = 0; m <= k; m++) {

dp[i][j][m] = INT\_MAX;

}

}

}

for (int i = 0; i < numVertices; i++) {

dp[i][i][0] = 0;

}

for (int m = 1; m <= k; m++) {

for (int i = 0; i < numVertices; i++) {

for (int j = 0; j < numVertices; j++) {

for (int l = 0; l < numEdges; l++) {

int u = edges[l].src;

int v = edges[l].dest;

int weight = edges[l].weight;

if (dp[u][i][m - 1] != INT\_MAX && dp[u][i][m - 1] + weight < dp[v][j][m]) {

dp[v][j][m] = dp[u][i][m - 1] + weight;

}

}

}

}

}

int shortestPathWeight = dp[source][destination][k];

for (int i = 0; i < numVertices; i++) {

for (int j = 0; j < numVertices; j++) {

delete[] dp[i][j];

}

delete[] dp[i];

}

delete[] dp;

return shortestPathWeight;

}

int main()

{

int numVertices, numEdges;

cout << "Enter the number of vertices: ";

cin >> numVertices;

cout << "Enter the number of edges: ";

cin >> numEdges;

Edge\* edges = new Edge[numEdges];

cout << "Enter the edges (source destination weight):\n";

for (int i = 0; i < numEdges; i++)

{

cin >> edges[i].src >> edges[i].dest >> edges[i].weight;

}

int source, destination, k;

cout << "Enter the source vertex: ";

cin >> source;

cout << "Enter the destination vertex: ";

cin >> destination;

cout << "Enter the number of edges on the path (k): ";

cin >> k;

int shortestPathWeight = shortestPathWithKEdges(edges, numEdges, numVertices, source, destination, k);

if (shortestPathWeight != INT\_MAX)

{

cout << "The weight of the shortest path from source to destination with exactly " << k << " edges: " << shortestPathWeight << endl;

} else {

cout << "No path exists from source to destination with exactly " << k << " edges." << endl;

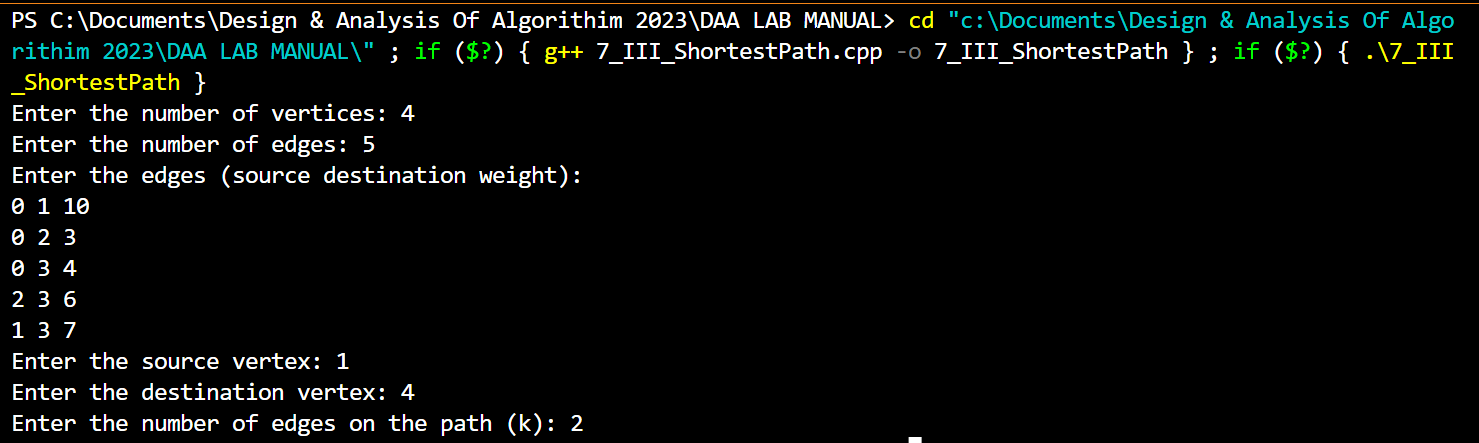
}

delete[] edges;

return 0;

}

**Output :**

****

**Week – 8:**

1. Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm   
   which will find minimum cost required to connect these cities.

**Prim's Algorithm**

#include <iostream>

#include <climits>

#define MAX\_CITIES 100

using namespace std;

int findMinKey(int key[], bool mstSet[], int V) {

int minKey = INT\_MAX, minIndex;

for (int v = 0; v < V; v++) {

if (!mstSet[v] && key[v] < minKey) {

minKey = key[v];

minIndex = v;

}

}

return minIndex;

}

void printMST(int parent[], int graph[MAX\_CITIES][MAX\_CITIES], int V) {

cout << "City\t-\tCity\tCost" << endl;

for (int i = 1; i < V; i++) {

cout << parent[i] << "\t-\t" << i << "\t" << graph[i][parent[i]] << endl;

}

}

void primMST(int graph[MAX\_CITIES][MAX\_CITIES], int V)

{

int parent[MAX\_CITIES];

int key[MAX\_CITIES];

bool mstSet[MAX\_CITIES];

for (int i = 0; i < V; i++)

{

key[i] = INT\_MAX;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++)

{

int u = findMinKey(key, mstSet, V);

mstSet[u] = true;

for (int v = 0; v < V; v++)

{

if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

}

printMST(parent, graph, V);

}

int main()

{

int V;

int graph[MAX\_CITIES][MAX\_CITIES];

cout << "Enter the number of cities: ";

cin >> V;

cout << "Enter the adjacency matrix (0 for no connection):" << endl;

for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

cin >> graph[i][j];

}

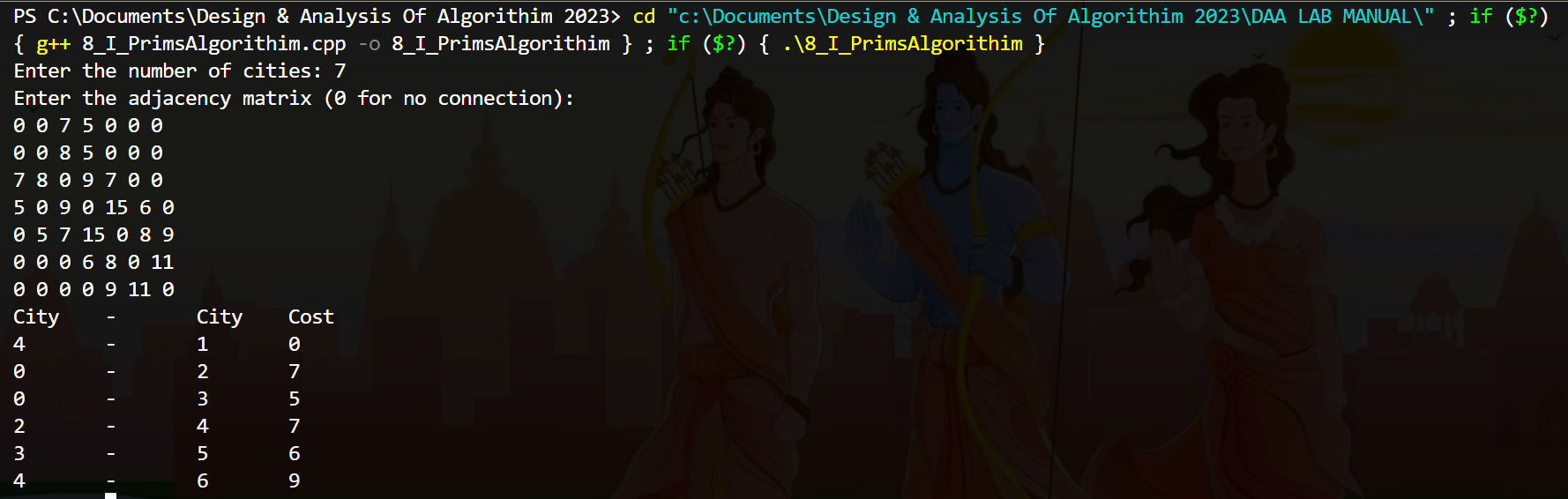
}

primMST(graph, V);

return 0;

}

**Output :**

****

1. Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm   
   which will find minimum cost required to connect these cities.

**Kruskal’s Algorithm**

#include <iostream>

#include <algorithm>

#define MAX\_CITIES 100

using namespace std;

struct Edge

{

int src, dest, cost;

};

bool compareEdges(const Edge& a, const Edge& b)

{

return a.cost < b.cost;

}

class Graph

{

int V; // Number of cities

Edge edges[MAX\_CITIES]; // Array to store all the edges of the graph

public:

Graph(int V) : V(V) {}

void addEdge(int src, int dest, int cost)

{

Edge edge = {src, dest, cost};

edges[src] = edge;

}

int findParent(int parent[], int i)

{

if (parent[i] == i)

return i;

return findParent(parent, parent[i]);

}

void unionSets(int parent[], int x, int y)

{

int xRoot = findParent(parent, x);

int yRoot = findParent(parent, y);

parent[yRoot] = xRoot;

}

void kruskalMST() {

Edge mst[MAX\_CITIES];

int parent[MAX\_CITIES];

int mstCount = 0;

for (int i = 0; i < V; i++)

parent[i] = i;

sort(edges, edges + V, compareEdges);

for (int i = 0; i < V; i++) {

int src = edges[i].src;

int dest = edges[i].dest;

int srcParent = findParent(parent, src);

int destParent = findParent(parent, dest);

if (srcParent != destParent) {

mst[mstCount++] = edges[i];

unionSets(parent, srcParent, destParent);

}

}

cout << "City\t-\tCity\tCost" << endl;

for (int i = 0; i < mstCount; i++) {

cout << mst[i].src << "\t-\t" << mst[i].dest << "\t" << mst[i].cost << endl;

}

}

};

int main()

{

int V;

int graph[MAX\_CITIES][MAX\_CITIES];

cout << "Enter the number of cities: ";

cin >> V;

cout << "Enter the adjacency matrix (0 for no connection):" << endl;

for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

cin >> graph[i][j];

}

}

Graph g(V);

for (int i = 0; i < V; i++) {

for (int j = i + 1; j < V; j++) {

if (graph[i][j] != 0) {

g.addEdge(i, j, graph[i][j]);

}

}

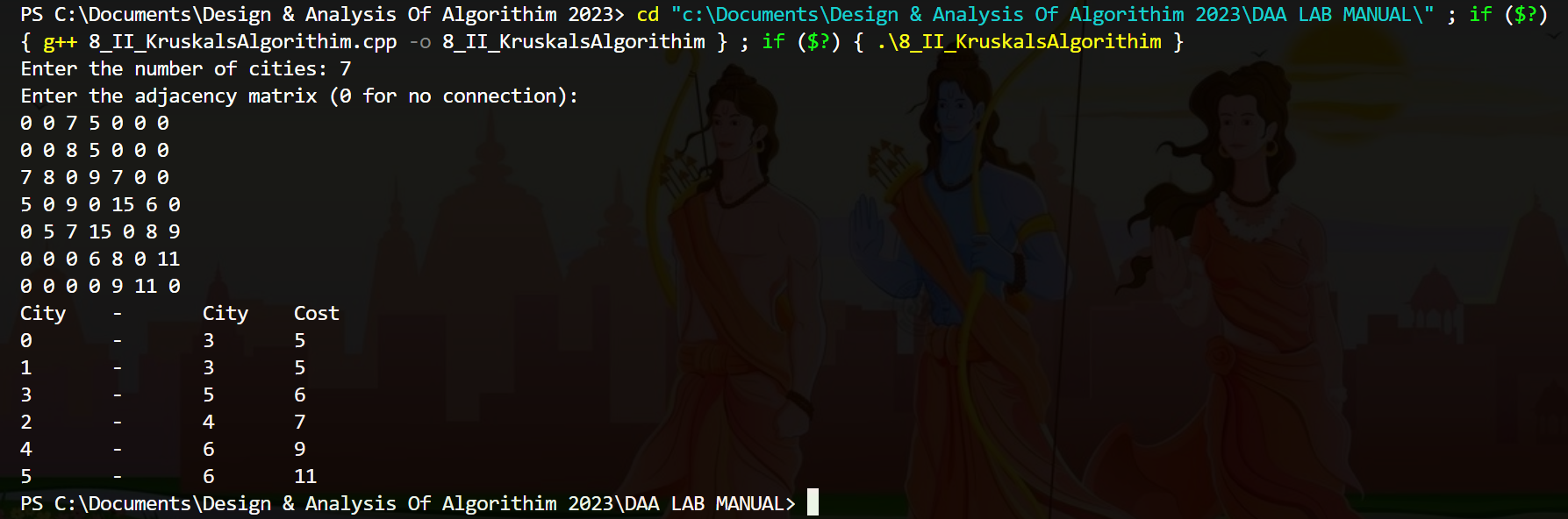
}

g.kruskalMST();

return 0;

}

**Output :**

****

1. Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.

**Maximum Greedy Approach**

#include <iostream>

#include <algorithm>

#define MAX\_CITIES 100

using namespace std;

struct Edge {

int src, dest, cost;

};

bool compareEdges(const Edge& a, const Edge& b) {

return a.cost > b.cost;

}

class Graph

{

int V;

Edge edges[MAX\_CITIES];

public:

Graph(int V) : V(V) {}

void addEdge(int src, int dest, int cost) {

Edge edge = {src, dest, cost};

edges[src] = edge;

}

int findParent(int parent[], int i) {

if (parent[i] == i)

return i;

return findParent(parent, parent[i]);

}

void unionSets(int parent[], int x, int y) {

int xRoot = findParent(parent, x);

int yRoot = findParent(parent, y);

parent[yRoot] = xRoot;

}

int findMaxBudget() {

int parent[MAX\_CITIES];

int maxBudget = 0;

for (int i = 0; i < V; i++)

parent[i] = i;

sort(edges, edges + V, compareEdges);

for (int i = 0; i < V; i++) {

int src = edges[i].src;

int dest = edges[i].dest;

int srcParent = findParent(parent, src);

int destParent = findParent(parent, dest);

if (srcParent != destParent) {

maxBudget += edges[i].cost;

unionSets(parent, srcParent, destParent);

}

}

return maxBudget;

}

};

int main() {

int V;

int graph[MAX\_CITIES][MAX\_CITIES];

cout << "Enter the number of cities: ";

cin >> V;

cout << "Enter the adjacency matrix (0 for no connection):" << endl;

for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

cin >> graph[i][j];

}

}

Graph g(V);

for (int i = 0; i < V; i++) {

for (int j = i + 1; j < V; j++) {

if (graph[i][j] != 0) {

g.addEdge(i, j, graph[i][j]);

}

}

}

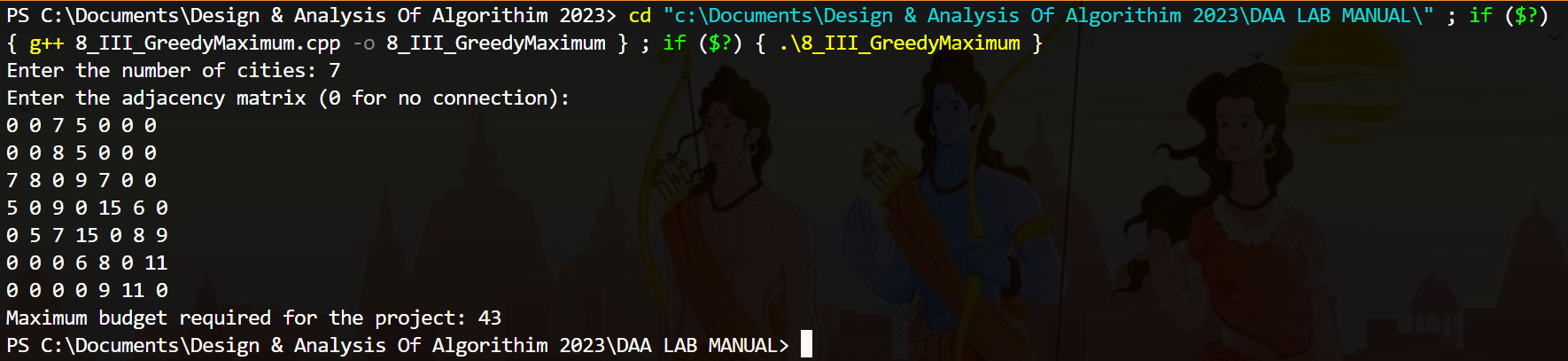
int maxBudget = g.findMaxBudget();

cout << "Maximum budget required for the project: " << maxBudget << endl;

return 0;

}

**Output :**



**Week – 9**

1. Given a graph, Design an algorithm and implement it using a program to implement **Floyd- Warshall all pair Shortest Path Algorithm**.

#include <bits/stdc++.h>

using namespace std;

#define V 5

#define INF 99999

void printSolution(int dist[][V])

{

cout<<"The following matrix shows the shortest distances between every pair of vertices \n";

for(int i=0;i<V;i++)

{

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

{

if(dist[i][j]==INF)

cout<<"INF"<<" ";

else

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

}

cout<<endl;

}

}

void floydWarshall(int graph[][V])

{

int dist[V][V],i,j,k;

for(i=0;i<V;i++)

for(j=0;j<V;j++)

dist[i][j]=graph[i][j];

for(k=0;k<V;k++)

{

for(i=0;i<V;i++)

{

for(j=0;j<V;j++)

{

if(dist[i][k]+dist[k][j]<dist[i][j])

dist[i][j]=dist[i][k]+dist[k][j];

}

}

}

printSolution(dist);

}

int main()

{

int graph[V][V]={{0,10,5,5,INF},

{INF,0,5,5,5},

{INF,INF,0,INF,10},

{INF,INF,INF,0,20},

{INF,INF,INF,5,0}};

floydWarshall(graph);

return 0;

}

**Output:**

**A picture containing text, screenshot, font, software

Description automatically generated**

1. Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value. You can take fractions of items,i.e. the items can be broken into smaller pieces so that you have to carry only a fraction xi of item i, where 0 ≤xi≤ 1.

**Fractional Knapsack**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int n,i,j;

float w;

cout<<"Enter the number of items: ";

cin>>n;

cout<<"Enter the maximum capacity of knapsack: ";

cin>>w;

float weight[n],value[n],ratio[n],temp;

cout<<"Enter the weights and values of the items:"<<endl;

for(i=0;i<n;i++)

{

cin>>weight[i]>>value[i];

ratio[i]=value[i]/weight[i];

}

for(i=0;i<n;i++)

{

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

{

if(ratio[i]<ratio[j])

{

temp=ratio[j];

ratio[j]=ratio[i];

ratio[i]=temp;

temp=weight[j];

weight[j]=weight[i];

weight[i]=temp;

temp=value[j];

value[j]=value[i];

value[i]=temp;

}

}

}

float x[n],profit=0;

for(i=0;i<n;i++)

x[i]=0;

for(i=0;i<n;i++)

{

if(weight[i]>w)

break;

else

{

x[i]=1;

profit+=value[i];

w-=weight[i];

}

}

if(i<n)

x[i]=w/weight[i];

profit+=x[i]\*value[i];

cout<<"The selected items are:"<<endl;

for(i=0;i<n;i++)

cout<<x[i]<<" ";

cout<<endl;

cout<<"The maximum profit is: "<<profit<<endl;

return 0;

}

**Output :**

**A screen shot of a computer

Description automatically generated with medium confidence**

1. Given an array of elements. Assume arr[i] represents the size of file i. Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files A and B with sizes m and n, computation cost of merging them is O(m+n).

**Greedy Approach**

#include <iostream>

using namespace std;

int mergeFiles(int arr[], int size) {

if (size < 2) {

return arr[0];

}

int totalCost = 0;

while (size > 1) {

int minIndex1 = 0;

int minIndex2 = 1;

if (arr[minIndex1] > arr[minIndex2]) {

swap(minIndex1, minIndex2);

}

for (int i = 2; i < size; i++) {

if (arr[i] < arr[minIndex1]) {

minIndex2 = minIndex1;

minIndex1 = i;

} else if (arr[i] < arr[minIndex2]) {

minIndex2 = i;

}

}

int mergedSize = arr[minIndex1] + arr[minIndex2];

totalCost += mergedSize;

arr[minIndex1] = mergedSize;

for (int i = minIndex2; i < size - 1; i++) {

arr[i] = arr[i + 1];

}

size--;

}

return totalCost;

}

int main() {

const int MAX\_FILES = 100; // Maximum number of files

int arr[MAX\_FILES];

int size;

cout << "Enter the number of files: ";

cin >> size;

if (size <= 0 || size > MAX\_FILES) {

cout << "Invalid number of files!" << std::endl;

return 0;

}cout << "Enter the sizes of files: ";

for (int i = 0; i < size; i++) {

cin >> arr[i];

}

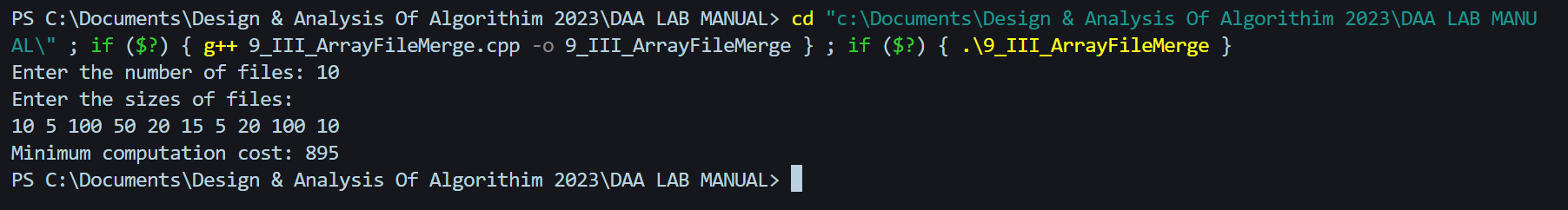
int minComputation = mergeFiles(arr, size);

cout << "Minimum computation cost: " << minComputation << std::endl;

return 0;

}

**Output:**

****

**Week – 10**

1. Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time.

**Maximum number of Activities**

#include <iostream>

const int MAX\_ACTIVITIES = 100;

using namespace std;

int selectActivities(int start[], int finish[], int size) {

if (size == 0) {

return 0;

}

int selectedCount = 1;

int lastFinish = finish[0];

for (int i = 1; i < size; i++) {

if (start[i] >= lastFinish) {

selectedCount++;

lastFinish = finish[i];

}

}

return selectedCount;

}

int main() {

int start[MAX\_ACTIVITIES];

int finish[MAX\_ACTIVITIES];

int size;

cout << "Enter the number of activities: ";

cin >> size;

if (size <= 0 || size > MAX\_ACTIVITIES) {

std::cout << "Invalid number of activities!" << std::endl;

return 0;

}

cout << "Enter the start and finish times of activities: ";

for (int i = 0; i < size; i++) {

cin >> start[i] >> finish[i];

}

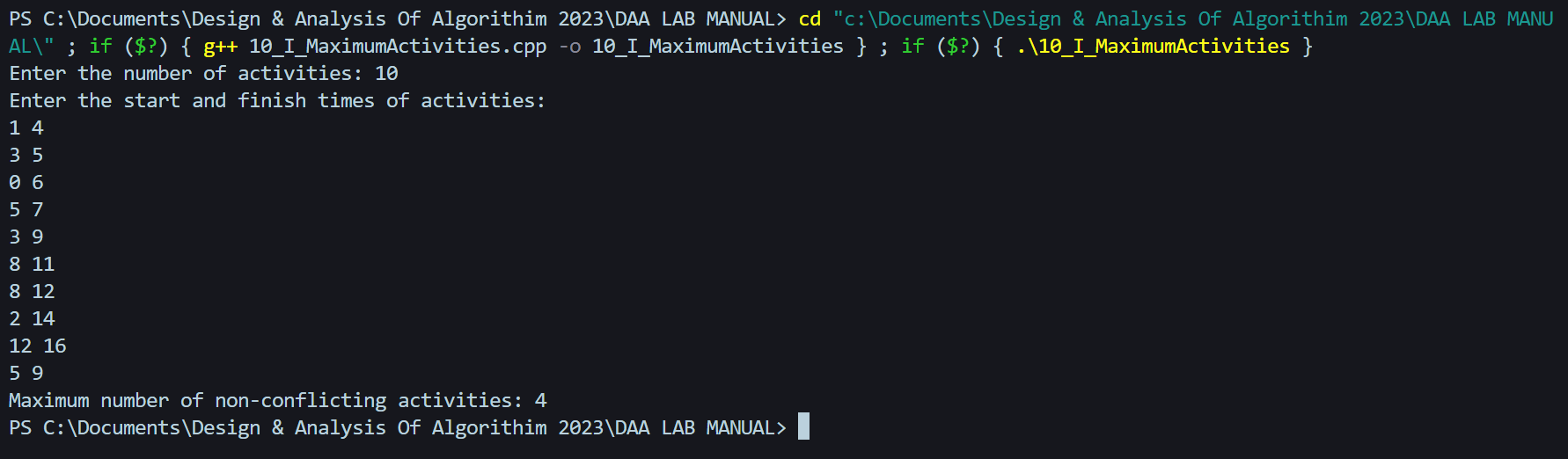
int maxActivities = selectActivities(start, finish, size);

cout << "Maximum number of non-conflicting activities: " << maxActivities << std::endl;

return 0;

}

**Output :**



1. Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks.

**Maximum number of Tasks**

#include <iostream>

#include <algorithm>

const int MAX\_TASKS = 100;

using namespace std;

bool compareTasks(int time[], int deadline[], int a, int b) {

return deadline[a] < deadline[b];

}

int findMaxTasks(int time[], int deadline[], int size, int selected[]) {

int maxTasks = 0;

bool slots[MAX\_TASKS] = {false};

int taskIndices[MAX\_TASKS];

for (int i = 0; i < size; i++) {

taskIndices[i] = i;

}

sort(taskIndices, taskIndices + size, [&](int a, int b) {

return compareTasks(time, deadline, a, b);

});

for (int i = 0; i < size; i++) {

for (int j = std::min(size, deadline[taskIndices[i]]) - 1; j >= 0; j--) {

if (!slots[j]) {

selected[j] = taskIndices[i];

slots[j] = true;

maxTasks++;

break;

}

}

}

return maxTasks;

}

int main() {

int time[MAX\_TASKS];

int deadline[MAX\_TASKS];

int selected[MAX\_TASKS];

int size;

cout << "Enter the number of tasks: ";

cin >> size;

if (size <= 0 || size > MAX\_TASKS) {

cout << "Invalid number of tasks!" << std::endl;

return 0;

}

cout << "Enter the time and deadline for each task: ";

for (int i = 0; i < size; i++) {

std::cin >> time[i] >> deadline[i];

}

int maxTasks = findMaxTasks(time, deadline, size, selected);

cout << "Maximum number of tasks that can be completed: " << maxTasks <<endl;

cout << "Selected tasks: ";

for (int i = 0; i < maxTasks; i++) {

cout << selected[i] << " ";

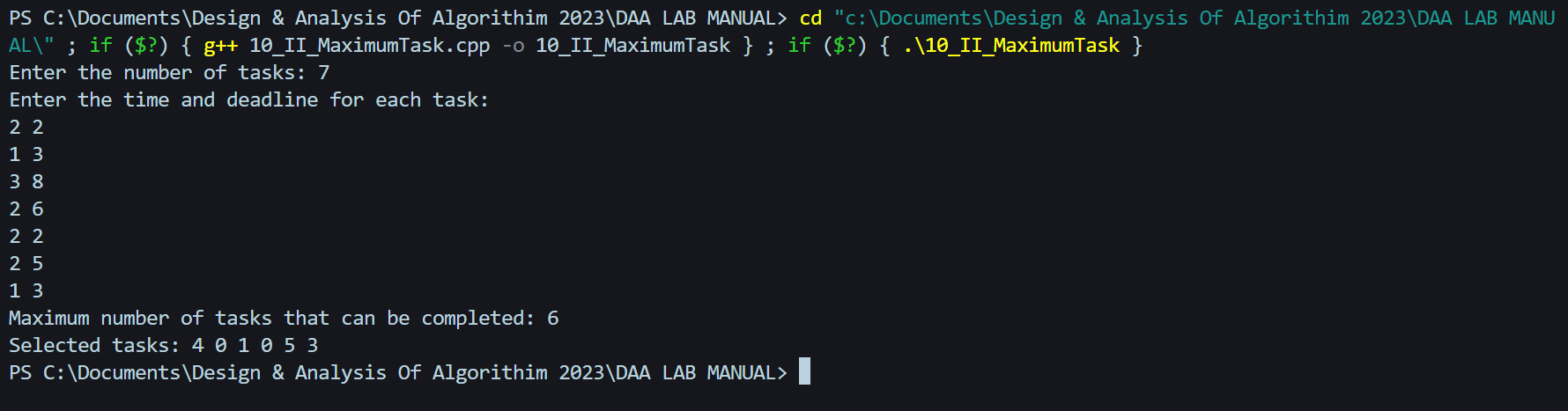
}

cout << endl;

return 0;

}

**Output :**

****

1. Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than n/2 times, where n is the size of array.

**Majority Element**

#include <iostream>

#include <algorithm>

using namespace std;

bool hasMajorityElement(int arr[], int size) {

int candidate = 0;

int count = 0;

for (int i = 0; i < size; i++) {

if (count == 0) {

candidate = arr[i];

count = 1;

} else if (arr[i] == candidate) {

count++;

} else {

count--;

}

}

count = 0;

for (int i = 0; i < size; i++) {

if (arr[i] == candidate) {

count++;

}

}

return count > size / 2;

}

double calculateMedian(int arr[], int size) {

sort(arr, arr + size);

if (size % 2 == 0) {

return (arr[size / 2 - 1] + arr[size / 2]) / 2.0;

} else {

return arr[size / 2];

}

}

int main() {

int arr[] = {2, 4, 5, 2, 2, 3, 2, 2, 6};

int size = sizeof(arr) / sizeof(arr[0]);

bool hasMajority = hasMajorityElement(arr, size);

double median = calculateMedian(arr, size);

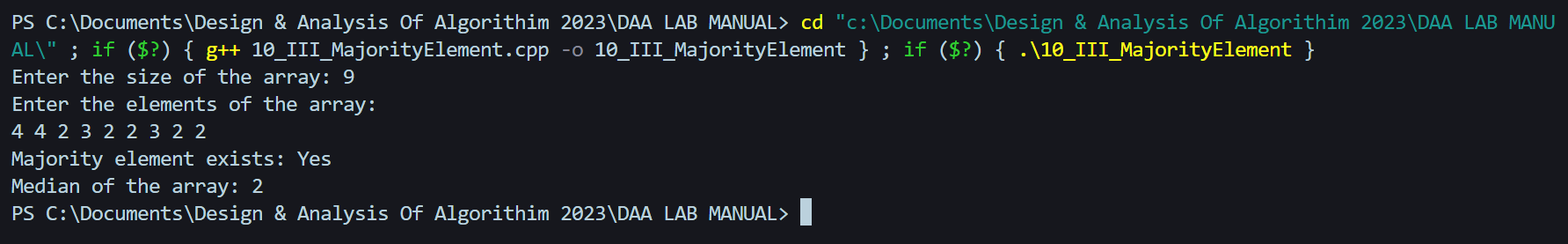
cout << "Majority element exists: " << (hasMajority ? "Yes" : "No") << endl;

cout << "Median of the array: " << median << endl;

return 0;

}

**Output :**

****

**Week – 11**

1. Given a sequence of matrices, write an algorithm to find most efficient way to multiply these matrices together. To find the optimal solution, you need to find the order in which these matrices should be multiplied.

**Chain Matrix Multiplication**

#include <bits/stdc++.h>

using namespace std;

int MatrixChainMultiplication(int arr[], int n)

{

int dp[n][n];

memset(dp, 0, sizeof(dp));

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

{

dp[i][i] = 0;

}

for (int l = 2; l < n; l++)

{

for (int i = 1; i < n - l + 1; i++)

{

int j = i + l - 1;

dp[i][j] = INT\_MAX;

for (int k = i; k <= j - 1; k++)

{

dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j] + arr[i - 1] \* arr[k] \* arr[j]);

}

}

}

return dp[1][n - 1];

}

int main()

{

cout<<"Enter the size of the array: ";

int n;

cin >> n;

int arr[n];

cout << "Enter the elements of the array: ";

for (int i = 0; i < n; i++){

cin >> arr[i];

}

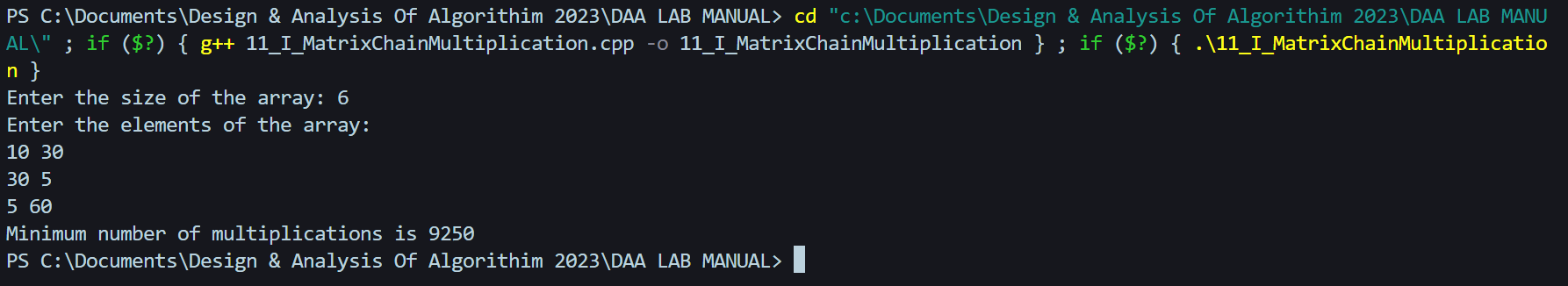
cout << "Minimum number of multiplications is "

<< MatrixChainMultiplication(arr, n);

return 0;

}

**Output :**

****

**Week – 12**

1. Given two sequences, Design an algorithm and implement it using a program to find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

**Dynamic Programming**

#include <bits/stdc++.h>

using namespace std;

int LongestSubsequence(string X, string Y, int n, int m)

{

int dp[n + 1][m + 1];

memset(dp, 0, sizeof(dp));

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

{

for (int j = 1; j < m + 1; j++)

{

if (X[i - 1] == Y[j - 1])

{

dp[i][j] = 1 + dp[i - 1][j - 1];

}

else

{

dp[i][j] = max(dp[i][j - 1], dp[i - 1][j]);

}

}

}

return dp[n][m];

}

int main()

{

cout<<"Enter the two strings: ";

string X, Y;

cin >> X >> Y;

int n = X.length();

int m = Y.length();

cout << "Length of Longest Subsequence is " << LongestSubsequence(X, Y, n, m);

return 0;

}

**Output**

**A picture containing text, screenshot, font

Description automatically generated**

1. Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight <= w and has maximum value. Here, you cannot break an item i.e. either pick the complete item or don't pick it. (0-1 property).

**Knapsack Problem 0-1**

**Time Complexity: O(n\*w)**

**Space Complexity: O(n\*w)**

#include <bits/stdc++.h>

using namespace std;

int Knapsack(int wt[], int val[], int w, int n)

{

int dp[n + 1][w + 1];

memset(dp, 0, sizeof(dp));

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

{

for (int j = 1; j < w + 1; j++)

{

if (wt[i - 1] <= j)

{

dp[i][j] = max(val[i - 1] + dp[i - 1][j - wt[i - 1]],

dp[i - 1][j]);

}

else

{

dp[i][j] = dp[i - 1][j];

}

}

}

return dp[n][w];

}

int main()

{

int val[] = {1,2,5,9,4};

int wt[] = {2,3,3,4,5};

int w = 10;

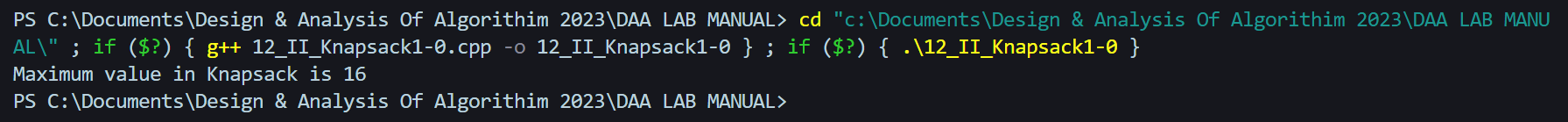
int n = sizeof(val) / sizeof(val[0]);

cout << "Maximum value in Knapsack is " << Knapsack(wt, val, w, n);

return 0;

}

**Output :**

****