**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

Analysis and Design of Algorithms

***Submitted by***

**UMA DEVI S A (1BM21CS413)**

***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 **UMA DEVI S A (1BM21CS413)** 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.

Name of the Lab-Incharge: **Manjunath D R**               **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 a. Solve Towers-of-Hanoi problem b. To find GCD |  |
| **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.** | 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. |  |
| **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. |  |
| **5.** | Sort a given set of N integer elements using Insertion Sort technique and compute its time taken. |  |
| **6.** | Write program to obtain the Topological ordering of vertices in a given digraph. |  |
| **7.** | Implement Johnson Trotter algorithm to generate permutations |  |
| **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. |  |
| **9.** | Sort a given set of N integer elements using Quick Sort technique and compute its time taken |  |
| **10.** | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. |  |
| **11.** | Implement Warshall’s algorithm using dynamic programming. |  |
| **12.** | Implement 0/1 Knapsack problem using dynamic programming. |  |
| **13.** | Implement All Pair Shortest paths problem using Floyd’s algorithm. |  |
| **14.** | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. |  |
| **15** | Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm. |  |
| **16.** | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. |  |

**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. |
| C04 | Ability to conduct practical experiments to solve problems using an appropriate designing method and find time efficiency. |

**Experiment-01**

Write a recursive program to Solve Towers-of-Hanoi problem

#include<stdio.h>

#include<conio.h>

#include<math.h>

void hanoi(int x, char from, char to, char aux)

{

if(x==1)

printf("Move Disk From %c to %c\n",from,to);

else

{

hanoi(x-1,from,aux,to);

printf("Move Disk From %c to %c\n",from,to);

hanoi(x-1,aux,to,from);

}

}

void main( )

{

int disk;

int moves;

printf("Enter the number of disks you want to play with:");

scanf("%d",&disk);

moves=pow(2,disk)-1;

printf("\nThe No of moves required is=%d \n",moves);

hanoi(disk,'A','C','B');

}

OUTPUT:

Enter the number of disks you want to play with:3

The No of moves required is=7

Move Disk From A to C

Move Disk From A to B

Move Disk From C to B

Move Disk From A to C

Move Disk From B to A

Move Disk From B to C

Move Disk From A to C

Write a recursive program to find GCD.

#include <stdio.h>

int GCD(int n1, int n2);

int main()

{

int n1, n2;

printf("Enter two positive integers: ");

scanf("%d %d", &n1, &n2);

printf("G.C.D of %d and %d is %d.", n1, n2, GCD(n1,n2));

return 0;

}

int GCD(int n1, int n2)

{

if (n2 != 0)

return hcf(n2, n1%n2);

else

return n1;

}

OUTPUT:

Enter two positive integers: 8 16

G.C.D of 8 and 16 is 8.

**Experiment-02**

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.

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

int key;

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

{

int mid;

mid=((low+high)/2);

if(low>high)

return -1;

if(key==a[mid])

return mid;

else

if(key<a[mid])

return binary(a,0,mid-1);

else

return binary(a,mid+1,high);

}

void main()

{

int a[30000],n,key,pos,i,t;

clock\_t end,start;

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

scanf("%d",&n);

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

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

{

a[i]=rand()%1000;

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

}

printf("\nEnter the element to be searched");

scanf("%d",&key);

start=clock();

pos=binary(a,0,n-1);

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

t=900/900;

if(pos==-1)

printf("Element is not found in an array\n");

else

printf("Element is found in an array\n");

end=clock();

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

LINEAR SERACH

#include <stdio.h>

#include<time.h>

#include<stdlib.h>

int linear(int arr[],int key,int i,int n)

{

int pos;

if(key>=n)

return -1;

else

if(arr[i]==key)

{

pos=i+1;

return pos;

}

else

return linear(arr,key,i+1,n);

return pos;

}

void main()

{

int n,key,pos,a[30000],i,t;

clock\_t end,start;

printf("Enter the number of elements in the array ");

scanf("%d", &n);

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

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

{

a[i]=i;

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

}

printf("Enter the element to search ");

scanf("%d", &key);

start=clock();

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

t=800/800;

pos=linear(a,key,0,n);

if (pos!=-1)

printf("Element found at pos %d ", pos);

else

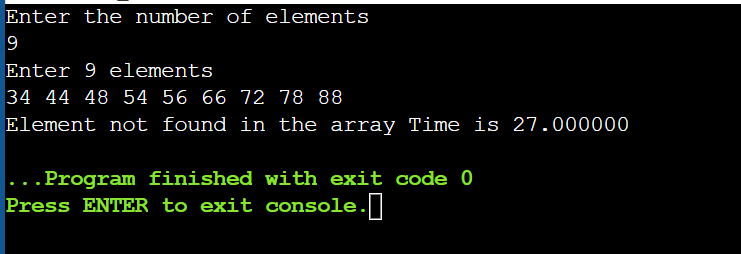
printf("Element not found");

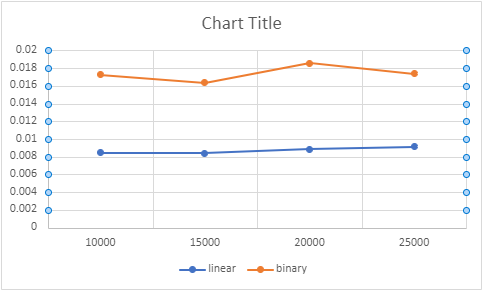
end=clock();

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

}

OUTPUT:





**Experiment-03**

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.

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<time.h>

void selection\_sort(int n,int a[])

{

int i,j,t,s,pos;

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

{

pos=i;

s=a[i];

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

{

if(a[j]<s)

{

s=a[j];

pos=j;

}

}

t=a[i];

a[i]=a[pos];

a[pos]=t;

}

}

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()%50;

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

}

start=clock();

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

t=900/900;

selection\_sort(n,a);

printf("Sorted array:\n");

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

{

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

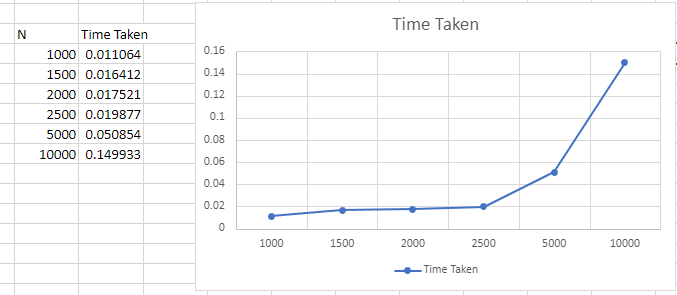
}

end=clock();

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

}

OUTPUT:



**Experiment-04**

Write program to do the following:

1. Print all the nodes reachable from a given starting node in a digraph using BFS method.
2. Check whether a given graph is connected or not using DFS method.

#include<stdio.h>

int a[20][20],vis[20],n;

void dfs(int v)

{

int i;

vis[v]=1;

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

if(a[v][i] && !vis[i])

{

printf("\n %d->%d",v,i);

dfs(i);

}

}

void main()

{

int i,j,count=0;

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

scanf("%d",&n);

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

{

vis[i]=0;

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

a[i][j]=0;

}

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

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

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

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

dfs(1);

printf("\n");

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

{

if(vis[i])

count++;

}

if(count==n)

printf("\n Graph is connected");

else

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

}

OUTPUT:

Enter number of vertices:4

Enter the adjacency matrix:

0 1 0 0

0 0 1 0

0 0 0 1

1 0 0 0

1->2

2->3

3->4

Graph is connected

**BFS**

#include<stdio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v)

{

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

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

q[++r]=i;

if(f<=r)

{

visited[q[f]]=1;

bfs(q[f++]);

}

}

void main()

{

int v;

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

scanf("%d",&n);

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

{

q[i]=0;

visited[i]=0;

}

printf("\n Enter graph data in matrix form:\n");

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

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

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

}

printf("\n Enter the starting vertex:");

scanf("%d",&v);

bfs(v);

printf("\n The node which are reachable are:\n");

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

if(visited[i])

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

}

OUTPUT:

Enter the number of vertices:4

Enter graph data in matrix form:

0 1 1 1

0 0 0 1

0 0 0 0

0 0 1 0

Enter the starting vertex:1

The node which are reachable are:

2 3 4

**Experiment-05**

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

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<time.h>

void insertion\_sort(int n,int array[])

{

int item,i,j;

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

{

item=array[i];

j=i-1;

while(item<array[j] && j>=0)

{

array[j+1]=array[j];

j--;

}

array[j+1]=item;

}

}

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;

insertion\_sort(n,a);

printf("Sorted array:\n");

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

{

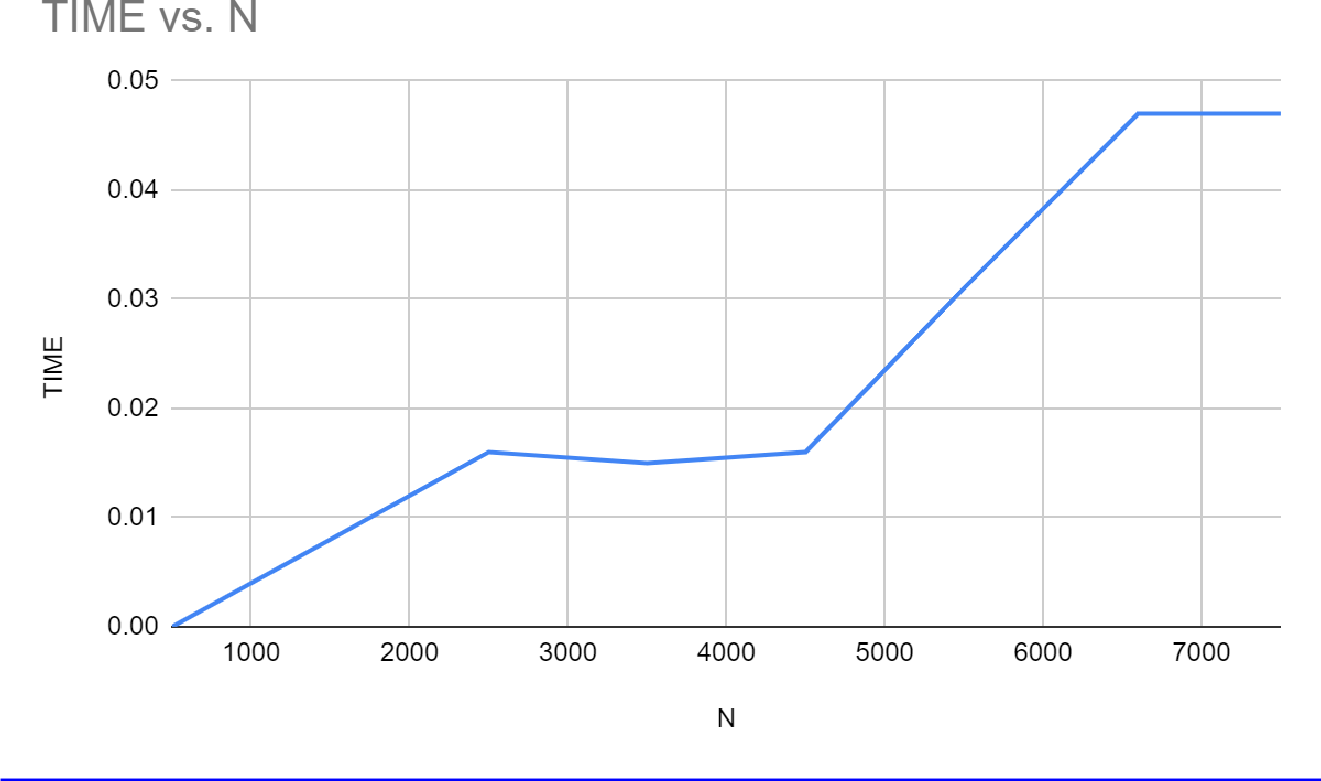
printf("%d\n",a[i]);

end=clock();

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

}

OUTPUT:



**Experiment-06**

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:

Enter number of nodes:

4

Enter the adjacency matrix

0 1 1 0

0 0 0 1

0 0 0 1

0 0 0 0

0

2

1

3

**Experiment-07**

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:

Johnson trotter algorithm to find all permutations of given numbers

Enter the number

3

The total permutations are 6

All possible permutations are:

1 2 3

1 3 2

3 1 2

3 2 1

2 3 1

2 1 3

**Experiment-08**

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>

#include<conio.h>

#include<stdlib.h>

#include<time.h>

#define MAX 50

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

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

void main()

{

int array[MAX],size,i;

int clock\_t,start\_t,end\_t;

double timeTaken;

printf("Enter the size of an array...\n");

scanf("%d",&size);

printf("Before sorting array elements are...\n");

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

{

array[i]=rand()%100;

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

}

start\_t=clock();

delay(100);

mergeSort(array,0,size-1);

printf("\nAfter sorting array elements are...\n");

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

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

end\_t=clock();

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

printf("\nTimetaken to sort array of %d is %0.2f\n",size,timeTaken);

}

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

{

int mid,i;

if(low<high)

{

mid=(low+high)/2;

mergeSort(a,low,mid);

mergeSort(a,mid+1,high);

simpleMerge(a,low,mid,high);

}

}

void simpleMerge(int a[],int low,int mid,int high)

{

int i=low,j=mid+1,k=low;

int c[50];

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

{

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

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

else

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

}

while(i<=mid)

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

while(j<=high)

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

for(i=low;i<=high;i++)

a[i]=c[i];

}

OUTPUT:

Enter the size of an array...

10

Before sorting array elements are...

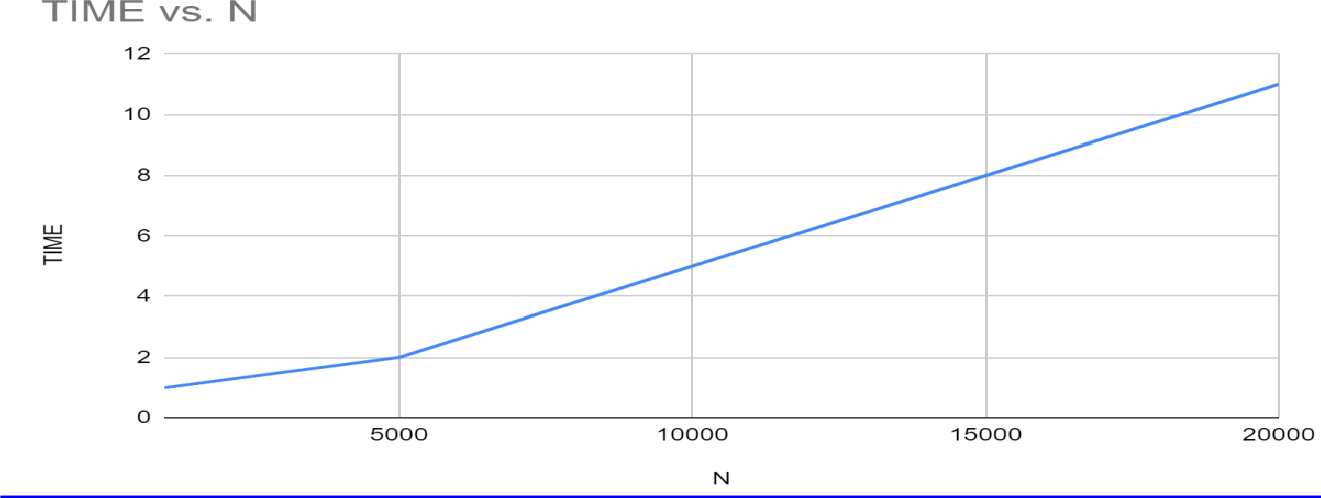
83 86 77 15 93 35 86 92 49 21

After sorting array elements are...

15 21 35 49 77 83 86 86 92 93

Timetaken to sort array of 10 is 0.00123

GRAPH:



**Experiment-09**

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:

Enter the size of an array...

10

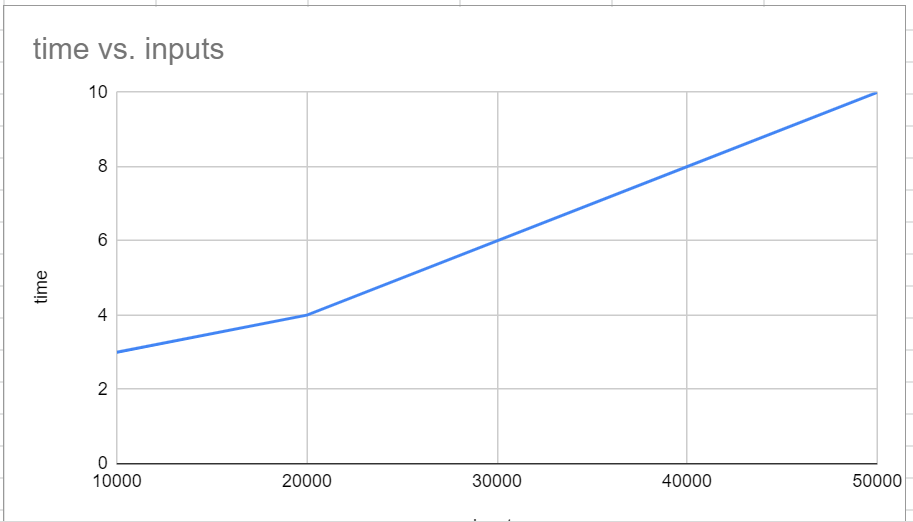
Before sorting array elements are...

83 86 77 15 93 35 86 92 49 21

After sorting array elements are...

15 21 35 49 77 83 86 86 92 93

Timetaken to sort array of 10 is 0.00013

GRAPH:

**Experiment-10**

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

#include<stdio.h>

#include <conio.h>

#include<stdlib.h>

#include<time.h>

void heapify\_function(int arr[])

{

int i,n;

n=arr[0];

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

adjust(arr,i);

}

void adjust(int arr[],int i)

{

int j,temp,n,k=1;

n=arr[0];

while(2\*i<=n && k==1)

{

j=2\*i;

if(j+1<=n && arr[j+1] > arr[j])

j=j+1;

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

k=0;

else

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

i=j;

}

}

}

void main()

{

int arr[10000],n,temp,last,i;

clock\_t start,end;

printf("Enter the no. of array elements \n");

scanf("%d",&n);

printf("Enter Elements in array:\n");

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

{

arr[i]=rand()%100;

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

}

arr[0]=n;

start = clock();

heapify\_function(arr);

end = clock();

while(arr[0] > 1)

{

last=arr[0];

temp=arr[1];

arr[1]=arr[last];

arr[last]=temp;

arr[0]--;

adjust(arr,1);

}

printf("\nArray After Heap Sort\n");

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

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

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

}

OUTPUT:

Enter the no. of array elements

10

Enter Elements in array:

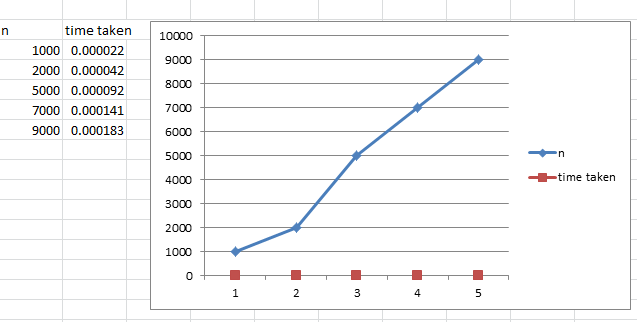
83 86 77 15 93 35 86 92 49 21

Array After Heap Sort

15 21 35 49 77 83 86 86 92 93

Time taken to sort is 0.000001

GRAPH:



**Experiment-11**

Implement Warshall’s algorithm using dynamic programming.

#include<stdio.h>

#include<conio.h>

int a[10][10];

void main()

{

int i,j,k,n;

printf("\n enter the number of vertices\n");

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]);

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

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

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

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

printf("\n\t the tranitive closure is\n");

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

{

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

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

printf("\n");

}

}

OUTPUT:

enter the number of vertices

4

enter the adjacency matrix

0 1 0 0

0 0 0 1

1 0 1 0

0 0 0 0

the tranitive closure is

0 1 0 1

0 0 0 1

1 1 1 1

0 0 0 0

**Experiment-12**

Implement 0/1 Knapsack problem using dynamic programming.

#include<stdio.h>

#include<conio.h>

void knapsack();

int max(int,int);

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

void main()

{

printf("\nEnter the number of items:\n");

scanf("%d",&n);

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

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

{

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

}

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

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

{

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

}

printf("\n\Enter the knapsack's 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("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("Optimal solution is %d",v[n][m]);

printf("The 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:

Enter the number of items:

4

Enter the weight of the each item:

2 1 3 2

Enter the profit of each item:

12 10 20 15

Enter the knapsack's capacity: 5

Output is:

0 0 0 0 0 0

0 0 12 12 12 12

0 10 12 22 22 22

0 10 12 22 30 32

0 10 15 25 30 37

Optimal solution is 37

The solution vector is:

1 1 0 1

**Experiment-13**

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

#include<stdio.h>

#include<conio.h>

int a[10][10],n;

void floyd();

int min(int,int);

void main()

{

int i,j;

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

scanf("%d",&n);

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

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

{

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

{

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

}

}

floyd();

}

void floyd()

{

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("All pair of 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:

Enter the number of vertices

4

Enter the adjacency matrix:

9999 9999 3 9999

2 9999 9999 9999

9999 7 9999 1

6 9999 9999 9999

All pair of shortest path matrix is:

10 10 3 4

2 12 5 6

7 7 10 1

6 16 9 10

**Experiment-14**

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

#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"); 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;

}

}

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

while(no\_e<n)

{

min=1000;

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(visited[b]==0)

{

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

min\_cost=min\_cost+min;

no\_e++;

}

visited[b]=1;

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

}

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

return 0;

}

OUTPUT:

Enter number of nodes 6

Enter cost in form of adjacency matrix

0 3 1 6 0 0

3 0 5 0 3 0

1 5 0 5 6 4

6 0 5 0 0 2

0 3 6 0 0 6

0 0 4 2 6 0

1 to 3 cost=1

1 to 2 cost=3

2 to 5 cost=3

3 to 6 cost=4

6 to 4 cost=2

minimum weight is 13

**Experiment-15**

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

#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:

enter the no. of vertices: 6

enter the cost matrix:

9999 3 1 6 9999 9999

3 9999 5 9999 3 9999

1 5 9999 5 6 4

6 9999 5 9999 9999 2

9999 3 6 9999 9999 6

9999 9999 4 2 6 9999

1----->3=1

4----->6=2

1----->2=3

2----->5=3

3----->6=4

mincost=13

**Experiment-16**

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:

enter the no of vertices: 5

enter the cost matrix:

9999 3 9999 7 9999

3 9999 4 2 9999

9999 4 9999 5 6

7 2 5 9999 4

9999 9999 6 4 9999

enter the source node: 3

the shortest distance is:

3----->1=7

3----->2=4

3----->3=0

3----->4=5

3----->5=6