

Katihar Engineering College, Katihar
Department of Science & Technology, Govt. of Bihar



Department of Computer Science & Engineering
(Aryabhatta Knowledge University, Patna Bihar)

Design and Analysis of Algorithm Lab
Code: PCC CS 404P
4th Semester B.Tech. CSE (2020)

Submitted By

Name: Sameer Kumar

Branch & Sem.: CSE 4th Sem

Reg. No.: 18105129008

Submitted to:

Md. Talib Ahmad
(Assistant Professor)

INDEX

S.No.	Topic	Page No.
1	Quick Sort	3 – 4
2	Merge Sort	5 – 6
3	Topological ordering of vertices in a given digraph	7 - 8
4	Warshall's algorithm	9 - 10
5	0/1 Knapsack	11 – 13
6	Dijkstra's algorithm	14 – 15
7	Kruskal's algorithm	16 - 17
8	BFS	18 – 19
9	DFS	20
10	Prism Algorithm	21 – 22

Practical 1

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

```
#include <stdio.h>
#include <conio.h>
#include <time.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, key, k;
    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);
}

Void main()
{
    Int n,a[1000],k;
    clock_t st,et;
    double ts;
    clrscr();
```

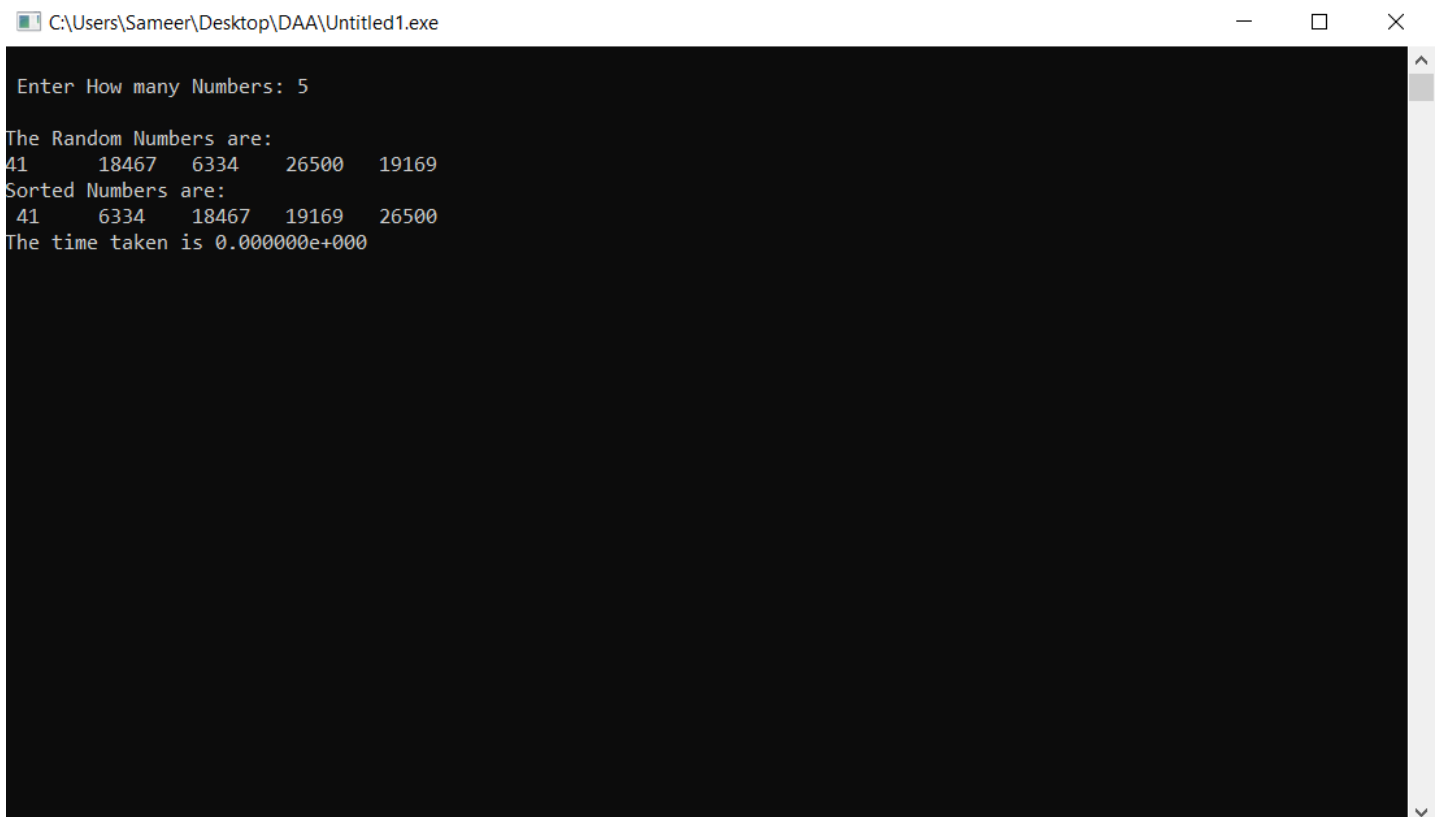
```

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); getch();
}

```

Output :



```

C:\Users\Sameer\Desktop\DAA\Untitled1.exe
Enter How many Numbers: 5
The Random Numbers are:
41      18467  6334   26500   19169
Sorted Numbers are:
41      6334   18467   19169   26500
The time taken is 0.000000e+000

```

PRACTICAL 2

2. Using OpenMP, implement a parallelized 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 and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
#include <stdio.h>
#include <conio.h>
#include <time.h>

void Merge(int a[], int low, int mid, int high)
{
    int i, j, k, b[20];
    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);
}
void main()
{

int n, a[2000],k; clock_t st,et;
double 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();
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);
getch();
}

```

PRACTICAL 3

3. Obtain the Topological ordering of vertices in a given digraph.

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n,indeg[10];
void find_indeg()
{
    int j,i,sum;
    for(j=0;j<n;j++)
    {
        sum=0;
        for(i=0;i<n;i++)
            sum+=a[i][j];
        indeg[j]=sum;
    }
}
void topology()
{
    int i,u,v,t[10],s[10],top=-1,k=0;
    find_indeg();
    for(i=0;i<n;i++)
    {
        if(indeg[i]==0) s[++top]=i;
    }
    while(top!=-1)
    {
        u=s[top--]; t[k++]=u; for(v=0;v<n;v++)
        {
            if(a[u][v]==1)
            {
                indeg[v]--; if(indeg[v]==0) s[++top]=v;
            }
        }
    }
    printf("The topological Sequence is:\n");
    for(i=0;i<n;i++)
        printf("%d ",t[i]);
```

```

}
void main()
{
int i,j;
printf("Enter number of jobs:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
}
topology(); getch();
}

```

Output :

```

Enter number of jobs:6
Enter the adjacency matrix:
0      0      1      1      0      0
0      0      0      1      1      0
0      0      0      1      0      1
0      0      0      0      0      1
0      0      0      0      0      1
0      0      0      0      0      0
The topological Sequence is:
1 4 0 2 3 5

```


PRACTICAL 4

4. Compute the transitive closure of a given directed graph using Warshall's algorithm.

```
#include <stdio.h>
#include <conio.h>
int n,a[10][10],p[10][10];
void path()
{
    int i,j,k;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            p[i][j]=a[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",&a[i][j]);
    path();
    printf("\nThe path matrix is showm below\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
```

```
printf("%d ",p[i][j]);  
printf("\n");  
}  
}
```

Output :

```
Enter the number of nodes:4  
Enter the adjacency matrix:  
0 1 0 0  
0 0 1 0  
0 0 0 1  
0 0 0 0  
  
The path matrix is shown below  
0 1 1 1  
0 0 1 1  
0 0 0 1  
0 0 0 0
```

PRACTICAL 5

5. Implement 0/1 Knapsack problem using Dynamic Programming.

```
#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);

}

```

Output :

```
Enter the number of elements
3
Enter the profit and weights of the elements
For item no 1
10      30
For item no 2
20      15
For item no 3
30      50

Enter the capacity
45
Items included are
Sl.no   weight  profit
1       30     10
2       15     20
Total profit = 30
```

PRACTICAL 6

6. From a given vertex in a weighted connected graph, find show paths to other vertices using Dijkstra's algorithm.

```
#include<stdio.h>
#include<conio.h>
#define infinity 999
void dij(int n,int v,int cost[10][10],int dist[100])
{
int i,u,count,w,flag[10],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];
}
}

void main()
{
int n,v,i,j,cost[10][10],dist[10];
printf("\n 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 :

```

Enter the number of nodes:5

Enter the cost matrix:
0      5      12     17     999
999     0      999     8       7
999     999     0       9     999
999     999     999     0     999
999     999     999     999    0

Enter the source matrix:1

Shortest path:
1->2,cost=5
1->3,cost=12
1->4,cost=13
1->5,cost=12

```

PRACTICAL 7

7. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

```
#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\n\tImplementation 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 :

```

Implementation of Kruskal's algorithm

Enter the no. of vertices
4

Enter the cost adjacency matrix
0      20      10      50
20      0      60      999
10      60      0      40
50      999      40      0

The edges of Minimum Cost Spanning Tree are

1 edge (1,3) =10
2 edge (1,2) =20
3 edge (3,4) =40

Minimum cost = 70

```

PRACTICAL 8

8. Print all the nodes reachable from a given starting node in a digraph using BFS method.

```
#include<stdio.h>
#include<conio.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      -
```

PRACTICAL 9

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

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
    int i; reach[v]=1;
    for(i=1;i<=n;i++)
        if(a[v][i] && !reach[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++){
        reach[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(reach[i]) count++;
    }
    if(count==n)
        printf("\n Graph is connected");
    else
        printf("\n Graph is not connected");
}
```

PRACTICAL 10

10.Find Minimum Cost Spanning Tree of a given undirected graph using Prism's algorithm.

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void 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) cost:%d",ne++,a,b,min);
mincost+=min;
visited[b]=1;
}
}
```

```
cost[a][b]=cost[b][a]=999;  
}  
printf("\n Minimun cost=%d",mincost);  
}
```

Output :

```
Enter the number of nodes:4  
Enter the adjacency matrix:  
0      20     10     50  
20      0      60     999  
10      60      0      40  
50      999     40      0  
  
Edge 1:(1 3) cost:10  
Edge 2:(1 2) cost:20  
Edge 3:(3 4) cost:40  
Minimum cost=70
```