St. Aloysius Degree College

BANGALORE

Faculty of Computer Applications

DAA Lab

V Semester BCA

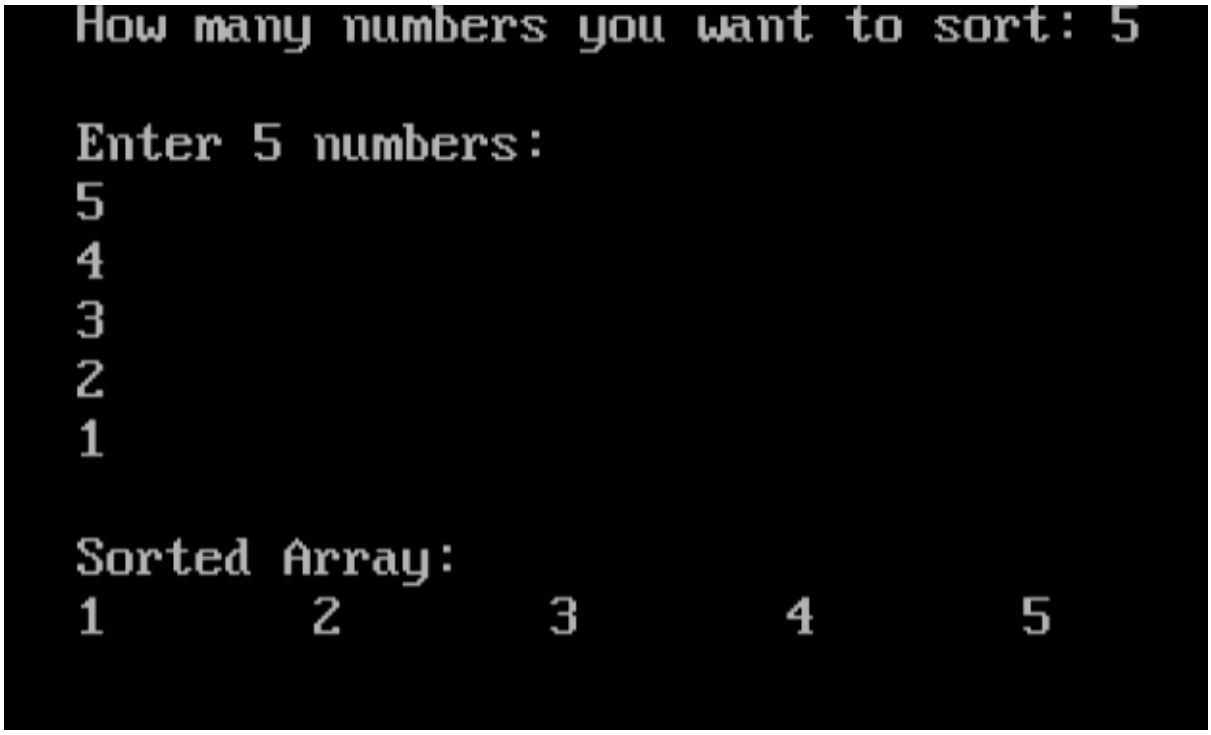
LAB MANUAL

1. Write a program to sort a list of N elements using Selection Sort Technique.

```
#include<stdio.h>
#include<conio.h>
void selectionsort(int a[],int);
void main()
{
   int a[10],i,n;
   clrscr();
   printf("How many numbers you want to sort: ");
   scanf("%d", &n);
   printf("\nEnter %d numbers: ", n);
   for (i = 0; i < n; i++)
         scanf("%d", &a[i]);
   selectionsort(a,n);
}
void selectionsort(int *a,int n)
 {
   int i,j,t,min;
   for (i = 0 ;i < n-1;i++)
   {
         min=i;
         for (j = i+1; j < n; j++)
          {
            if (a[j] < a[min])
                  min=j;
          }
         t=a[i];
         a[i]=a[min];
         a[min]=t;
```

```
    }
    printf("\nSorted Array:\n");
    for (i = 0; i < n;i++)
            printf("%d\t", a[i]);
    getch();
}
```

OUTPUT:



2. Write C program that accepts the vertices and edges for a graph and stores it as an adjacency matrix.

```
#include <stdio.h>
void main()
  {
    int n, i,j,adj[5][5] ;
    clrscr();
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &n);
    for(i=1;i<=n;i++)
```

```
  {
   for(j=1;j<=n;j++)
     {
        printf("Edge (V%d,V%d) exists? (Yes=1, No=0):",i,j);
        scanf("%d",&adj[i][j]);
      }
  }
 printf("\nAdjacency Matrix:\n");
 for (i = 1; i <= n; i++)
  {
       for (j = 1; j <= n; j++)
         {
           printf("%d ", adj[i][j]);
         }
       printf("\n");
  }
 getch();
}
```

```
Enter the number of vertices in the graph: 3
Edge (V1,V1) exists? (Yes=1, No=0):0
Edge (V1,V2) exists? (Yes=1, No=0):1
Edge (V1,V3) exists? (Yes=1, No=0):1
Edge (V2,V1) exists? (Yes=1, No=0):1
Edge (V2,V2) exists? (Yes=1, No=0):0
Edge (V2,V3) exists? (Yes=1, No=0):1
Edge (V3,V1) exists? (Yes=1, No=0):0
Edge (V3,V2) exists? (Yes=1, No=0):0
Edge (V3,V3) exists? (Yes=1, No=0):0

Adjacency Matrix:
0 1 1
1 0 1
0 0 0
```

3. Write a program to print In-Degree, Out-Degree and to display that adjacency matrix.

```
#include <stdio.h>
void main()
 {
   int n, i,j,sum, adj[10][10];
   clrscr();
   printf("Enter the number of vertices in the graph: ");
   scanf("%d", &n);
   for(i=1;i<=n;i++)
    {
     for(j=1;j<=n;j++)
       {
         printf("Edge (V%d,V%d) exists? (Yes=1, No=0):",i,j);
         scanf("%d",&adj[i][j]);
       }
    }
   printf("\nAdjacency Matrix:\n");
   for (i = 1; i <= n; i++)
    {
         for (j = 1; j <= n; j++)
           {
             printf("%d ", adj[i][j]);
           }
         printf("\n");
    }
   for(i=1;i<=n;i++)
    {
     sum=0;
    for(j=1;j<=n;j++)
      {
```

```
        sum = sum+ adj[j][i];
       }
      printf("InDegree of (V%d)=%d\n",i,sum);
     }
   for(i=1;i<=n;i++)
    {
     sum=0;
     for(j=1;j<=n;j++)
       {
         sum = sum+adj[i][j];
       }
     printf("OutDegree of (V%d)=%d\n",i,sum);
     }
   getch();
}
```

```
Enter the number of vertices in the graph: 2
Edge (V1,V1) exists? (Yes=1, No=0):0
Edge (V1,V2) exists? (Yes=1, No=0):1
Edge (V2,V1) exists? (Yes=1, No=0):1
Edge (V2,V2) exists? (Yes=1, No=0):0

Adjacency Matrix:
0 1
1 0
InDegree of (V1)=1
InDegree of (V2)=1
OutDegree of (V1)=1
OutDegree of (V2)=1
```

4. Write program to implement the DFS and BFS algorithm for a graph.

```
#include<stdio.h>
#include<conio.h>
int adj[20][20],visited[10],n,q[10],f=0,r=-1;
void dfs(int v)
   {
         int i;
         visited[v]=1;
         for (i=1;i<=n;i++)
           if(adj[v][i] && !visited[i])
             {
                  printf("\n %d->%d",v,i);
                  dfs(i);
             }
   }
void bfs(int v)
 {
         int i;
         for (i=1;i<=n;i++)
           if(adj[v][i] && visited[i]==0)
            q[++r]=i;
         if(f<=r)
            {
              visited[q[f]]=1;
              bfs(q[f++]);
            }
 }
void main()
{
   int i,j,v;
   clrscr();
   printf("Enter the number of vertices in the graph: ");
   scanf("%d", &n);
   printf("\nEnter the Adjacency Matrix:\n");
   for(i=1;i<=n;i++)
    {
     for(j=1;j<=n;j++)
           scanf("%d",&adj[i][j]);
    }
  for (i=1;i<=n;i++)
     visited[i]=0;
  printf("\n Enter the starting vertex for DFS traversal:");
  scanf("%d",&v);
  printf("\n DFS Traversal:");
  dfs(v);
  for (i=1;i<=n;i++)
    visited[i]=0;
```

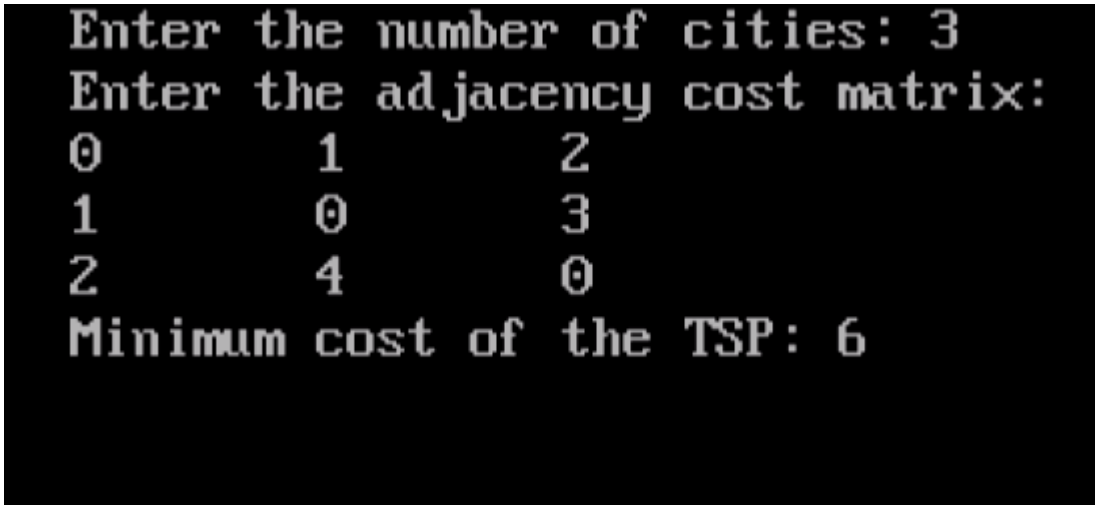
```
  printf("\n Enter the starting vertex for BFS traversal:");
  scanf("%d",&v);
  bfs(v);
  printf("\n BFS Traversal:\n");
  for (i=1;i<=n;i++)
     if(visited[i])
          printf("%d\t",i);
     else
         printf("\n BFS is not possible");
 getch();
}
```

```
 Enter the number of vertices in the graph: 3

 Enter the Adjacency Matrix:
 0       1       1
 1       0       1
 1       1       0

  Enter the starting vertex for DFS traversal:1

  DFS Traversal:
  1->2
  2->3
  Enter the starting vertex for BFS traversal:1

  BFS Traversal:
 1       2       3
```

5. Write a program to perform Travelling Salesman Problem.

```
#include <stdio.h>

int n, graph[10][10], visited[10], minCost = 9999;

void tsp(int city, int count, int cost)

{   int i;

   if (count == n)

   {

        if (graph[city][1] + cost < minCost)

            minCost = graph[city][1] + cost;

        return;
```

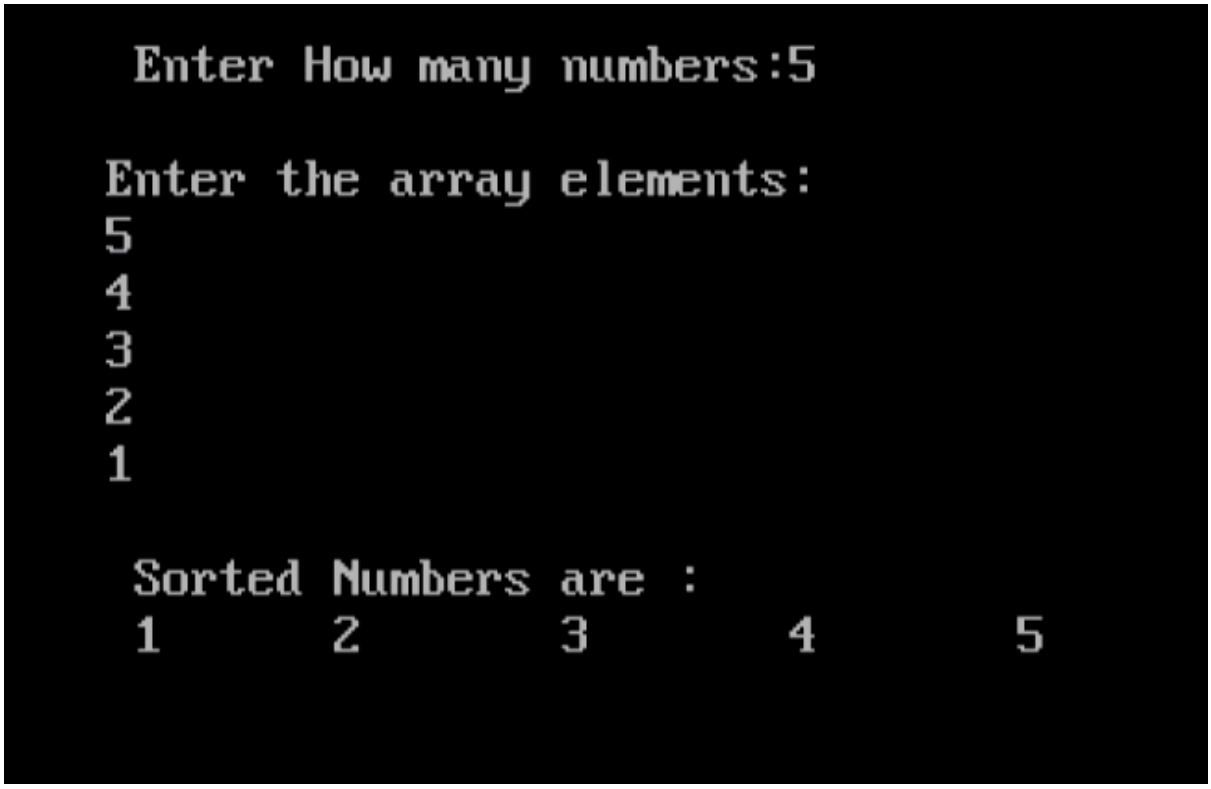
```
    }
   for (i = 1; i <= n; i++)
    {
          if (visited[i]==0 && graph[city][i] != 0)
          {
             visited[i] = 1;
             tsp(i, count + 1, cost + graph[city][i]);
             visited[i] = 0;
          }
    }
}
void main()
{
    int i,j;
    clrscr();
    printf("Enter the number of cities: ");
    scanf("%d", &n);
    printf("Enter the adjacency cost matrix:\n");
    for (i = 1; i <= n; i++)
    {
          for (j = 1; j <= n; j++)
             scanf("%d", &graph[i][j]);
    }
    visited[1] = 1;
    tsp(1, 1, 0);
    printf("Minimum cost of the TSP: %d\n", minCost);
    getch();
}
```



6. Write a program to implement Merge sort algorithm for sorting a list of integers in ascending order.

```
# include <stdio.h>
#include<stdlib.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[20],k;
        printf("\n Enter How many numbers:");
        scanf("%d", &n);
        printf("\nEnter the array elements:\n");
        for(k=0; k<n; k++)
          scanf("%d",&a[k]);
        MergeSort(a, 0, n-1);

        printf("\n Sorted Numbers are : \n ");
        for(k=0; k<n; k++)
            printf("%d\t", a[k]);

  }
```



7. 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.

```
# include <stdio.h>
#include<stdlib.h>
#include<time.h>
void Merge(int a[], int low, int mid, int high)
    {  int i, j, k, b[10000];
        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[10000],k;
  clock_t t;
double time_taken;
  t = clock();
        clrscr();
        printf("\n Enter How many numbers:");
        scanf("%d", &n);
        printf("\nThe Random Numbers are:\n");
         for(k=0; k<n; k++)
            {
             a[k]=rand()%n;
             printf("%d\t",a[k]);
```

```
        }
    MergeSort(a, 0, n-1);
  t = clock() - t;
  time_taken = ((double)t)/CLOCKS_PER_SEC;
    printf("\n Sorted Numbers are : \n ");
    for(k=0; k<n; k++)
        printf("%d\t", a[k])        ;
    printf("\nThe time taken is %f",time_taken);
    getch();
 }
```

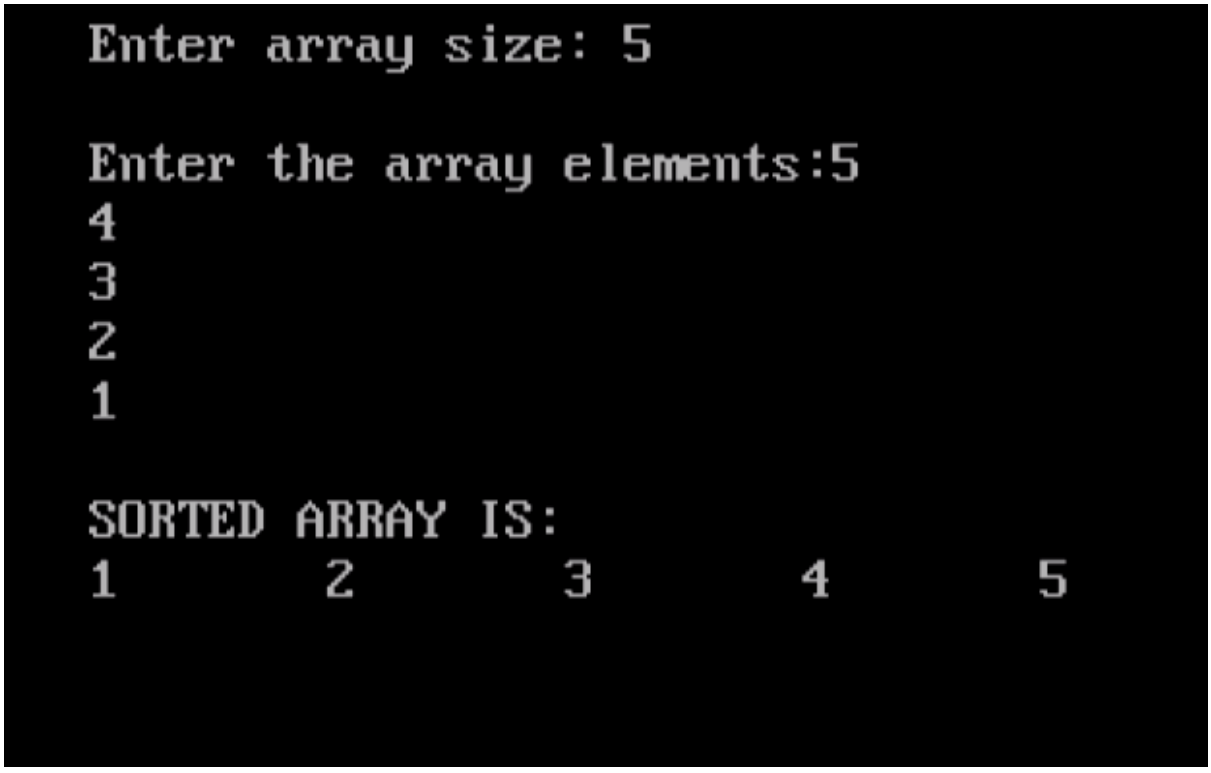
```
 Enter How many numbers:50

The Random Numbers are:
46      30      32      40      6       17      45      15      48      26
4       8       21      29      42      10      12      21      13      47
19      41      40      35      14      9       2       21      29      16
31      1       45      43      34      10      29      45      11      42
39      38      16      14      42      13      16      14      39      1

 Sorted Numbers are :
 1      1       2       4       6       8       9       10      10      11
12      13      13      14      14      14      15      16      16      16
17      19      21      21      21      26      29      29      29      30
31      32      34      35      38      39      39      40      40      41
42      42      42      43      45      45      45      46      47      48

The time taken is 1.978022_
```

8. Write a program to implement Quick Sort algorithm using Divide and Conquer Strategy for sorting list of integers in ascending order.

```
#include<stdio.h>
int partition(int a[],int,int);
void quick(int a[],int,int);
void main()
{
  int i,a[10],n;
  clrscr();
  printf("\nEnter array size: ");
```

```
   scanf("%d",&n);
   printf("\nEnter the array elements:");
   for(i=0;i<n;i++)
      scanf("%d",&a[i]);
   quick(a,0,n-1);
   printf("\nSORTED ARRAY IS:\n");
   for(i=0;i<n;i++)
     printf("%d\t",a[i]);
   getch();
}
void quick(int a[],int low,int high)
{
  int pos;
  if(low<high)
   {
     pos=partition(a,low,high);
     quick(a,low,pos-1);
     quick(a,pos+1,high);
   }
}
int partition(int a[],int low,int high)
  {
     int down,up,temp,key;
     key=a[low];
     down=low+1;
     up=high;
     while(1)
       {
              while(down<high&&a[down]<=key)
                down++;
```

```
        while(a[up]>key)
          up--;
        if(down<up)
          {
                temp=a[down];
                a[down]=a[up];
                a[up]=temp;
          }
        else
              {
                 temp=a[low];
                 a[low]=a[up];
                 a[up]=temp;
                 return up;
              }
    }
}
```



9. 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.

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

int partition(int a[],int,int);
void quick(int a[],int,int);

void main()
{
  int i,a[10000],n;
  clock_t t;
double time_taken;
  t = clock();
  clrscr();
        printf("\n Enter How many numbers:");
        scanf("%d", &n);
        printf("\nThe Random Numbers are:\n");
         for(i=0; i<n; i++)
           {
             a[i]=rand()%n;
             printf("%d\t",a[i]);
           }
  quick(a,0,n-1);
  t = clock() - t;
  time_taken = ((double)t)/CLOCKS_PER_SEC;
  printf("\nSORTED ARRAY IS:\n");
  for(i=0;i<n;i++)
    printf("%d\t",a[i]);
  printf("\nThe time taken is %f",time_taken);
```

```
   getch();
}

void quick(int a[],int low,int high)
{
  int pos;

  if(low<high)
   {
     pos=partition(a,low,high);
     quick(a,low,pos-1);
     quick(a,pos+1,high);
   }
}

int partition(int a[],int low,int high)
  {
     int down,up,temp,key;
     key=a[low];
     down=low+1;
     up=high;
     while(1)
       {
              while(down<high&&a[down]<=key)
                down++;
              while(a[up]>key)
                up--;
              if(down<up)
                {
                      temp=a[down];
```

```
                a[down]=a[up];
                a[up]=temp;
            }
        else
            {
                temp=a[low];
                a[low]=a[up];
                a[up]=temp;
                return up;
            }
    }
}
```

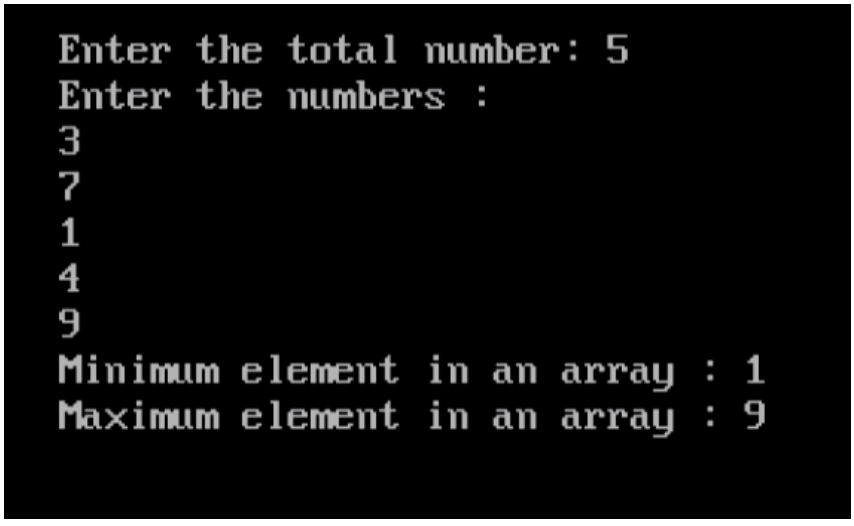
```
 Enter How many numbers:50

The Random Numbers are:
46      30      32      40      6       17      45      15      48      26
4       8       21      29      42      10      12      21      13      47
19      41      40      35      14      9       2       21      29      16
31      1       45      43      34      10      29      45      11      42
39      38      16      14      42      13      16      14      39      1

SORTED ARRAY IS:
1       1       2       4       6       8       9       10      10      11
12      13      13      14      14      14      15      16      16      16
17      19      21      21      21      26      29      29      29      30
31      32      34      35      38      39      39      40      40      41
42      42      42      43      45      45      45      46      47      48

The time taken is 3.186813
```

10. Write a program to find minimum and maximum value in an array using divide and conquer.

```
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j)
{
 int max1, min1, mid;
 if(i==j)
```

```
{
 max = min = a[i];
}
else
{
 if(i == j-1)
 {
  if(a[i] <a[j])
  {
   max = a[j];
   min = a[i];
  }
  else
  {
   max = a[i];
   min = a[j];
  }
 }
 else
 {
  mid = (i+j)/2;
  maxmin(i, mid);
  max1 = max; min1 = min;
  maxmin(mid+1, j);
  if(max <max1)
   max = max1;
  if(min > min1)
   min = min1;
 }
}
```

```
}
void main ()
{
 int i, n;
 printf ("\nEnter the total number: ");
 scanf ("%d",&n);
 printf ("Enter the numbers : \n");
 for (i=1;i<=n;i++)
  scanf ("%d",&a[i]);

 max = a[0];
 min = a[0];
 maxmin(1, n);
 printf ("Minimum element in an array : %d\n", min);
 printf ("Maximum element in an array : %d\n", max);
}
```



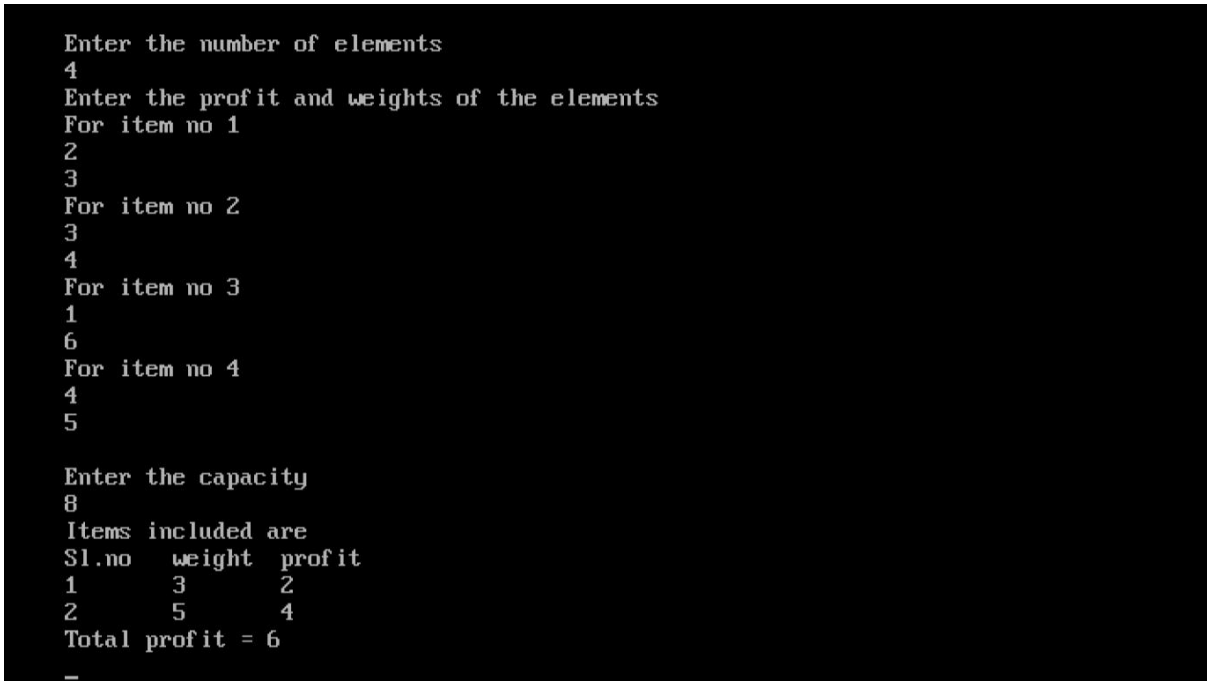
11. Write program to implement Dynamic Programming algorithm for the 0/1 Knapsack

problem.

```
#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),knap(i-1,j-w[i])+p[i]);
       v[i][j]=value;
   }
 return(v[i][j]);
}
void main()
   {  int profit,count=0;
      clrscr();
      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();
}
```



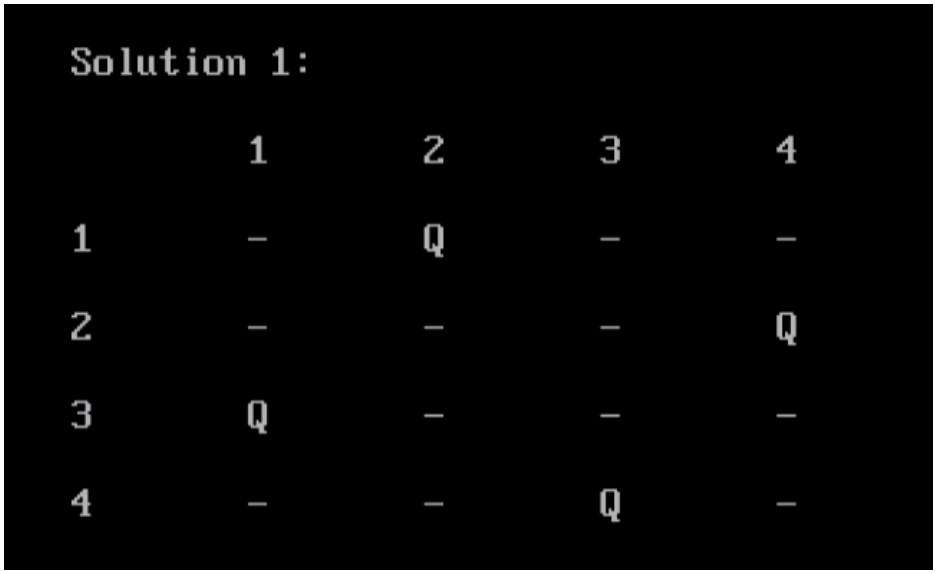
12. Write program to implement Dynamic Programming algorithm for the Optimal Binary Search Tree Problem.

13. Write program to implement backtracking algorithm for solving problems like N queens .

```
#include<stdio.h>
#include<math.h>

int board[20],count;

int main()
{
int n,i,j;
void queen(int row,int n);
clrscr();
printf("N Queens Problem Using Backtracking");
printf("\n\nEnter number of Queens:");
scanf("%d",&n);
queen(1,n);
```

```
return 0;
}

//function for printing the solution
void print(int n)
{
int i,j;
printf("\n\nSolution %d:\n\n",++count);

for(i=1;i<=n;++i)
  printf("\t%d",i);

for(i=1;i<=n;++i)
{
  printf("\n\n%d",i);
  for(j=1;j<=n;++j) //for nxn board
  {
   if(board[i]==j)
    printf("\tQ"); //queen at i,j position
   else
    printf("\t-"); //empty slot
  }
}
}

/*funtion to check conflicts
If no conflict for desired postion returns 1 otherwise returns 0*/
int place(int row,int column)
{
int i;
```

```
for(i=1;i<=row-1;++i)
{
  //checking column and digonal conflicts
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
}

return 1; //no conflicts
}

//function to check for proper positioning of queen
void queen(int row,int n)
{
int column;
for(column=1;column<=n;++column)
{
  if(place(row,column))
  {
   board[row]=column; //no conflicts so place queen
   if(row==n) //dead end
    print(n); //printing the board configuration
   else //try queen with next position
    queen(row+1,n);
  }
}
}
```

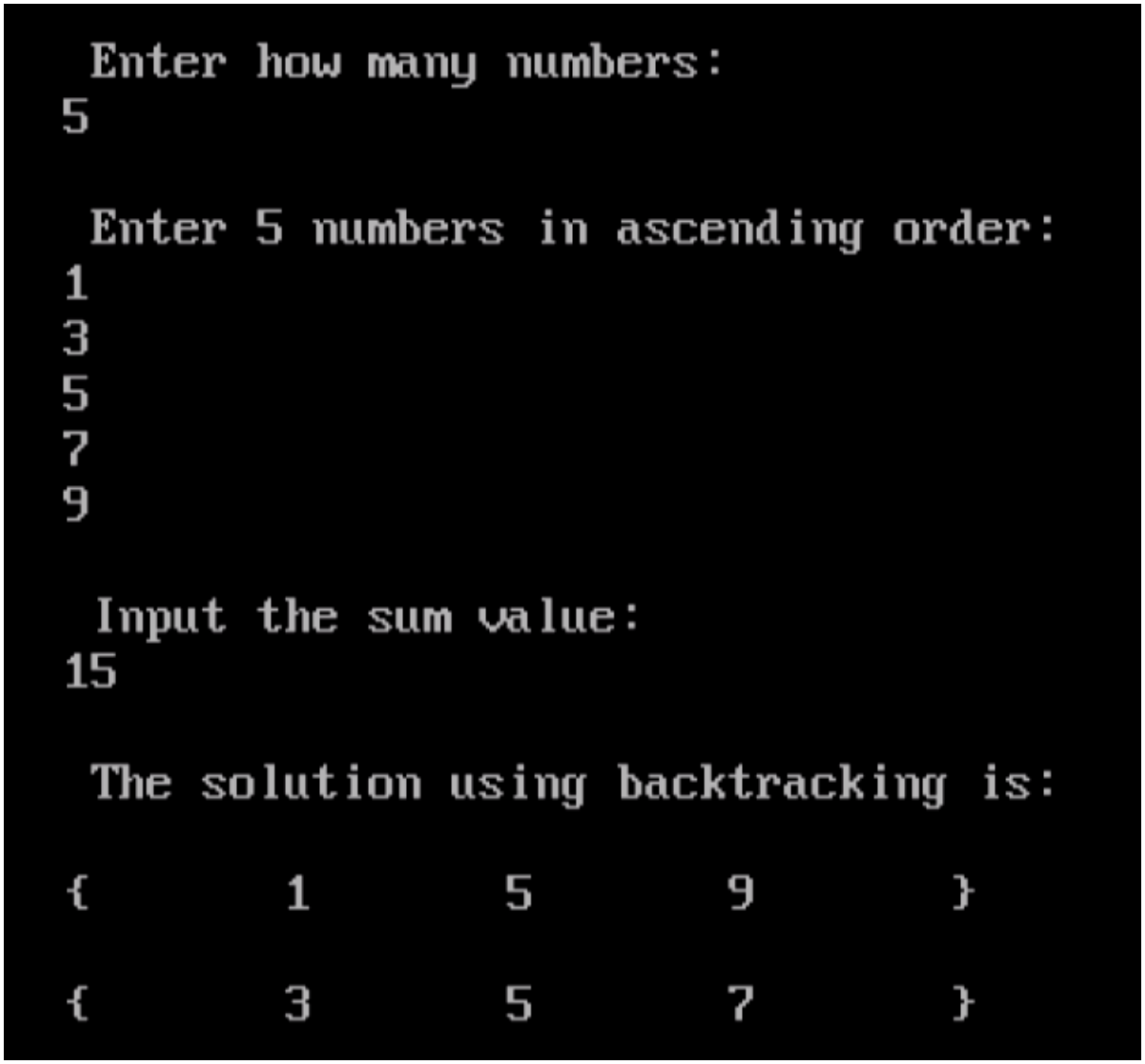


14. Write a program to implement the backtracking algorithm for the sum of subsets problem

```
#include<conio.h>
void sumset(int i,int wt,int total);
int inc[10],w[10],sum,n;
int promising(int i,int wt,int total)
{
 return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));
}
void main()
{
 int i,j,n,temp,total=0;
 clrscr();
 printf("\n Enter how many numbers:\n");
 scanf("%d",&n);
 printf("\n Enter %d numbers in ascending order:\n",n);
 for(i=0;i<n;i++)
 {
  scanf("%d",&w[i]);
  total+=w[i];
 }
```

```
 printf("\n Input the sum value:\n");
 scanf("%d",&sum);

 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);
 }
 getch();
}
void sumset(int i,int wt,int total)
{
 int j;
 if(promising(i,wt,total))
 {
  if(wt==sum)
  {
   printf("\n{\t");
   for(j=0;j<=i;j++)
    if(inc[j]==1)
     printf("%d\t",w[j]);
   printf("}\n");
  }
  else
  {
   inc[i+1]=1;
```

```
  sumset(i+1,wt+w[i+1],total-w[i+1]);
  inc[i+1]=0;
  sumset(i+1,wt,total-w[i+1]);
 }
}
}
```



15. Write program to implement greedy algorithm for job sequencing with deadlines.

16. Write a program that implements Prim's algorithm to generate minimum cost spanning Tree.

17. Write a program that implements Kruskal's algorithm to generate minimum cost spanning tree.

18. Write a program to perform Knapsack Problem using Greedy Solution

```
#include <stdio.h>
void main()
{
    int capacity, no_items, cur_weight, item;
    int used[10];
    float total_profit;
    int i;
    int weight[10];
    int value[10];

    printf("Enter the capacity of knapsack:\n");
    scanf("%d", &capacity);

    printf("Enter the number of items:\n");
    scanf("%d", &no_items);

    printf("Enter the weight and value of %d item:\n", no_items);
    for (i = 0; i < no_items; i++)
    {
        printf("Weight[%d]:\t", i);
        scanf("%d", &weight[i]);
        printf("Value[%d]:\t", i);
        scanf("%d", &value[i]);
    }

    for (i = 0; i < no_items; ++i)
        used[i] = 0;

    cur_weight = capacity;
    while (cur_weight > 0)
```

```
    {
        item = -1;
        for (i = 0; i < no_items; ++i)
            if ((used[i] == 0) &&
                ((item == -1) || ((float) value[i] / weight[i] > (float) value[item] / weight[item])))
                item = i;

        used[item] = 1;
        cur_weight -= weight[item];
        total_profit += value[item];
        if (cur_weight >= 0)
            printf("Added object %d (%d Rs., %dKg) completely in the bag. Space left: %d.\n",
item + 1, value[item], weight[item], cur_weight);
        else
        {
            int item_percent = (int) ((1 + (float) cur_weight / weight[item]) * 100);
            printf("Added %d%% (%d Rs., %dKg) of object %d in the bag.\n", item_percent,
value[item], weight[item], item + 1);
            total_profit -= value[item];
            total_profit += (1 + (float)cur_weight / weight[item]) * value[item];
        }
    }

    printf("Filled the bag with objects worth %.2f Rs.\n", total_profit);
}
```

```
Enter the capacity of knapsack:
8
Enter the number of items:
4
Enter the weight and value of 4 item:
Weight[0]:      3
Value[0]:       2
Weight[1]:      4
Value[1]:       3
Weight[2]:      6
Value[2]:       1
Weight[3]:      5
Value[3]:       4
Added object 4 (4 Rs., 5Kg) completely in the bag. Space left: 3.
Added 75% (3 Rs., 4Kg) of object 2 in the bag.
Filled the bag with objects worth 6.25 Rs.
```