**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**Analysis and Design of Algorithms**

***Submitted by***

**POORVIKA S K(1BM22CS412)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**June-2023 to September-2023**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by **POORVIKA S K (1BM22CS412),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge: Basavaraju Jakkalli Dr. Jyothi S Nayak

Associative Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Lab Program No.** | **Program Details** | **Page No.** |
| 1 | Write program to do the following:  a. Print all the nodes reachable from a given starting node in a digraph using BFS method.  b. Check whether a given graph is connected or not using DFS method. | 5-11 |
| 2 | Write program to obtain the Topological ordering of vertices in a given digraph. | 12-14 |
| 3 | Implement Johnson Trotter algorithm to generate permutations. | 15-20 |
| 4 | Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. | 21-24 |
| 5 | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 25-29 |
| 6 | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | 30-35 |
| 7 | Implement 0/1 Knapsack problem using dynamic programming. | 36-40 |
| 8 | Implement All Pair Shortest paths problem using Floyd’s algorithm. | 41-44 |
| 9 | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm. | 45-51 |
| 10 | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 52-55 |
| 11 | Implement “N-Queens Problem” using Backtracking. | 56-59 |

**Course Outcome**

|  |  |
| --- | --- |
| CO1 | Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations. |
| CO2 | Apply various design techniques for the given problem. |
| CO3 | Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain  problems are NP-Complete |
| CO4 | Design efficient algorithms and conduct practical experiments to solve problems. |

**LAB PROGRAM-01**

**Write program to do the following:**

**a. Print all the nodes reachable from a given starting node in a digraph**

**using BFS method.**

**b. Check whether a given graph is connected or not using DFS method.**

**a. BFS:-**

#include<stdio.h>

#include<conio.h>

int a[10][10],n;

void bfs(int);

void main()

{

int i,j,src;

printf("\n enter the no of nodes:\t");

scanf("%d",&n);

printf("\n enter the adjacency matrix:\n");

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

{

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

{

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

}

}

printf("\n enter the source node:\n");

scanf("%d",&src);

bfs(src);

}

void bfs(int src)

{

int q[10],f=0,r=-1,vis[10],i,j;

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

{

vis[j]=0;

}

vis[src]=1;

r=r+1;

q[r]=src;

while(f<+r)

{

i=q[f];

f=f+1;

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

{

if(a[i][j]==1&&vis[j]!=1)

{

vis[j]=1;

r=r+1;

q[r]=j;

}

}

}

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

{

if(vis[j]!=1)

printf("\n node %d is not reachable\n",j);

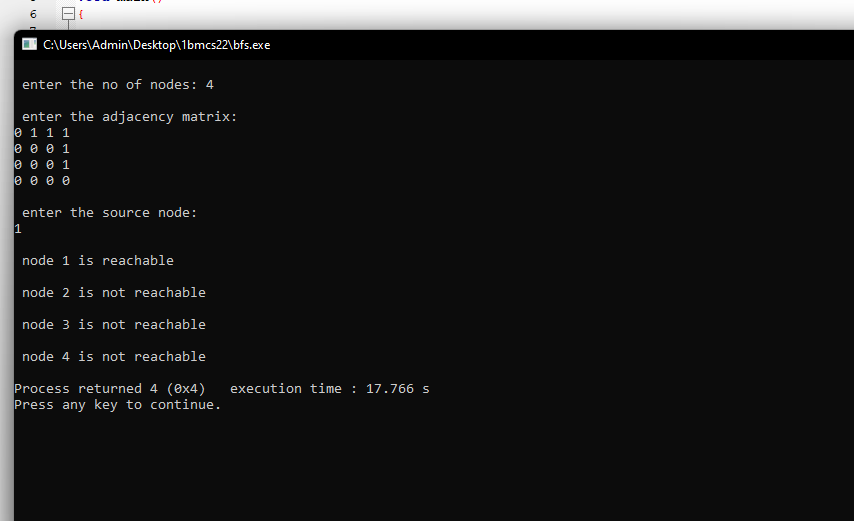
else

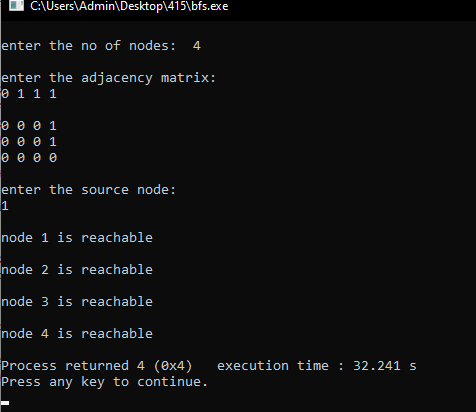
printf("\n node %d is reachable\n",j);

}

}

**OUTPUT:-**





**b. DFS:-**

#include<stdio.h>

#include<conio.h>

int a[10][10],n,vis[10];

int dfs(int);

void main()

{

int i,j,src,ans;

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

{

vis[j]=0;

}

printf("\nenter the no of nodes:\t");

scanf("%d",&n);

printf("\nenter the adjacency matrix:\n");

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

{

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

{

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

}

}

ans=dfs(src);

if(ans==1)

{

printf("\ngraph is connected\n");

}

else

{

printf("\ngraph is not connected\n");

}

}

int dfs(int src)

{

int j;

vis[src]=1;

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

{

if(a[src][j]==1 && vis[j]!=1)

dfs(j);

}

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

{

if(vis[j]!=1)

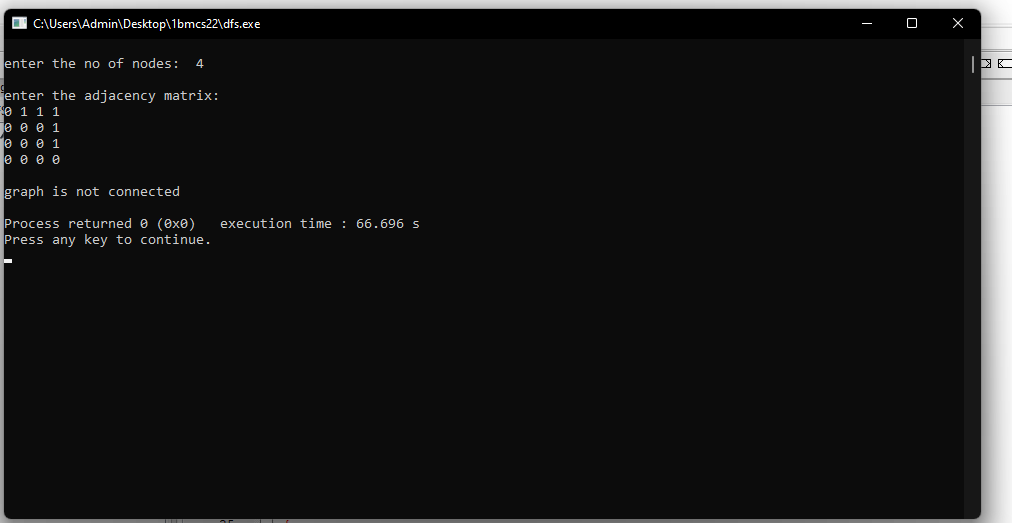
return 0;

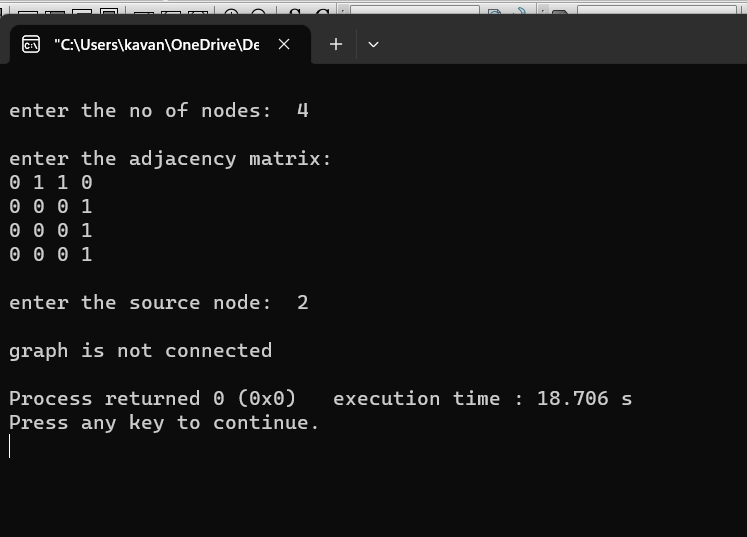
}

return 1;

}

**OUTPUT:-**





**LAB PROGRAM-02**

**Write program to obtain the Topological ordering of vertices in a given digraph.**

#include<stdio.h>

#include<conio.h>

void source\_removal(int n,int a[10][10])

{

int i,j,k,u,v,top,s[10],t[10],indeg[10],sum;

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

{

sum=0;

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

sum=sum+a[j][i];

indeg[i]=sum;

}

top=-1;

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

{

if(indeg[i]==0)

s[++top]=i;

}

k=0;

while(top!=-1)

{

u=s[top--];

t[k++]=u;

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

{

if(a[u][v]==1)

{

indeg[v]=indeg[v]-1;

if(indeg[v]==0)

s[++top]=v;

}

}

}

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

{

printf("%d\n",t[i]);

}

}

void main()

{

int i,j,a[10][10],n;

printf("Enter number of nodes:\n");

scanf("%d",&n);

printf("Enter the adjacency matrix\n");

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

{

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

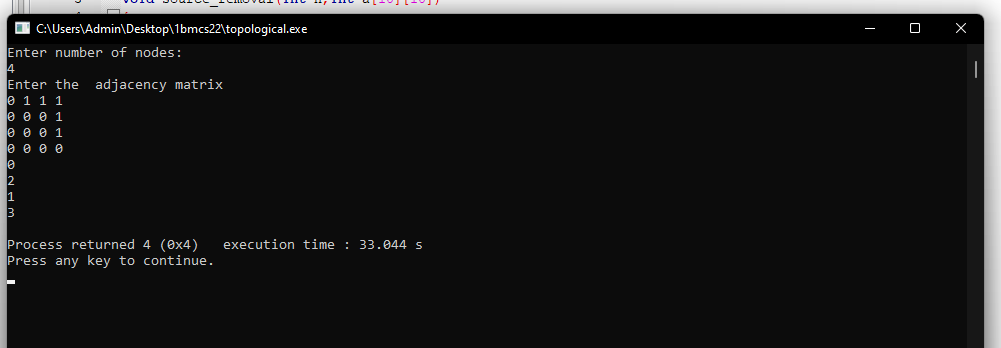
scanf("%d",&a[i][j]);

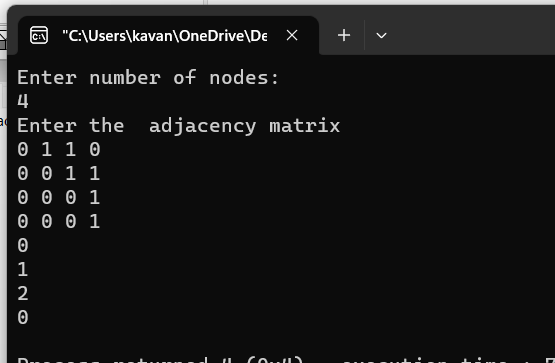
}

source\_removal(n,a);

}

**OUTPUT:-**





**LAB PROGRAM-03**

**Implement Johnson Trotter algorithm to generate permutations.**

#include <stdio.h>

#include <stdlib.h>

int swap(int \*a,int \*b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int search(int arr[],int num,int mobile)

{

int g;

for(g=0;g<num;g++)

{

if(arr[g] == mobile)

{

return g;

}

}

return -1;

}

int find\_Moblie(int arr[],int d[],int num)

{

int mobile = 0;

int i;

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

{

if((d[arr[i]-1] == 0) && i != 0)

{

if(arr[i]>arr[i-1] && arr[i]>mobile)

{

mobile = arr[i];

}

}

else if((d[arr[i]-1] == 1) & i != num-1)

{

if(arr[i]>arr[i+1] && arr[i]>mobile)

{

mobile = arr[i];

}

}

}

if(mobile == 0)

return 0;

else

return mobile;

}

void permutations(int arr[],int d[],int num)

{

int i;

int mobile = find\_Moblie(arr,d,num);

int pos = search(arr,num,mobile);

if(d[arr[pos]-1]==0)

swap(&arr[pos],&arr[pos-1]);

else

swap(&arr[pos],&arr[pos+1]);

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

{

if(arr[i] > mobile)

{

if(d[arr[i]-1]==0)

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

else

d[arr[i]-1] = 0;

}

}

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

{

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

}

}

int factorial(int k)

{

int f = 1;

int i = 0;

for(i=1;i<k+1;i++)

{

f = f\*i;

}

return f;

}

int main()

{

int num = 0;

int i;

int j;

int z = 0;

printf("Johnson trotter algorithm to find all permutations of given numbers \n");

printf("Enter the number\n");

scanf("%d",&num);

int arr[num],d[num];

z = factorial(num);

printf("The total permutations are %d",z);

printf("\nAll possible permutations are: \n");

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

{

d[i] = 0;

arr[i] = i+1;

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

}

printf("\n");

for(j=1;j<z;j++)

{

permutations(arr,d,num);

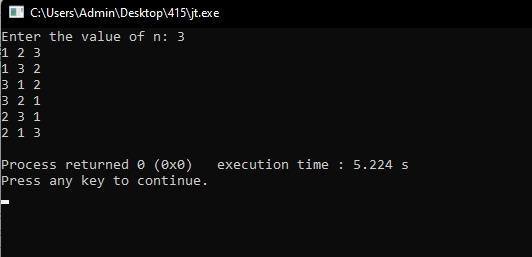
printf("\n");

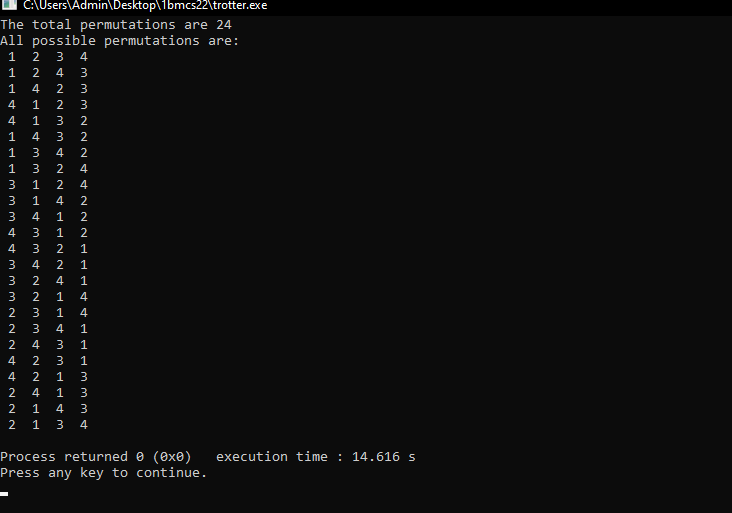
}

return 0;

}

**OUTPUT:-**





**LAB PROGRAM-04**

**Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.**

#include<stdio.h>

void mergesort(int a[],int i,int j);

void merge(int a[],int i1,int j1,int i2,int j2);

int main()

{

int a[30],n,i;

printf("Enter no of elements:");

scanf("%d",&n);

printf("Enter array elements:");

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

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

mergesort(a,0,n-1);

printf("\nSorted array is :");

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

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

return 0;

}

void mergesort(int a[],int i,int j)

{

int mid;

if(i<j)

{

mid=(i+j)/2;

mergesort(a,i,mid);

mergesort(a,mid+1,j);

merge(a,i,mid,mid+1,j);

}

}

void merge(int a[],int i1,int j1,int i2,int j2)

{

int temp[50];

int i,j,k;

i=i1;

j=i2;

k=0;

while(i<=j1 && j<=j2)

{

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

temp[k++]=a[i++];

else

temp[k++]=a[j++];

}

while(i<=j1)

temp[k++]=a[i++];

while(j<=j2)

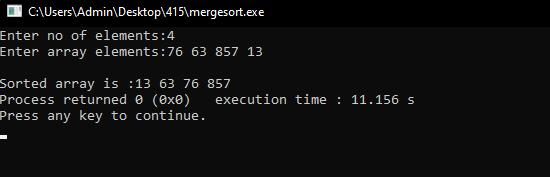
temp[k++]=a[j++];

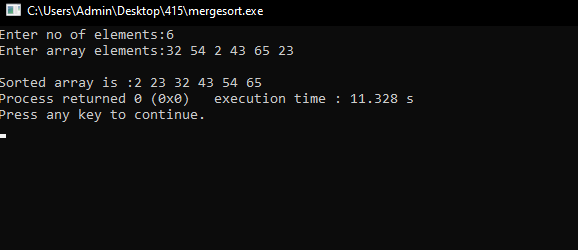
for(i=i1,j=0;i<=j2;i++,j++)

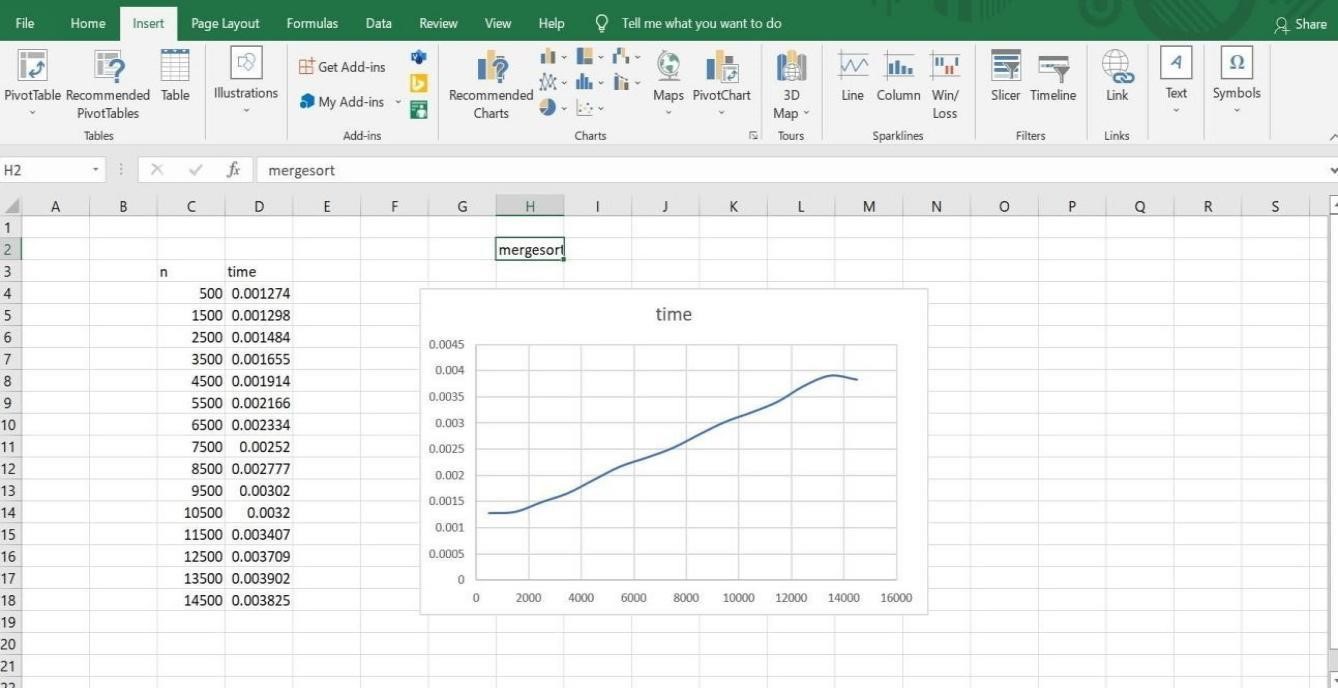
a[i]=temp[j];

}

**OUTPUT:-**







**LAB PROGRAM-05**

**Sort a given set of N integer elements using Quick Sort technique and compute its time taken.**

#include<stdio.h>

#include<conio.h>

#include<time.h>

#include<stdlib.h>

int partition(int a[],int low,int high)

{

int key,i,j,temp;

key=a[low];

i=low+1;

j=high;

while(1)

{

while(i<high && key>=a[i])

i++;

while(key<a[j])

j--;

if(i<j)

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

else

{

temp=a[low];

a[low]=a[j];

a[j]=temp;

return j;

}

}

}

void quicksort(int a[],int low,int high)

{

int j;

if(low<high)

{

j=partition(a,low,high);

quicksort(a,low,j-1);

quicksort(a,j+1,high);

}

}

void main()

{

int a[10000],n,t,i;

clock\_t end,start;

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

scanf("%d",&n);

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

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

{

a[i]=rand()%1000;

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

}

start=clock();

for(int j=0;j<5000000;j++)

t=900/900;

quicksort(a,0,n-1);

end=clock();

printf("Sorted array:\n");

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

{

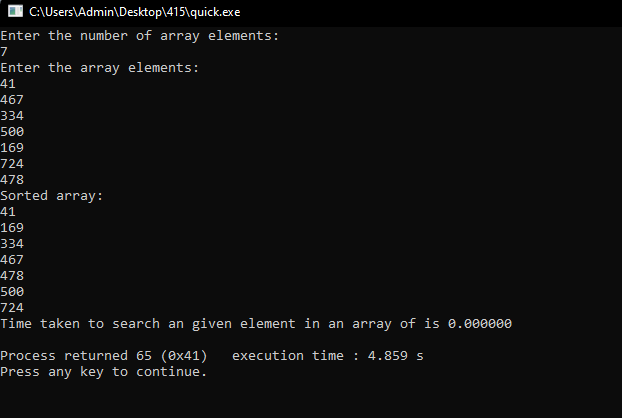
printf("%d\n",a[i]);

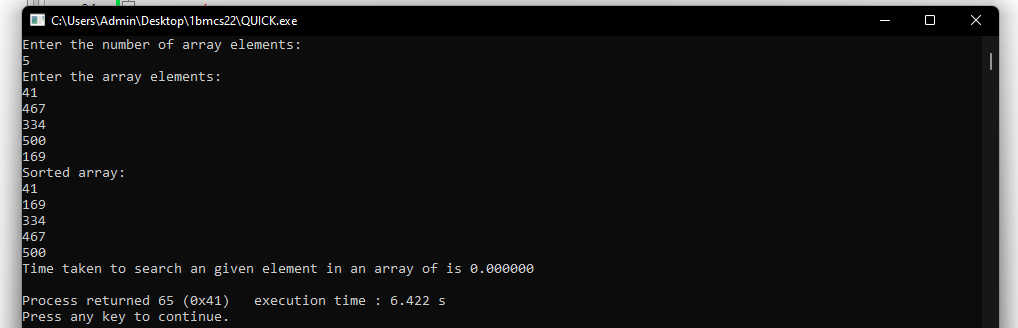
}

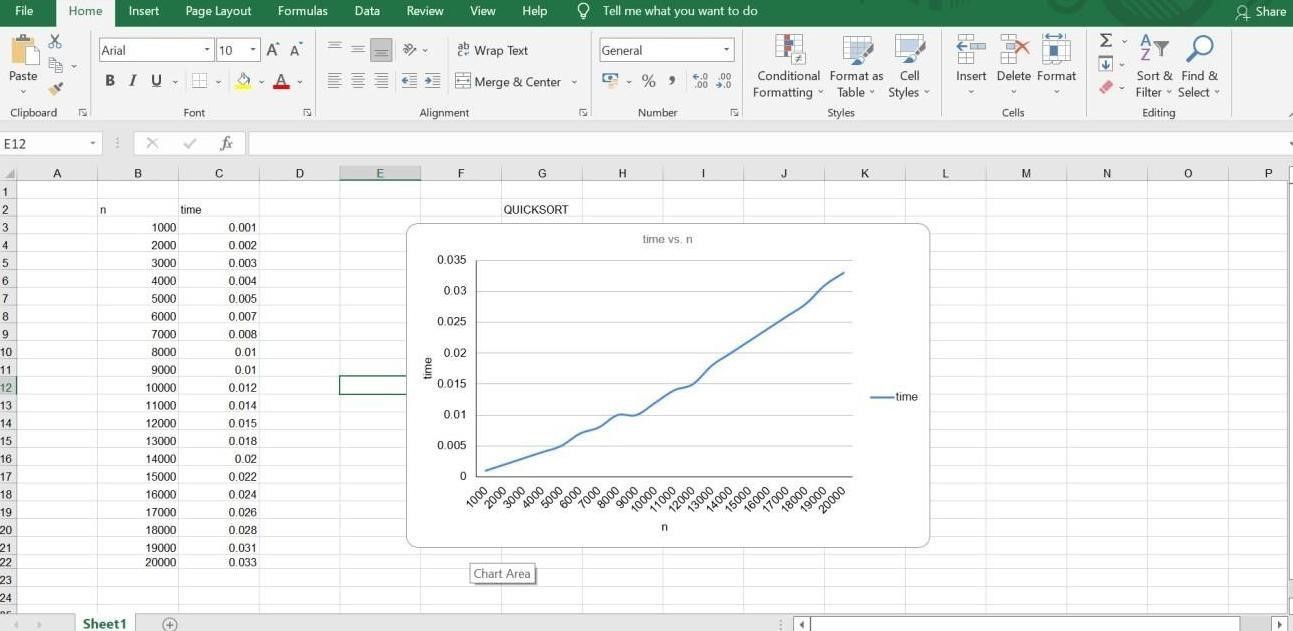
printf("Time taken to search an given element in an array of is %f\n",(((double)(end-start))/CLOCKS\_PER\_SEC));

}

**OUTPUT:-**







**LAB PROGRAM-06**

**Sort a given set of N integer elements using Heap Sort technique and compute its time taken.**

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <math.h>

void swap(int \*,int \*);

void heapify(int [],int,int);

void heapSort(int[], int);

int main()

{

int a[15000],n,i,j,ch,temp;

clock\_t start,end;

while(1)

{

printf("\n 1: For manual entry of N values and array elements:");

printf("\n 2: To display time taken for sorting number of elements N in the range 500 to 14500:");

printf("\n 3: To exit");

printf("\n Enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\n Enter the number of elements:");

scanf("%d",&n);

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

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

{

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

}

start=clock();

heapSort(a, n);

end=clock();

printf("\n Sorted array is:");

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

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

}

printf("\n Time taken to sort %d numbers is %f secs",n,((double)(end-start)/CLOCKS\_PER\_SEC));

break;

case 2:

n=500;

while(n<=14500){

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

a[i]=n-i;

}

start=clock();

heapSort(a, n);

for(j=0;j<50000000;j++){

temp=38/600;

}

end=clock();

printf("\n Time taken to sort %d numbers is %f secs",n,((double)(end-start)/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

case 3: exit(0);

}

}

}

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

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

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

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

largest = left;

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

largest = right;

if (largest != i)

{

swap(&arr[i], &arr[largest]);

heapify(arr, n, largest);

}

}

void heapSort(int arr[], int n)

{

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

heapify(arr, n, i);

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

{

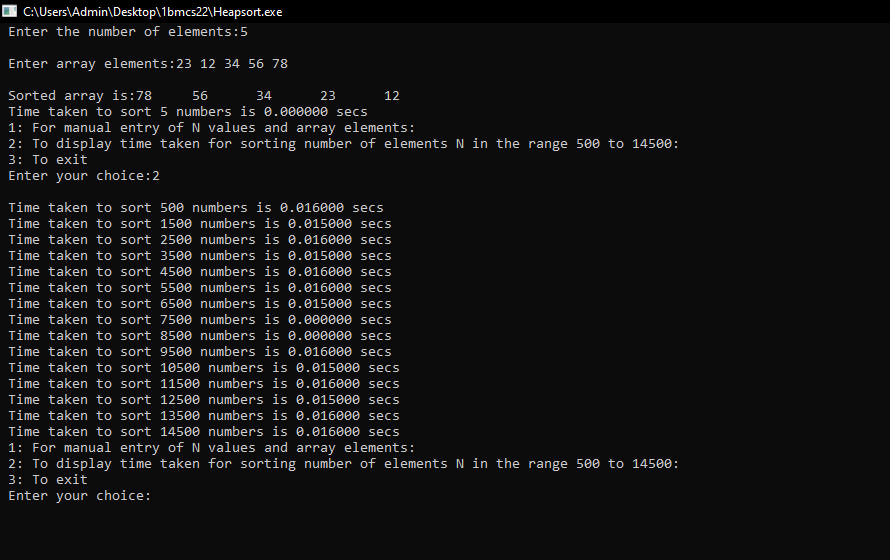
swap(&arr[0], &arr[i]);

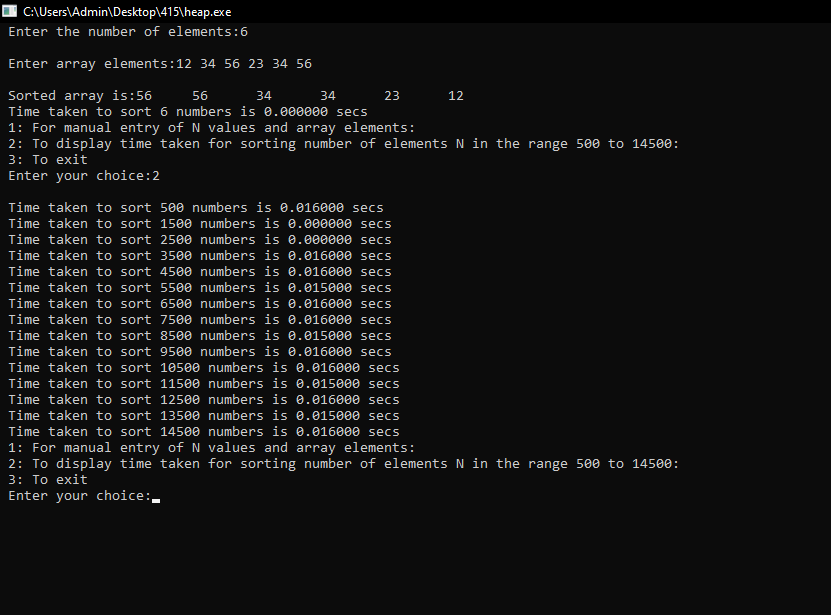
heapify(arr, i, 0);

}

}

**OUTPUT:-**





**LAB PROGRAM-07**

**Implement 0/1 Knapsack problem using dynamic programming.**

#include<stdio.h>

void knapsack();

int max(int,int);

int i,j,n,m,p[10],w[10],v[10][10];

void main()

{

printf("\n enter the no. of items:\t");

scanf("%d",&n);

printf("\n enter the weight of the each item:\n ");

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

{

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

}

printf("\n enter the profit of each item:\n ");

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

{

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

}

printf("\n enter the capacity:\t ");

scanf("%d",&m);

knapsack();

}

void knapsack()

{

int x[10];

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

{

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

{

if(i==0||j==0)

{

v[i][j]=0;

}

else if(j-w[i]<0)

{

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

}

else

{

v[i][j]=max(v[i-1][j],v[i-1][j-w[i]]+p[i]);

}

}

}

printf("\n the output is:\n");

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

{

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

{

printf("%d\t",v[i][j]);

}

printf("\n\n");

}

printf("\nthe optimal solution is %d",v[n][m]);

printf("\nthe solution vector is:\n");

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

{

if(v[i][m]!=v[i-1][m])

{

x[i]=1;

m=m-w[i];

}

else

{

x[i]=0;

}

}

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

{

printf("%d\t",x[i]);

}

}

int max(int x,int y)

{

if(x>y)

{

return x;

}

else

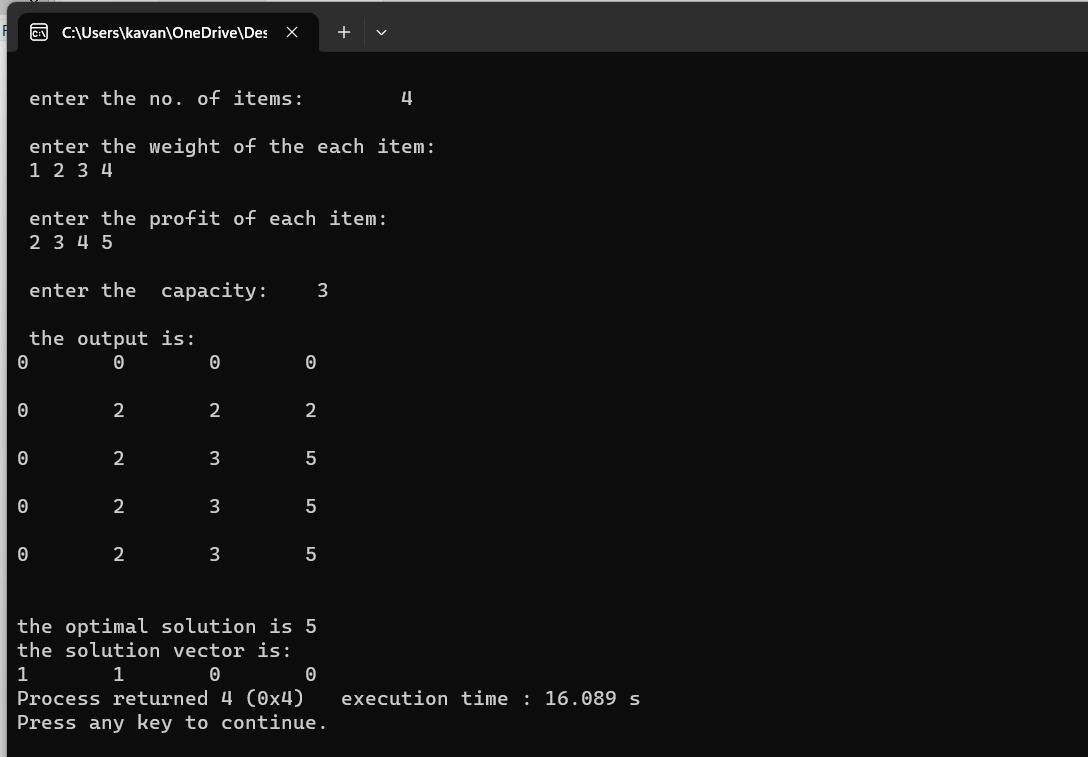
{

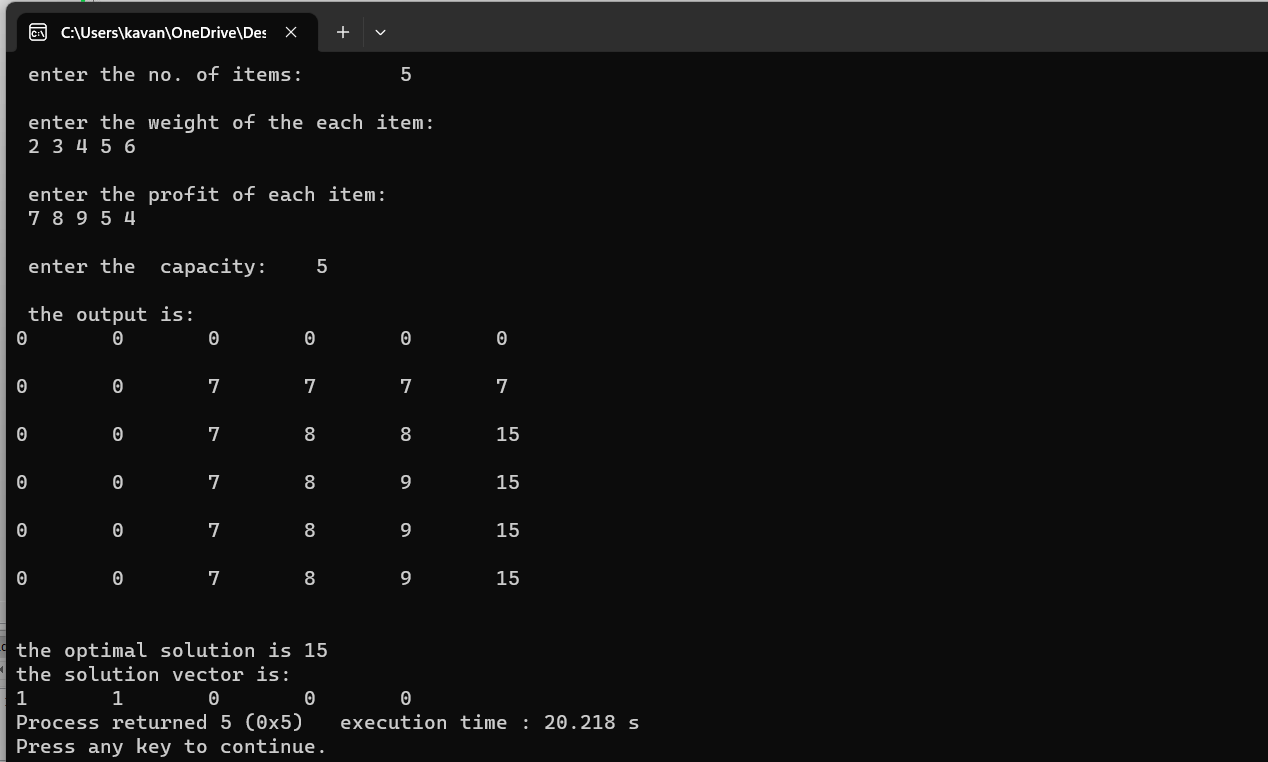
return y;

}

}

**OUTPUT:-**





**LAB PROGRAM-08**

**Implement All Pair Shortest paths problem using Floyd’s algorithm**.

#include<stdio.h>

int a[10][10],n;

void floyds();

int min(int,int);

void main()

{

int i,j;

printf("\n enter the no. of vertices:\t");

scanf("%d",&n);

printf("\n enter the cost matrix:\n");

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

{

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

{

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

}

}

floyds();

}

void floyds()

{

int i,j,k;

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

{

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

{

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

{

a[i][j]=min(a[i][j],a[i][k]+a[k][j]);

}

}

}

printf("\n all pair shortest path matrix is:\n");

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

{

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

{

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

}

printf("\n\n");

}

}

int min(int x,int y)

{

if(x<y)

{

return x;

}

else

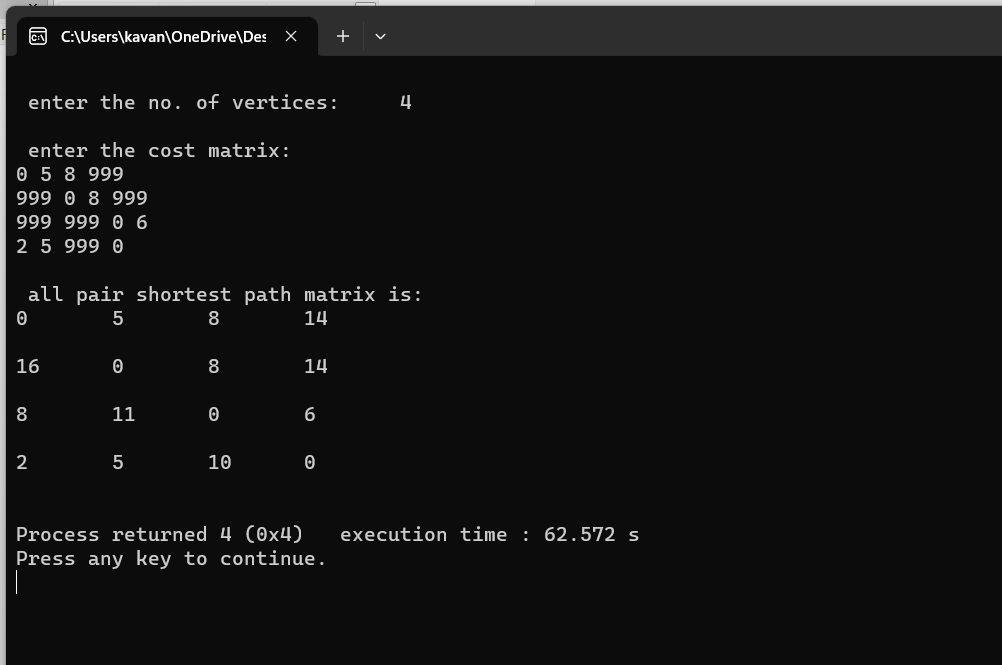
{

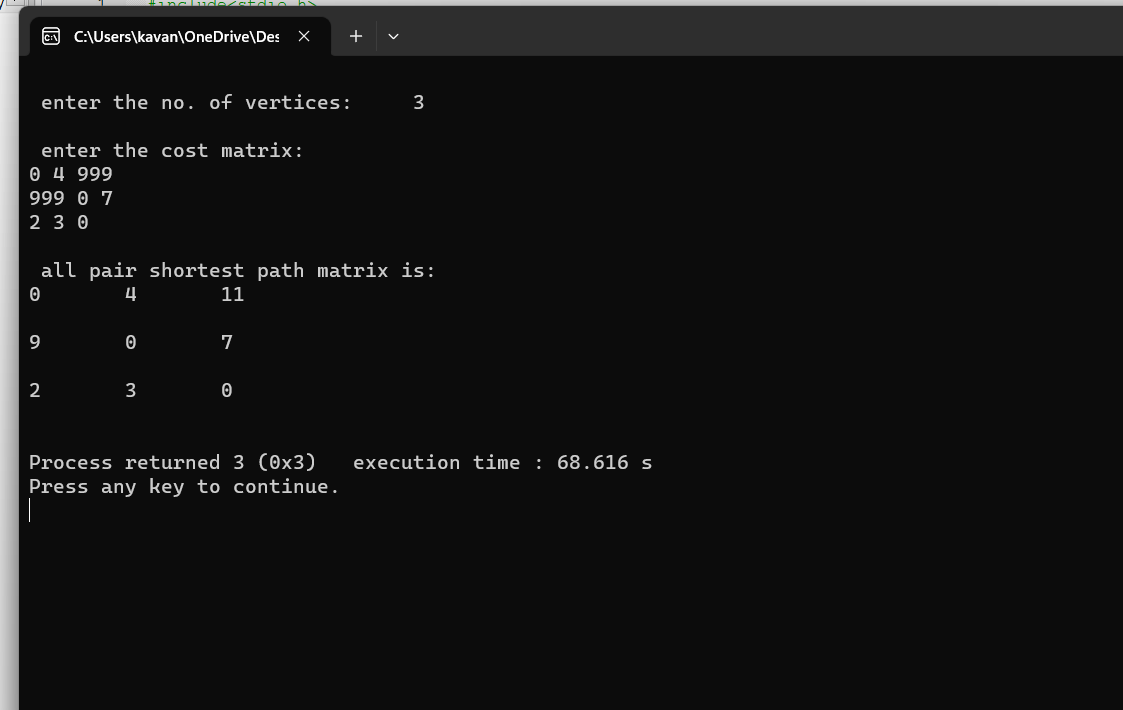
return y;

}

}

**OUTPUT:-**





**LAB PROGRAM-09**

**Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm.**

**PRIM’S:-**

#include<stdio.h>

int main()

{

int cost[10][10],visited[10]={0},i,j,n,no\_e=1,min,a,b,min\_cost=0;

printf("Enter number of nodes ");

scanf("%d",&n);

printf("Enter cost in form of adjacency matrix\n");

//input graph

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

{

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

{

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

// cost is 0 then initialize it by maximum value

if(cost[i][j]==0)

cost[i][j]=1000;

}

}

// logic for finding minimum cost spanning tree

visited[1]=1; // visited first node

while(no\_e<n)

{

min=1000;

// in each cycle find minimum cost

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

{

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

{

if(cost[i][j]<min)

{

if(visited[i]!=0)

{

min=cost[i][j];

a=i;

b=j;

}

}

}

}

//if node is not visited

if(visited[b]==0)

{

printf("\n%d to %d cost=%d",a,b,min);

min\_cost=min\_cost+min;

no\_e++;

}

visited[b]=1;

// initialize with maximum value you can also use any other value

cost[a][b]=cost[b][a]=1000;

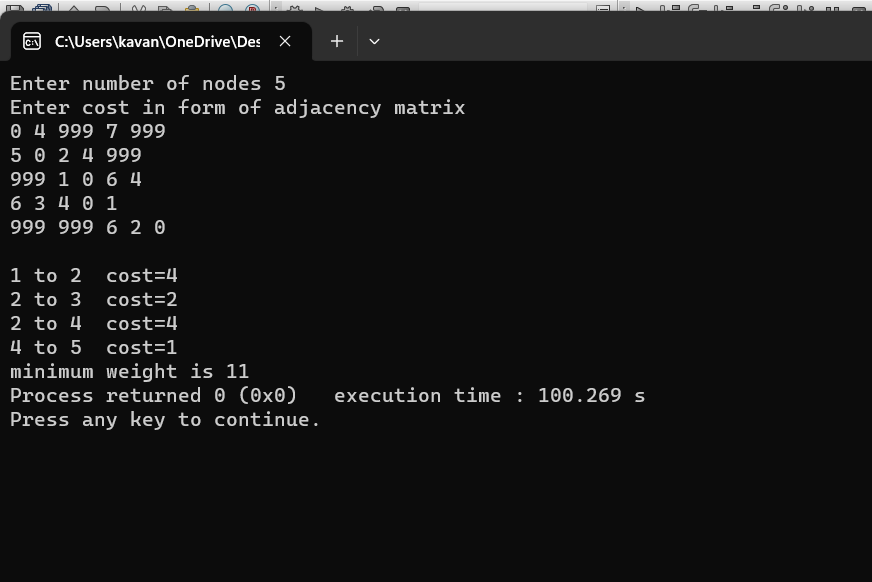
}

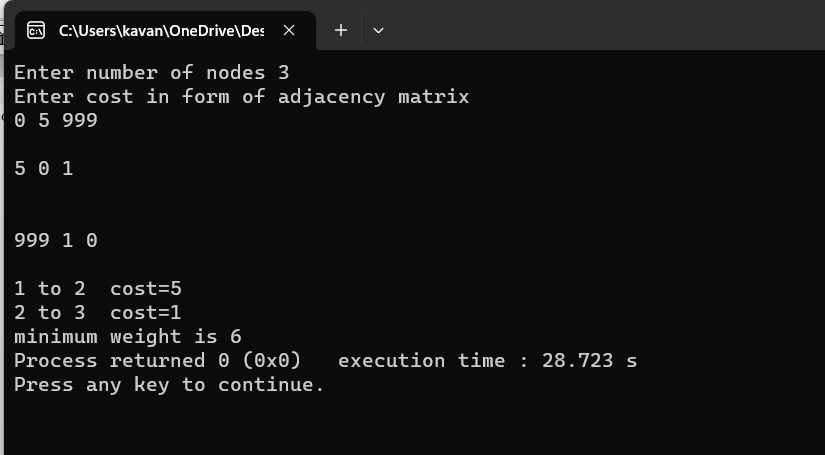
printf("\nminimum weight is %d",min\_cost);

return 0;

}

**OUTPUT:-**





**KRUSKAL’S:-**

#include<stdio.h>

#include<conio.h>

void kruskals();

int c[10][10],n;

void main()

{

int i,j;

printf("\nenter the no. of vertices:\t");

scanf("%d",&n);

printf("\nenter the cost matrix:\n");

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

{

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

{

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

}

}

kruskals();

}

void kruskals()

{

int i,j,u,v,a,b,min;

int ne=0,mincost=0;

int parent[10];

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

{

parent[i]=0;

}

while(ne!=n-1)

{

min=9999;

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

{

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

{

if(c[i][j]<min)

{

min=c[i][j];

u=a=i;

v=b=j;

}

}

}

while(parent[u]!=0)

{

u=parent[u];

}

while(parent[v]!=0)

{

v=parent[v];

}

if(u!=v)

{

printf("\n%d----->%d=%d\n",a,b,min);

parent[v]=u;

ne=ne+1;

mincost=mincost+min;

}

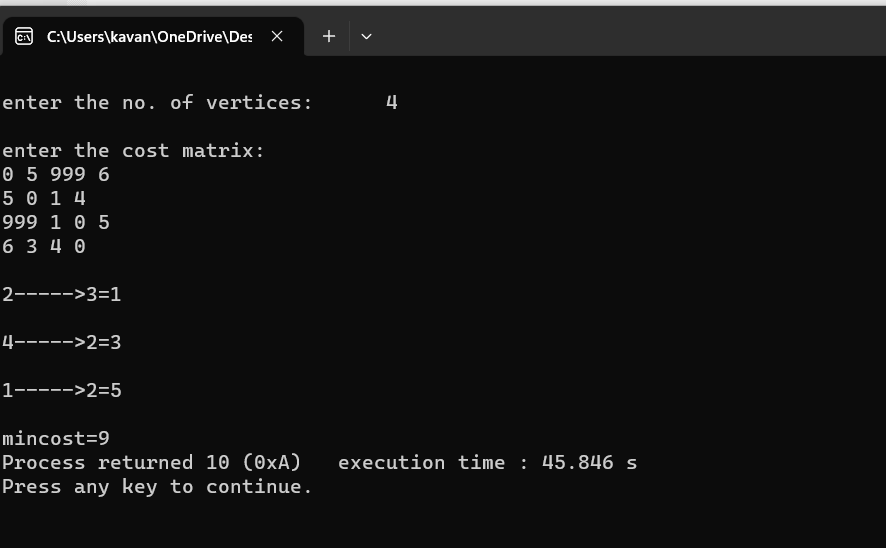
c[a][b]=c[b][a]=9999;

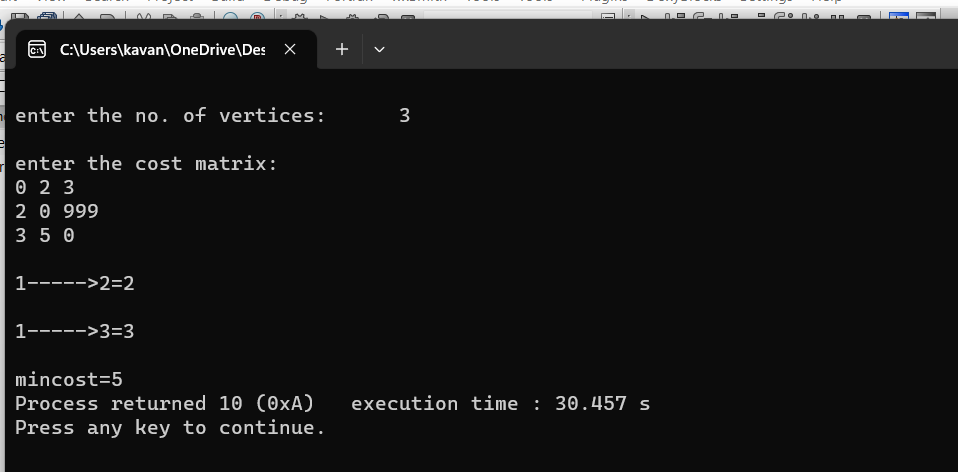
}

printf("\nmincost=%d",mincost);

}

**OUTPUT:-**





**LAB PROGRAM-10**

**From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm**.

#include<stdio.h>

#include<conio.h>

void dijkstras();

int c[10][10],n,src;

void main()

{

int i,j;

printf("\nenter the no of vertices:\t");

scanf("%d",&n);

printf("\nenter the cost matrix:\n");

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

{

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

{

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

}

}

printf("\nenter the source node:\t");

scanf("%d",&src);

dijkstras();

}

void dijkstras()

{

int vis[10],dist[10],u,j,count,min;

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

{

dist[j]=c[src][j];

}

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

{

vis[j]=0;

}

dist[src]=0;

vis[src]=1;

count=1;

while(count!=n)

{

min=9999;

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

{

if(dist[j]<min&&vis[j]!=1)

{

min=dist[j];

u=j;

}

}

vis[u]=1;

count++;

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

{

if(min+c[u][j]<dist[j]&&vis[j]!=1)

{

dist[j]=min+c[u][j];

}

}

}

printf("\nthe shortest distance is:\n");

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

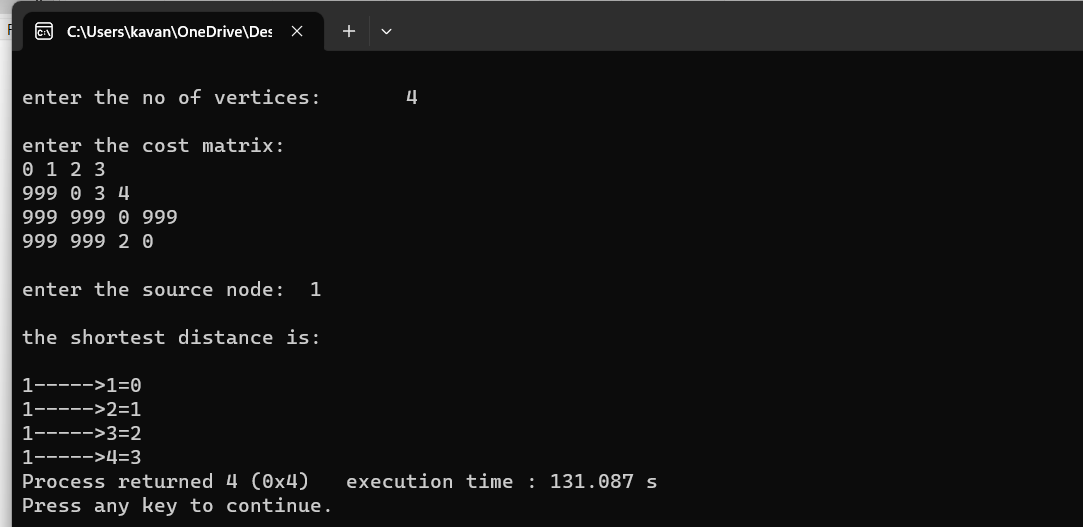
{

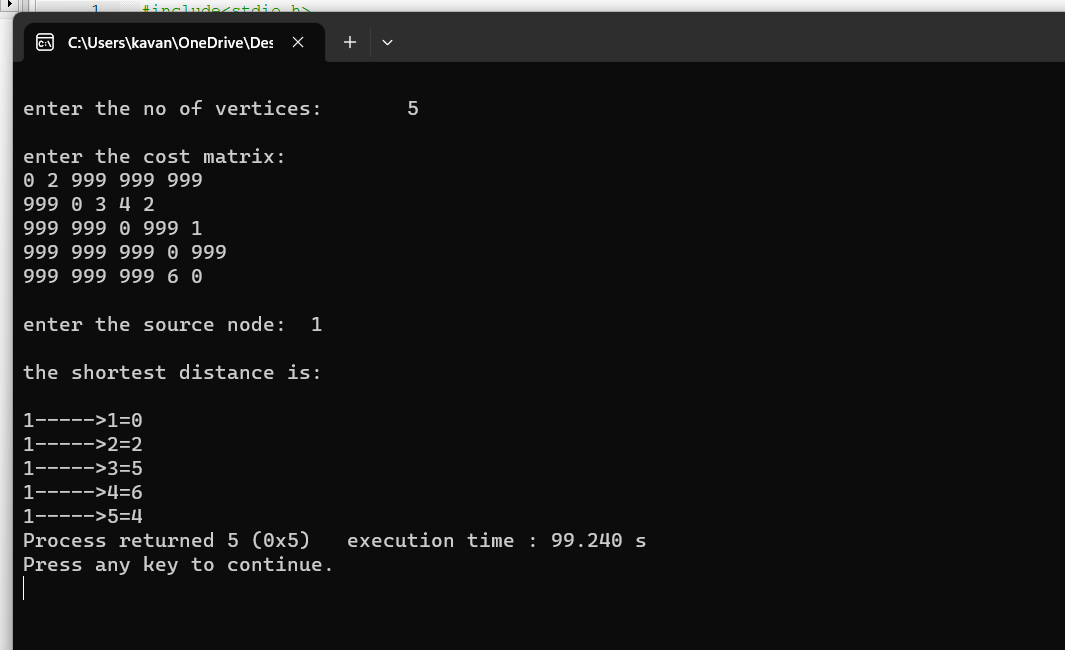
printf("\n%d----->%d=%d",src,j,dist[j]);

}

}

**OUTPUT:-**





**LAB PROGRAM-11**

**Implement “N-Queens Problem” using Backtracking.**

#include<stdio.h>

#include<math.h>

int a[30],count=0;

int place(int pos)

{

int i;

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

if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))

return 0;

}

return 1;

}

void print\_sol(int n) {

int i,j;

count++;

printf("\n\nSolution #%d:\n",count);

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

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

if(a[i]==j)

printf("Q\t"); else

printf("\*\t");

}

printf("\n");

}

}

void queen(int n) {

int k=1;

a[k]=0;

while(k!=0) {

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

while((a[k]<=n)&&!place(k))

a[k]++;

if(a[k]<=n) {

if(k==n)

print\_sol(n); else {

k++;

a[k]=0;

}

} else

k--;

}

}

void main() {

int i,n;

printf("Enter the number of Queens\n");

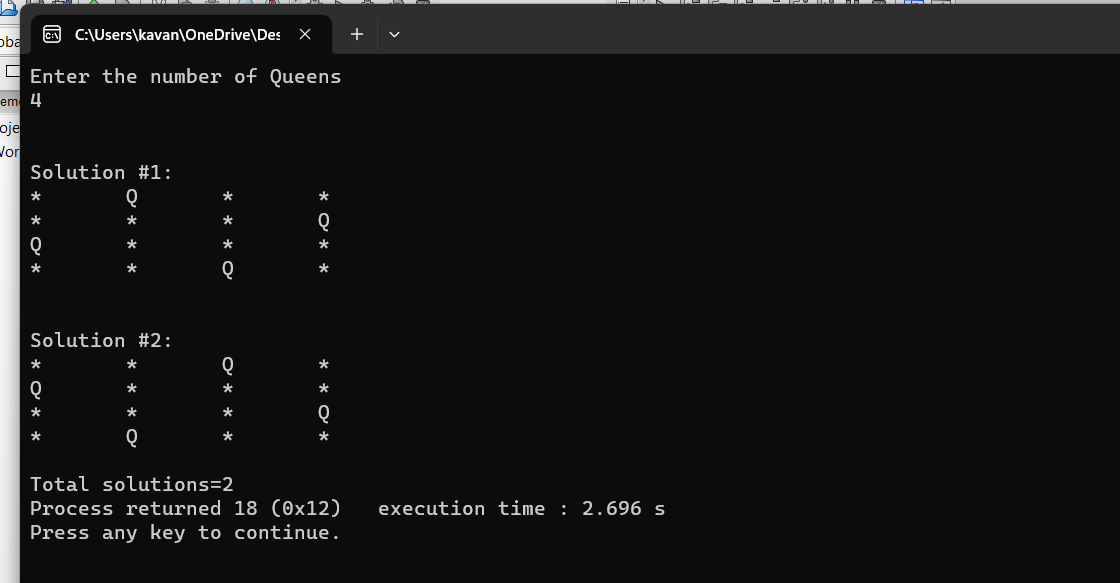
scanf("%d",&n);

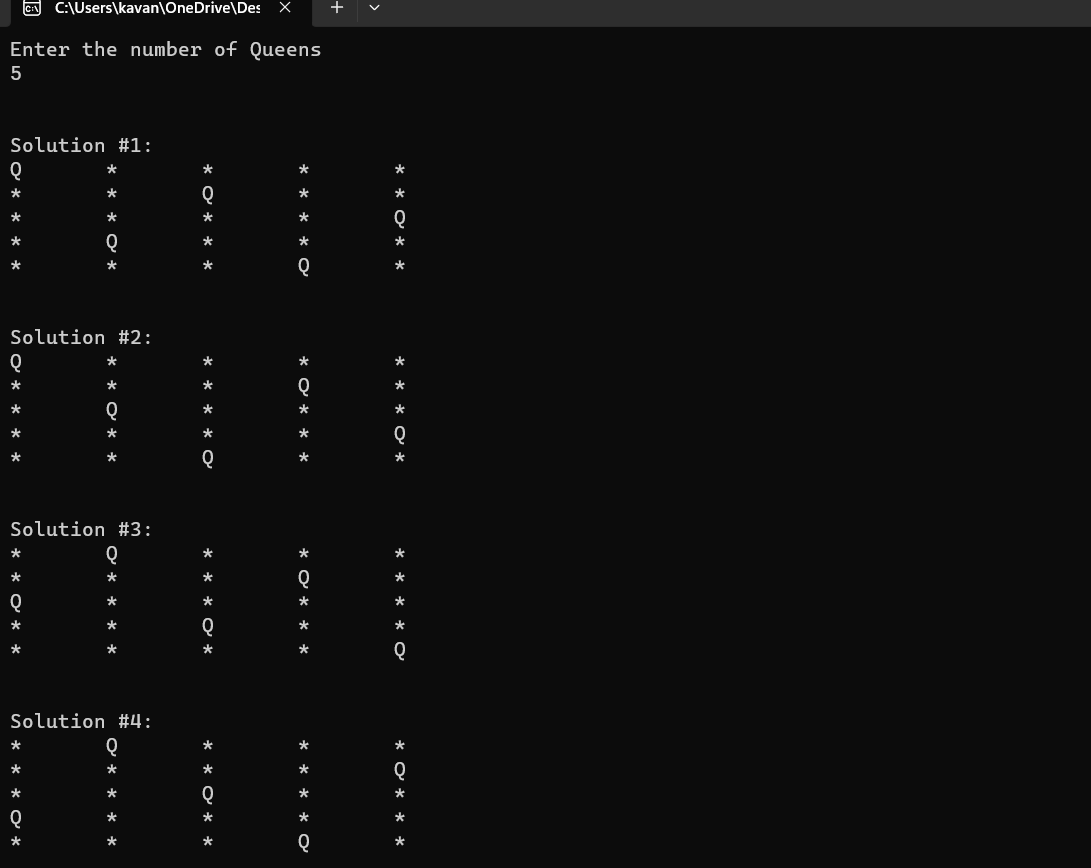
queen(n);

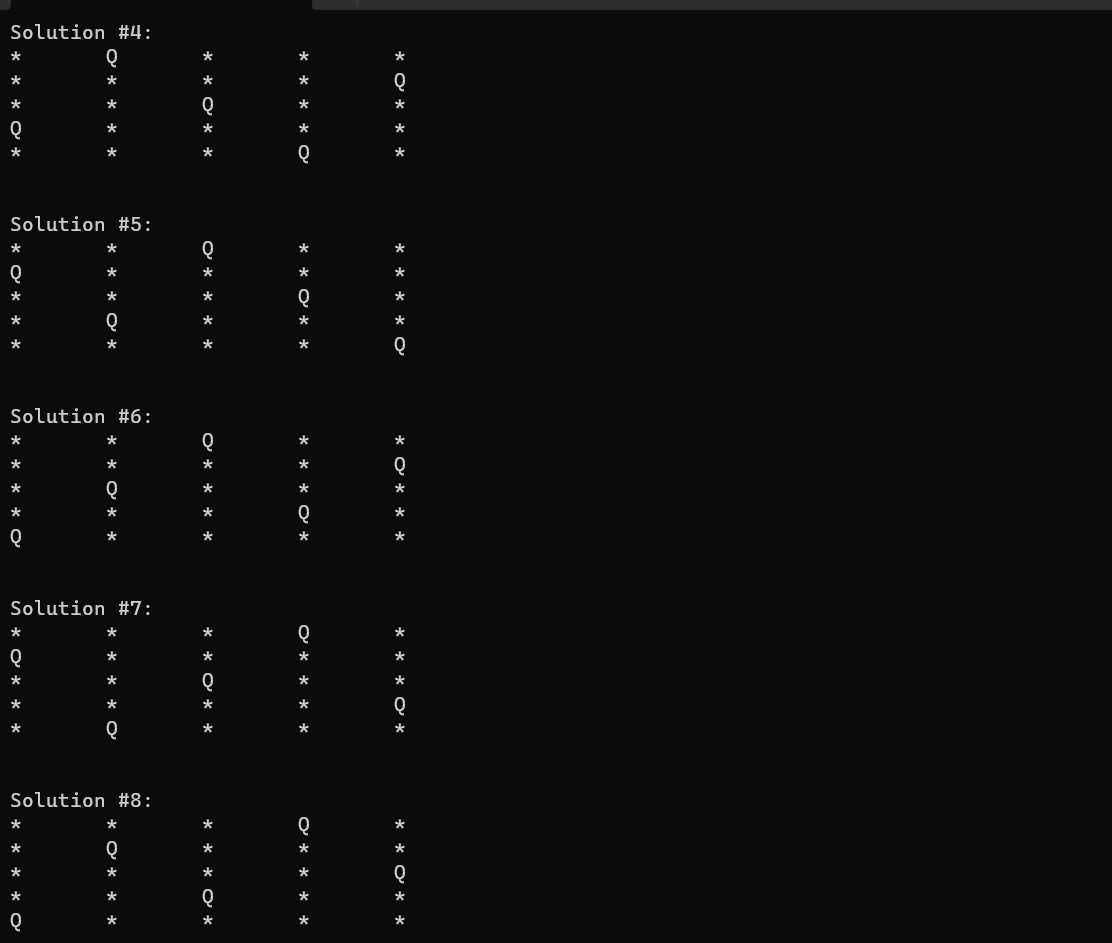
printf("\nTotal solutions=%d",count);

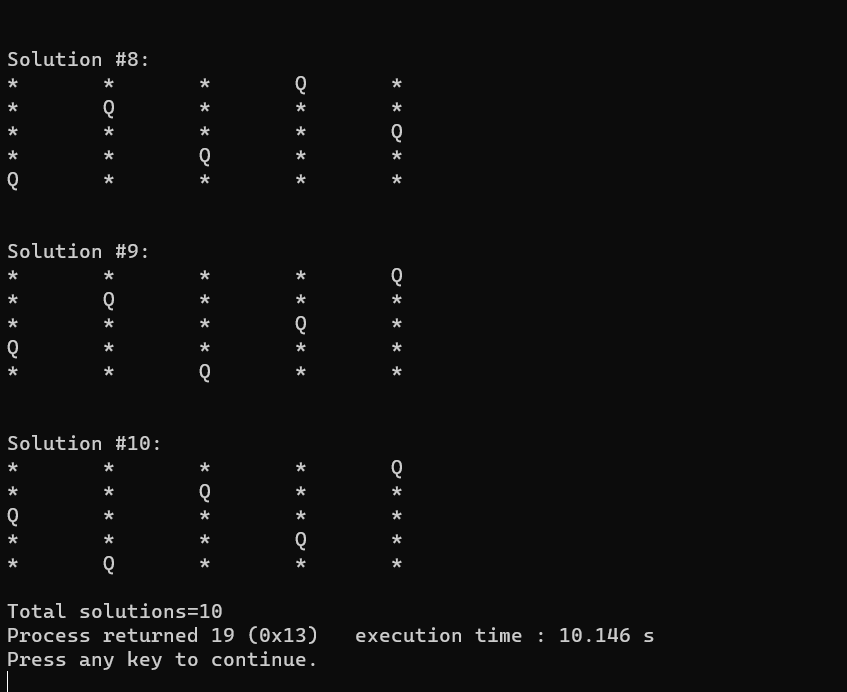
}

**OUTPUT:-**



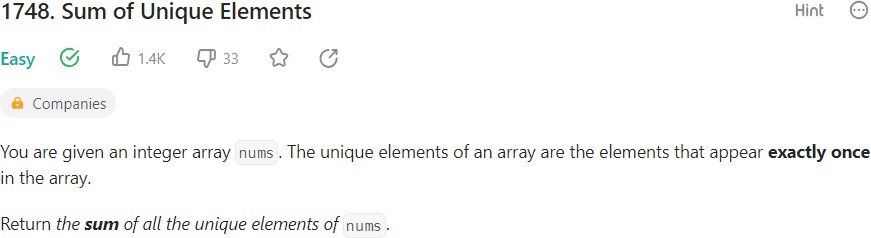


****

****

LEETCODE PROBLEMS:

1.



Solution:

class Solution { public:

int maxi(vector<int>& nums){ int m=INT\_MIN;

for(int i=0;i<nums.size();i++){ if(nums[i]>m)m=nums[i];

}

return m;

}

int sumOfUnique(vector<int>& nums) { int n=maxi(nums);

int aux[n+1];

for(int i=0;i<n+1;i++){ aux[i]=0;

}

for(int i=0;i<nums.size();i++){ aux[nums[i]]++;

}

int sum=0;

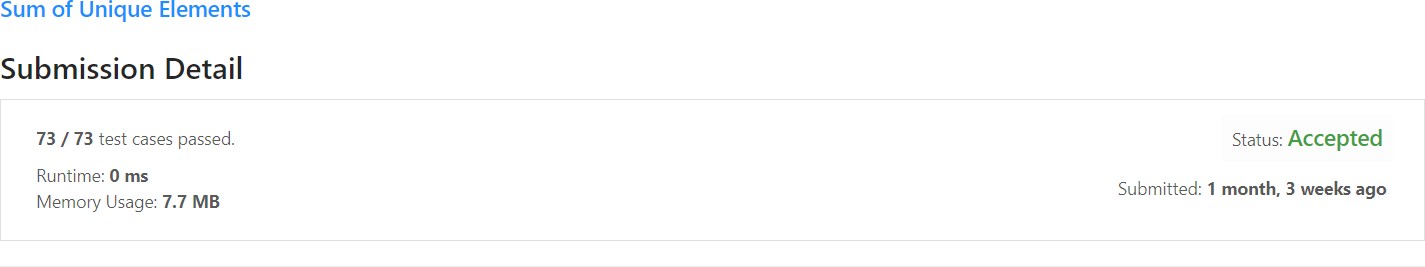
for(int i=0;i<n+1;i++){ if(aux[i]==1)sum+=i;

}

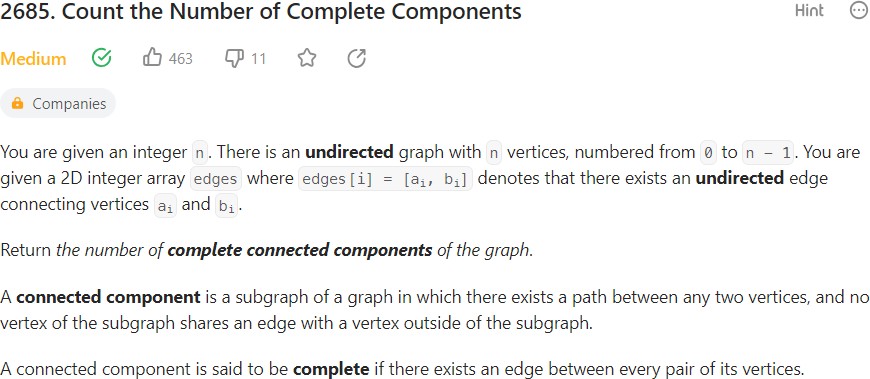
return sum;

}

};



2.



#include <stdio.h> #include <stdlib.h> #include <stdbool.h>

struct Node { int val;

struct Node\* next;

};

void dfs(int src, struct Node\*\* g, int\* nodes, int\* vis, int\* nodeCount) { if (vis[src])

return; vis[src] = 1;

nodes[\*nodeCount] = src;

(\*nodeCount)++;

struct Node\* current = g[src]->next; while (current != NULL) {

int x = current->val;

dfs(x, g, nodes, vis, nodeCount); current = current->next;

}

}

int countCompleteComponents(int n, int\*\* edges, int edgesSize, int\* edgesColSize) { struct Node\*\* g = (struct Node\*\*)malloc(n \* sizeof(struct Node\*));

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

g[i] = (struct Node\*)malloc(sizeof(struct Node)); g[i]->next = NULL;

}

for (int i = 0; i < edgesSize; i++) { int u = edges[i][0];

int v = edges[i][1];

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->val = v;

newNode->next = g[u]->next; g[u]->next = newNode;

newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->val = u;

newNode->next = g[v]->next; g[v]->next = newNode;

}

int\* vis = (int\*)malloc(n \* sizeof(int)); for (int i = 0; i < n; i++) {

vis[i] = 0;

}

int res = 0;

for (int i = 0; i < n; i++) { if (vis[i] == 0) {

int\* nodes = (int\*)malloc(n \* sizeof(int)); int nodeCount = 0;

dfs(i, g, nodes, vis, &nodeCount);

int count = 0;

for (int j = 0; j < nodeCount; j++) { int pathNode = nodes[j];

struct Node\* current = g[pathNode]->next; int neighborCount = 0;

while (current != NULL) { neighborCount++; current = current->next;

}

if (neighborCount >= nodeCount - 1) { count++;

}

}

if (count == nodeCount) { res++;

}

free(nodes);

}

}

for (int i = 0; i < n; i++) { struct Node\* current = g[i]; while (current != NULL) { struct Node\* temp = current; current = current->next; free(temp);

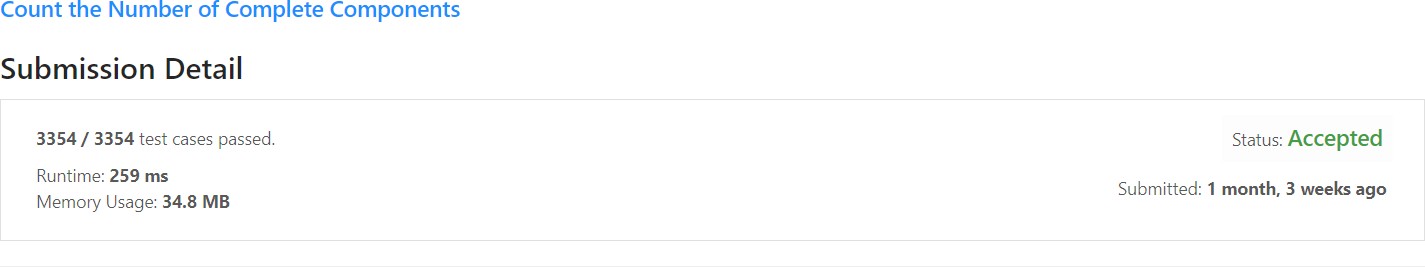
}

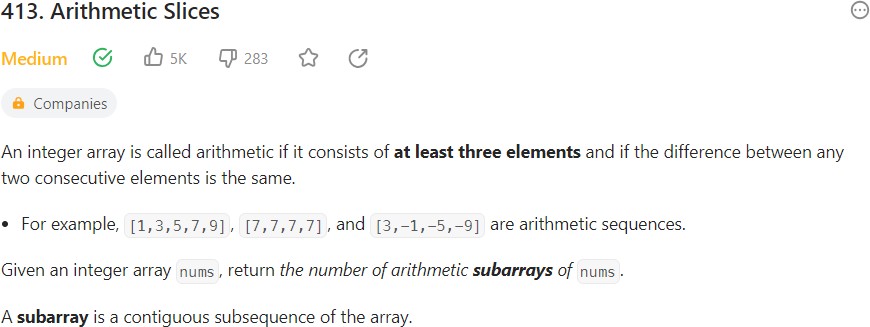
}

free(g); free(vis);

return res;

}



3.

int numberOfArithmeticSlices(int\* nums, int numsSize)

{

if (numsSize<3) return 0;

int count=0, diff;

for(int i = 0; i<numsSize-2; ++i)

{

diff = nums[i+1] - nums[i];

for(int j = i+2; j<numsSize; ++j)

{

if(nums[j] - nums[j-1] == diff)

++count; else

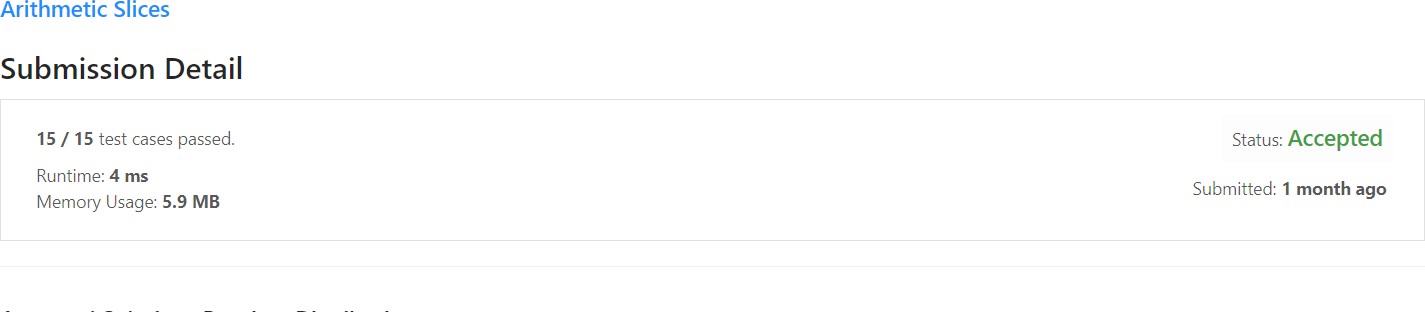
break;

}

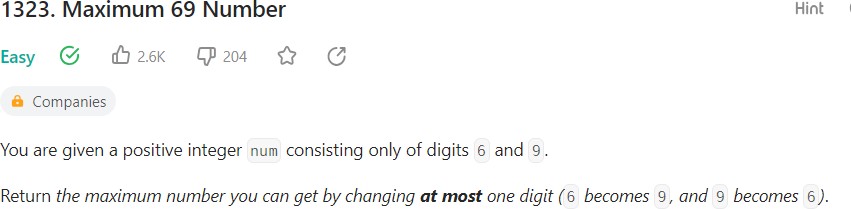
}

return count;

}



4.



class Solution { public:

int maximum69Number (int n) { int k=0;

string s=""; while(n){ s+=n%10+'0';

n/=10;

}

int size=s.length(); for(int i=size-1;i>=0;i--){

if(s[i]=='9' || k==1){

n=n\*10+(s[i]-'0');

};

}

}

return n;

}

};

