**Q1. Sort a given set of elements using the Quick 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. The elements can be read from a file or can be generated using the random number generator.**

**PROGRAM:-**

//

// main.c

// Quick\_Sort

//

// Created by Harsh vikram Shahi on 29/09/21.

//

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

**void** Exch(**int** \*p,**int** \*q)

{

**int** temp = \*p;

\*p = \*q;

\*q = temp;

}

**void** QuickSort(**int** a[],**int** low,**int** high)

{

**int** i,j,k,key;

**if**(low>=high)

**return**;

key=low;

i=low+1;

j=high;

**while**(i<=j)

{

**while**(a[i]<=a[key])

i=i+1;

**while**(a[j]>a[key])

j=j-1;

**if**(i<j)

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

}

Exch(&a[j],&a[key]);

QuickSort(a,low,j-1);

QuickSort(a,j+1,high);

}

**int** main()

{

**int** n,a[10000],k;

//double clock\_tst;

**double** et;

**double** st,ts;

//clrscr();

printf("\n Enter How many Numbers: ");

scanf("%d", &n);

printf("\nThe Random Numbers are:\n");

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

a[k]=rand();

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

}

st=clock();

QuickSort(a, 1, n);

et=clock();

ts=(**double**)(et-st)/CLOCKS\_PER\_SEC;

printf("\nSorted Numbers are: \n ");

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

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

printf("\nThe time taken is %e",ts);

}

**OUTPUT:-**

Graphical user interface, text, application, email

Description automatically generated

**Q2. Implement a Merge Sort algorithm to sort a given set of elements 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 .The elements can be read from a file or can be generated using the random number generator.**

**PROGRAM:-**

//

// main.c

// Merge\_Sort

//

// Created by Harsh vikram Shahi on 06/10/21.

//

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

**int** b[50000];

**void** Merge(**int** a[],**int** low,**int** mid,**int** high)

{

**int** i,j,k;

i=low;

j=mid+1;

k=low;

**while**(i<=mid && j<=high)

{

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

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

**else**

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

}

**while**(i<=mid)

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

**while**(j<=high)

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

**for**(k=low;k<=high;k++)

a[k]=b[k];

}

**void** MergeSort(**int** a[],**int** low,**int** high)

{

**int** mid;

**if**(low>=high)

**return**;

mid=(low+high)/2;

MergeSort(a,low,mid);

MergeSort(a,mid+1,high);

Merge(a,low,mid,high);

}

**int** main()

{

**int** n,a[50000],k;

//double clock\_tst;

**double** et;

**double** ts,st;

printf("\n Enter How many Numbers:");

scanf("%d", &n);

printf("\nThe Random Numbers are:\n");

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

a[k]=rand();

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

}

st=clock();

MergeSort(a, 1, n);

et=clock();

ts=(**double**)(et-st)/CLOCKS\_PER\_SEC;

printf("\n Sorted Numbers are : \n ");

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

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

printf("\nThe time taken is %e",ts);

}

**OUTPUT:-**

****

**Q3. Implement N Queen's problem using Back Tracking.**

**PROGRAM: -**

//

// main.c

// N\_Queen

//

// Created by Harsh vikram Shahi on 20/10/21.

//

#include <stdio.h>

#include <math.h>

#include<stdlib.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("0\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--;

}

}

**int** main(){

**int** n;

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

scanf("%d", &n);

queen(n);

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

}

**OUTPUT:-**

**Graphical user interface, text, application

Description automatically generated**

**Q4. 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.**

**PROGRAM:-**

//

// main.c

// Subset\_problem

//

// Created by Harsh vikram Shahi on 27/10/21.

//

#include<stdio.h>

#define TRUE 1

#define FALSE 0

#include<stdlib.h>

**int** inc[50],w[50],sum,n;

**void** sumset(**int** ,**int** ,**int**);

**int** promising(**int** i,**int** wt,**int** total) {

**return** (((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));

}

**int** main() {

**int** i,j,n,temp,total=0;

printf("\n Enter how many numbers: ");

scanf("%d",&n);

printf("\n Enter %d numbers : ",n);

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

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

total+=w[i];

}

printf("\n Input the sum value to create sub set: ");

scanf("%d",&sum);

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

**for** (j=0;j<n-1;j++) **if**(w[j]>w[j+1]) {

temp=w[j];

w[j]=w[j+1];

w[j+1]=temp;

}

printf("\n The given %d numbers in ascending order: ",n);

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

printf("%3d",w[i]);

**if**((total<sum))

printf("\n Subset construction is not possible");

**else**{

}

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

inc[i]=0;

printf("\n The solution using backtracking is:\n");

sumset(-1,0,total);

}

**void** sumset(**int** i,**int** wt,**int** total){

**int** j;

**if**(promising(i,wt,total)) {

**if**(wt==sum){

printf("\n{");

**for** (j=0;j<=i;j++)

**if**(inc[j])

printf("%3d",w[j]);

printf(" }\n");

} **else** {

inc[i+1]=**TRUE**;

sumset(i+1,wt+w[i+1],total-w[i+1]);

inc[i+1]=**FALSE**;

sumset(i+1,wt,total-w[i+1]);

}

}

}

**OUTPUT:-**

****

**Q5. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm.**

**PROGRAM:-**

//

// main.c

// Minimum\_cost\_spanning\_tree

//

// Created by Harsh vikram Shahi on 10/11/21.

//

#include<stdio.h>

#include<stdlib.h>

#include<stdlib.h>

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

**int** min,mincost=0,cost[9][9],parent[9];

**int** find(**int**);

**int** uni(**int**,**int**);

**int** main() {

printf("\n Implementation of Kruskal's algorithm\n\n");

printf("\nEnter the no. of vertices\n");

scanf("%d",&n);

printf("\nEnter the cost 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;

}

}

printf("\nThe edges of Minimum Cost Spanning Tree are\n\n");

**while**(ne<n){

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

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

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

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u);

v=find(v);

**if**(uni(u,v)){

printf("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

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

}

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

}

**int** find(**int** i){

**while**(parent[i])

i=parent[i];

**return** i;

}

**int** uni(**int** i,**int** j){

**if**(i!=j) {

parent[j]=i;

**return** 1;

}

**return** 0;

}

**OUTPUT:-**

**Graphical user interface, application

Description automatically generated**

**Q6. Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.**

**PROGRAM:-**

//

// main.c

// MinimumCost\_SpanningTree\_Prism

//

// Created by Harsh vikram Shahi on 01/12/21.

//

#include<stdio.h>

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

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

**int** main()

{

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

scanf("%d",&n);

printf("\n Enter 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;

}

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("\n Edge %d:(%d %d)");

printf("cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

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

}

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

}

**OUTPUT: -**

**Graphical user interface, text, application

Description automatically generated**

**Q7. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijikstra’s algorithm.**

**PROGRAM:-**

//

// main.c

// Dijkstra\_algorithm

//

// Created by Harsh vikram Shahi on 08/12/21.

//

#include<stdio.h>

#define infinity 999

**void** dij(**int** n, **int** v,**int** cost[20][20], **int** dist[]){

**int** i,u,count,w,flag[20],min;

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

flag[i]=0,dist[i]=cost[v][i];

count=2;

**while**(count<=n){

min=99;

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

**if**(dist[w]<min && !flag[w]) {

min=dist[w];

u=w;

}

flag[u]=1;

count++;

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

**if**((dist[u]+cost[u][w]<dist[w]) && !flag[w])

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

}

}

**int** main(){

**int** n,v,i,j,cost[20][20],dist[20];

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

scanf("%d",&n);

printf("\n enter the 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]=infinity;

}

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

scanf("%d",&v);

dij(n,v,cost,dist);

printf("\n shortest path : \n");

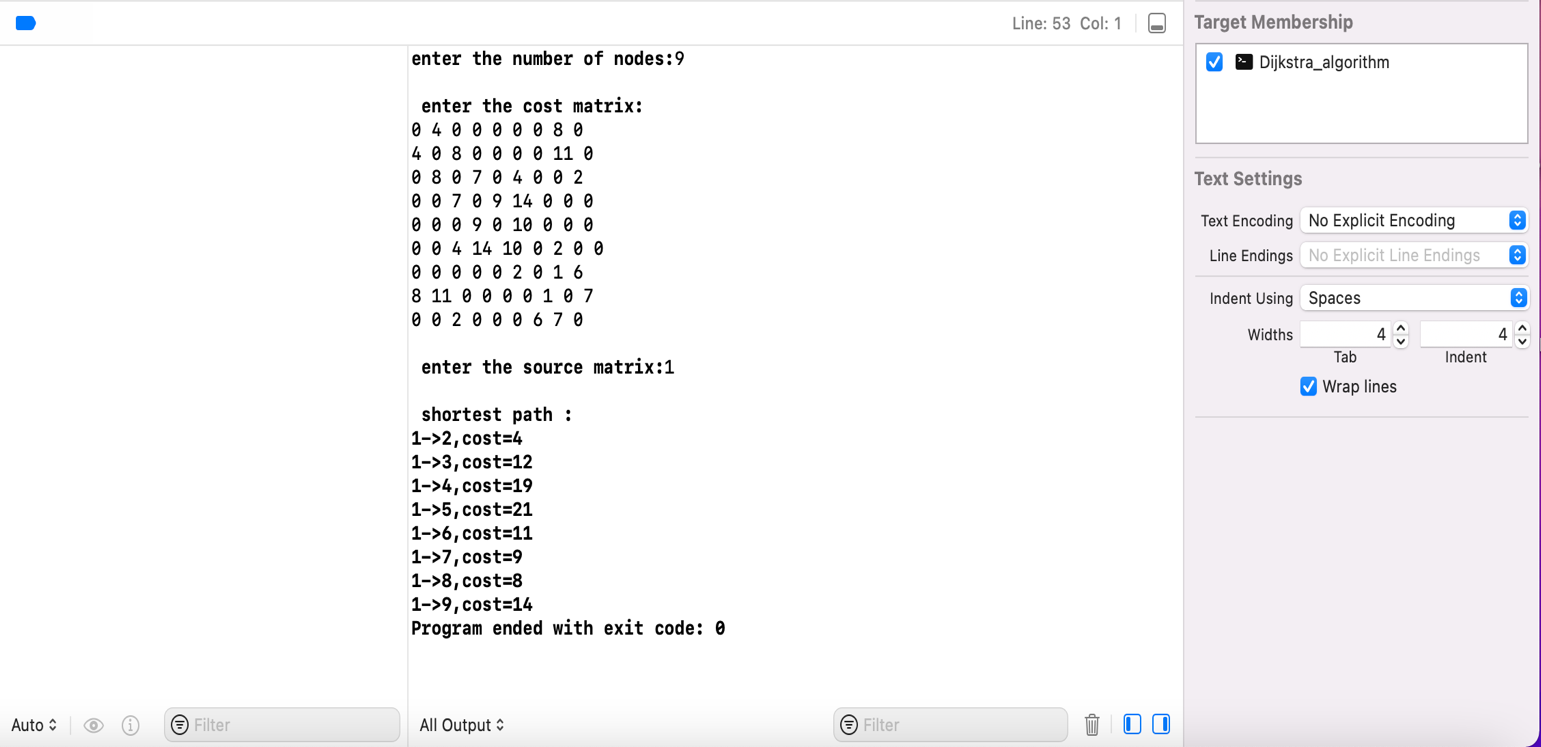
**for**(i=1;i<=n;i++)

**if**(i!=v)

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

}

**OUTPUT:-**

****

**Q8. Implement 0/1 Knapsack problem using Dynamic Programming.**

**PROGRAM:-**

//

// main.c

// Knapsack

//

// Created by Harsh vikram Shahi on 15/12/21.

//

#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** knap(**int** i,**int** j){

**int** value;

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

**if**(j<w[i])

value=knap(i-1,j);

**else**

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

v[i][j]=value;

}

**return**(v[i][j]);

}

**int** main(){

**int** profit,count=0;

printf("\nEnter the number of objects ");

scanf("%d",&n);

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

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

{

printf("\nEnter profit and weight For object no %d :",i);

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

}

printf("\nEnter the capacity");

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=knap(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("objectincluded are \n ");

printf("Sl.no\tweight\tprofit\n");

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

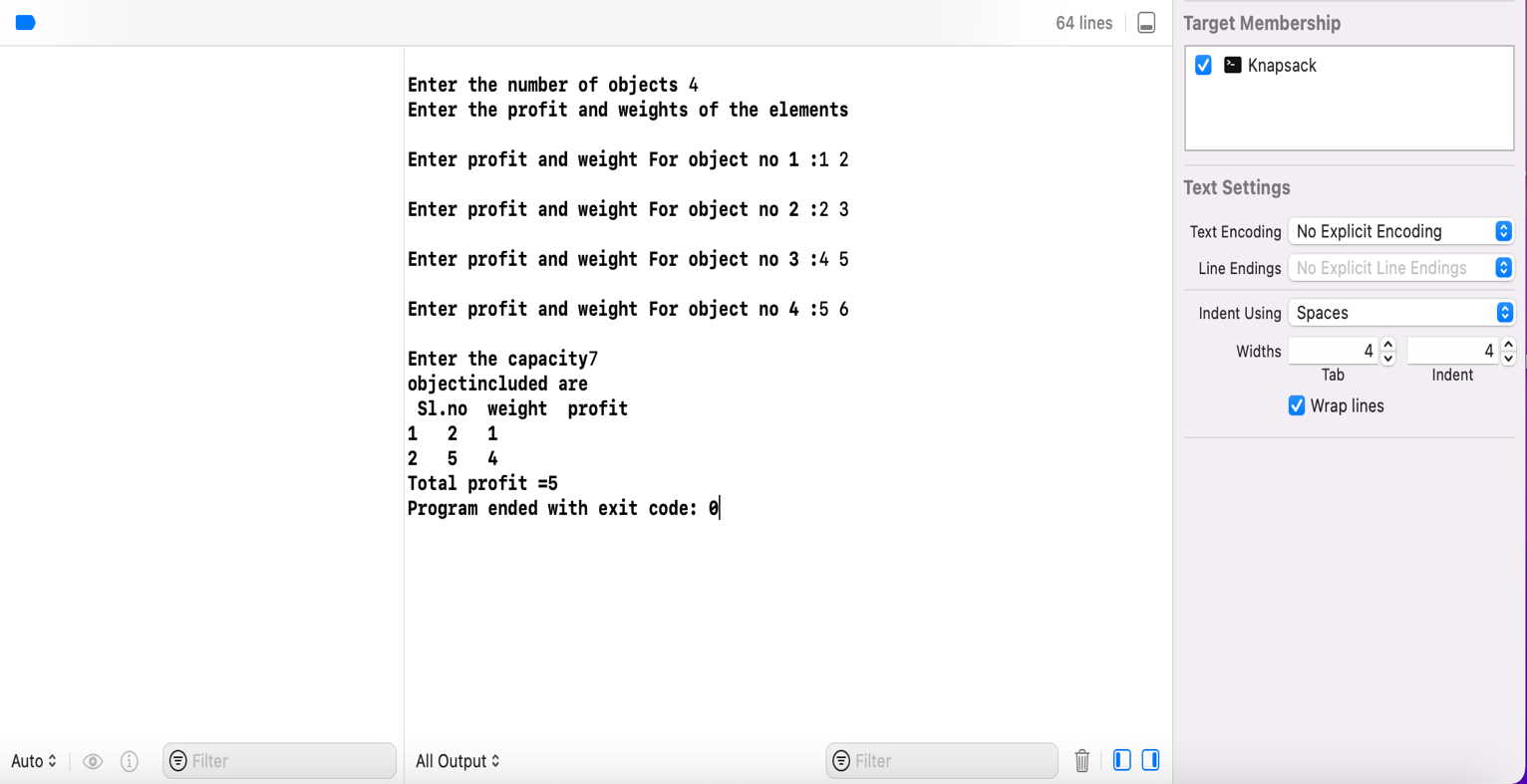
**if**(x[i])

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

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

}

**OUTPUT:-**

****