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

**Bull Temple Road, Bangalore-560019**

(Affiliated to Visvesvaraya Technological

University, Belgaum)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY RECORD**

**NAME:** ANAGHA ACHARYA

**USN:** 1BM19CS224

**PROGRAM:** BACHELOR OF ENGINEERING

**SEMESTER:** IV

**SESSION:** APR-JUL 2021

**COURSE CODE:** 19CS4PCDBM

**COURSE TITLE:** ANALYSIS AND DESIGN OF ALGORITHMS

**CREDITS:** 4

PROGRAM 10: Sort a given set of elements using heap sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int temp;

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

{

int largest,left,right;

largest = i;

left = 2\*i + 1;

right = 2\*i + 2;

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

largest=left;

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

largest = right;

if(largest != i)

{

temp = a[i];

a[i]= a[largest];

a[largest] = temp;

heapify(a,n, largest);

}

}

void heapsort(int a[],int n)

{

int i;

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

heapify(a,n,i);

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

{

temp=a[0];

a[0]=a[i];

a[i]=temp;

heapify(a, i, 0);

}

}

void main(){

int a[1000],i,n;

clock\_t start,end;

double time;

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

scanf("%d",&n);

printf("The numbers are:\n");

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

{

a[i]=(int)rand()%10000;

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

}

start=clock();

heapsort(a,n);

end=clock();

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

printf("\nSorted array:\n");

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

{

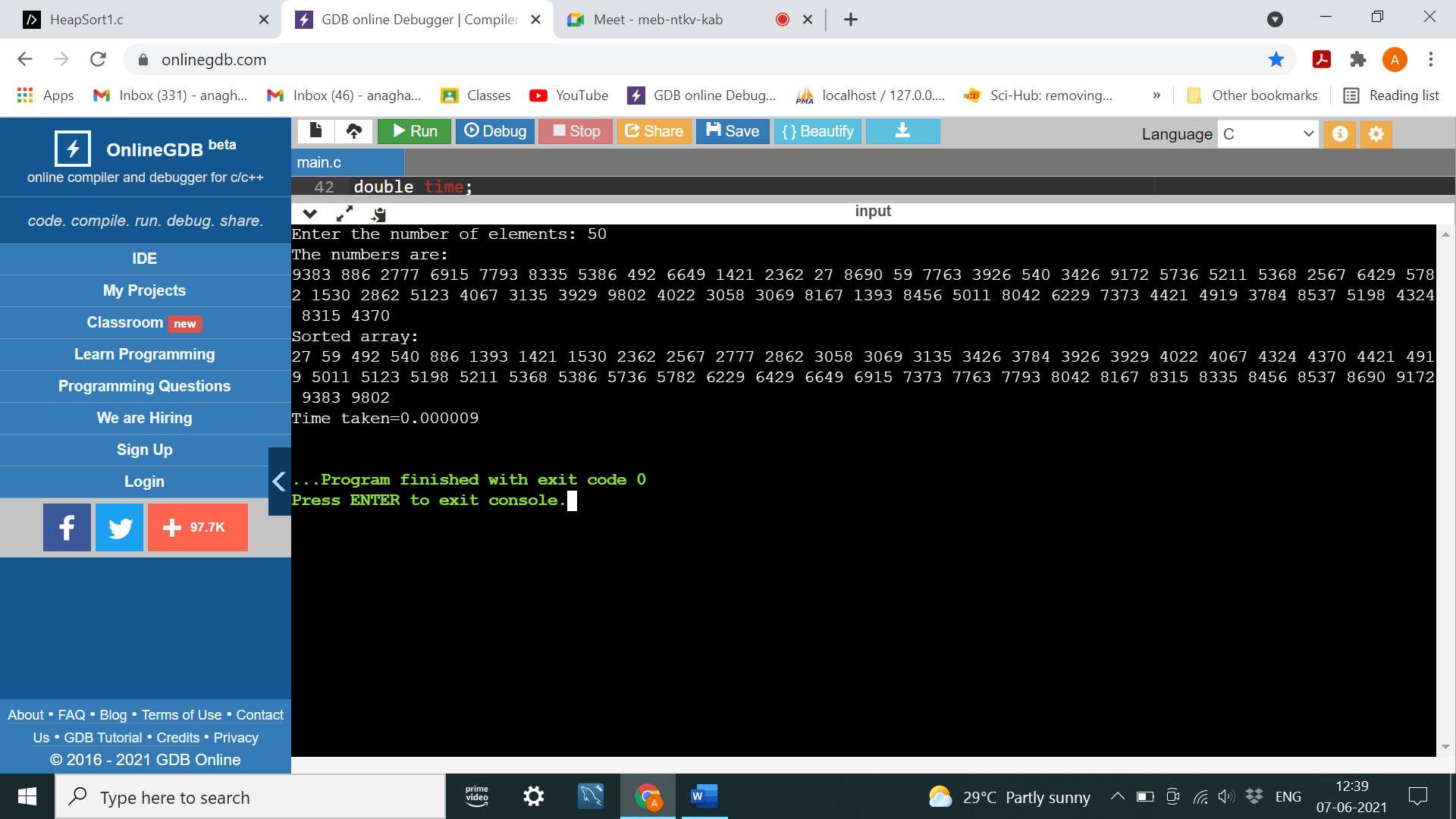
printf("%d ",a[i]);

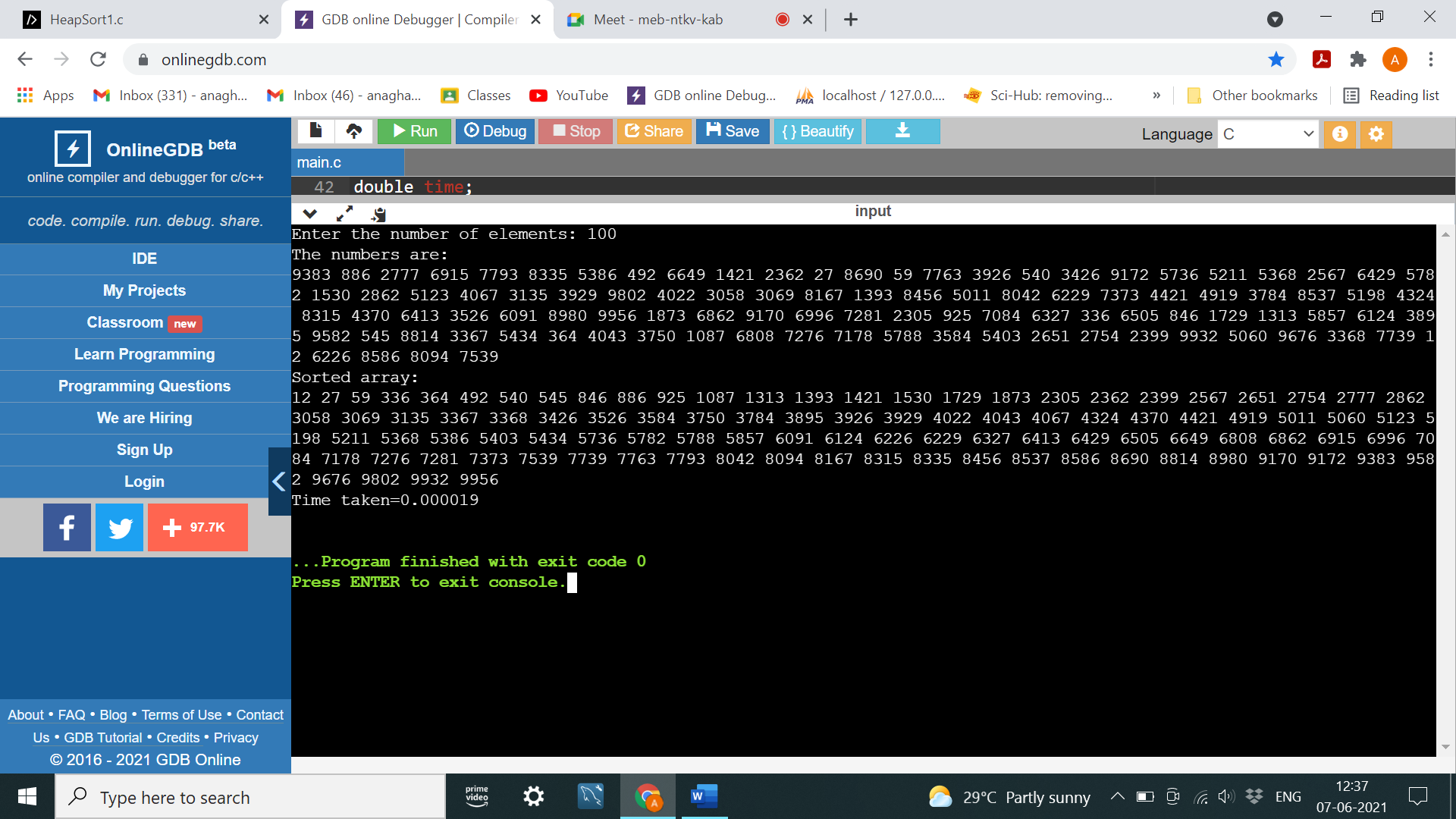
}

printf("\nTime taken=%1f\n",time);

}

OUTPUT:





PROGRAM 11: Implement Warshall’s algorithm using dynamic programming.

#include <stdio.h>

#include<time.h>

int n,adj[10][10],p[10][10];

void warshalls()

{

int i,j,k;

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

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

p[i][j]=adj[i][j];

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

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

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

if(p[i][k]==1&&p[k][j]==1)

p[i][j]=1;

}

void main()

{

int i,j;

printf("Enter the number of nodes:");

scanf("%d",&n);

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

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

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

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

warshalls();

printf("\nThe transitive closure:\n");

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

{

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

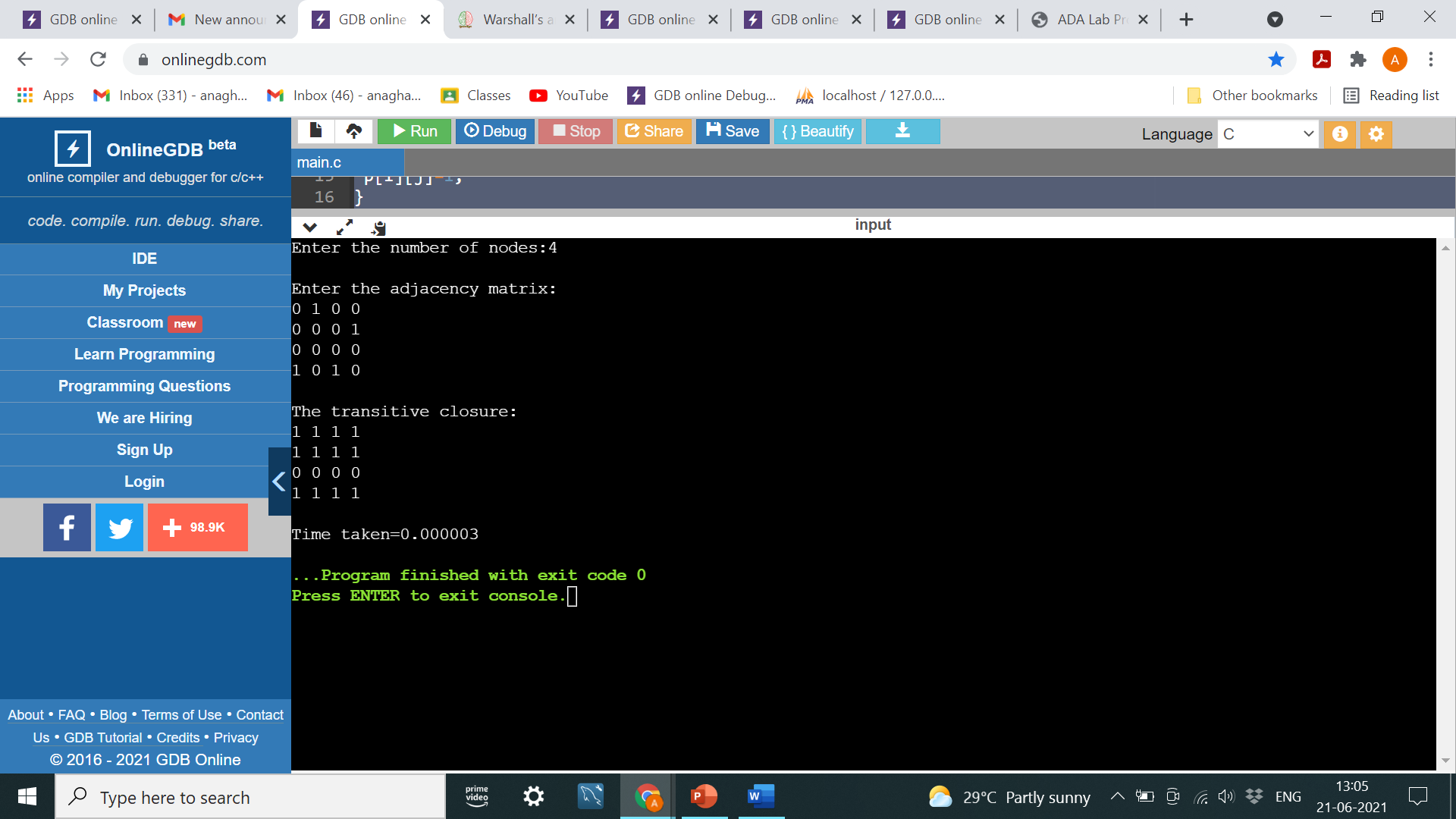
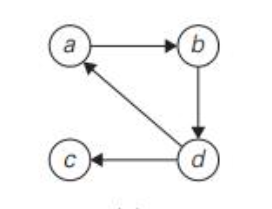
printf("%d ",p[i][j]);

printf("\n");

}

}

OUTPUT:



PROGRAM 12: Implement 0/1 Knapsack problem using dynamic programming.

#include<stdio.h>

int w[10],p[10],v[10][10],n,i,j,cap,x[10]={0};

int max(int i,int j)

{

return ((i>j)?i:j);

}

int knapsack(int i,int j)

{

int value;

if(v[i][j]<0)

{

if(j<w[i])

value=knapsack(i-1,j);

else

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

v[i][j]=value;

}

return(v[i][j]);

}

void main()

{

int profit,count=0;

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

scanf("%d",&n);

printf("Enter the profit and weights of the elements\n");

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

{

printf("Item no %d: ",i);

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

}

printf("\nEnter the capacity \n");

scanf("%d",&cap);

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

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

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

v[i][j]=0;

else

v[i][j]=-1;

profit=knapsack(n,cap);

i=n;

j=cap;

while(j!=0&&i!=0)

{

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

{

x[i]=1;

j=j-w[i];

i--;

}

else

i--;

}

printf("\nItems included:\n");

printf("SI No.\tWeight\tProfit\n");

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

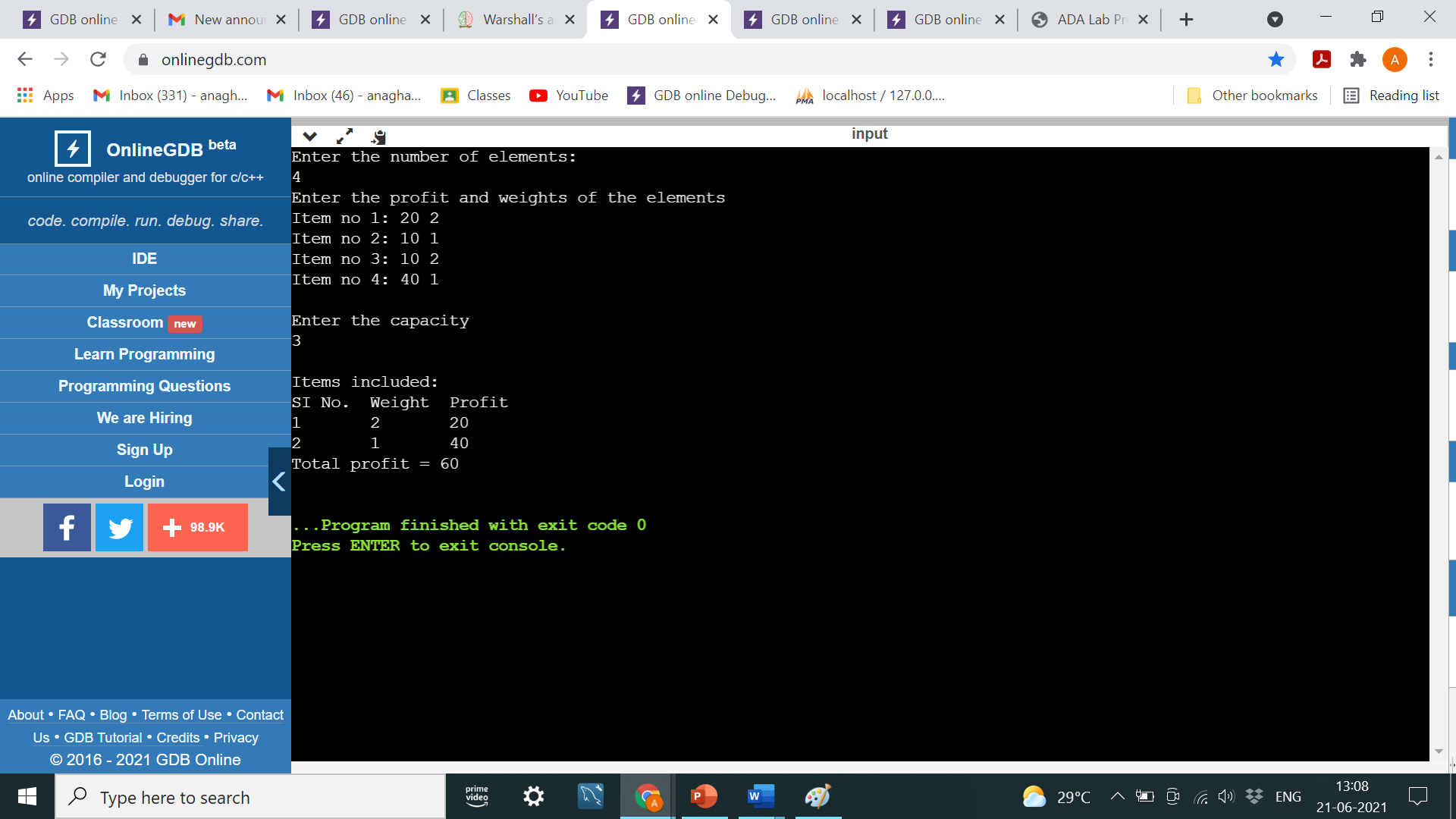
if(x[i])

printf("%d\t%d\t%d\n",++count,w[i],p[i]);

printf("Total profit = %d\n",profit);

}

OUTPUT:



PROGRAM 13: Implement All Pair Shortest paths problem using Floyd’s algorithm.

#include<stdio.h>

#include<conio.h>

int min(int,int);

void floyds(int p[10][10],int n)

{

int i,j,k;

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

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

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

if(i==j)

p[i][j]=0;

else

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

}

int min(int a,int b)

{

if(a<b)

return(a);

else

return(b);

}

void main()

{

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

printf("Enter the number of vertices:");

scanf("%d",&n);

printf("\nEnter the number of edges: ");

scanf("%d",&e);

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

{

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

p[i][j]=999;

}

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

{

printf("\nEnter the end vertices of edge %d with its weight \n",i);

scanf("%d%d%d",&u,&v,&w);

p[u][v]=w;

}

floyds(p,n);

printf("\n Matrix of input data:\n");

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

{

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

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

printf("\n");

}

printf("\nThe shortest paths are:\n");

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

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

{

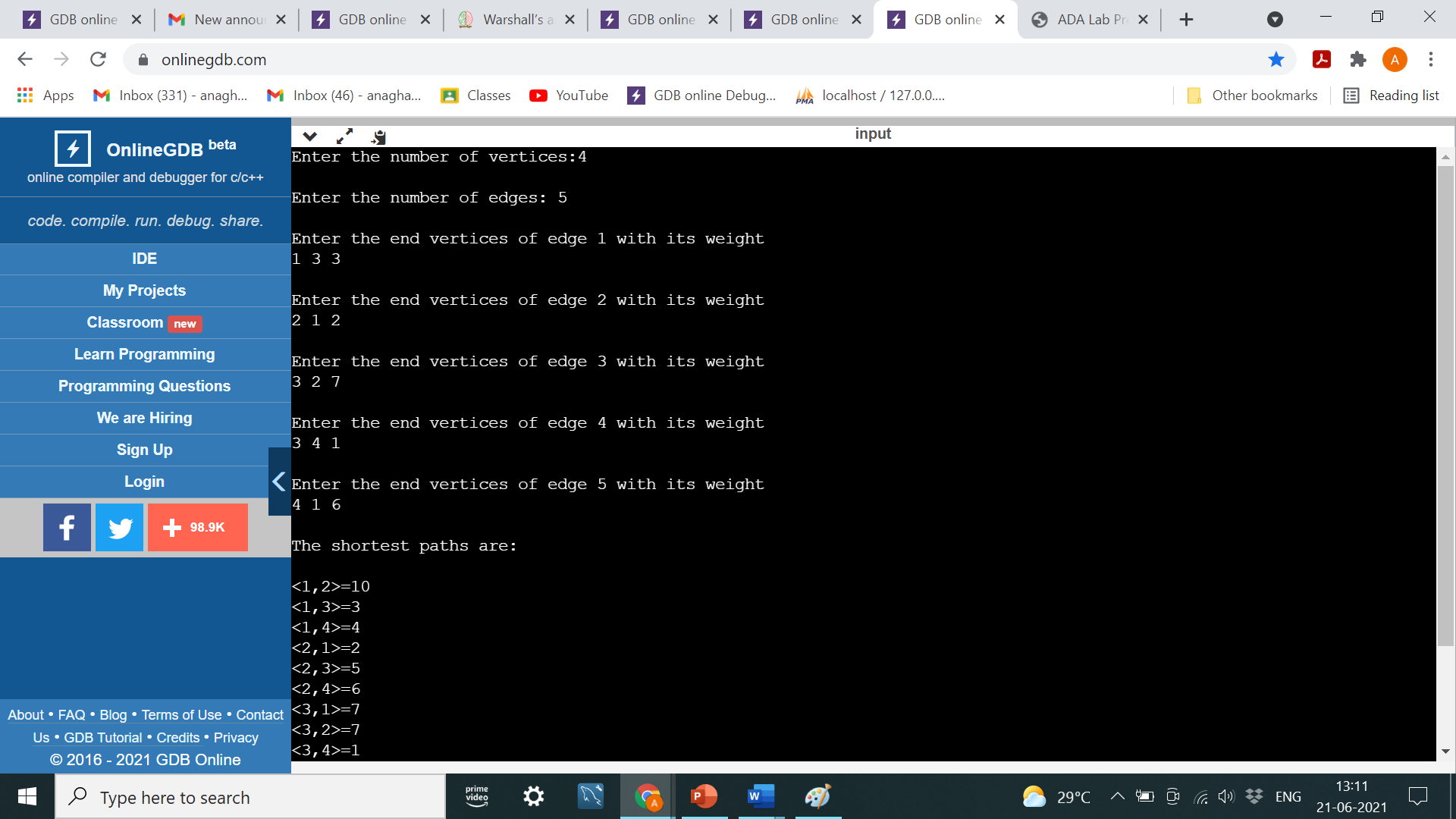
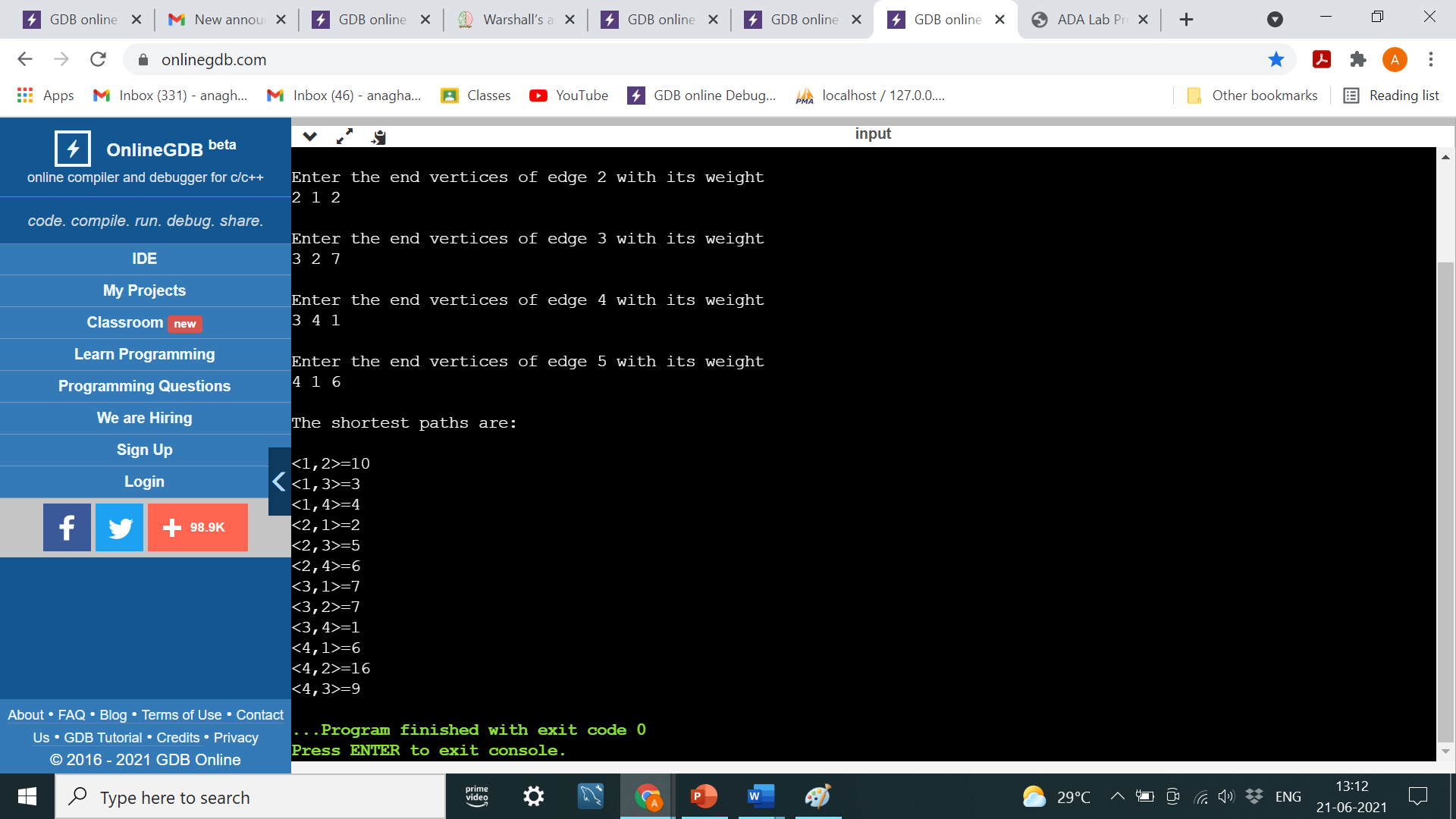
if(i!=j)

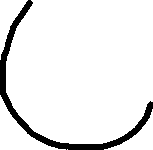
printf("\n<%d,%d>=%d",i,j,p[i][j]);

}

}

OUTPUT:





PROGRAM 14: Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.

#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void prims(){

visited[1]=1;

printf("\n");

while(ne<n)

{

for(i=1,min=999;i<=n;i++)

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

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

if(visited[i]!=0)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0)

{

printf("\nEdge %d(%d %d); Cost=%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

}

printf("\nMinimum cost=%d",mincost);

}

void main()

{

printf("Enter the number of nodes:");

scanf("%d",&n);

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

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

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

{

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

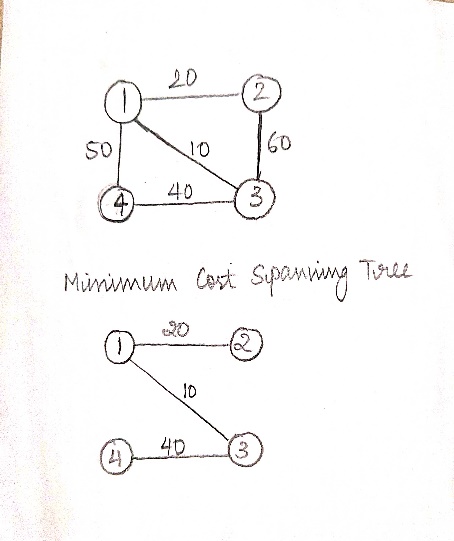
if(cost[i][j]==0)

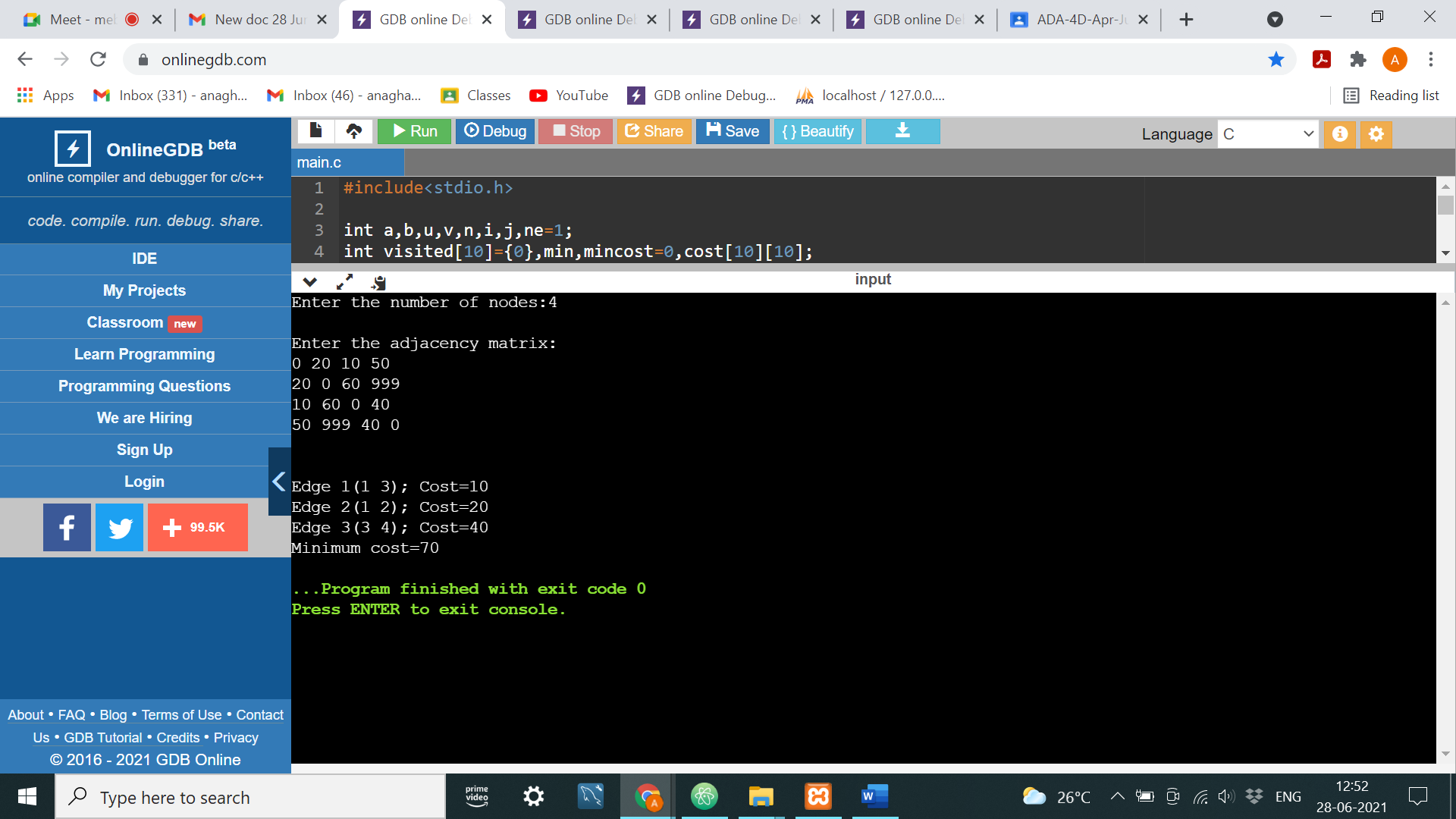
cost[i][j]=999;

}

prims();

}

OUTPUT:



PROGRAM 15: Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm.

#include<stdio.h>

int n, v, u,cost[10][10], parent[10]={0}, i, j;

int count=1, mincost=0, min, a, b;

void kruskals(){

while(count<n)

{

min=999;

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

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

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

{

min=cost[i][j];

a=u=i;

b=v=j;

}

while(parent[u])

u=parent[u];

while(parent[v])

v=parent[v];

if(u!=v)

{

count++;

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

mincost+=min;

parent[v]=u;

}

cost[a][b]=cost[b][a]=999;

}

printf("\nMinimum cost=%d", mincost);

}

void main()

{

printf("Enter number of vertices:");

scanf("%d",&n);

printf("\nEnter cost matrix:\n");

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

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

{

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

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

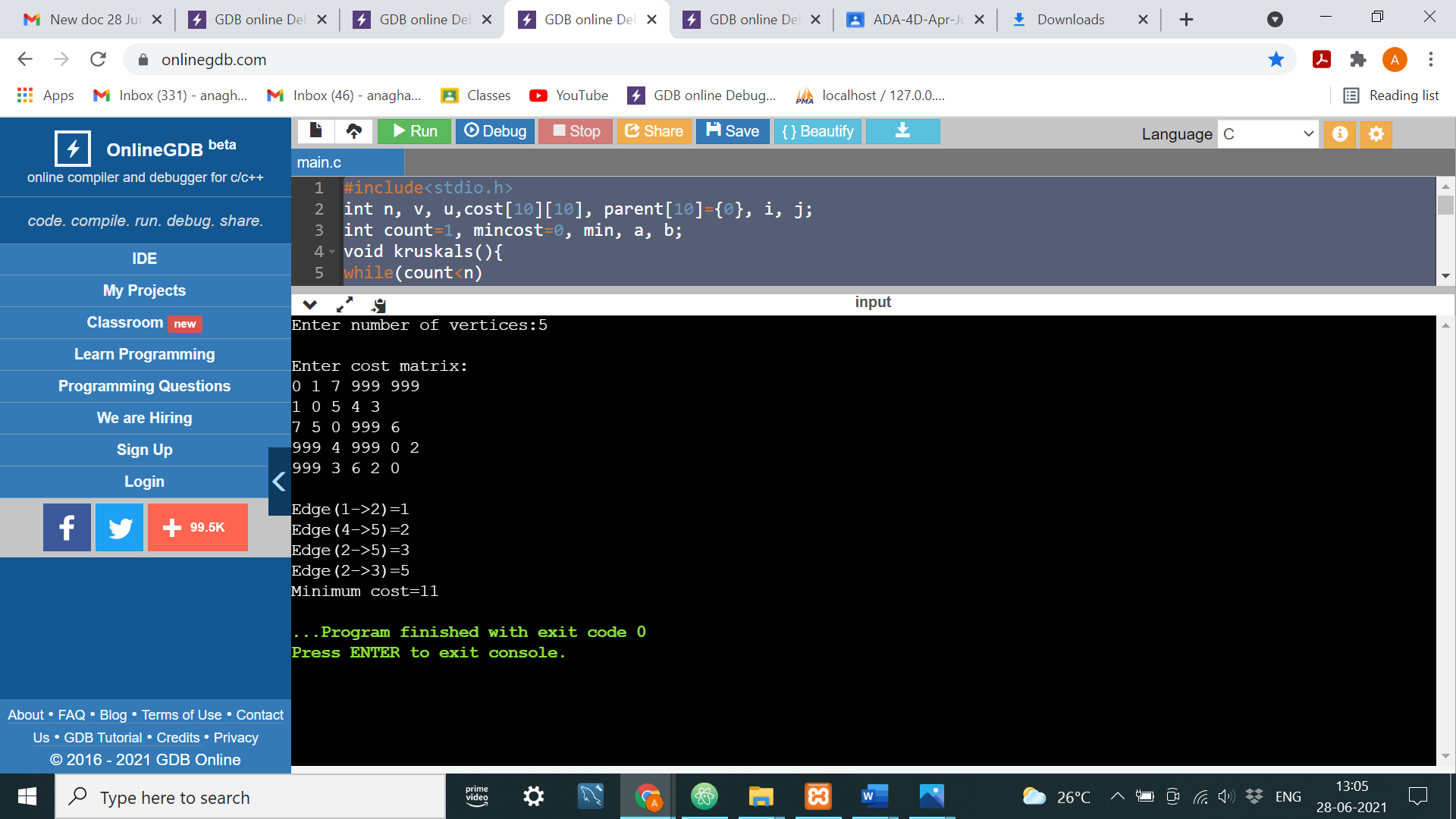
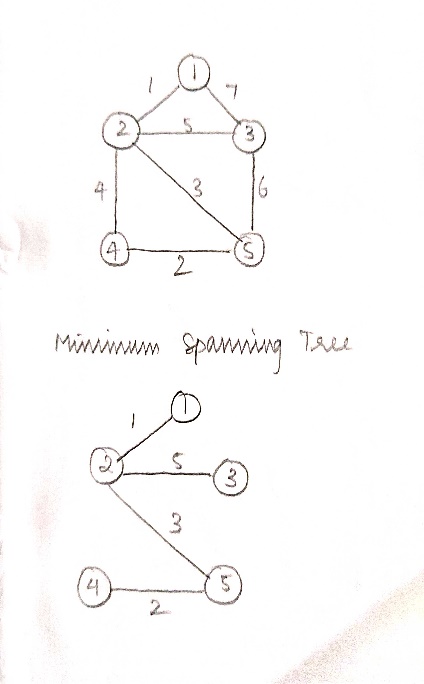
cost[i][j]=999;

}

kruskals();

}

OUTPUT:

PROGRAM 16: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

#include<stdio.h>

void dijkstra(int n, int v, int cost[10][10],int dist[10])

{

int count,u,i,w,visited[10],min;

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

{

visited[i]=0;

dist[i]=cost[v][i];

}

visited[v]=1;

dist[v]=1;

count=2;

while(count<=n)

{

min=999;

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

if((dist[w]<min) && (visited[w]!=1))

{

min=dist[w];

u=w;

}

visited[u]=1;

count++;

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

if((dist[u]+cost[u][w]<dist[w]) && (visited[w]!=1))

dist[w]=dist[u]+cost[u][w];

}

}

void main()

{

int n,v,cost[10][10],dist[10], i, j;

printf("Enter number of vertices: ");

scanf("%d",&n);

printf("\nEnter the cost matrix(for infinity, enter 999):\n");

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

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

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

printf("\nEnter source vertex:");

scanf("%d",&v);

dijkstra(n,v,cost,dist);

printf("\nShortest path from");

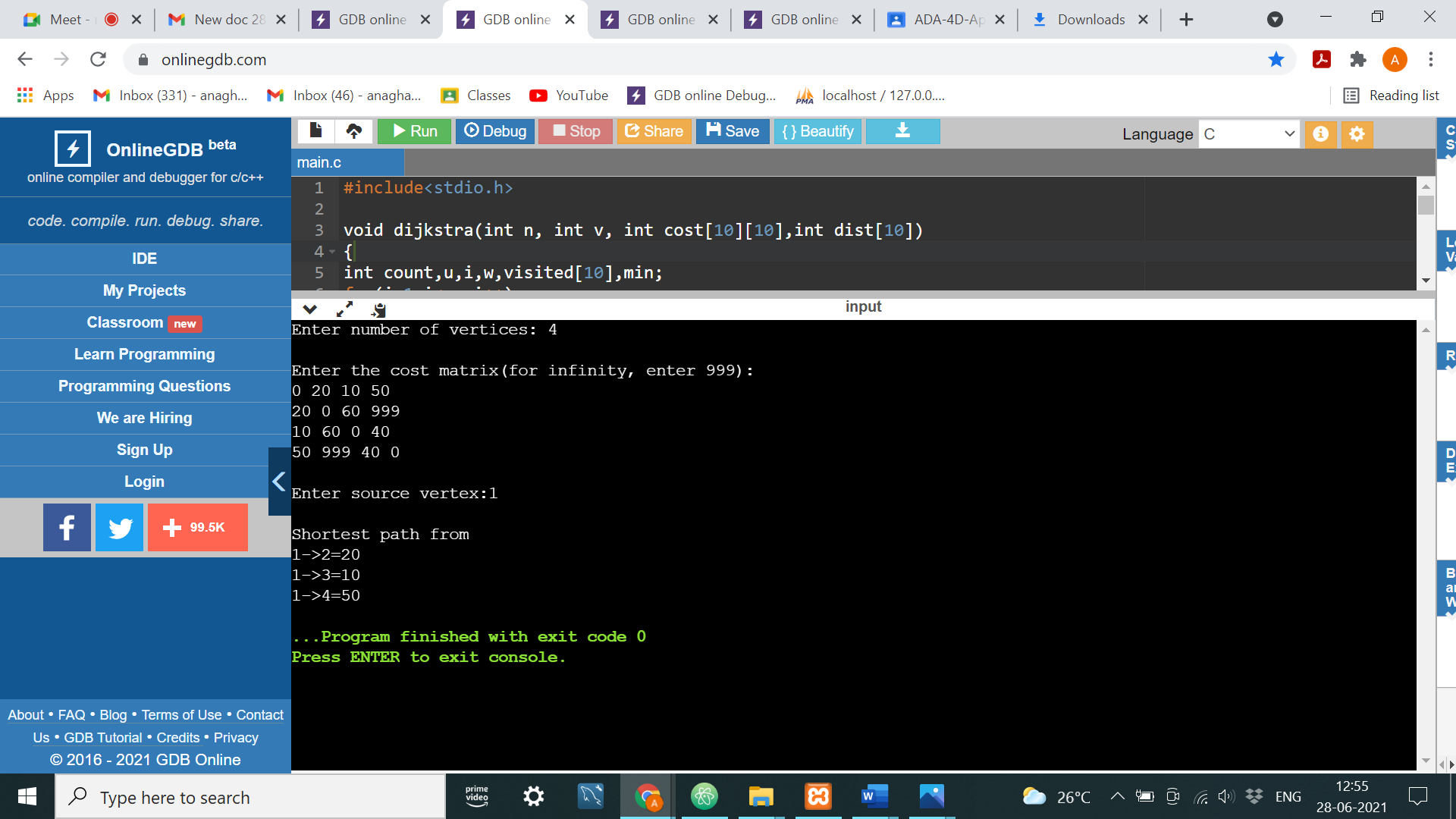
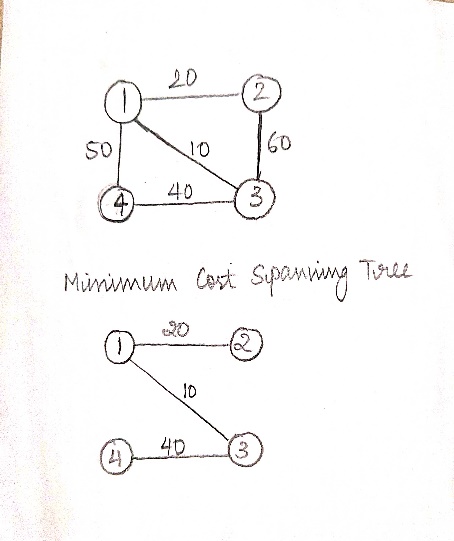
for(i=1;i<=n;i++)

if(i!=v)

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

}

OUTPUT:



PROGRAM 17: Implement “Sum of Subsets” using Backtracking. “Sum of Subsets” problem: Find a subset of a given set S = {s1,s2,……,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S = {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn’t have a solution.

#include<stdio.h>

int x[10],w[10],d,count=0;

void subset(int cs,int k,int r)

{

int i;

x[k]=1;

if((cs+w[k])==d)

{

printf("\nSubset %d: ",++count);

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

if(x[i]==1)

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

}

else

if(cs+w[k]+w[k+1]<=d)

subset(cs+w[k],k+1,r-w[k]);

if(cs+r-w[k]>=d && cs+w[k]<=d)

{

x[k]=0;

subset(cs,k+1,r-w[k]);

}

}

void main()

{

int i,n,sum=0;

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

scanf("%d",&n);

printf("\nEnter the elements in ascending order: ");

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

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

printf("\nEnter the sum(d): ");

scanf("%d",&d);

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

sum=sum+w[i];

if(sum<d || w[1]>d)

{

printf("No subset possible");

}

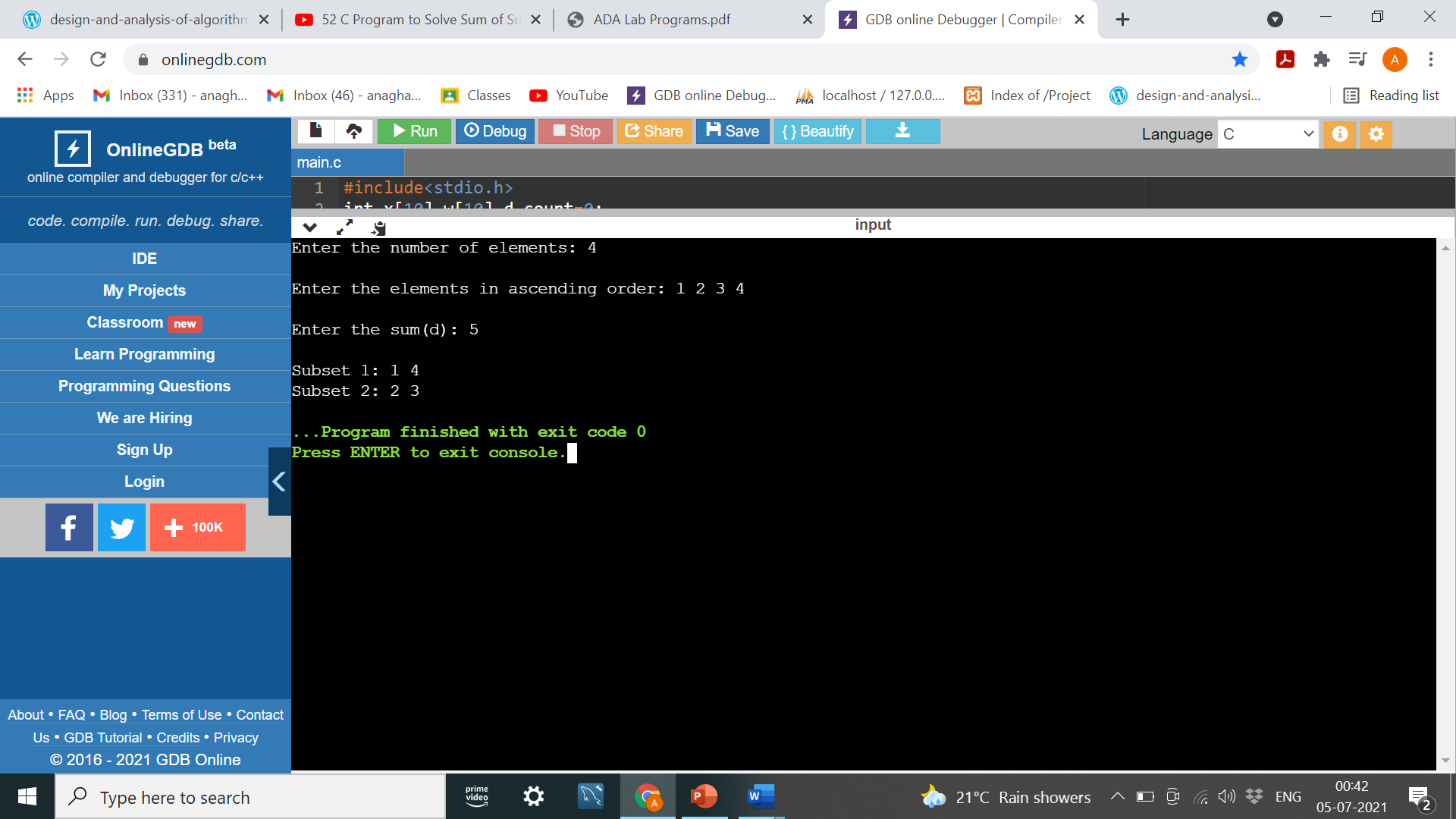
else{

subset(0,1,sum);

}

}

OUTPUT:



PROGRAM 18: Implement “N-Queens Problem” using Backtracking.

#include<stdio.h>

#include<stdlib.h>

void nqueens(int);

int place(int[],int);

void printsolution(int,int[]);

int place(int x[],int k)

{

int i;

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

if(x[i]==x[k]||(abs(x[i]-x[k]))==abs(i-k))

return 0;

return 1;

}

void nqueens(int n)

{

int x[10],count=0,k=1;

x[k]=0;

while(k!=0)

{

x[k]=x[k]+1;

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

x[k]=x[k]+1;

if(x[k]<=n)

{

if(k==n)

{

count++;

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

printsolution(n,x);

}

else

{

k++;

x[k]=0;

}

}

else

{

k--;

}

}

return;

}

void printsolution(int n,int x[])

{

int i,j;

char c[10][10];

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

{

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

c[i][j]='-';

}

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

c[i][x[i]]='Q';

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

{

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

{

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

}

printf("\n");

}

}

void main()

{

int n;

printf("Enter the no.of queens: ");

scanf("%d",&n);

nqueens(n);

}

OUTPUT:

