**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**Analysis and Design of Algorithms**

***Submitted by***

**NAYANTARA K KUMAR (1BM20CS228)**

***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**

**May-2022 to July-2022**

**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 **NAYANTARA K KUMAR (1BM20CS228),** 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 year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

**Rekha G S**             **Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
| **1** | Write a recursive program to Solve  **a)** Towers-of-Hanoi problem **b)** To find GCD | **1** |
| **2** | Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N. | **3** |
| **3** | Sort a given set of N integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. | **7** |
| **4** | 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. | **9** |
| **5** | Sort a given set of N integer elements using Insertion Sort technique and compute its time taken. | **13** |
| **6** | Write program to obtain the Topological ordering of vertices in a given digraph. | **15** |
| **7** | Implement Johnson Trotter algorithm to generate permutations. | **17** |
| **8** | 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. | **22** |
| **9** | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | **25** |
| **10** | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | **28** |
| **11** | Implement Warshall’s algorithm using dynamic programming | **31** |
| **12** | Implement 0/1 Knapsack problem using dynamic programming. | **33** |
| **13** | Implement All Pair Shortest paths problem using Floyd’s algorithm. | **36** |
| **14** | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. | **38** |
| **15** | Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm. | **41** |
| **16** | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | **44** |
| **17** | 1. 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. | **47** |
| **18** | Implement “N-Queens Problem” using Backtracking. | **50** |

**Course Outcome**

|  |  |
| --- | --- |
| **CO1** | Ability to **analyze** time complexity of Recursive and Non-Recursive algorithms using asymptotic notations. |
| **CO2** | Ability to **design** efficient algorithms using various design techniques. |
| **CO3** | Ability to **apply** the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete |
| **CO4** | Ability to **conduct** practical experiments to solve problems using an appropriate designing method and find time efficiency. |

**Program 1**

**Write a recursive program to**

1. **Solve Towers-of-Hanoi problem b. To find GCD**

**Tower of Hanoi Program**

#include<stdio.h>

#include<stdlib.h>

void toh(int n, char src, char des, char aux){

if(n==1){

printf("Move disc from %c to %c\n",src, des);

}

else{

toh(n-1, src, aux, des);

printf("Move disc from %c to %c\n", src, des);

toh(n-1 ,aux, des, src);

}

}

void main(){

int n;

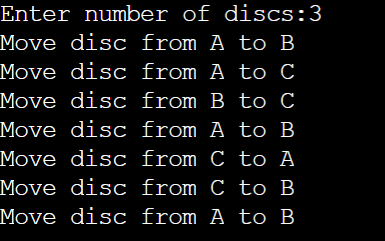
printf("Enter number of discs:");

scanf("%d",&n);

toh(n,'A','B','C');

}

**Result**

****

**GCD Program**

#include<stdio.h>

#include<stdlib.h>

int gcd(int m,int n){

if(n==0)

return m;

else

gcd(n,m%n);

}

void main(){

int m,n;

printf("Enter 2 numbers:");

scanf("%d%d",&m,&n);

printf("GCD=%d",gcd(m,n));

}

**Result**



**Program 2**

**Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N**

**Linear Search Program**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void delay(){

int i,n;

for(i=0;i<5000;i++)

n=3000/13;

}

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

delay();

int indx;

n--;

if(n>=0){

if(a[n]==key)

return n;

else{

indx=linearSearch(a,n,key);

}

}

else return -1;

return indx;

}

void main(){

clock\_t start,end;

int n=3000;

int arr[n];

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

arr[i]=n-i;

int key;

printf("Enter key:");

scanf("%d",&key);

start=clock();

int idx=linearSearch(arr,n,key);

end=clock();

if(idx!=-1)

printf("Element found at index %d",idx);

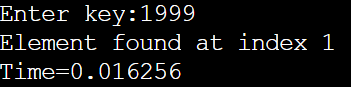
else

printf("Element not found");

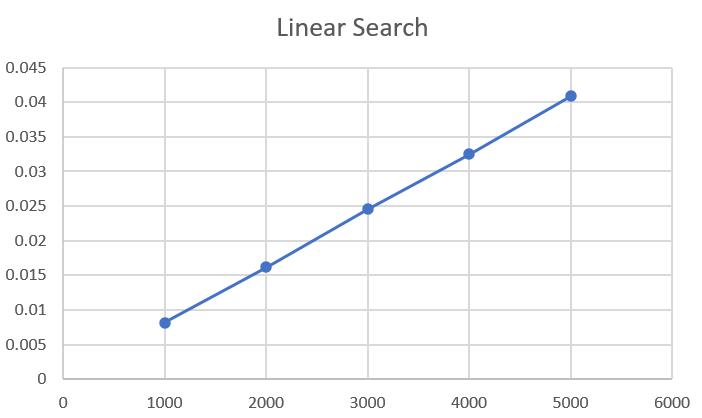
printf("\nTime=%f",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**Result**



**Graph**

****

**Binary Search Program**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void delay(){

int i,n;

for(i=0;i<500000;i++)

n=3000/13;

}

int binarySearch(int a[],int l,int r,int key){

delay();

int indx;

if(r>=l){

int m=l+(r-l)/2;

if(a[m]==key)

return m;

else if(key<a[m])

indx=binarySearch(a,l,m-1,key);

else

indx=binarySearch(a,m+1,r,key);

}

else return -1;

}

void main(){

clock\_t start,end;

int n=5000;

int arr[n];

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

arr[i]=i;

int key;

printf("Enter key:");

scanf("%d",&key);

start=clock();

int idx=binarySearch(arr,0,n-1,key);

end=clock();

if(idx!=-1)

printf("Element found at index %d",idx);

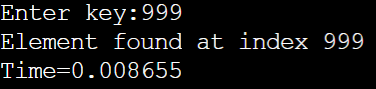
else

printf("Element not found");

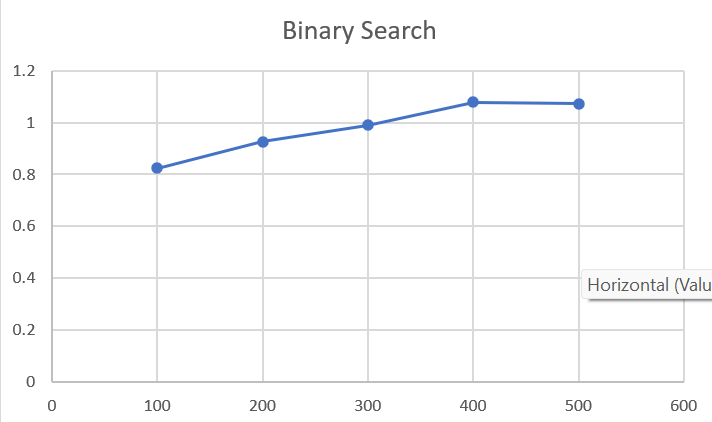
printf("\nTime=%f",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**Result**

****

**Graph**

****

**Program 3**

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

**Program**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void delay(){

int i,n;

for(i=0;i<500;i++)

n=3000/13;

}

int main(){

int n,i;

clock\_t start,end;

printf("Enter number of elements to be sorted:");

scanf("%d",&n);

int arr[n];

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

arr[i]=n-i;

start=clock();

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

int min=i;

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

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

delay();

min=j;

}

int temp=arr[i];

arr[i]=arr[min];

arr[min]=temp;

}

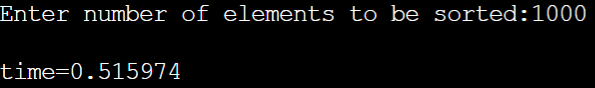
}

end=clock();

printf("\ntime=%f",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**Result**

****

**Graph**

****

**Program 4**

**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**.

**BFS Program**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int n;

int adj[MAX][MAX],state[MAX],queue[MAX];

int front=-1,rear=-1;

void create\_graph();

void BF\_traversal();

void BFS();

void insert\_queue(int);

int isEmpty();

int deleteQueue();

void main(){

create\_graph();

BF\_traversal();

}

void create\_graph(){

int count,max,src,dest;

printf("Enter number of vertices in graph:");

scanf("%d",&n);

max=n\*(n-1);

for(count=1;count<max;count++){

printf("Enter edge %d (-1 -1) to exit:",count);

scanf("%d %d",&src,&dest);

if(src==-1&&dest==-1)

break;

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

printf("Invalid edge");

count--;

}

else{

adj[src][dest]=1;

}

}

}

void BF\_traversal(){

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

state[i]=initial;

printf("Enter start vertex:");

int v;

scanf("%d",&v);

BFS(v);

}

void BFS(int v){

int i,j;

insert\_queue(v);

state[v]=waiting;

while(!isEmpty()){

i=deleteQueue();

printf("%d\t",i);

state[i]=visited;

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

if(adj[i][j]==1 && state[j]==initial){

insert\_queue(j);

state[j]=waiting;

}

}

}printf("\n");

}

void insert\_queue(int v){

if(front==-1)

front++;

rear++;

queue[rear]=v;

}

int isEmpty(){

if(front==-1||front>rear)

return 1;

else

return 0;

}

int deleteQueue(){

int item;

if(front==-1||front>rear)

exit(1);

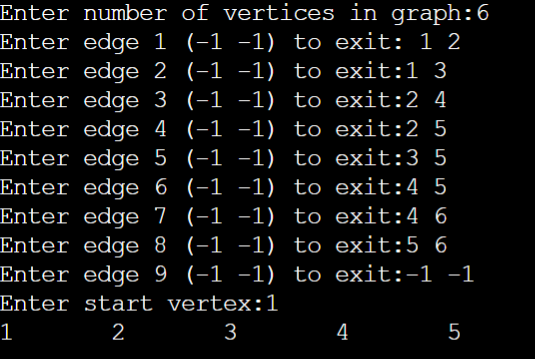
item=queue[front];

front++;

return item;

}

**Result**

****

**DFS Program**

#include<stdio.h>

void DFS(int);

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

void main(){

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

scanf("%d",&n);

int i,j;

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

visited[i]=0;

printf("\n Enter adjacency matrix of graph:\n");

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

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

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

}

printf("DFS:");

DFS(0);

}

void DFS(int v){

int i;

printf("%d\t",v);

visited[v]=1;

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

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

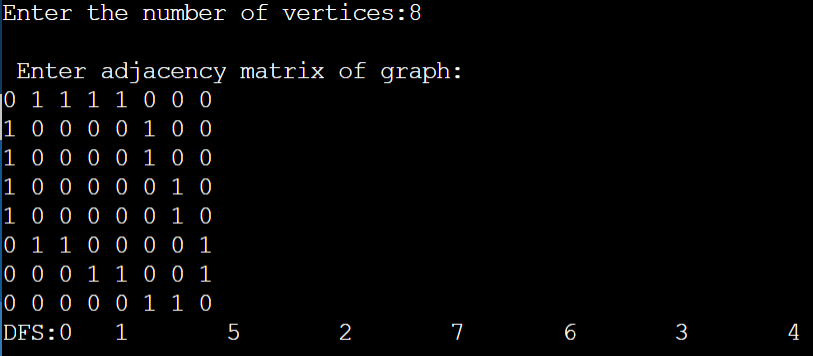
DFS(i);

}

}

}

**Result**

****

**Program 5**

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

**Program**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void delay(){

int temp;

for(int i=0;i<500;i++)

temp=3000/13;

}

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

int i;

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

int key=a[i];

int j=i-1;

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

delay();

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

j--;

}

a[j+1]=key;

}

}

int main(){

clock\_t start,end;

int n=5000;

int arr[n];

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

arr[i]=n-i;

start=clock();

insertionSort(arr,n);

end=clock();

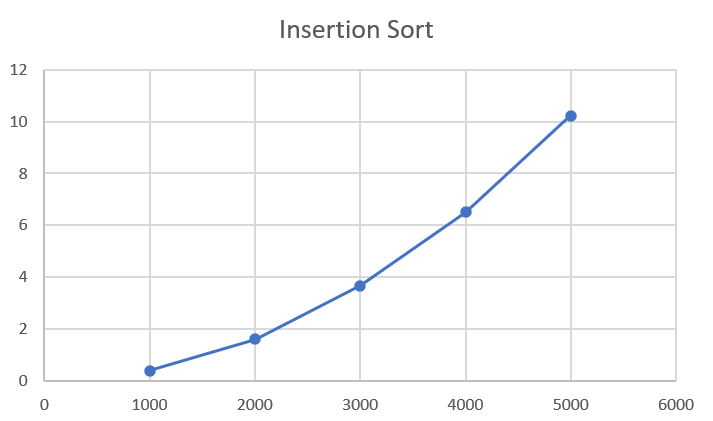
printf("Time taken=%f",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**Result**



**Graph**



**Program 6**

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

**Program**

#include<stdio.h>

#include<stdlib.h>

int main(){

int count=0;

printf("Enter number of vertices:");

int n;

scanf("%d",&n);

int a[n][n];

printf("Enter adjacency matrix:");

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

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

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

int indegree[n];

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

indegree[i]=0;

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

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

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

}

while (count<n){

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

if(indegree[i]==0){

printf("%d\t",i+1);

indegree[i]=-1;

count++;

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

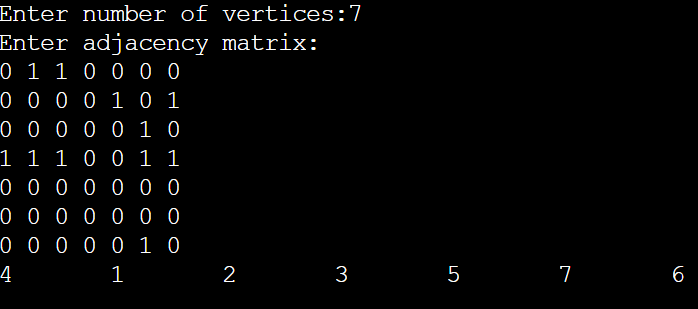
if(a[i][k]==1)

indegree[k]--;

}}}}

}

**Result**



**Program 7**

**Implement Johnson Trotter algorithm to generate permutations**

**Program**

#include <stdio.h>

#include <stdlib.h>

int flag = 0;

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+1;

}

else

{

flag++;

}

}

return -1;

}

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

{

int mobile = 0;

int mobile\_p = 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\_p)

{

mobile = arr[i];

mobile\_p = mobile;

}

else

{

flag++ ; }

}

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

{

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

{

mobile = arr[i];

mobile\_p = mobile;

}

else

{

flag++;

}

}

else

{

flag++;

}

}

if((mobile\_p == 0) && (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]-1]==0)

swap(&arr[pos-1],&arr[pos-2]);

else

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

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);

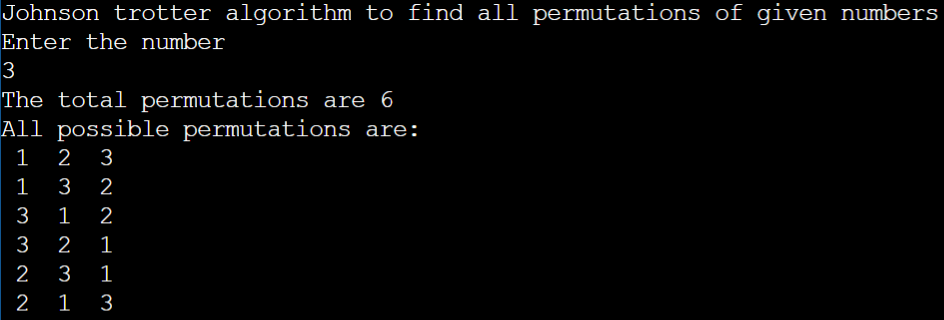
printf("\n");

}

return 0;

}

**Result**



**Program 8**

**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.**

**Program**

#include <stdio.h>

#include <stdlib.h>

#include<time.h>

void delay(){

int i,n;

for(i=0;i<2000;i++)

n=3000/13;

}

void merge(int a[],int l,int m,int r){

int i,j,k;

int n1=m-l+1;

int n2=r-m;

int L[n1],R[n2];

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

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

for(j=0;j<n2;j++)

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

i=0;

j=0;

k=l;

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

if(L[i]<=R[j]){

a[k]=L[i];

k++;

i++;

}

else{

a[k]=R[j];

k++;

j++;

}

}

while(i<n1){

a[k]=L[i];

k++;

i++;

}

while(j<n2){

a[k]=R[j];

k++;

j++;

}

}

void mergeSort(int a[],int l,int r){

if(l<r){

delay();

int m=l+(r-l)/2;

mergeSort(a,l,m);

mergeSort(a,m+1,r);

merge(a,l,m,r);

}

}

void main(){

clock\_t start,end;

int n=1000;

int a[n];

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

a[i]=n-i;

start=clock();

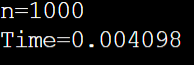
mergeSort(a,0,n-1);

end=clock();

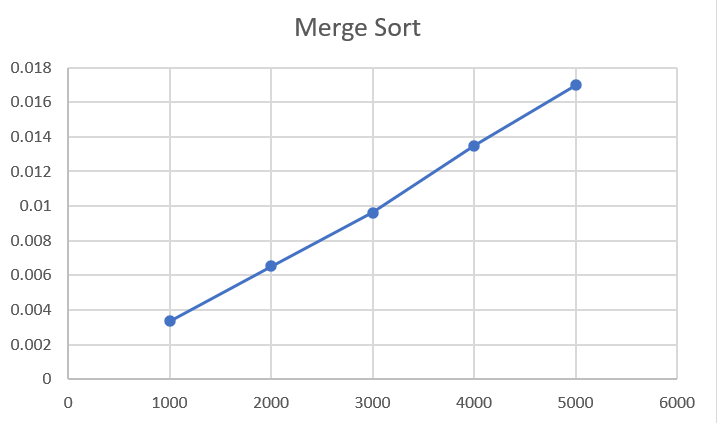
printf("Time=%f",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**Result**

****

**Graph**

****

**Program 9**

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

**Program**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void delay(){

int i,n;

for(i=0;i<5000;i++)

n=3000/13;

}

void quickSort(int a[],int first,int last){

int i,j,p,temp;

if(first<last){

p=first;

i=first;

j=last;

while(i<j){

while(a[i]<=a[p]&&i<last)

i++;

while(a[j]>a[p])

j--;

if(i<j){

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

temp=a[p];

a[p]=a[j];

a[j]=temp;

quickSort(a,first,j-1);

quickSort(a,j+1,last);

}

}

void main(){

clock\_t start,end;

int n=1000;

int a[n];

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

a[i]=n-i;

start=clock();

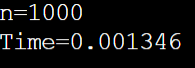
quickSort(a,0,n-1);

end=clock();

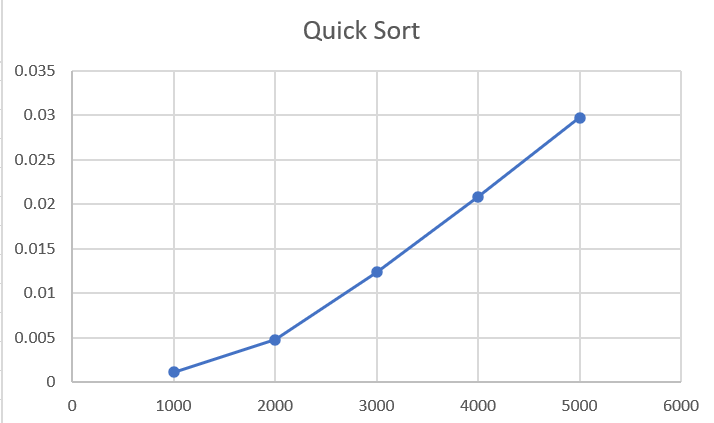
printf("Time=%f",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**Result**

****

**Graph**

****

**Program 10**

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

**Program**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void delay(){

int i,n;

for(i=0;i<5000;i++){

n=3000/13;

}

}

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

int temp=\*a;

\*a=\*b;

\*b=temp;

}

void heapify(int arr[],int N,int i){

delay();

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);

}

}

void main(){

clock\_t start,end;

printf("Enter number of elements:");

int N,i;

scanf("%d",&N);

int arr[N];

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

arr[i]=rand()%(N+1);

start=clock();

heapsort(arr,N);

end=clock();

printf("Sorted array:");

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

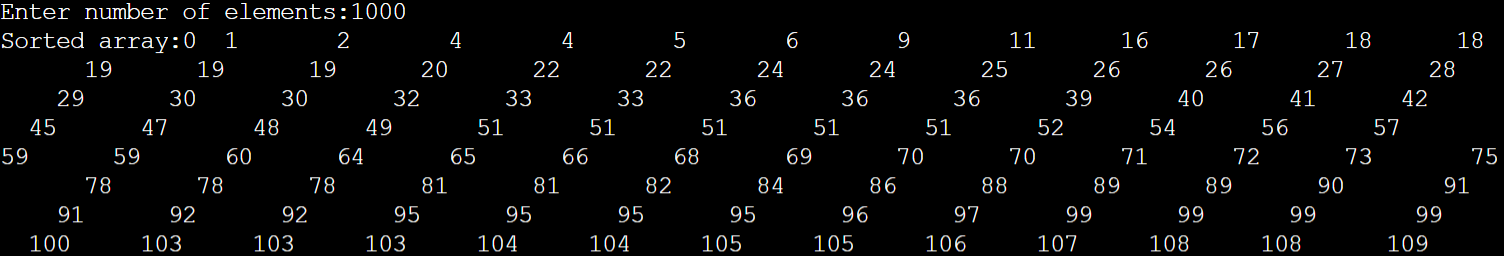
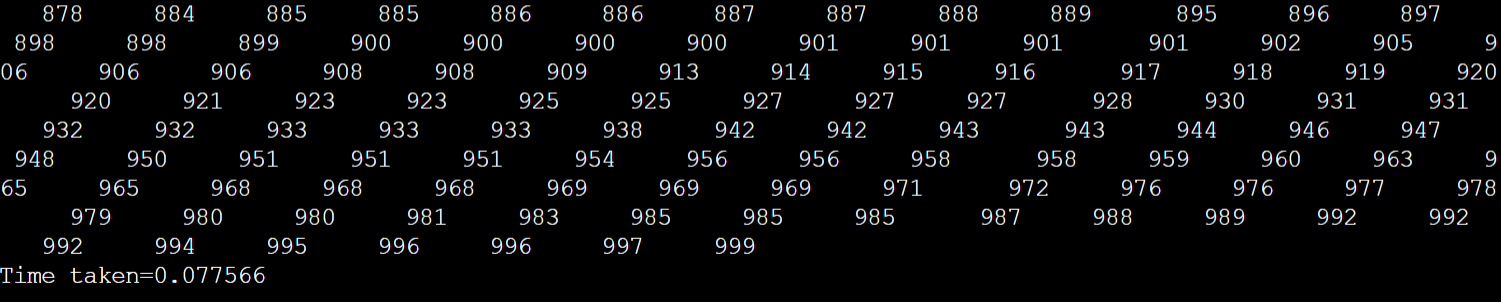
printf("%d\t",arr[i]);

}

printf("\nTime taken=%f",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**Result**

**** ****

**Graph**

**Program 11**

**Implement Warshall’s algorithm using dynamic programming.**

**Program**

#include<stdio.h>

#include<stdlib.h>

void main(){

printf("Enter number of vertices:");

int n;

scanf("%d",&n);

printf("Enter adjacent matrix:");

int a[n][n];

int i,j;

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

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

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

}

int k=0;

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

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

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

a[i][j]=a[i][j]||(a[i][k]&&a[k][j]);

}

}

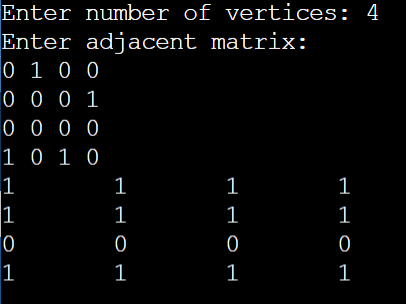
for(i=0;i<n;i++){

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

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

printf("\n");}}

**Result**



**Program 12**

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

**Program**

#include<stdio.h>

#include<conio.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]);

}

void main() {

int profit,count=0;

printf("\nEnter 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("For item no %d\n",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=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("Items included are\n");

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

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

if(x[i])

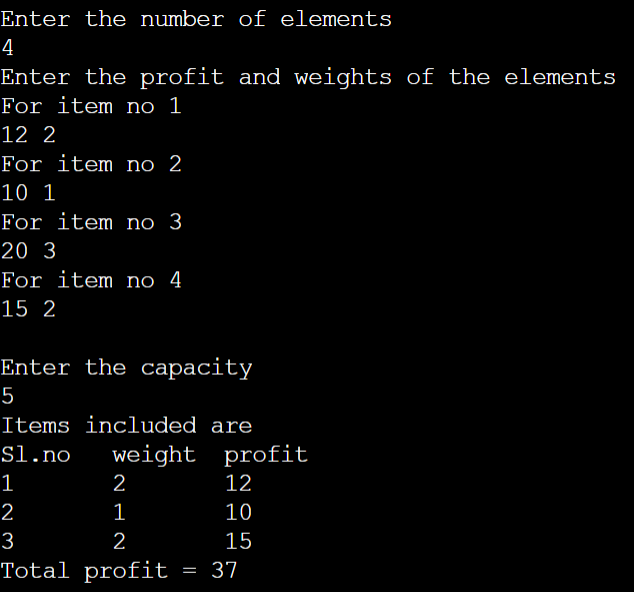
printf("%d\t%d\t%d\n",++count,w[i],p[i]);

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

getch();

}

**Result**



**Program 13**

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

**Program**

#include<stdio.h>

#include<stdlib.h>

void main(){

printf("Enter number of vertices:");

int n;

scanf("%d",&n);

printf("Enter weighted matrix:");

int a[n][n];

int i,j;

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

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

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

}

int k=0;

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

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

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

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

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

}

}

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

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

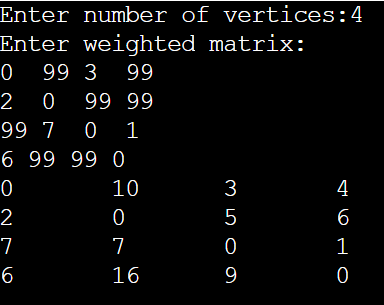
printf("%d\t",a[i][j]);

printf("\n");

}

}

**Result**



**Program 14**

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

**Program**

#include<stdio.h>

#include<stdlib.h>

int G[10][10];

int V;

int minKEY(int key[],int visited[]){

int min=999;

int min\_index;

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

if(visited[v]==0&&key[v]<min){

min=key[v];

min\_index=v;

}

}

return min\_index;

}

void printMST(int parent[]){

printf("\nEdge\tWeight");

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

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

}

}

void prim(){

int parent[V];

int key[V];

int visited[V];

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

key[i]=999;

visited[i]=0;

}

key[0]=0;

parent[0]=-1;

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

int u=minKEY(key,visited);

visited[u]=1;

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

if(G[u][v]&&visited[v]==0&&G[u][v]<key[v]){

parent[v]=u;

key[v]=G[u][v];

}

}

}

printMST(parent);

}

void main(){

printf("Enter number of vertices:");

scanf("%d",&V);

printf("Enter adjacency matrix:");

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

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

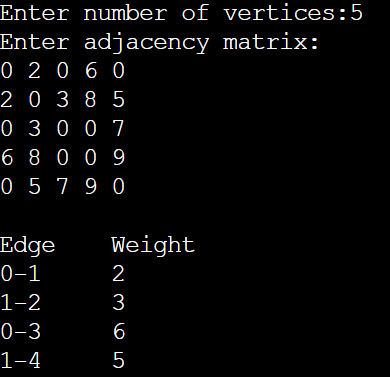
scanf("%d",&G[i][j]);

}

prim();

}

**Result**



**Program 15**

**Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.**

**Program**

#include <stdio.h>

#include <conio.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);

void main()

{

printf("\n\tImplementation of Kruskal's Algorithm\n");

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

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("The edges of Minimum Cost Spanning Tree are\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("%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);

getch();

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

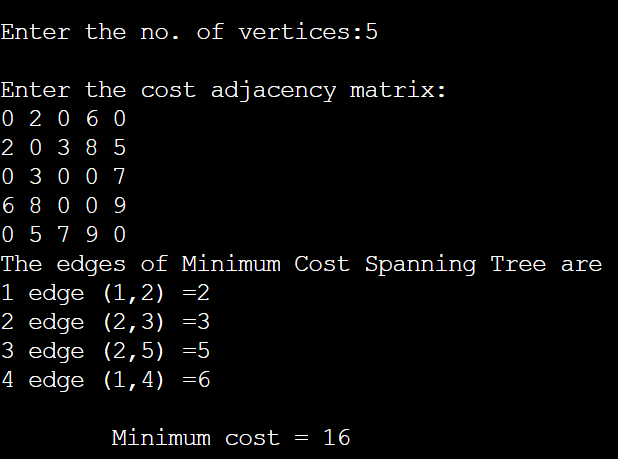
return 1;

}

return 0;

}

**Result**



**Program 16**

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

**Program**

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

void main(){

int G[MAX][MAX], i, j, n, u;

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

scanf("%d", &n);

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

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

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

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

printf("\nEnter the starting node: ");

scanf("%d", &u);

dijkstra(G,n,u);

getch();

}

void dijkstra(int G[MAX][MAX], int n, int startnode)

{ int cost[MAX][MAX], distance[MAX], pred[MAX];

int visited[MAX], count, mindistance, nextnode, i,j;

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

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

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

cost[i][j]=INFINITY;

else

cost[i][j]=G[i][j];

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

{ distance[i]=cost[startnode][i];

pred[i]=startnode;

visited[i]=0;

}

distance[startnode]=0;

visited[startnode]=1;

count=1;

while(count < n-1){

mindistance=INFINITY;

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

if(distance[i] < mindistance&&!visited[i])

{ mindistance=distance[i];

nextnode=i;

}

visited[nextnode]=1;

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

if(!visited[i])

if(mindistance+cost[nextnode][i] < distance[i])

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode;}

count++;}

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

if(i!=startnode) {

printf("\nDistance of %d = %d", i, distance[i]);

printf("\nPath = %d ", i);

j=i;

do{

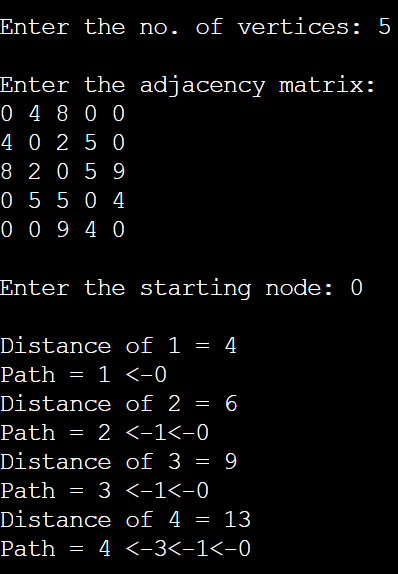
j=pred[j];

printf("<-%d", j);

}

while(j!=startnode);}}

**Result**



**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.**

**Program**

#include<stdio.h>

#include<conio.h>

#define TRUE 1

#define FALSE 0

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

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

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

}

void sumset(int i,int wt,int total);

void main() {

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

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

scanf("%d",&n);

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

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

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

total+=w[i];

}

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

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

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

if((total<sum))

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

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

inc[i]=0;

printf("\nSolution:\n");

sumset(-1,0,total);

}

getch();

}

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

int j;

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

if(wt==sum) {

printf("{");

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

if(inc[j])

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

printf("}\n");

} else {

inc[i+1]=TRUE;

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

inc[i+1]=FALSE;

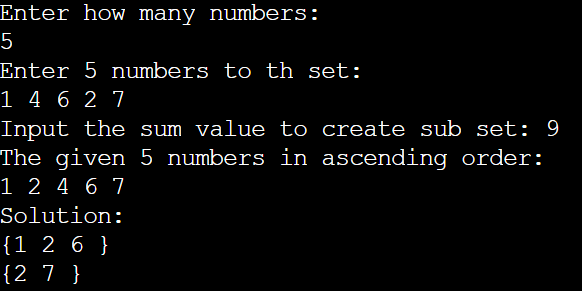
sumset(i+1,wt,total-w[i+1]);

}

}

}

**Result**



**Program 18**

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

**Program**

#include<stdio.h>

#include<conio.h>

#define TRUE 1

#define FALSE 0

#include<stdio.h>

#include<conio.h>

void nqueens(int n)

{

int k,x[20],count=0;

k=1;

x[k]=0;

while(k!=0)

{

x[k]++;

while(place(x,k)!=1 && x[k]<=n)

x[k]++;

if(x[k]<=n)

{

if(k==n)

{

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

printf("Queen\t\tPosition\n");

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

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

}

else

{

k++;

x[k]=0;

}

}

else

k--;

}

}

int place(int x[], int k)

{

int i;

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

{

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

return 0;

}

return 1;

}

void main()

{

int n;

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

scanf("%d", &n);

nqueens(n);}

**Result**

