ADA lab manual for cseise

Analysis & Design of Algorithms (Visvesvaraya Technological University)

1. **Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected 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("Kruskal's algorithm in C\n");
        printf("========================\n");
        printf("Enter 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("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("\nMinimum 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;
}
```

OUTPUT 1

```
Kruskal's algorithm in C
========================
Enter the no. of vertices:
4

Enter the cost adjacency matrix:
0 3 4 1
5 0 3 2
1 2 0 6
8 5 7 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,4) =1
2 edge (3,1) =1
3 edge (2,4) =2

Minimum cost = 4
_
```

OUTPUT 2

```
Kruskal's algorithm in C
========================
Enter the no. of vertices:
6

Enter the cost 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
The edges of Minimum Cost Spanning Tree are
1 edge (1,3) =1
2 edge (4,6) =2
3 edge (1,2) =3
4 edge (2,5) =3
5 edge (3,6) =4

Minimum cost = 13
_
```

2. Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim'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()
 {
        clrscr();
        printf("\nEnter 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;
        }
        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);
        getch();

 }
```

OUTPUT 1

```
Enter the number of nodes: 6

Enter the 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


 Edge 1:(1 3) cost:1
 Edge 2:(1 2) cost:3
 Edge 3:(2 5) cost:3
 Edge 4:(3 6) cost:4
 Edge 5:(6 4) cost:2
 Minimun cost=13_
```

OUTPUT 2

```
Enter the number of nodes:6

Enter the adjacency matrix:
0 4 0 0 0 2
4 0 6 0 0 3
0 6 0 3 0 1
0 0 3 0 2 0
0 0 0 2 0 4
2 3 1 0 4 0


 Edge 1:(1 6) cost:2
 Edge 2:(6 3) cost:1
 Edge 3:(3 4) cost:3
 Edge 4:(4 5) cost:2
 Edge 5:(6 2) cost:3
 Minimun cost=11_
```

3. a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.

```
#include<stdio.h>
#include<iostream>
#include<omp.h>
#include<conio.h>
void floyd(int[10][10],int);
int min(int,int);
void main()
{
int n,a[10][10],i,j;
printf("Enter the no.of nodes : ");
scanf("%d",&n);
printf("\nEnter the cost adjacency matrix\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
floyd(a,n);
getch();
}
void floyd(int a[10][10],int n)
{
int d[10][10],i,j,k;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
d[i][j]=a[i][j];
}
#pragma omp parallel for
for(k=1;k<=n;k++)
{
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
}
}
}
printf("\nThe distance matrix is\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf("%d\t",d[i][j]);
}
printf("\n");
}
}
```

```
int min (int a,int b)
{
if(a<b)
return a;
else
return b;
}
```

OUTPUT 1

```
Enter the no.of nodes : 4

Enter the cost adjacency matrix
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0

The distance matrix is
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0
```

OUTPUT 2

```
Enter the no.of nodes : 5

Enter the cost adjacency matrix
0 2 999 1 8
6 0 3   2 999
999 999 0 4 999
999 999 2 0 3
3 999 999 999 0

The distance matrix is
0       2       3       1       4
6       0       3       2       5
10      12      0       4       7
6       8       2       0       3
3       5       6       4       0
```

b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.

```
#include<stdio.h>
void warshall(int[10][10],int);
void main()
{
        int a[10][10],i,j,n;
        clrscr();
        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",&a[i][j]);
        printf("The adjacency matirx is:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        printf("%d\t",a[i][j]);
                }
                printf("\n");
        }
        warshall(a,n);
        getch();
}
void warshall(int p[10][10],int n)
{
        int i,j,k;
        for(k=1;k<=n;k++)
        {
                for(j=1;j<=n;j++)
                {
                        for(i=1;i<=n;i++)
                        {
                                if((p[i][j]==0) && (p[i][k]==1) && (p[k][j]==1))
                                {
                                        p[i][j]=1;
                                }
                        }
                }
        }
printf("\nThe path matrix is:\n");
for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
        {
                printf("%d\t",p[i][j]);
        }
```

```
            printf("\n");
            }
    }
```

```
Enter the number of nodes:5

Enter the adjacency matrix:
0 1 1 0 1
0 0 1 0 0
0 0 0 1 0
0 0 0 0 0
0 0 0 1 0
The adjacency matirx is:
0       1       1       0       1
0       0       1       0       0
0       0       0       1       0
0       0       0       0       0
0       0       0       1       0

The path matrix is:
0       1       1       1       1
0       0       1       1       0
0       0       0       1       0
0       0       0       0       0
0       0       0       1       0
-
```

OUTPUT 2

```
Enter the number of nodes:4

Enter the adjacency matrix:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
The adjacency matirx is:
0       1       0       0
0       0       0       1
0       0       0       0
1       0       1       0

The path matrix is:
1       1       1       1
1       1       1       1
0       0       0       0
1       1       1       1
```

4. Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

```
/*To find shortest paths to other vertices using Dijkstra's algorithm.*/
#include<stdio.h>
void dij(int,int [20][20],int [20],int [20],int);
void main()
{
        int i,j,n,visited[20],source,cost[20][20],d[20];
        clrscr();
        printf("Enter no. of vertices: ");
        scanf("%d",&n);
        printf("Enter the cost adjacency matrix\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);
                }
        }
        printf("\nEnter the source node: ");
        scanf("%d",&source);
        dij(source,cost,visited,d,n);
        for(i=1;i<=n;i++)
        if(i!=source)
        printf("\nShortest path from %d to %d is %d",source,i,d[i]);
        getch();
}
void dij(int source,int cost[20][20],int visited[20],int d[20],int n)
{
        int i,j,min,u,w;
        for(i=1;i<=n;i++)
        {
                visited[i]=0;
                d[i]=cost[source][i];
        }
        visited[source]=1;
        d[source]=0;
        for(j=2;j<=n;j++)
        {
                min=999;
                for(i=1;i<=n;i++)
                {
                        if(!visited[i])
                        {
                                if(d[i]<min)
                                {
                                        min=d[i];
                                        u=i;
                                }
```

```
                        }
                } //for i
        visited[u]=1;
        for(w=1;w<=n;w++)
        {
        if(cost[u][w]!=999 && visited[w]==0)
        {
                if(d[w]>cost[u][w]+d[u])
                d[w]=cost[u][w]+d[u];
        }
         } //for w
        } // for j
}
```

```
Enter no. of vertices: 6
Enter the cost adjacency matrix
999 3 999 999 6 5
3 999 1 999 999 4
999 1 999 6 999 4
999 999 6 999 8 5
6 999 999 8 999 2
5 4 4 5 2 999

Enter the source node: 1

Shortest path from 1 to 2 is 3
Shortest path from 1 to 3 is 4
Shortest path from 1 to 4 is 10
Shortest path from 1 to 5 is 6
Shortest path from 1 to 6 is 5_
```

OUTPUT 1

5. **Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.**

```
/*To obtain the topological order in of vertices in a digraph.*/
#include<stdio.h>
void findindegree(int [10][10],int[10],int);
void topological(int,int [10][10]);
void main()
{
        int a[10][10],i,j,n;
        clrscr();
        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",&a[i][j]);
        printf("\nThe adjacency matirx is:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        printf("%d\t",a[i][j]);
                }
                printf("\n");
        }
        topological(n,a);
        getch();
}
void findindegree(int a[10][10],int indegree[10],int n)
{
        int i,j,sum;
        for(j=1;j<=n;j++)
        {
                sum=0;
                for(i=1;i<=n;i++)
                {
                        sum=sum+a[i][j];
                }
                indegree[j]=sum;
        }
}
void topological(int n,int a[10][10])
{
        int k,top,t[100],i,stack[20],u,v,indegree[20];
        k=1;
        top=-1;
        findindegree(a,indegree,n);
        for(i=1;i<=n;i++)
```

```
        {
                if(indegree[i]==0)
                {
                        stack[++top]=i;
                }
        }
                while(top!=-1)
        {
                u=stack[top--];
                t[k++]=u;
                for(v=1;v<=n;v++)
                {
                        if(a[u][v]==1)
                        {
                                indegree[v]--;
                                if(indegree[v]==0)
                                {
                                        stack[++top]=v;
                                }
                        }
                }
        }
printf("\nTopological sequence is\n");
for(i=1;i<=n;i++)
printf("%d\t",t[i]);
}
```

OUTPUT 1

```
Enter the number of nodes:5

Enter the adjacency matrix
0 0 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0

The adjacency matirx is:
0       0       1       0       0
0       0       1       0       0
0       0       0       1       1
0       0       0       0       1
0       0       0       0       0

Topological sequence is
2       1       3       4       5
```

6. **Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.**

```
#include<stdio.h>
#define MAX 50
int p[MAX],w[MAX],n;
int knapsack(int,int);
int max(int,int);
void main()
{
int m,i,optsoln;
clrscr();
printf("Enter no. of objects: ");
scanf("%d",&n);
printf("\nEnter the weights:\n");
for(i=1;i<=n;i++)
scanf("%d",&w[i]);
printf("\nEnter the profits:\n");
for(i=1;i<=n;i++)
scanf("%d",&p[i]);
printf("\nEnter the knapsack capacity:");
scanf("%d",&m);
optsoln=knapsack(1,m);
printf("\nThe optimal soluntion is:%d",optsoln);
getch();
}
int knapsack(int i,int m)
{
if(i==n)
return (w[n]>m) ? 0 : p[n];
if(w[i]>m)
return knapsack(i+1,m);
return max(knapsack(i+1,m),knapsack(i+1,m-w[i])+p[i]);
}
int max(int a,int b)
{
if(a>b)
return a;
else
return b;
}
```

```
Enter no. of objects: 3

Enter the weights:
100 14 10

Enter the profits:
20 18 15

Enter the knapsack capacity:116

The optimal solution is:38
```

7. **Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method**

```
#include<stdio.h>
int main()
{
    float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
    int n,i,j;
    printf("Enter the number of items :");
    scanf("%d",&n);
   for (i = 0; i < n; i++)
   {
      printf("Enter Weight and Profit for item[%d] :\n",i);
      scanf("%f %f", &weight[i], &profit[i]);
   }
   printf("Enter the capacity of knapsack :\n");
   scanf("%f",&capacity);

    for(i=0;i<n;i++)
       ratio[i]=profit[i]/weight[i];

   for (i = 0; i < n; i++)
    for (j = i + 1; j < n; j++)
      if (ratio[i] < ratio[j])
      {
         temp = ratio[j];
         ratio[j] = ratio[i];
         ratio[i] = temp;

         temp = weight[j];
         weight[j] = weight[i];
         weight[i] = temp;

         temp = profit[j];
         profit[j] = profit[i];
         profit[i] = temp;
      }

    printf("Knapsack problems using Greedy Algorithm:\n");
    for (i = 0; i < n; i++)
    {
     if (weight[i] > capacity)
        break;
      else
     {
        Totalvalue = Totalvalue + profit[i];
        capacity = capacity - weight[i];
      }
    }
      if (i < n)
```

```
    Totalvalue = Totalvalue + (ratio[i]*capacity);
  printf("\nThe maximum value is :%f\n",Totalvalue);
  return 0;
}
```

OUTPUT 1

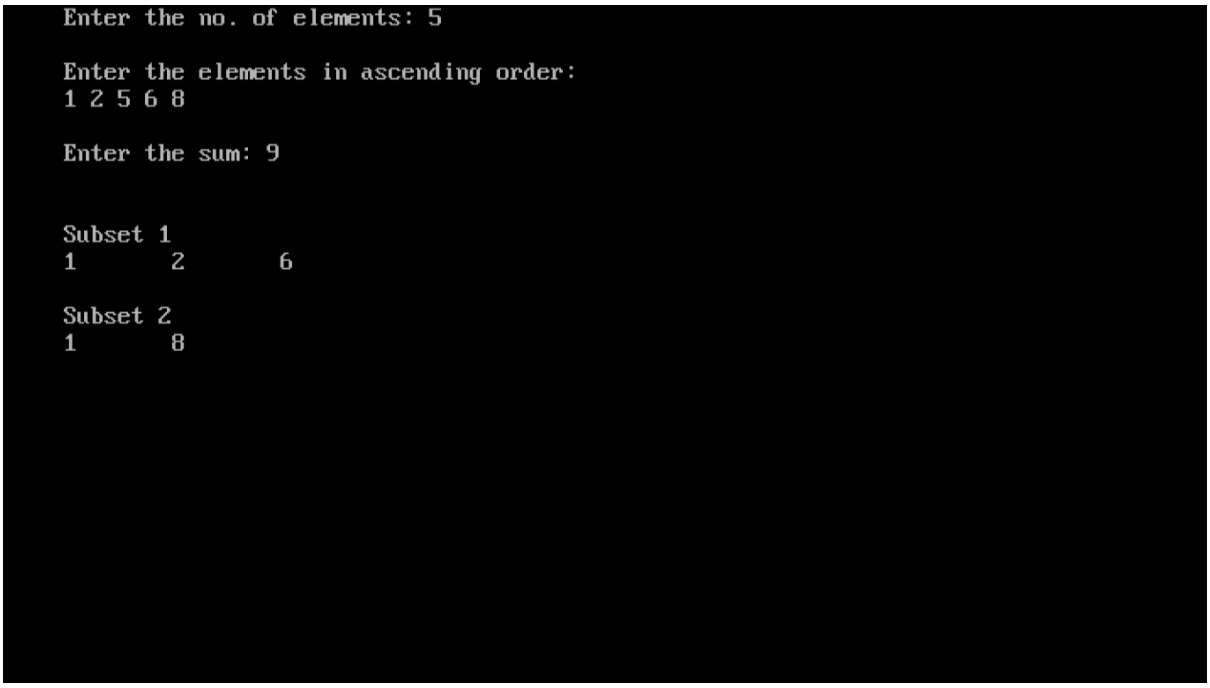
```
Enter the number of items :
4
Enter Weight and Profit for item[0] :
2
12
Enter Weight and Profit for item[1] :
1
10
Enter Weight and Profit for item[2] :
3
20
Enter Weight and Profit for item[3] :
2
15
Enter the capacity of knapsack :
5
Knapsack problems using Greedy Algorithm:

The maximum value is :38.333332
Enter the number of items :
```

8. **Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.**

```
#include<stdio.h>
void subset(int,int,int);
int x[10],w[10],d,count=0;
void main()
{
int i,n,sum=0;
clrscr();
printf("Enter the no. of elements: ");
scanf("%d",&n);
printf("\nEnter the elements in ascending order:\n");
for(i=0;i<n;i++)
scanf("%d",&w[i]);
printf("\nEnter the sum: ");
scanf("%d",&d);
for(i=0;i<n;i++)
sum=sum+w[i];
if(sum<d)
{
printf("No solution\n");
getch();
return;
}
subset(0,0,sum);
if(count==0)
{
printf("No solution\n");
getch();
return;
}
getch();
}
void subset(int cs,int k,int r)
{
int i;
x[k]=1;
if(cs+w[k]==d)
{
printf("\n\nSubset %d\n",++count);
for(i=0;i<=k;i++)
if(x[i]==1)
printf("%d\t",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]);
}
}
```

OUTPUT

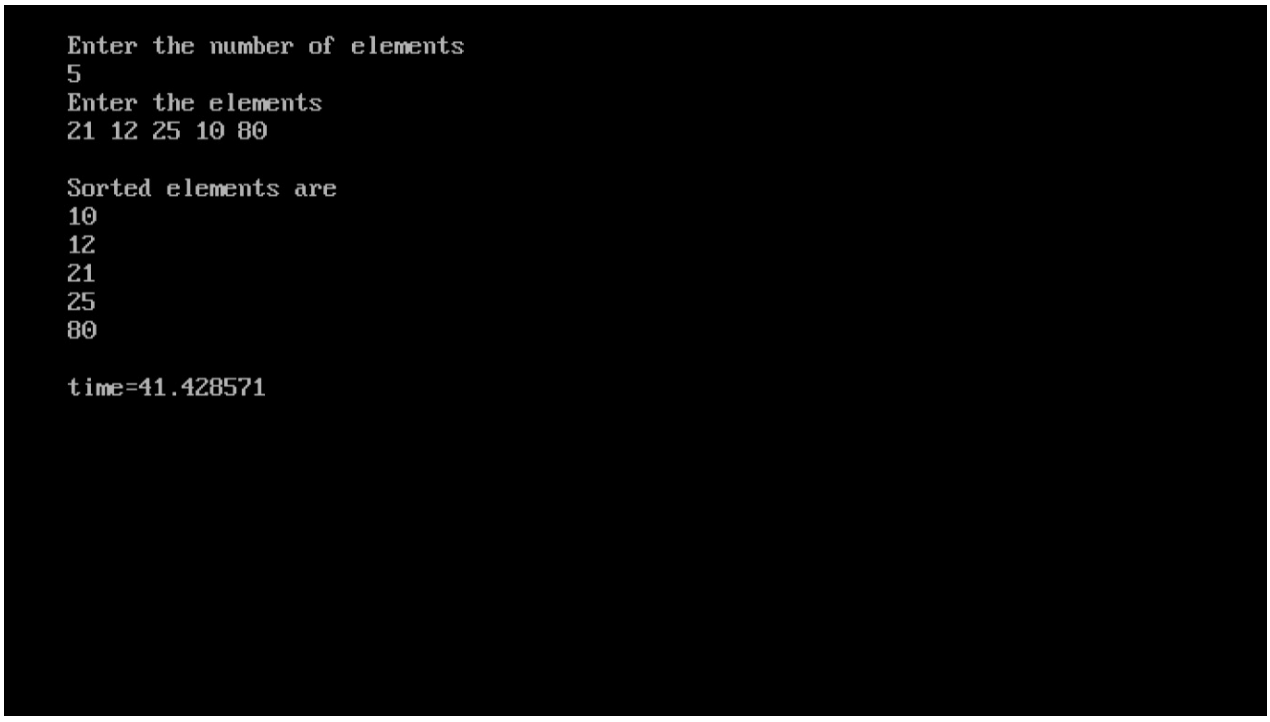


9. Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
#include <stdio.h>
#include<conio.h>
#include<time.h>
int a[25];
main()
{
        int i,n;
        clock_t start,end;
        clrscr();
        start=clock();
        printf("\nEnter the number of elements\n");
        scanf("%d",&n);
        printf("Enter the elements\n");
         for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        selsort(n);
        printf("\nSorted elements are\n");
         for(i=1;i<=n;i++)
        printf("%d\n",a[i]);
        end=clock();
        printf("\ntime=%f",(end−start)/CLK_TCK);
        getch();
}
selsort(int x)
{
        int i,j;
        for(i=1;i<=x;i++)
        {
        for(j=i+1;j<=x;j++)
         if(a[i]>a[j])
        swap(&a[i],&a[j]);
```

```
        }
    }
    swap(int *c,int *d)
    {

        int temp;
        temp = *C;
        *c=*d;
        *d=temp;
    }
```

OUTPUT 1

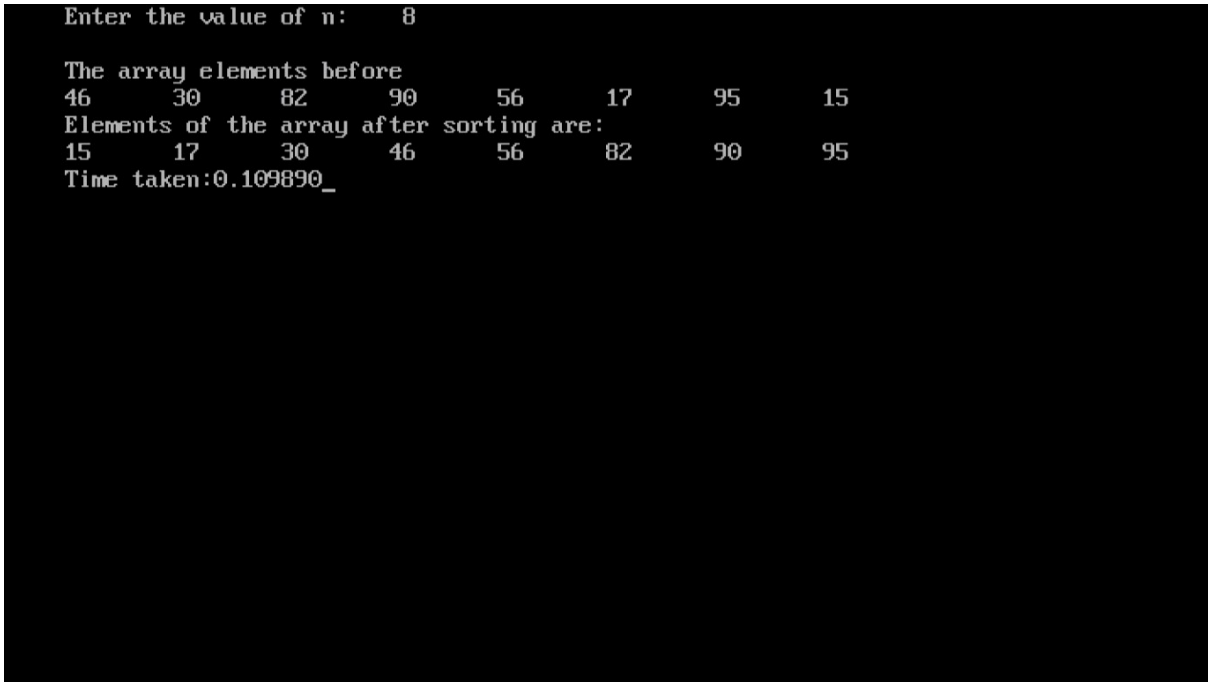


10. Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
/*To sort a set of elements using Quick sort algorithm.*/
#include<stdio.h>
#include<time.h>
#define max 500
void qsort(int [],int,int);
int partition(int [],int,int);
void main()
{
int a[max],i,n;
clock_t s,e;
clrscr();
printf("Enter the value of n:");
scanf("%d",&n);
for(i=0;i<n;i++)
a[i]=rand()%100;
printf("\nThe array elements before\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
s=clock();
delay(100);
qsort(a,0,n-1);
e=clock();
printf("\nElements of the array after sorting are:\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
printf("\nTime taken:%f",(e-s)/CLK_TCK);
getch();
}
void qsort(int a[],int low,int high)
{
int j;
if(low<high)
{
j=partition(a,low,high);
qsort(a,low,j-1);
qsort(a,j+1,high);
}
}
int partition(int a[], int low,int high)
{
int pivot,i,j,temp;
pivot=a[low];
i=low+1;
j=high;
```

```
while(1)
{
while(pivot>a[i] && i<=high)
i++;
while(pivot<a[j])
j--;
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
else
{
temp=a[j];
a[j]=a[low];
a[low]=temp;
return j;
}
}
}
```

OUTPUT



11. Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. 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

/*To sort a set of elements using Merge sort algorithm.*/

```
#include<time.h>
#include<stdio.h>
#include<iostream>
#include<conio.h>
#define max 100
void mergesort(int[100],int,int);
void merge(int[100],int,int,int);
int a[max];
void main()
{
int i,n;
clock_t s,e;
printf("Enter the no.of elements\n");
scanf_s("%d",&n);
printf("Elements of the array before sorting\n");
for(i=0;i<n;i++)
{
a[i]=rand()%1000;
printf("%d\t",a[i]);
}
s=clock();
mergesort(a,0,n-1);
e=clock();
printf("\nElements of the array after sorting\n");

for(i=0;i<n;i++)
printf("%d\t",a[i]);
printf("\nthe time taken=%f\n",(e-s)/CLK_TCK);
getch();
}
void mergesort(int a[100],int low,int high)
{
int mid;
if(high>low)
{
mid=(low+high)/2;
mergesort(a,low,mid);
mergesort(a,mid+1,high);
merge(a,low,mid,high);
}
}
void merge(int a[100],int low,int mid,int high)
```

```
{
int h=low,j=mid+1,i=low,b[max],k;

while((h<=mid)&&(j<=high))
{
if(a[h]<=a[j])
{
b[i]=a[h];
h=h+1;
}
else
{
b[i]=a[j];
j=j+1;
}
i=i+1;
}
if(h>mid)
{

for(k=j;k<=high;k++)
{
b[i]=a[k];
i++;
}
}
else
{

for(k=h;k<=mid;k++)
{
b[i]=a[k];
i++;
}
}

for(k=low;k<=high;k++)
a[k]=b[k];
}
```

OUTPUT

```
Enter the no.of elements
10
Elements of the array before sorting
346     130     982     90      656     117     595     415     948     126

Elements of the array after sorting
90      117     126     130     346     415     595     656     948     982

the time taken=0.000000
```

12. Design and implement C/C++ Program for N Queen's problem using Backtracking.

```
#include<stdio.h>
void nqueens(int);
int place(int[],int);
void printsolution(int,int[]);
void main()
{
int n;
clrscr();
printf("Enter the no.of queens: ");
scanf("%d",&n);
nqueens(n);
getch();
}
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--; //backtracking
}
}
return;
}
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 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]='X';
}
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");
}
}
```

OUTPUT