/\*

Q-01 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), where n is the size of input)

\*/

#include<iostream>

#include<vector>

using namespace std;

void linearSearch(vector<int> arr,int n, int key){

int i;

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

if(arr[i] == key)

break;

}

if(i != n)

cout << "Present "<<i+1<<endl;

else

cout<<"Not Present"<<endl;

}

int main(){

int n;

cout<<"Enter Size :";

cin>>n;

vector<int> arr(n);

cout<<"Enter Array :";

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

cin >> arr[i];

int key;

cout<<"Enter Key :";

cin >> key;

linearSearch(arr,n,key);

}



/\*

Q-02 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(nlogn), where n is the size of input).

\*/

#include<stdio.h>

int binarySearch(int a[],int n,int key){

int l=0,r=n-1,count=0;

while(l<=r){

int m = (l+r)/2;

if(a[m] == key){

count++;

return count;

}

if(a[m] < key){

l = m+1;

count++;

}

else{

r=m-1;

count++;

}

}

return -1;

}

void main(){

int a[50],n,key,i;

printf("Enter Size :");

scanf("%d",&n);

printf("Enter Elements :");

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

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

}

printf("Enter Key :");

scanf("%d",&key);

int flag=binarySearch(a,n,key);

if(flag != -1){

printf("Present %d",flag);

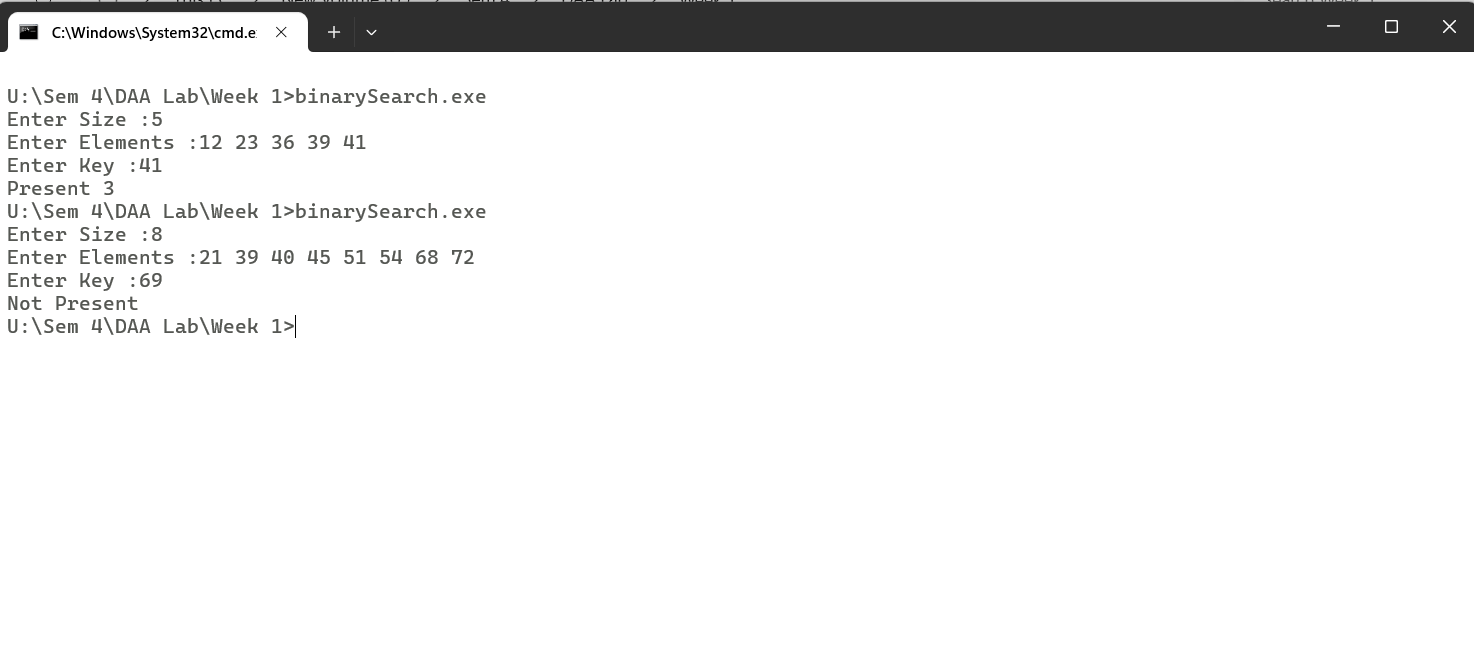
}

else{

printf("Not Present");

}

}



/\*

Q-03 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.

\*/

#include<stdio.h>

#include<math.h>

int jumpSearch(int a[],int n,int key){

int k=0,count=0;

count++;

if(a[0] == key){

return count;

}

while(pow(2,k) < n){

count++;

if(a[(int)pow(2,k)] == key){

return count;

}else if(a[(int)pow(2,k+1)] == key){

return count;

}else if(a[(int)pow(2,k)] < key && a[(int)pow(2,k+1)] > key)

break;

k++;

}

for(int i=(int)pow(2,k)+1;i<=(int)pow(2,k+1);i++){

count++;

if(a[i] == key)

return 1;

}

return -1;

}

void main(){

int n,a[50],i,key,flag;

printf("Enter Size :");

scanf("%d",&n);

printf("Enter Array Elements :");

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

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

}

printf("Enter Key :");

scanf("%d",&key);

flag=jumpSearch(a,n,key);

if(flag != -1){

printf("Key Found after %d Comparisons\n",flag);

}else

printf("Key Not Found\n");

}



/\*

Q-01 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))

\*/

#include<stdio.h>

void main(){

int i,a[50],n,key,count=0;

printf("Enter Size :");

scanf("%d",&n);

printf("Enter Array Elements :");

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

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

}

printf("Enter Key :");

scanf("%d",&key);

int l=0,r=n-1;

while(l<=r){

int mid = (l+r)/2;

if(a[mid] == key){

count++;

for(i=mid-1;i>=0;i--){

if(a[i]!=key)

break;

else

count ++;

}

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

if(a[i]!=key)

break;

else

count++;

}

}

if(a[mid]<key){

l=mid+1;

}else{

r=mid-1;

}

}

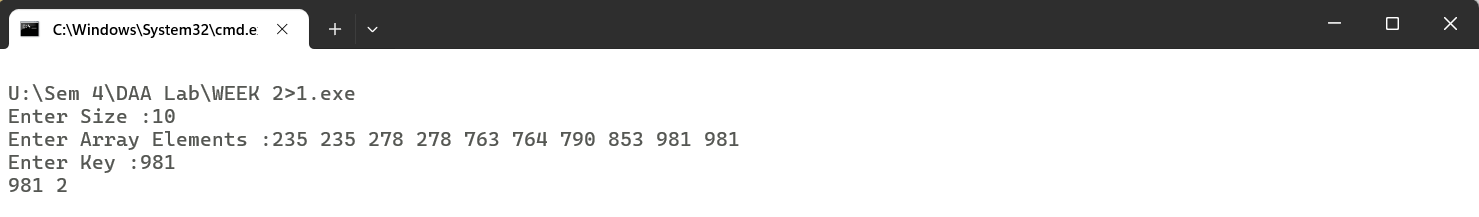
if(count == 0)

printf("Not Found\n");

else

printf("%d %d\n",key,count);

}



/\*

Q-02 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].

\*/

#include<stdio.h>

void main(){

int n,a[50],i,j,k;

printf("Enter Size :");

scanf("%d",&n);

printf("Enter Array :");

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

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

for(i=n-1;i>=0;i--){

j=0;

k=i-1;

while(j<k && a[j]+a[k] != a[i]){

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

j++;

else if(a[j]+a[k] > a[i])

k--;

}

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

break;

}

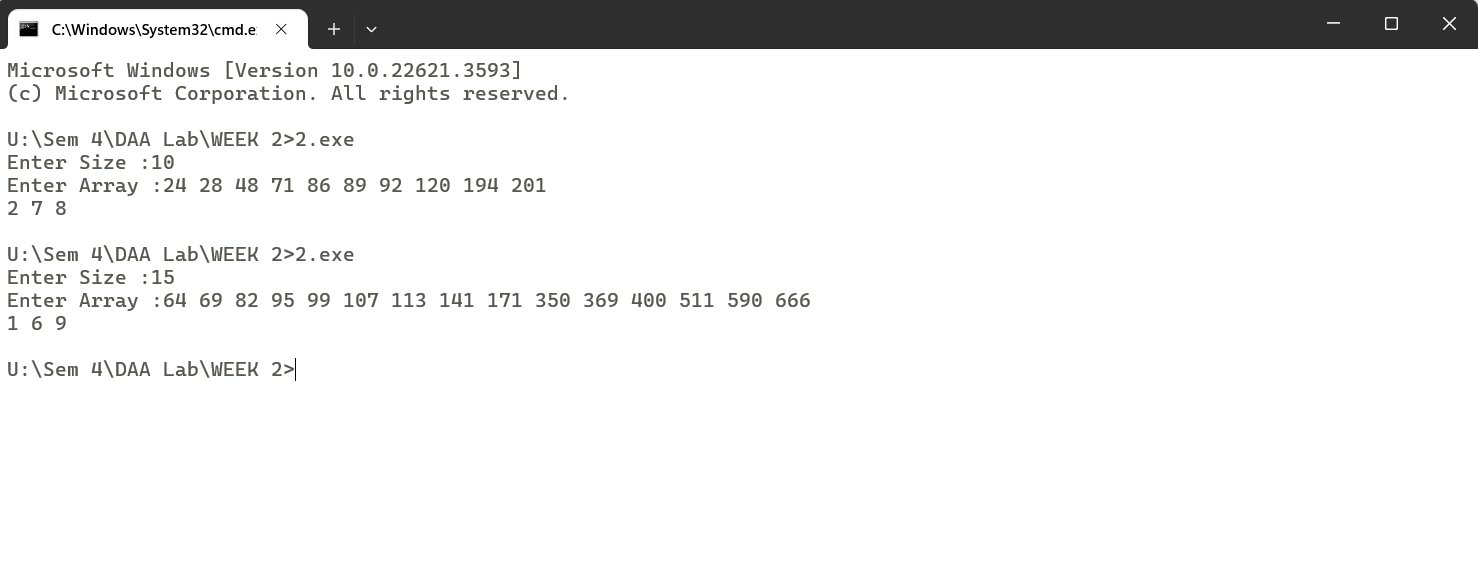
if(a[j] + a[k] != a[i])

printf("No Sequence Found\n");

else

printf("%d %d %d\n",j+1,k+1,i+1);

}



/\*

Q-03 Given an array of nonnegative integers, design an algorithm and a program to count the numberof pairs of integers such that their difference is equal to a given key, K.

\*/

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

void findDiff(vector<int> arr,int n,int key){

int c=0;

sort(arr.begin(),arr.end());

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

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

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

c++ ;

break;

}

}

}

cout<<"Total Pairs :"<<c;

}

int main(){

int n;

cout<<"Enter Size :";

cin>>n;

vector<int> arr(n);

cout<<"Enter Array :";

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

cin >> arr[i];

}

int key;

cout<<"Enter Key :";

cin >> key;

findDiff(arr,n,key);

}



/\*

Q-01 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.

\*/

#include<stdio.h>

void insertionSort(int a[],int n){

int i,j,t;

int shifts=0,comparisons=0;

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

comparisons++;

t=a[i];

j=i-1;

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

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

j--;

shifts++;

comparisons++;

}

a[j+1]=t;

}

printf("Sorted Array is :");

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

printf("%d ",a[i]);

}

printf("\nShifts =%d\nComparisons =%d",comparisons,shifts);

}

void main()

{

int n,a[50],key,i;

printf("Enter Size :");

scanf("%d",&n);

printf("Enter Elements :");

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

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

}

insertionSort(a,n);

}



/\*

Q-02 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.

\*/

#include<stdio.h>

void selectionSort(int a[],int n){

int i,j,pos,comp=0,swaps=0;

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

pos =i;

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

comp++;

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

pos = j;

}

}

swaps++;

int temp = a[pos];

a[pos] = a[i];

a[i]=temp;

}

printf("Swaps :%d\nComparisons :%d\n",swaps,comp);

}

void main()

{

int n,i,a[50];

printf("Enter Size :");

scanf("%d",&n);

printf("Enter Array Elements :");

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

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

}

selectionSort(a,n);

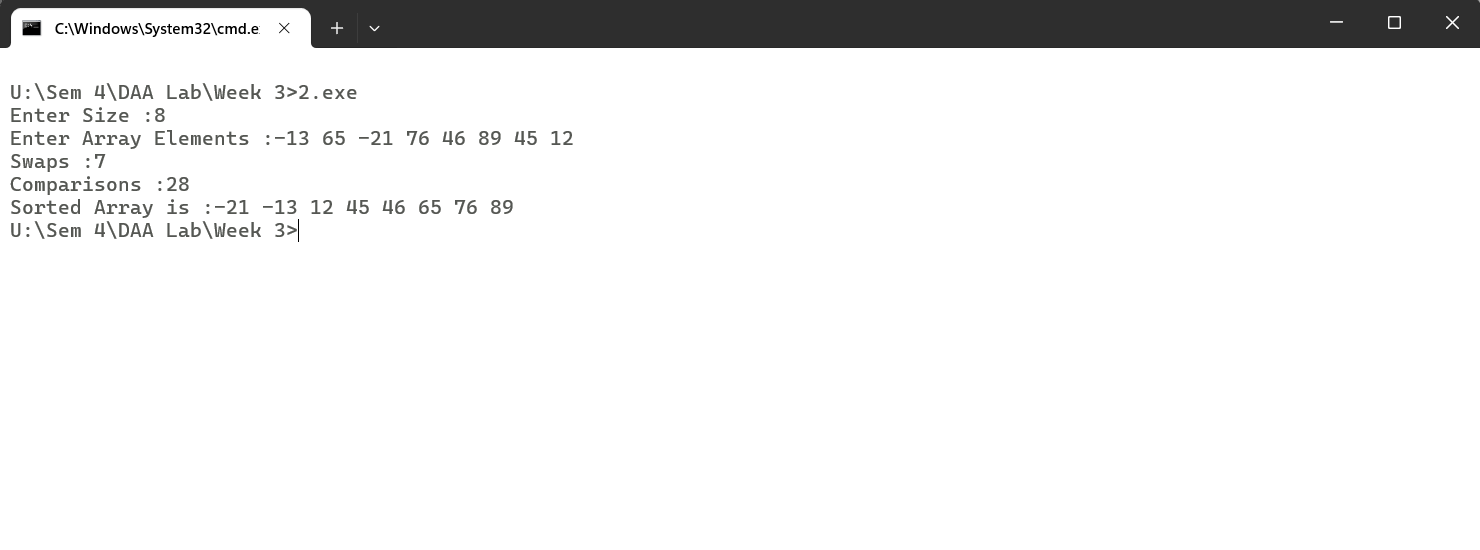
printf("Sorted Array is :");

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

printf("%d ",a[i]);

}

}



/\*

Q-03 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))

\*/

#include<stdio.h>

void merge(int a[],int lb,int mid,int ub)

{

int temp[50],i=lb,j=mid+1,k=lb;

while(i<=mid && j<=ub){

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

temp[k]=a[i];

k++;

i++;

}

else{

temp[k]=a[j];

k++;

j++;

}

}

while(i<=mid){

temp[k]=a[i];

k++;

i++;

}

while(j<=ub){

temp[k]=a[j];

k++;

j++;

}

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

a[i] = temp[i];

}

void Sort(int a[],int lb,int ub)

{

int mid;

if(lb < ub){

mid =(lb+ub)/2;

Sort(a,lb,mid);

Sort(a,mid+1,ub);

merge(a,lb,mid,ub);

}

}

void check(int a[],int n){

int i;

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

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

break;

if(i != n-1) printf("YES");

else printf("NO");

}

void main()

{

int n,i,a[50];

printf("Enter Size :");

scanf("%d",&n);

printf("Enter Array Elements :");

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

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

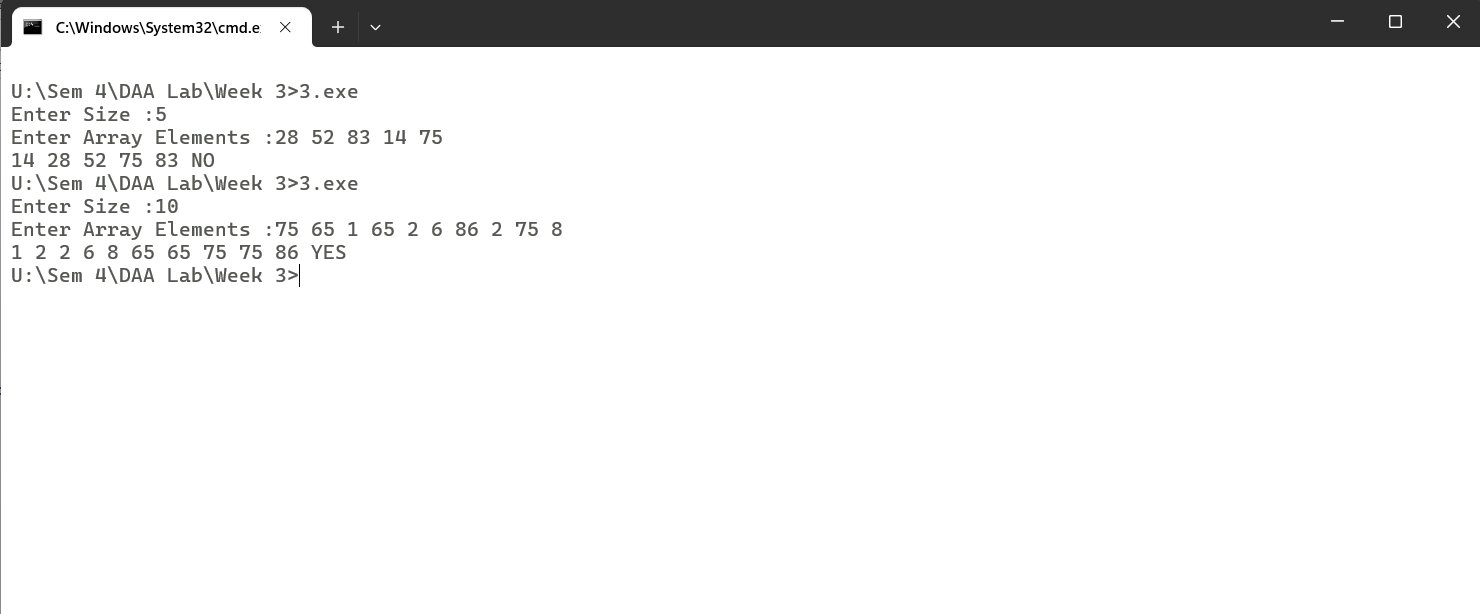
Sort(a,0,n-1);

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

printf("%d ",a[i]);

check(a,n);

}



/\*

Q-01 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.

\*/

#include<stdio.h>

void mSort(int a[],int t[],int lb,int mid,int ub,int \*c,int \*in){

int k=lb,le=mid;

while(lb <= le && mid+1 <= ub){

(\*in)++;

if(a[lb]<a[mid+1]){

(\*c)++;

t[k] = a[lb];

k++;

lb++;

}

else{

(\*c)++;

t[k] = a[mid+1];

k++;

mid++;

}

}

while(lb<=le){

(\*in)++;

t[k] = a[lb];

k++;

lb++;

}

while(mid+1 <= ub){

(\*in)++;

t[k] = a[mid+1];

k++;

mid++;

}

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

a[i] = t[i];

}

void merge(int a[],int t[],int lb,int ub,int \*c,int \*in){

int mid;

if(lb < ub){

mid = (lb +ub)/2;

merge(a,t,lb,mid,c,in);

merge(a,t,mid+1,ub,c,in);

mSort(a,t,lb,mid,ub,c,in);

}

}

void main(){

int n,i,a[50],t[50];

printf("Enter Size :");

scanf("%d",&n);

printf("Enter Array Elements :");

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

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

}

int c=0,in=0;

merge(a,t,0,n-1,&c,&in);

printf("Sorted Array is :");

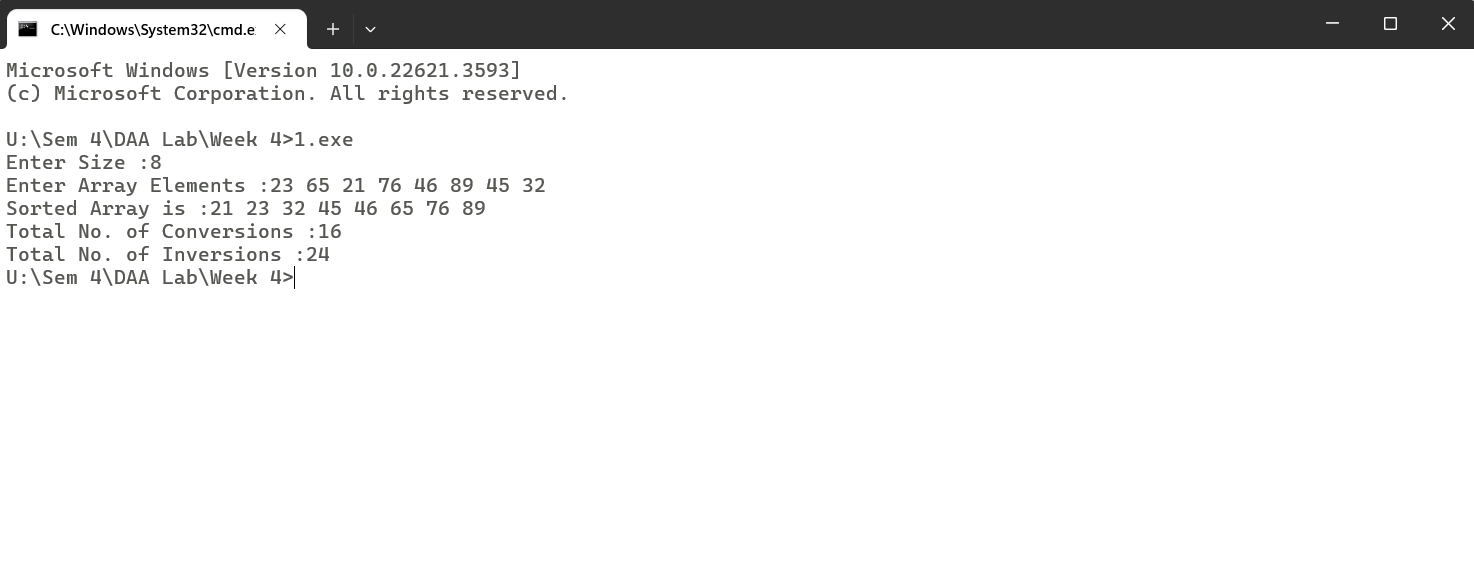
for(i=0;i<n;i++){

printf("%d ",a[i]);

}

printf("\nTotal No. of Conversions :%d\nTotal No. of Inversions :%d",c,in);

}



/\*

Q-02 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.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void swap1(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int a[], int lb, int ub, int \*swap, int \*comp) {

int randomIndex = lb + rand() % (ub - lb + 1);

swap1(&a[lb], &a[randomIndex]);

(\*swap)++;

int pivot = a[lb];

int i = lb;

int j = ub;

while (i < j) {

while (i <= ub && a[i] <= pivot) {

(\*comp)++;

i++;

}

while (a[j] > pivot) {

(\*comp)++;

j--;

}

if (i < j) {

swap1(&a[i], &a[j]);

(\*swap)++;

}

}

swap1(&a[lb], &a[j]);

(\*swap)++;

return j;

}

void quickSort(int a[], int lb, int ub, int \*swap, int \*comp) {

if (lb < ub) {

int loc = partition(a, lb, ub, swap, comp);

quickSort(a, lb, loc - 1, swap, comp);

quickSort(a, loc + 1, ub, swap, comp);

}

}

int main() {

int n, a[50], i;

printf("Enter Size: ");

scanf("%d", &n);

printf("Enter Elements: ");

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

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

}

int swap = 0, comp = 0;

srand(time(0));

quickSort(a, 0, n - 1, &swap, &comp);

printf("Sorted Array: ");

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

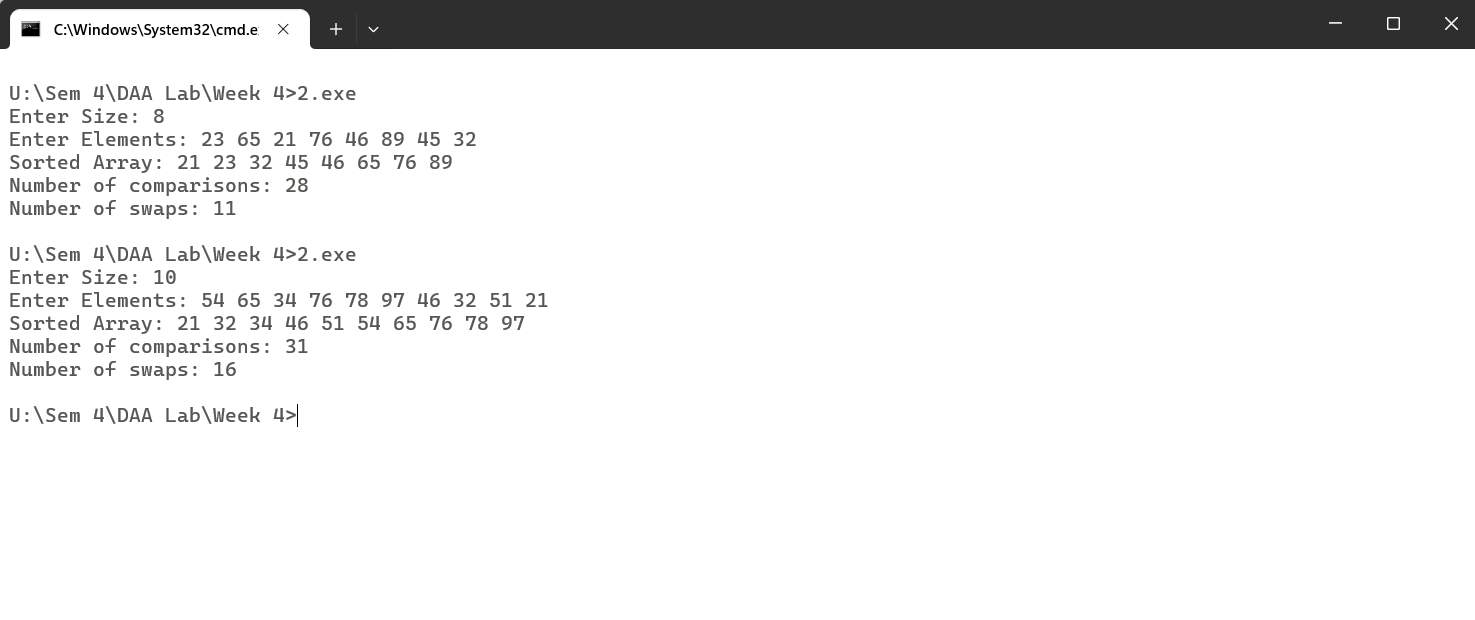
printf("%d ", a[i]);

}

printf("\nNumber of comparisons: %d\nNumber of swaps: %d\n", comp, swap);

return 0;

}



/\*

Q-03 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))

\*/

#include<stdio.h>

int partition(int arr[], int l, int r)

{

int x = arr[r], i = l,temp;

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

if (arr[j] <= x) {

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

i++;

}

}

temp=arr[i];

arr[i]=arr[r];

arr[r]=temp;

return i;

}

int kthSmallest(int arr[], int l, int r, int k)

{

if (k > 0 && k <= r - l + 1) {

int index = partition(arr, l, r);

if (index - l == k - 1)

return arr[index];

if (index - l > k - 1)

return kthSmallest(arr, l, index - 1, k);

return kthSmallest(arr, index + 1, r,k - index + l - 1);

}

}

void main()

{

int n,a[50],i;

printf("Enter Size :");

scanf("%d",&n);

printf("Enter Elements :");

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

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

}

int k;

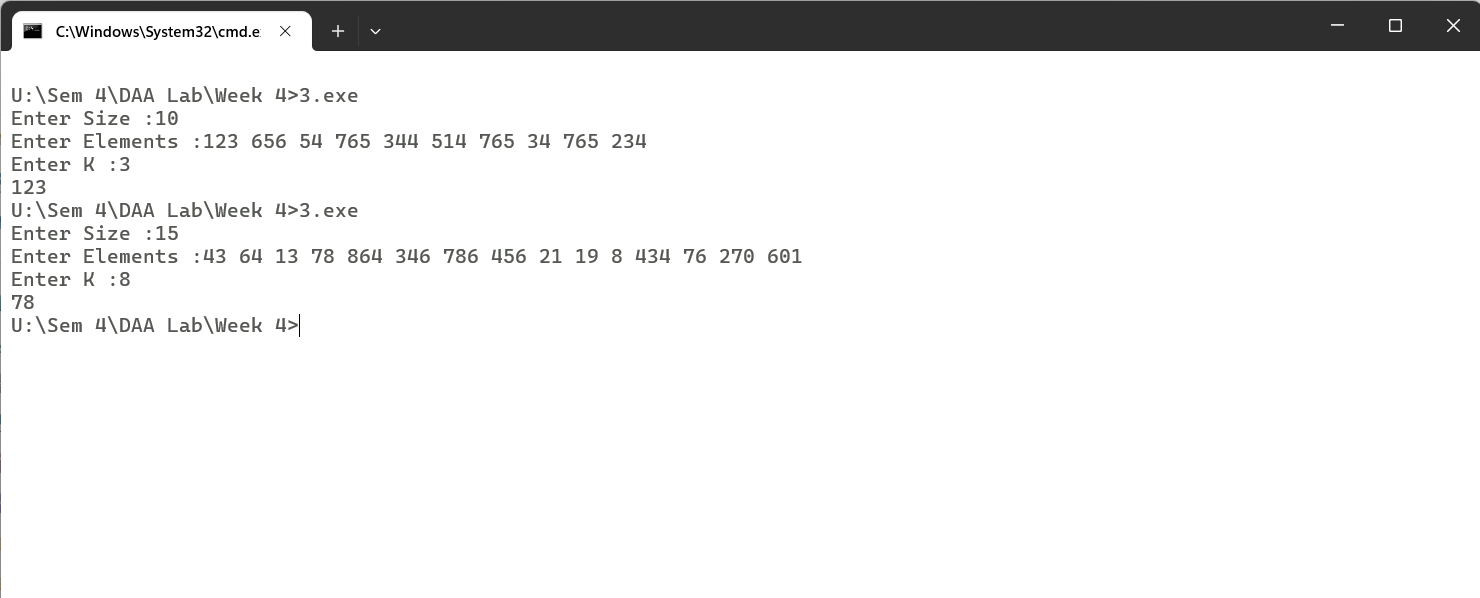
printf("Enter K :");

scanf("%d",&k);

int number= kthSmallest(a,0,n-1,k);

printf("%d",number);

}



/\*

Q-01 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)) (Hint: Use counting sort)

\*/

#include<iostream>

using namespace std;

int main(){

int i,a[26]={0};

string str;

cout << "Enter String :";

cin >> str;

for(i=0;i<str.length();i++){

a[str[i]-97]++;

}

int max=0,pos=0;

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

if(max < a[i]){

max = a[i];

pos =i;

}

}

cout<<char('a'+pos)<<" - "<<max<<endl;

}



/\*

Q-02 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>

#include<algorithm>

using namespace std;

void check(vector<int> arr, int target)

{

sort(arr.begin(),arr.end());

int left=0,right=arr.size()-1,sum;

while(left < right){

sum =arr[left]+arr[right];

if(sum == target)

break;

else if(sum < target)

left++;

else

right--;

}

if(sum == target){

cout<<arr[left]<<" "<<arr[right];

}

else

cout<<"Does not exist";

}

int main()

{

int n;

cout<<"Enter Size :";

cin>>n;

vector<int> arr(n);

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

cin>>arr[i];

}

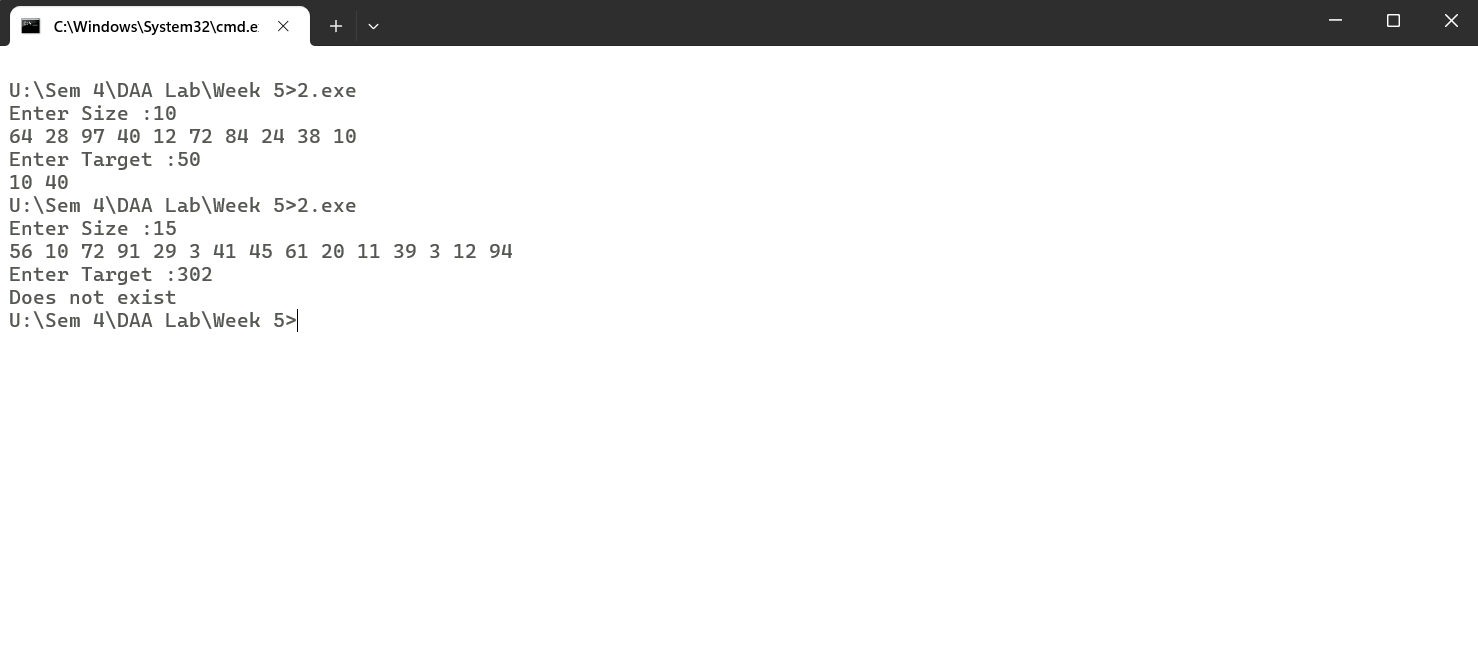
int target;

cout<<"Enter Target :";

cin>>target;

check(arr,target);

}



/\*

Q-03 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<stdio.h>

void show(int a[],int m,int b[],int n){

int i=0,j=0;

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

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

printf("%d ",a[i]);

i++;

j++;

}

else if(a[i] < b[j]) i++;

else j++;

}

}

void main()

{

int m,n,a[50],b[50],i;

printf("Enter Size :");

scanf("%d",&m);

printf("Enter Elements :");

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

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

printf("Enter Size :");

scanf("%d",&n);

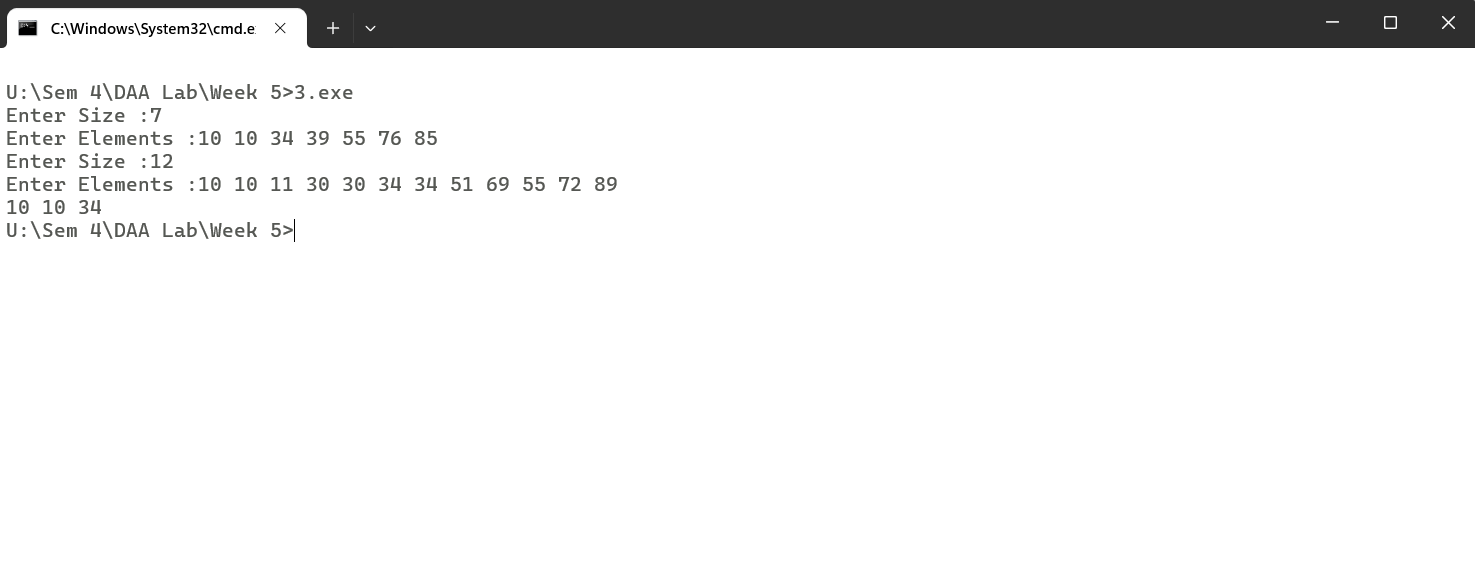
printf("Enter Elements :");

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

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

show(a,m,b,n);

}



/\*

Q-01 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. (Hint: use DFS)

\*/

#include<bits/stdc++.h>

using namespace std;

void input(int \*\*graph,int n){

cout<<"Enter Adjacency Matrix :";

int i,j;

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

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

cin>>graph[i][j];

}

}

}

void dfs(int \*\*graph, int n){

stack<int> s;

vector<int> status(n,0);

int source,dest;

cout<<"Enter Source and Destination Nodes :";

cin>>source>>dest;

source-=1;

dest-=1;

s.push(source);

int poped;

do{

poped = s.top();

status[poped] =1;

s.pop();

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

if(graph[poped][i] == 1 && status[i] == 0){

s.push(i);

}

}

}while(poped != dest && !s.empty());

if(poped == dest)

cout<<"Yes Path Exists\n";

else

cout<<"No Path exists\n";

}

int main()

{

int i,n;

cout<<"Enter No. of Vertices :";

cin>>n;

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

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

graph[i] = new int[n];

input(graph,n);

dfs(graph,n);

}



/\*

Q-02 Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS)

\*/

#include<bits/stdc++.h>

using namespace std;

void input(int \*\*graph,int n){

cout<<"Enter Adjacency Matrix :";

int i,j;

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

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

cin>>graph[i][j];

}

}

}

bool bfs(int \*\*graph,int n)

{

queue<int> q;

vector<int> status(n,0);

q.push(0);

status[0] =1;

while(!q.empty()){

int popped = q.front();

q.pop();

if(graph[popped][popped] == 1)

return false;

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

if(graph[popped][i]==1 && status[i] == 0){

status[i] = 1 + status[popped];

q.push(i);

}

else if(graph[popped][i] ==1 && status[i] == status[popped])

return false;

}

}

return true;

}

int main()

{

int i,n;

cout<<"Enter No. of Vertices :";

cin>>n;

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

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

graph[i] = new int[n];

input(graph,n);

bool ans = bfs(graph,n);

if(ans)

cout<<"Yes, Bipartite Graph\n";

else

cout<<"No";

}



/\*

Q-03 Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.

\*/

#include<bits/stdc++.h>

using namespace std;

void input(int \*\*graph,int n){

cout<<"Enter Adjacency Matrix :";

int i,j;

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

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

cin>>graph[i][j];

}

}

}

bool checkCycle(int \*\*graph, int n){

stack<int> s;

vector<int> status(n,0);

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

if (status[i] == 0) {

s.push(i);

while (!s.empty()) {

int popped = s.top();

s.pop();

if (status[popped] == 0) {

status[popped] = 1;

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

if (graph[popped][j] == 1) {

if (status[j] == 1)

return true; // Cycle detected

s.push(j);

}

}

}

status[popped] = 2; // Node is completely processed

}

}

}

return false;

}

int main()

{

int i,n;

cout<<"Enter No. of Vertices :";

cin>>n;

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

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

graph[i] = new int[n];

input(graph,n);

bool ans =checkCycle(graph,n);

if(ans)

cout<<"Cycle Exists";

else

cout<<"No Cycle Exists";

}



/\*

Q-01 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? (Hint: Print shortest path and distance from friends’ location to Akshay’s house)

\*/

#include <iostream>

#include <vector>

#include <utility>

#include <climits>

#include <stack>

using namespace std;

int getMin(const vector<pair<int, int>>& dist, const vector<bool>& status) {

int minVal = INT\_MAX, minIndex;

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

if (!status[i] && minVal >= dist[i].first) {

minVal = dist[i].first;

minIndex = i;

}

}

return minIndex;

}

void printPath(const vector<int>& prev, int src, int dest) {

stack<int> path;

for (int at = dest; at != -1; at = prev[at]) {

path.push(at);

}

while (!path.empty()) {

cout << path.top() + 1 << " ";

path.pop();

}

cout << endl;

}

void dijkstra(const vector<vector<int>>& graph, int src) {

int n = graph.size();

vector<pair<int, int>> dist(n, {INT\_MAX, -1});

vector<bool> status(n, false);

vector<int> prev(n, -1);

dist[src - 1] = {0, -1};

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

int index = getMin(dist, status);

status[index] = true;

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

if (!status[j] && graph[index][j] && dist[index].first != INT\_MAX && dist[index].first + graph[index][j] < dist[j].first) {

dist[j].first = dist[index].first + graph[index][j];

dist[j].second = index;

prev[j] = index;

}

}

}

cout << "Shortest paths from node " << src << ":\n";

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

cout << "Node " << i + 1 << ": ";

if (dist[i].first == INT\_MAX)

cout << "Not reachable";

else {

cout << "Weight: " << dist[i].first << ", Path: ";

printPath(prev, src - 1, i);

}

cout << endl;

}

}

int main() {

int n;

cout << "Enter number of vertices: ";

cin >> n;

vector<vector<int>> graph(n, vector<int>(n));

cout << "Enter adjacency matrix:\n";

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

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

cin >> graph[i][j];

}

}

int src;

cout << "Enter source node: ";

cin >> src;

if (src < 1 || src > n) {

cout << "Invalid source node!";

return 1;

}

dijkstra(graph, src);

return 0;

}



/\*

Q-02 Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm.

\*/

#include<bits/stdc++.h>

using namespace std;

struct Edge {

int src, dest, weight;

};

void printPath(const vector<int>& prev, int src, int dest) {

stack<int> path;

for (int at = dest; at != -1; at = prev[at]) {

path.push(at);

}

while (!path.empty()) {

cout << path.top() + 1 << " ";

path.pop();

}

cout << endl;

}

void bellmanFord(const vector<Edge>& edges, int n, int src) {

vector<int> dist(n, INT\_MAX);

vector<int> prev(n, -1);

dist[src - 1] = 0;

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

for (const Edge& edge : edges) {

if (dist[edge.src] != INT\_MAX && dist[edge.src] + edge.weight < dist[edge.dest]) {

dist[edge.dest] = dist[edge.src] + edge.weight;

prev[edge.dest] = edge.src;

}

}

}

for (const Edge& edge : edges) {

if (dist[edge.src] != INT\_MAX && dist[edge.src] + edge.weight < dist[edge.dest]) {

cout << "Graph contains a negative cycle\n";

return;

}

}

cout << "Shortest paths from node " << src << ":\n";

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

cout << "Node " << i + 1 << ": ";

if (dist[i] == INT\_MAX)

cout << "Not reachable";

else {

cout << "Weight: " << dist[i] << ", Path: ";

printPath(prev, src - 1, i);

}

cout << endl;

}

}

int main() {

int n;

cout << "Enter number of vertices: ";

cin >> n;

vector<Edge> edges;

cout << "Enter adjacency matrix (source, destination, weight):\n";

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

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

int weight;

cin >> weight;

if (weight != 0)

edges.push\_back({i, j, weight});

}

}

int src;

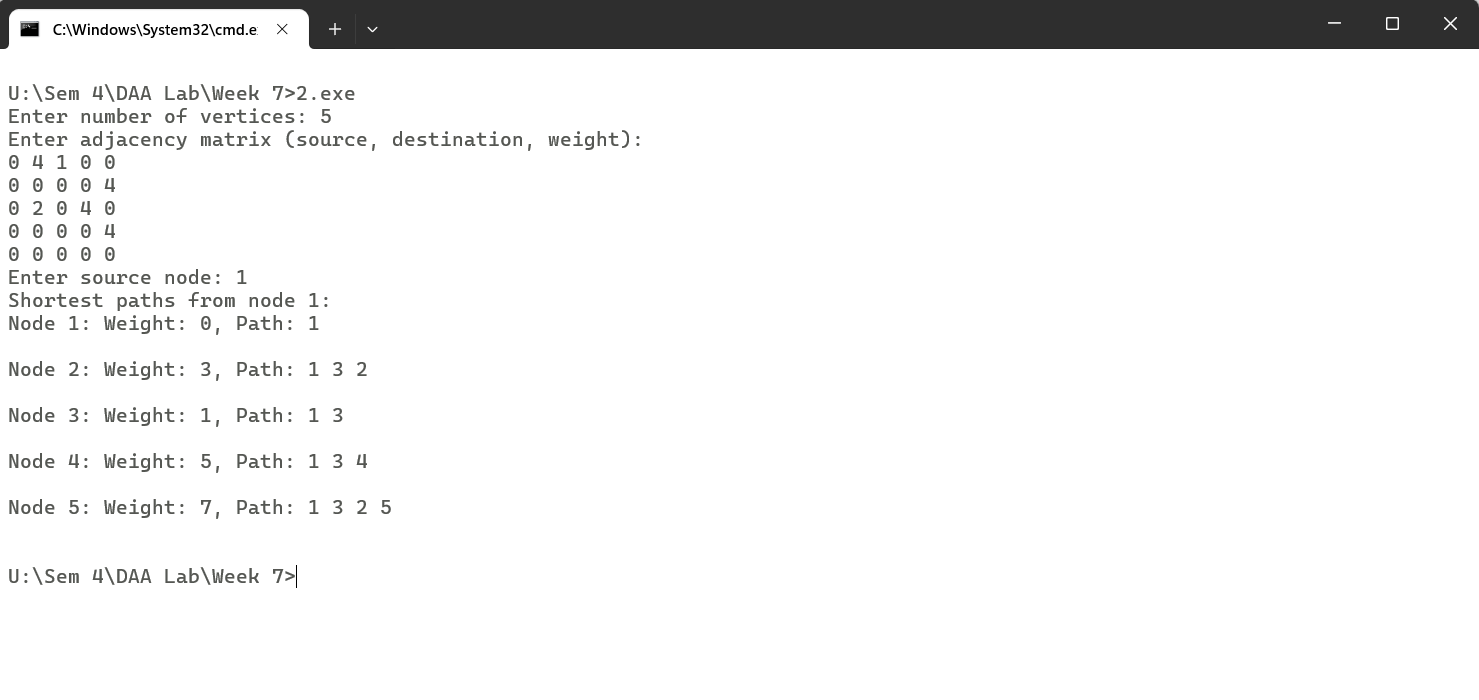
cout << "Enter source node: ";

cin >> src;

bellmanFord(edges, n, src);

return 0;

}



/\*

Q-03 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.

\*/

#include <bits/stdc++.h>

using namespace std;

#define INF INT\_MAX

int shortestPath(int \*\*graph, int u, int v, int k,int n)

{

if (k == 0 && u == v) return 0;

if (k == 1 && graph[u][v] != INF) return graph[u][v];

if (k <= 0) return INF;

int res = INF;

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

{

if (graph[u][i] != INF && u != i && v != i)

{

int rec\_res = shortestPath(graph, i, v, k-1, n);

if (rec\_res != INF)

res = min(res, graph[u][i] + rec\_res);

}

}

return res;

}

int main()

{

cout<<"Enter Size ";

int n;

cin >> n;

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

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

graph[i] = new int[n];

}

cout<<"Enter Adjacency Matrix :"<<endl;

cin.ignore();

string line;

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

getline(cin, line);

stringstream ss(line);

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

string value;

ss >> value;

if (value == "INF") {

graph[i][j] = INF;

} else {

graph[i][j] = stoi(value);

}

}

}

cout<<"Enter Source and Destination :";

int u,v,k;

cin >> u >> v;

u--;

v--;

cout<<"Enter k edges :";

cin>>k;

cout << "Weight of the shortest from source to destination is " <<shortestPath(graph, u, v, k, n);

return 0;

}



/\*

Q-01 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. (use Prim's algorithm)

\*/

#include<iostream>

#include<vector>

#include<climits>

using namespace std;

int findMinimum(vector<int> values, vector<bool> status){

int value =INT\_MAX,index;

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

if(!status[i] && values[i] < value){

value = values[i];

index = i;

}

}

return index;

}

int prims(vector<vector<int>> graph,int n){

vector<bool> status(n, false);

int minimumWeight = 0;

vector<int> values(n, INT\_MAX);

values[0] = 0;

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

int index = findMinimum(values, status);

status[index] = true;

minimumWeight += values[index];

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

if(status[j] == false && graph[index][j] != 0 &&values[j] > graph[index][j])

values[j] = graph[index][j];

}

}

return minimumWeight;

}

int main(){

int n;

cout<<"Enter Size :";

cin>>n;

cout<<"Enter Adjacency Matrix :";

vector<vector<int>> graph(n, vector<int>(n,0));

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

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

cin >> graph[i][j];

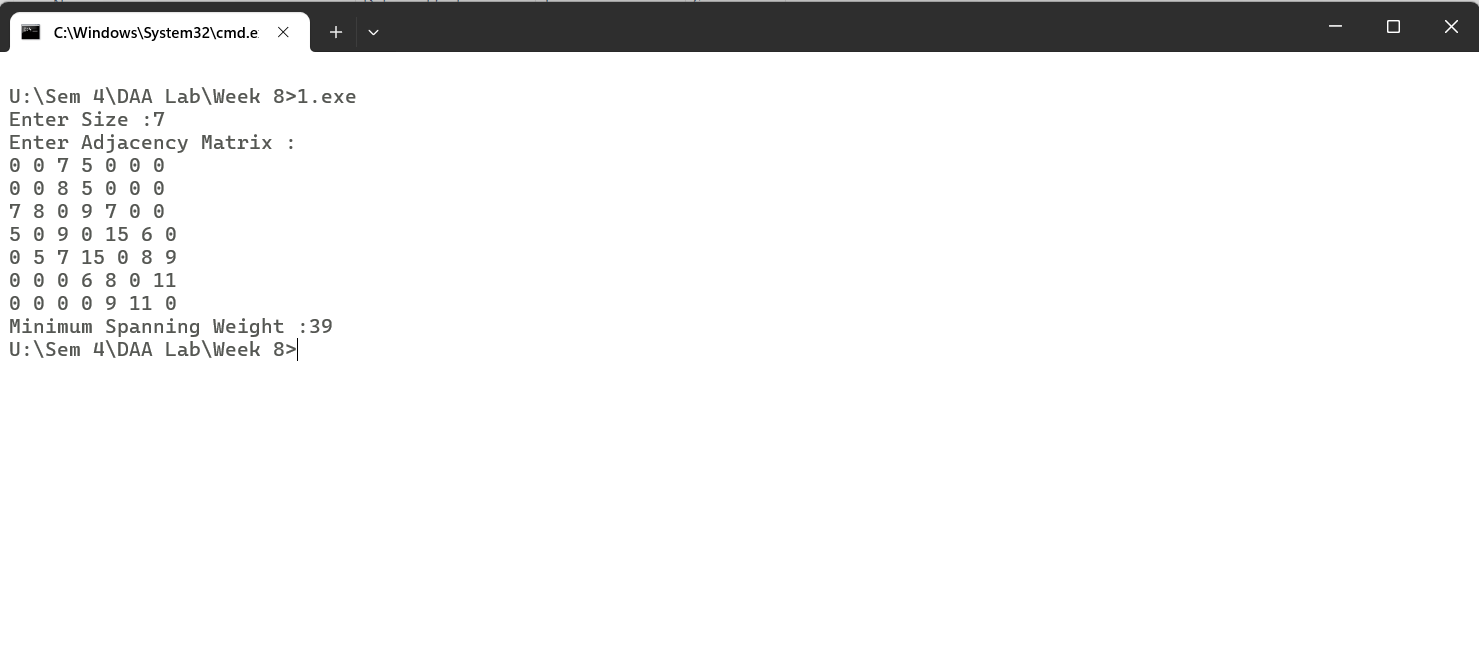
}

}

int minimumWeight = prims(graph,n);

cout<<"Minimum Spanning Weight :"<<minimumWeight;

}



/\*

Q-02 Implement the previous problem using Kruskal's algorithm

\*/

#include<bits/stdc++.h>

using namespace std;

int findParent(vector<int> disjoint, int u){

while(disjoint[u] > 0){

u = disjoint[u];

}

return u;

}

bool checkLoop(vector<int> &disjoint, int u,int v){

int w = findParent(disjoint, u);

int x = findParent(disjoint, v);

if(w == x)

return true;

if(disjoint[w] < disjoint[x]){

disjoint[w] += disjoint[x];

disjoint[x] = w;

}else{

disjoint[x] += disjoint[w];

disjoint[w] = x;

}

return false;

}

int kruskal(vector<vector<int>> graph, int n){

set<pair<int, pair<int,int>>> edges;

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

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

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

edges.insert({graph[i][j], {min(i,j),max(i,j)}});

}

}

}

int minimumWeight =0;

vector<int> disjoint(n, -1);

for(auto it : edges){

bool loop = checkLoop(disjoint, it.second.first, it.second.second);

if(!loop)

minimumWeight += it.first;

}

return minimumWeight;

}

int main(){

int n;

cout<<"Enter Size :";

cin>>n;

cout<<"Enter Adjacency Matrix :";

vector<vector<int>> graph(n, vector<int>(n,0));

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

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

cin >> graph[i][j];

}

}

int minimumWeight = kruskal(graph,n);

cout<<"Minimum Spanning Weight :"<<minimumWeight;

}



/\*

Q-03 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.

\*/

#include<bits/stdc++.h>

using namespace std;

int findParent(vector<int> disjoint, int u){

while(disjoint[u] > 0){

u = disjoint[u];

}

return u;

}

bool checkLoop(vector<int> &disjoint, int u,int v){

int w = findParent(disjoint, u);

int x = findParent(disjoint, v);

if(w == x)

return true;

if(disjoint[w] < disjoint[x]){

disjoint[w] += disjoint[x];

disjoint[x] = w;

}else{

disjoint[x] += disjoint[w];

disjoint[w] = x;

}

return false;

}

int maximumKruskal(vector<vector<int>> graph, int n){

set<pair<int, pair<int, int>>, greater<pair<int,pair<int, int>>> > edges;

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

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

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

edges.insert({graph[i][j], {min(i,j),max(i,j)}});

}

}

}

int maximumWeight =0;

vector<int> disjoint(n, -1);

for(auto it : edges){

bool loop = checkLoop(disjoint, it.second.first, it.second.second);

if(!loop)

maximumWeight += it.first;

}

return maximumWeight;

}

int main(){

int n;

cout<<"Enter Size :";

cin>>n;

cout<<"Enter Adjacency Matrix :";

vector<vector<int>> graph(n, vector<int>(n,0));

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

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

cin >> graph[i][j];

}

}

cout<<endl;

int maximumWeight = maximumKruskal(graph,n);

cout<<"Maximum Spanning Weight :"<<maximumWeight;

}



/\*

Q-01 Given a graph, Design an algorithm and implement it using a program to implement FloydWarshall all pair shortest path algorithm

\*/

#include <bits/stdc++.h>

using namespace std;

#define INF 99999

void printSolution(int \*\*dist, int V);

void floydWarshall(int \*\*dist,int V)

{

int i, j, k;

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

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

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

if (dist[i][j] > (dist[i][k] + dist[k][j])&& (dist[k][j] != INF && dist[i][k] != INF))

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

}

}

}

printSolution(dist,V);

}

void printSolution(int \*\*dist, int V)

{

cout << "Shortest Distance Matrix "<<endl;

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;

}

}

int main()

{

int n;

cout<<"Enter No. of Vertices :";

cin>>n;

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

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

graph[i] = new int[n];

}

cout<<"Enter Adjacency Matrix :"<<endl;

cin.ignore();

string line;

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

getline(cin, line);

stringstream ss(line);

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

string value;

ss >> value;

if (value == "INF") {

graph[i][j] = INF;

} else {

graph[i][j] = stoi(value);

}

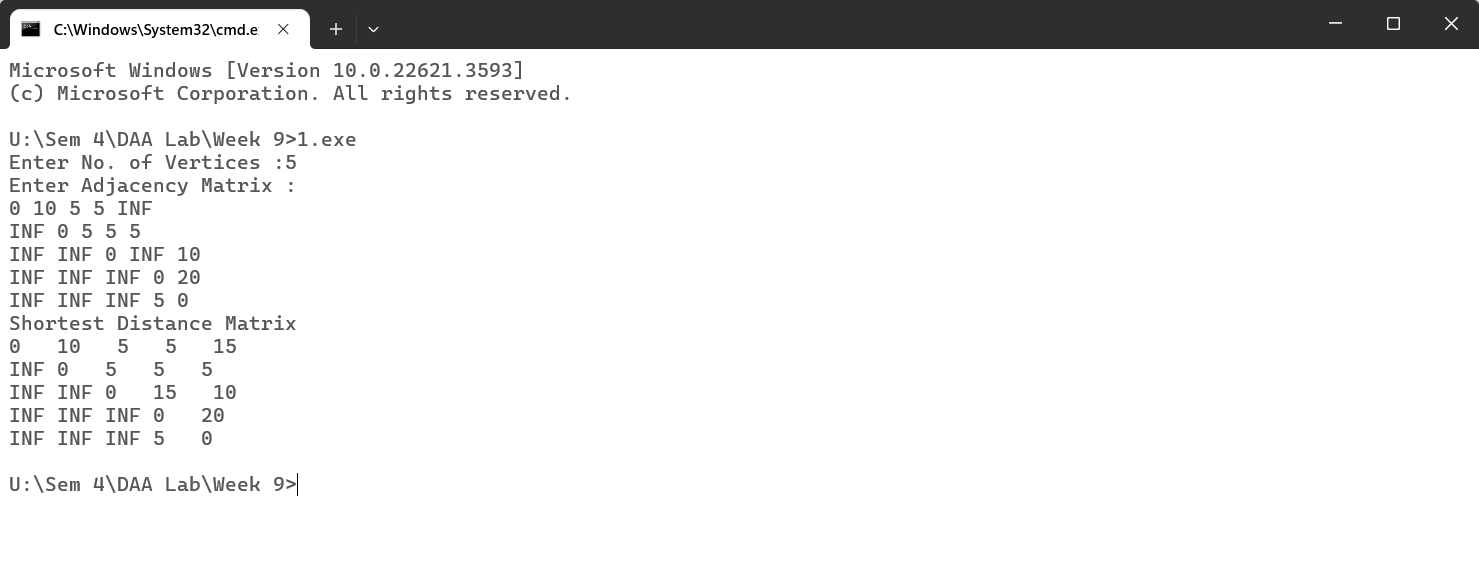
}

}

floydWarshall(graph,n);

return 0;

}



/\*

Q-02 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.

\*/

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Item {

double weight;

double value;

double ratio;

};

bool compare(Item a, Item b) {

return a.ratio > b.ratio;

}

pair<double, vector<pair<double, double>>> fractional\_knapsack(int n, vector<double>& weights, vector<double>& values, double capacity) {

vector<Item> items(n);

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

items[i] = {weights[i], values[i], values[i] / weights[i]};

sort(items.begin(), items.end(), compare);

double total\_value = 0.0;

vector<pair<double, double>> knapsack\_contents;

for (const auto& item : items) {

if (capacity > 0) {

if (item.weight <= capacity) {

total\_value += item.value;

capacity -= item.weight;

knapsack\_contents.push\_back({item.weight, 1.0});

} else {

double fraction = capacity / item.weight;

total\_value += item.value \* fraction;

knapsack\_contents.push\_back({item.weight, fraction});

break;

}

} else

break;

}

return {total\_value, knapsack\_contents};

}

int main() {

int n;

cout<<"Enter Size :";

cin >> n;

vector<double> weights(n);

vector<double> values(n);

double capacity;

cout<<"Enter Weights :";

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

cin >> weights[i];

cout<<"Enter Values :";

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

cin >> values[i];

cout<<"Enter Capacity :";

cin >> capacity;

auto result = fractional\_knapsack(n, weights, values, capacity);

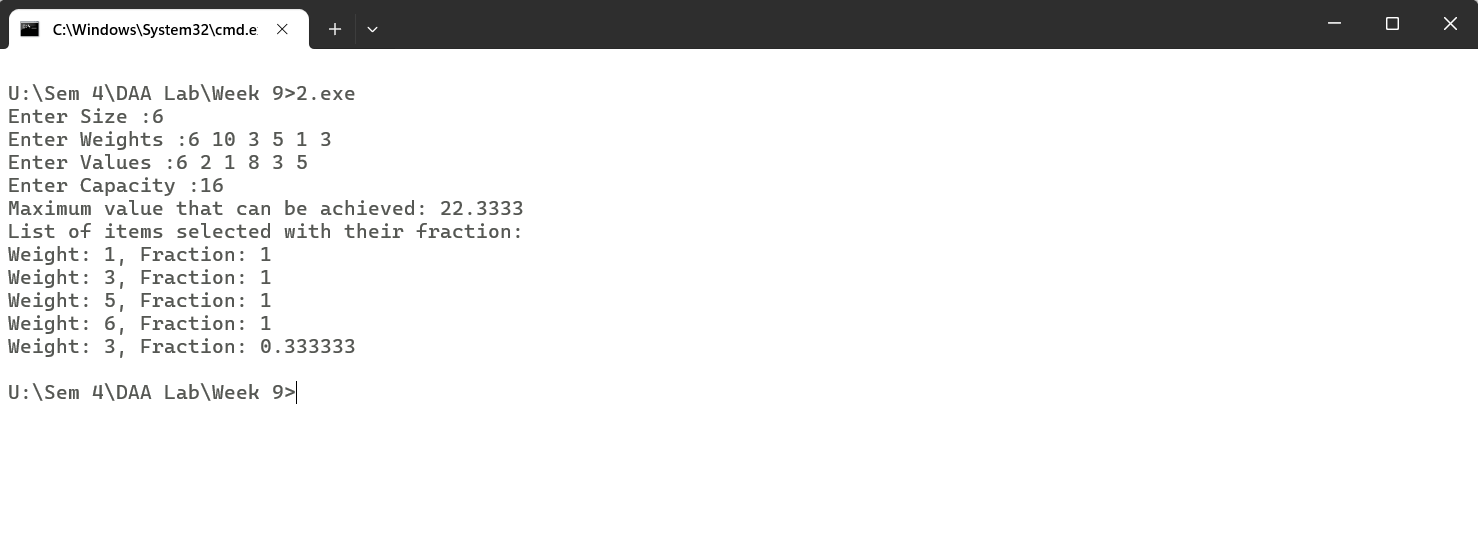
cout << "Maximum value that can be achieved: " << result.first << endl;

cout << "List of items selected with their fraction:" << endl;

for (const auto& item : result.second)

cout << "Weight: " << item.first << ", Fraction: " << item.second << endl;

}



/\*

Q-03 . 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). (Hint: use greedy approach)

\*/

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int main(){

int n;

cout<<"Enter Array Size :";

cin>>n;

vector<int> arr(n);

cout<<"Enter File Size :";

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

cin >> arr[i];

}

sort(arr.begin(),arr.end());

int ans = 0;

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

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

ans += arr[i];

}

cout<<"Minimum Computation is :"<<ans<<endl;

}



/\*

Q-01 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.

\*/

#include<bits/stdc++.h>

using namespace std;

bool comp(pair<pair<int,int>,int> p1,pair<pair<int,int>,int> p2){

return p1.first.second < p2.first.second;

}

void activitySelection(vector<pair<pair<int,int>,int>> arr){

sort(arr.begin(),arr.end(),comp);

vector<int> ans;

ans.push\_back(1);

int finish = arr[0].first.second;

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

if(arr[i].first.first >= finish){

ans.push\_back(i+1);

finish = arr[i].first.second;

}

}

cout<<"No. of non-conflicting Activities :" << ans.size()<<endl;

cout<<"List of Selected Activities :";

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

cout<<ans[i]<<" ";

}

}

int main(){

int n;

cout<<"Enter Total Values :";

cin>>n;

vector<pair<pair<int,int>,int>> arr(n);

cout<<"Enter Starting Time :";

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

cin >> (arr[i].first.first);

}

cout<<"Enter Finishing Time :";

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

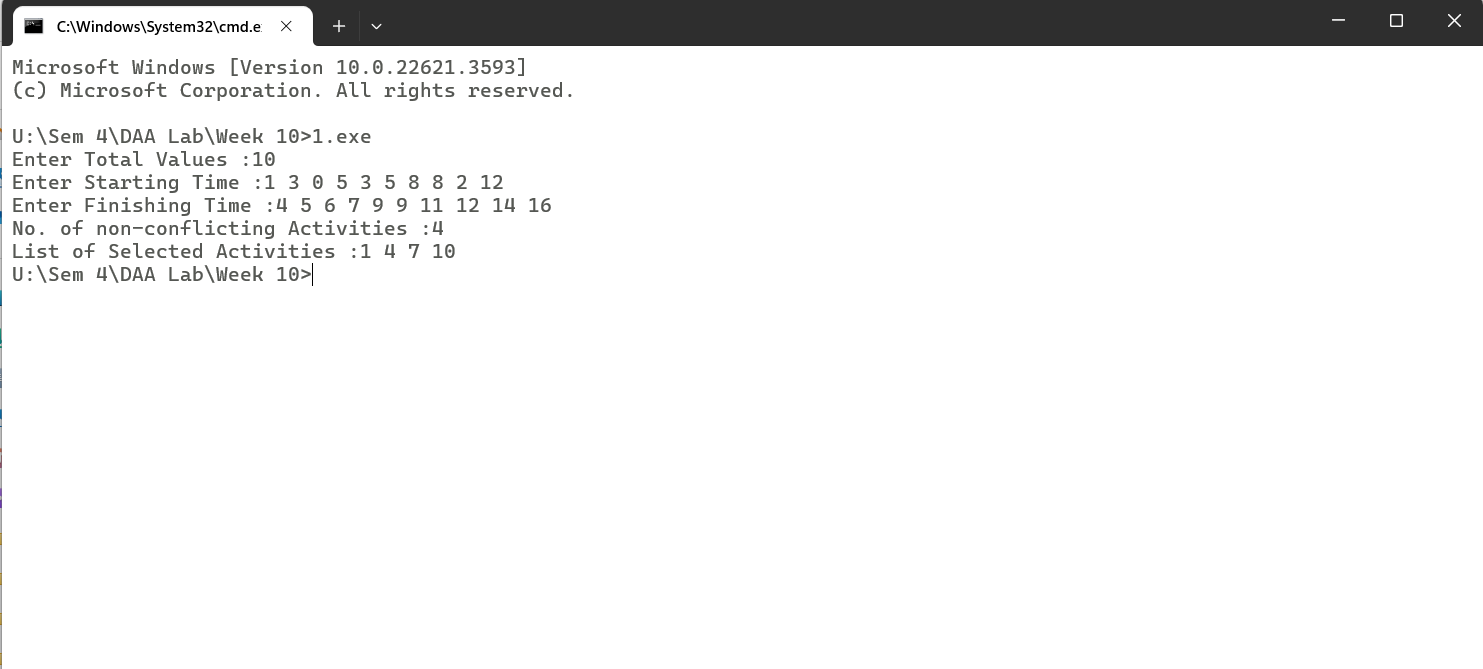
cin>>arr[i].first.second;

arr[i].second = i;

}

activitySelection(arr);

}



/\*

Q-02 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.

\*/

#include<bits/stdc++.h>

using namespace std;

bool comp(pair<pair<int,int>,int> p1,pair<pair<int,int>,int> p2){

if(p1.first.second == p2.first.second)

return p1.first.first < p2.first.first;

return p1.first.second < p2.first.second;

}

void activitySelection(vector<pair<pair<int,int>,int>> arr){

sort(arr.begin(),arr.end(),comp);

vector<int> ans;

ans.push\_back(1);

int time = arr[0].first.first;

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

if(arr[i].first.first + time <= arr[i].first.second){

ans.push\_back(arr[i].second+1);

time += arr[i].first.first;

}

}

cout<<"No. of non-conflicting Activities :" << ans.size()<<endl;

cout<<"List of Selected Activities :";

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

cout<<ans[i]<<" ";

}

}

int main(){

int n;

cout<<"Enter Total Tasks :";

cin>>n;

vector<pair<pair<int,int>,int>> arr(n);

cout<<"Enter Time Taken :";

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

cin >> (arr[i].first.first);

}

cout<<"Enter Deadline :";

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

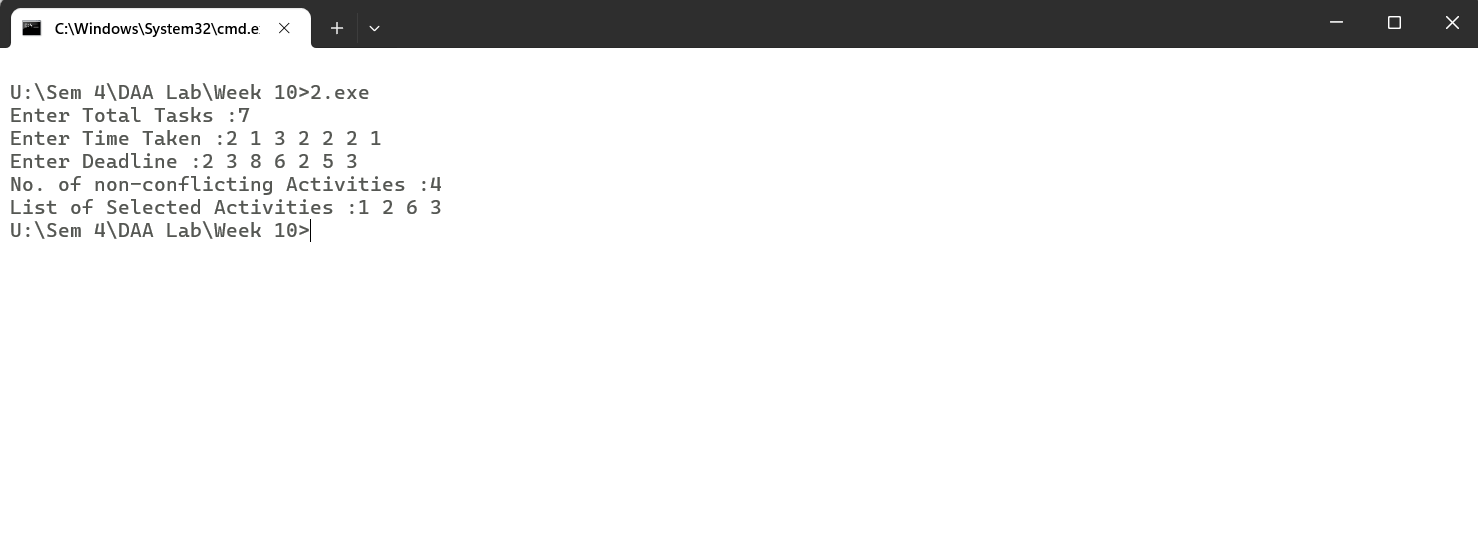
cin>>arr[i].first.second;

arr[i].second = i;

}

activitySelection(arr);

}



/\*

Q-03 . 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.

\*/

#include<iostream>

#include<vector>

using namespace std;

void calculateMajority(vector<int> arr){

int count=1,value=arr[0];

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

if(count == 0)

value = arr[i];

if(arr[i] == value){

count++;

}else{

count--;

}

}

count=0;

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

if(arr[i] == value) count++;

}

if(count > arr.size()/2){

cout<<"Yes"<<endl;

cout<<value;

}else{

cout<<"No"<<endl;

}

}

int main(){

int n;

cout << "Enter Size :";

cin>>n;

vector<int> arr(n);

cout<<"Enter Elements :";

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

cin >> arr[i];

}

calculateMajority(arr);

}



/\*

Q-01 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.

\*/

#include<iostream>

#include<vector>

#include<climits>

using namespace std;

void matrixChainMultiplication(vector<vector<int>> arr){

int n = arr.size();

vector<int> position(n+1);

vector<vector<int>> result(n , vector<int>(n,0));

position[0] = arr[0][0];

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

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

}

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

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

int j = i+d;

int val = INT\_MAX;

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

val = min(val, result[i][k]+result[k+1][j]+(position[i]\*position[k+1]\*position[j+1]));

}

result[i][j] = val;

}

}

cout<<"Minimum No. of Operations :"<<result[0][n-1];

}

int main(){

int n;

cout<<"Enter No. of Matrix Sequences :";

cin>>n;

cout<<"Enter Matrix Sequences :";

vector<vector<int>> arr(n , vector<int>(2));

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

cin>>arr[i][0]>>arr[i][1];

}

matrixChainMultiplication(arr);

return 0;

}



/\*

Q-02 Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N.

\*/

#include<iostream>

#include<vector>

using namespace std;

void findWays(vector<int> coins,int sum){

int n = coins.size();

vector<vector<int> > dp(n + 1, vector<int>(sum + 1, 0));

dp[0][0] = 1;

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

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

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

if ((j - coins[i - 1]) >= 0) {

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

}

}

}

cout<<"Total No. of Ways :"<<dp[n][sum];

}

int main(){

int n;

cout<<"Enter No. of Coins :";

cin>>n;

vector<int> coins(n);

cout<<"Enter Coin Values :";

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

cin>>coins[i];

int sum;

cout<<"Enter Sum :";

cin>>sum;

findWays(coins,sum);

}



/\*

Q-03 Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem.

\*/

#include<iostream>

#include<vector>

using namespace std;

//The given problem is solved using memoization approach

int checkSubset(vector<int> arr,int index,int n,int sum,vector<vector<int>> &dp){

if(sum == 0)

return 1;

if(index == n-1 && sum !=0)

return 0;

if(dp[n][sum] != -1)

return dp[n][sum];

return dp[n][sum] = checkSubset(arr,index+1,n,sum,dp) || checkSubset(arr,index+1,n,sum-arr[index],dp);

}

void findHalf(vector<int> arr,int n){

int sum=0;

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

sum += arr[i];

}

if(sum%2 != 0){

cout<<"NO"<<endl;

return;

}

vector<vector<int>> dp(n+1, vector<int>(sum+1,-1));

if(checkSubset(arr,0,n,sum/2,dp))

cout <<"YES";

else

cout<<"NO";

}

int main(){

int n;

cout<<"Enter Size :";

cin>>n;

vector<int> arr(n);

cout<<"Enter Element :";

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

cin>>arr[i];

}

findHalf(arr,n);

}



/\*

Q-01 Given two sequences, Design an algorithm and implement it using a program to find the lengthof longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

\*/

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

void longestCommonSequence(string str1,string str2){

vector<vector<int>> dp(str1.length()+1, vector<int>(str2.length()+1));

vector<vector<char>> s(str1.length()+1, vector<char>(str2.length()+1, '0'));

for(int i=0;i<str1.length()+1;i++)

dp[i][0] = 0;

for(int i=0;i<str2.length()+1;i++)

dp[0][i] = 0;

for(int i=1;i<str1.length()+1;i++){

for(int j=1;j<str2.length()+1;j++){

if(str1[i-1] == str2[j-1]){

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

s[i][j] = 'D';

}else if(dp[i-1][j] >= dp[i][j-1]){

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

s[i][j] = 'U';

}else{

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

s[i][j] = 'L';

}

}

}

string ans;

int i=str1.length(),j=str2.length();

while(s[i][j] != '0'){

if(s[i][j] == 'D'){

ans += str1[i-1];

i--;

j--;

}else if(s[i][j] == 'U'){

i--;

}else{

j--;

}

}

reverse(ans.begin(),ans.end());

cout<<"Longest Common Subsequence :"<<ans<<endl<<"Length :"<<ans.length();

}

int main(){

string str1,str2;

cout<<"Enter Two Strings :";

cin>>str1>>str2;

longestCommonSequence(str1,str2);

}



/\*

Q-02 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).

\*/

#include <iostream>

#include <vector>

#include<algorithm>

using namespace std;

void knapsack(int n,vector<int>& weights,vector<int>& values, int capacity) {

vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));

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

for (int w = 0; w <= capacity; ++w) {

if (weights[i-1] <= w) {

dp[i][w] = max(dp[i-1][w], dp[i-1][w-weights[i-1]] + values[i-1]);

} else {

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

}

}

}

int w = capacity;

vector<pair<int, int>> selected\_items;

for (int i = n; i > 0; --i) {

if (dp[i][w] != dp[i-1][w]) {

selected\_items.push\_back({weights[i-1], values[i-1]});

w -= weights[i-1];

}

}

reverse(selected\_items.begin(), selected\_items.end());

int max\_value = dp[n][capacity];

cout << "Value = "<<max\_value<<endl;

cout<<"Weights Selected : ";

for(int i=0;i<selected\_items.size();i++){

cout<<selected\_items[i].first << " ";

}

cout<<endl;

cout<<"Values of Weights Selected : ";

for(int i=0;i<selected\_items.size();i++){

cout<<selected\_items[i].second << " ";

}

cout<<endl;

}

int main() {

int n;

cout<<"Enter Size :";

cin >> n;

vector<int> weights(n);

vector<int> values(n);

cout<<"Enter Weights :";

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

cin >> weights[i];

}

cout<<"Enter Values :";

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

cin >> values[i];

}

int capacity;

cout<<"Enter Capacity :";

cin >> capacity;

knapsack(n, weights, values, capacity);

return 0;

}



/\*

Q-03 Given a string of characters, design an algorithm and implement it using a program to print all

possible permutations of the string in lexicographic order.

\*/

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

void getPermutation(string str, int curr, int size,vector<string> &ans){

if(curr == size){

ans.push\_back(str);

return;

}

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

swap(str[curr], str[i]);

getPermutation(str, curr+1, size, ans);

swap(str[curr], str[i]);

}

}

int main(){

string str;

cout<<"Enter String :";

cin>>str;

vector<string> ans;

getPermutation(str, 0, str.length()-1,ans);

sort(ans.begin(),ans.end());

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

cout<<ans[i]<<endl;

}

}



/\*

Q-01 Given an array of characters, you have to find distinct characters from this array. Design an algorithm and implement it using a program to solve this problem using hashing. (Time Complexity = O(n))

\*/

#include<iostream>

#include<map>

using namespace std;

void countFrequency(const string &str) {

map<char, int> frequency;

for (char c : str) {

if (c != ' ') // Ignoring spaces

frequency[c]++;

}

for (auto pair : frequency) {

cout << pair.first << " - " << pair.second << endl;

}

}

int main(){

int n;

cout<<"Enter Size :";

cin>>n;

string str;

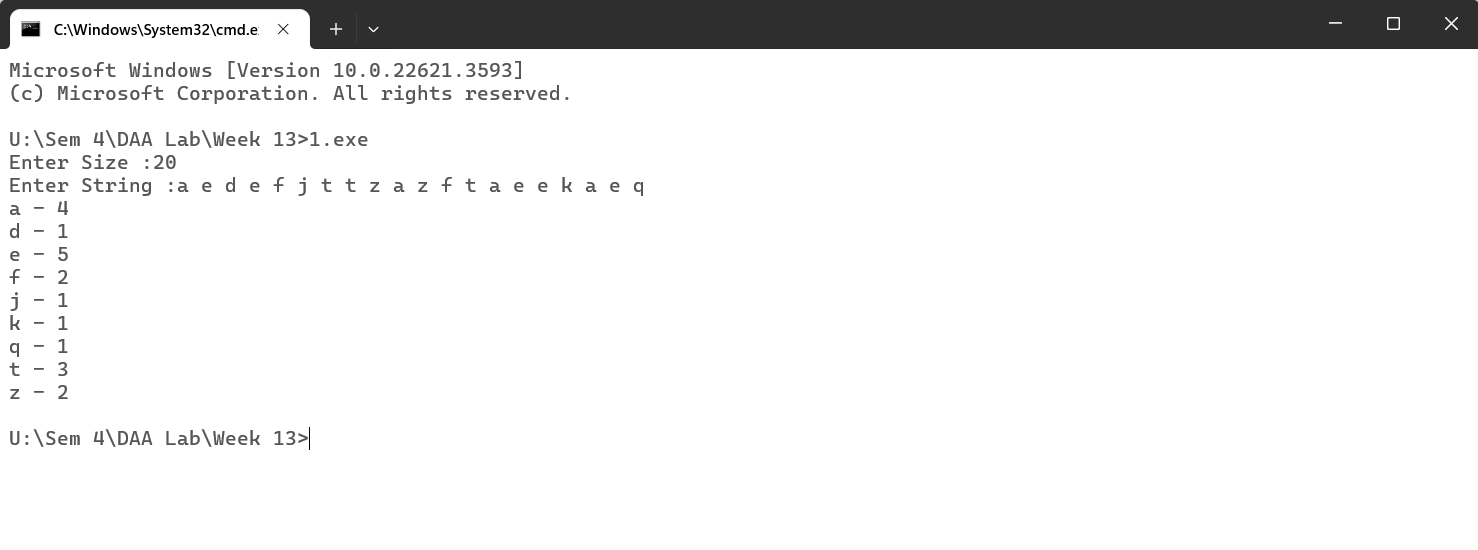
cin.ignore();

cout<<"Enter String :";

getline(cin,str);

countFrequency(str);

}



/\*

Q-02 Given an array of integers of size n, design an algorithm and write a program to check whether this array contains duplicate within a small window of size k < n.

\*/

#include<iostream>

#include<vector>

#include<unordered\_map>

using namespace std;

void checkDuplicates(vector<int> arr,int k){

unordered\_map<int,int> frequency;

int j;

for(j=0;j<k;j++){

frequency[arr[j]]++;

if(frequency[arr[j]] > 1){

cout << "Duplicates present in window "<<k<<endl;

return;

}

}

int i=0;

for(;j<arr.size();j++){

frequency[arr[i]]--;

i++;

frequency[arr[j]]++;

if(frequency[arr[j]] > 1){

cout << "Duplicates present in window "<<k<<endl;

return;

}

}

cout << "Duplicates not present in window "<<k<<endl;

}

int main(){

int t;

cout<<"Enter Test-Case :";

cin>>t;

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

int n;

cout<<"Enter Size :";

cin>>n;

cout<<"Enter Values :";

vector<int> arr(n);

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

cin>>arr[j];

}

cout<<"Enter Window Size :";

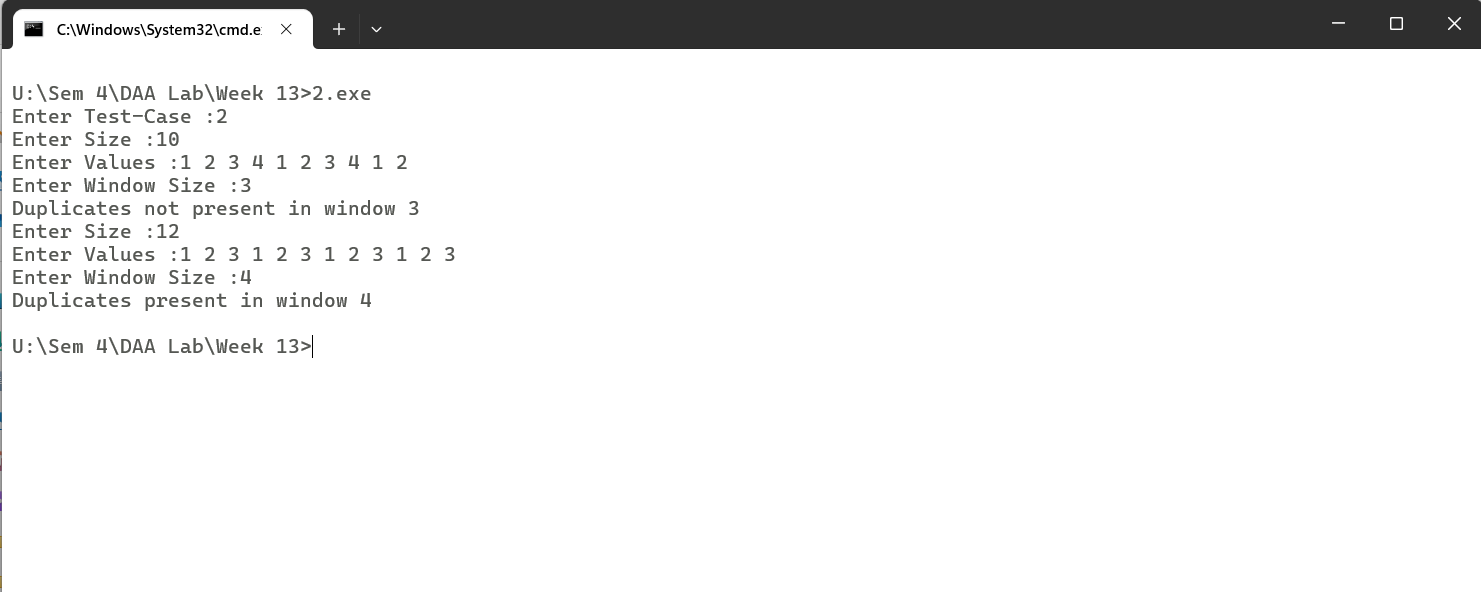
int k;

cin>>k;

checkDuplicates(arr,k);

}

}



/\*

Q-03 Given an array of nonnegative integers, Design an algorithm and implement it using a program to find two pairs (a,b) and (c,d) such that a\*b = c\*d, where a, b, c and d are distinct elements of array.

\*/

#include<iostream>

#include<unordered\_map>

#include<vector>

using namespace std;

void findPair(vector<int> arr){

unordered\_map<int, pair<int,int>> map;

for(int i=0;i<arr.size()-1;i++){

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

if(map.find(arr[i]\*arr[j]) != map.end()){

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

cout<<map[arr[i]\*arr[j]].first<<" "<<map[arr[i]\*arr[j]].second<<endl;

return;

}

map[arr[i]\*arr[j]] = {arr[i],arr[j]};

}

}

}

int main(){

int n;

cout<<"Enter Size :";

cin>>n;

vector<int> arr(n);

cout<<"Enter Array Elements :";

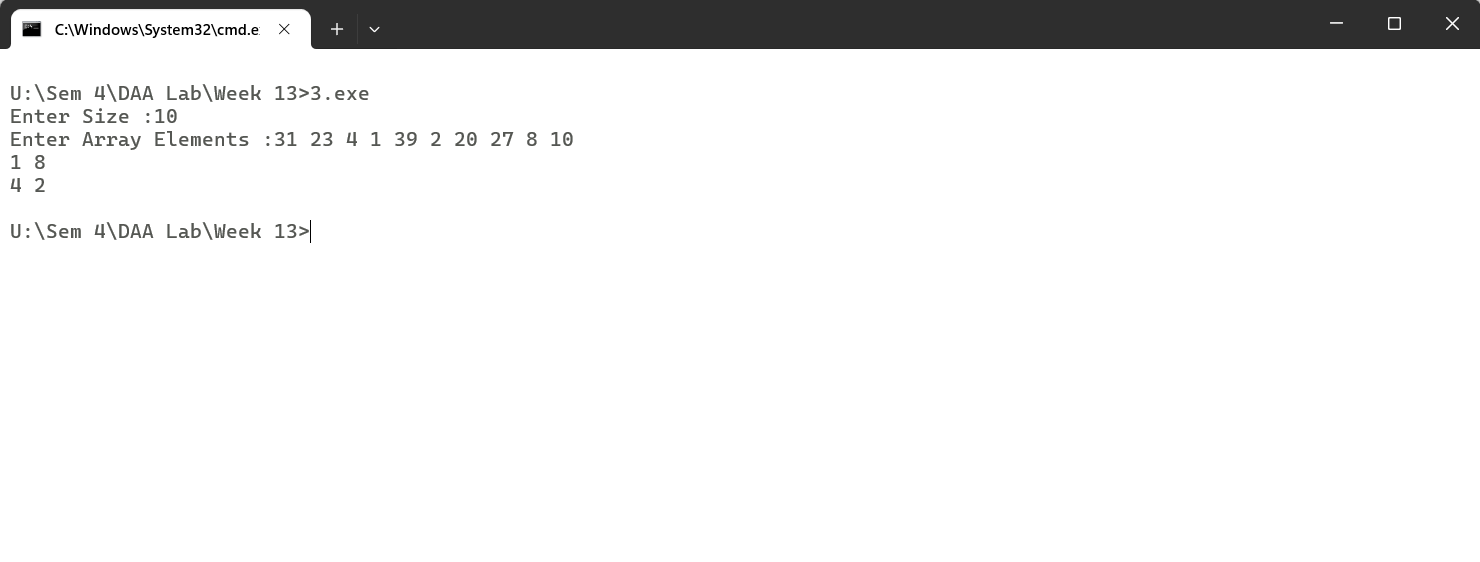
for(int i=0;i<n;i++){

cin>>arr[i];

}

findPair(arr);

}



/\*

Q-01 Given a number n, write an algorithm and a program to find nth ugly number. Ugly numbers are those numbers whose only prime factors are 2, 3 or 5. The sequence 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24,..... is sequence of ugly numbers.

\*/

#include<iostream>

#include<vector>

using namespace std;

int nthUglyNumber(int n) {

if(n == 1) return n;

int t2 = 0, t3 = 0, t5 = 0;

vector<int> k(n);

k[0] = 1;

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

k[i] = min(k[t2]\*2,min(k[t3]\*3,k[t5]\*5));

if(k[i] == k[t2]\*2) t2++;

if(k[i] == k[t3]\*3) t3++;

if(k[i] == k[t5]\*5) t5++;

}

return k[n-1];

}

int main(){

int t;

cout<<"Enter Test-Cases :";

cin >> t;

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

int n;

cout<<"Enter n :";

cin>>n;

cout<<nthUglyNumber(n)<<endl;

}

}



/\*

Q-02 Given a directed graph, write an algorithm and a program to find mother vertex in a graph. A mother vertex is a vertex v such that there exists a path from v to all other vertices of the graph.

\*/

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

class Graph {

int V;

vector<vector<int>> adj;

void DFSUtil(int v, vector<bool>& visited) {

visited[v] = true;

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

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

DFSUtil(i, visited);

}

}

public:

Graph(int V) {

this->V = V;

adj.resize(V, vector<int>(V, 0));

}

void addEdge(int v, int w) {

adj[v][w] = 1;

}

int findMotherVertex() {

vector<bool> visited(V, false);

int last\_v = 0;

// Step 1: Perform DFS traversal from each vertex to find the last visited vertex

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

if (!visited[i]) {

DFSUtil(i, visited);

last\_v = i;

}

}

// Step 2: Check if last visited vertex can reach all other vertices

fill(visited.begin(), visited.end(), false);

DFSUtil(last\_v, visited);

if (all\_of(visited.begin(), visited.end(), [](bool val) { return val; }))

return last\_v;

else

return -1;

}

};

int main() {

int V;

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

cin >> V;

Graph g(V);

// Input adjacency matrix

cout << "Enter the adjacency matrix:" << endl;

vector<vector<int>> adjMatrix(V, vector<int>(V, 0));

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

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

cin >> adjMatrix[i][j];

if (adjMatrix[i][j])

g.addEdge(i, j);

}

}

cout << "Mother vertex is: " << g.findMotherVertex() << endl;

return 0;

}

