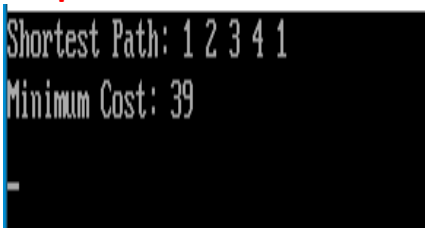
Program 1: Write a C Program to Perform Travelling Salesman Problem.

```
#include <stdio.h>
#include<conio.h>
int cm[10][10] = {
   {0, 10, 0, 20},
   {5, 0, 9, 10},
   {0, 13, 0, 12},
   {8, 8, 9, 0}
};
int visited[10], n, cost = 0;
void tsp(int c)
{
   int k, adj_vertex = 999;
   int min = 999;
   visited[c] = 1;
   printf("%d ", c + 1);
   for(k = 0; k < n; k++) {
      if((cm[c][k] != 0) && (visited[k] == 0)) {
            if(cm[c][k] < min) {
               min = cm[c][k];
               adj_vertex = k;
            }
      }
   }
   if(min != 999) {
      cost = cost + min;
   }
   if(adj_vertex == 999) {
      adj_vertex = 0;
      printf("%d", adj_vertex + 1);
      cost = cost + cm[c][adj_vertex];
      return;
   }
  tsp(adj_vertex);
}
void main(){
   int i;
   n = 4;
   clrscr();
   visited[i] = 0;

   printf("Shortest Path: ");
   tsp(0);
   printf("\nMinimum Cost: ");
   printf("%d\n", cost);
   getch();

}
```

Output:

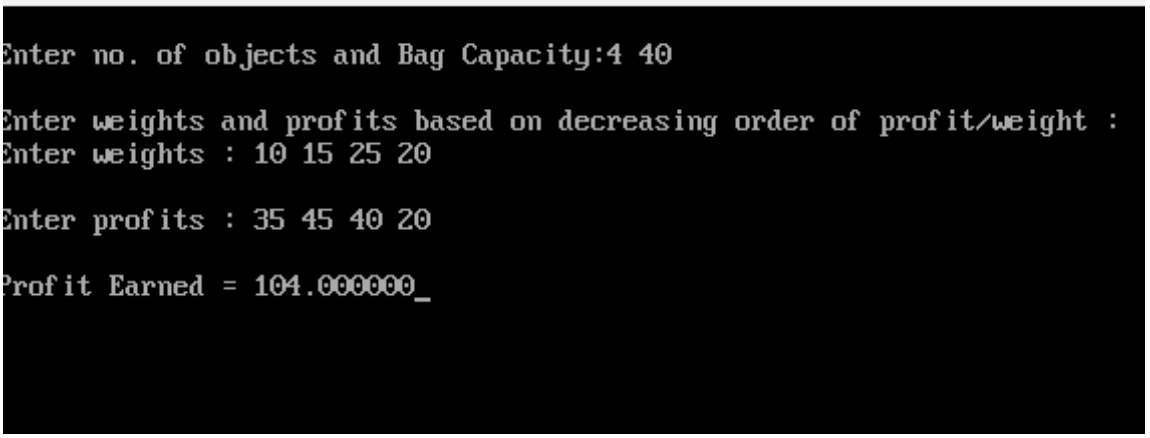


Program 2: Write a C Program to Perform Knapsack Problem.

```
#include<stdio.h>
#include<conio.h>
   void knapsack(int m,int n,int w[], int p[]);
   void main()
   {
            int m,n,w[10],p[10],i;
            clrscr();
            printf("\nEnter no. of objects and Bag Capacity:");
            scanf("%d%d",&n,&m);
            printf("\nEnter weights and profits based on decreasing order of profit/weight : ");
            printf("\nEnter weights : ");
            for(i=0;i<n;i++)
            scanf("%d",&w[i]);
            printf("\nEnter profits : ");
            for(i=0;i<n;i++)
            scanf("%d",&p[i]);
            knapsack(m,n,w,p);
            getch();
   }
   Void  knapsack(int m,int n,int w[], int p[])
   {
            double profit=0,rc=m,i;
            double x[10];
             for(i=0;i<n;i++)
            x[i]=0;
            for(i=0;i<n;i++)
            {
                     if(w[i]<rc)
                     {
```

```
    x[i]=1;
rc=rc-w[i];
    }
        else
        {       x[i]=rc/w[i];
                break;
        }
        }

for(i=0;i<n;i++)
profit=profit+x[i]*p[i];

printf("\nProfit Earned = %lf",profit);
}
```

Output of Knapsack

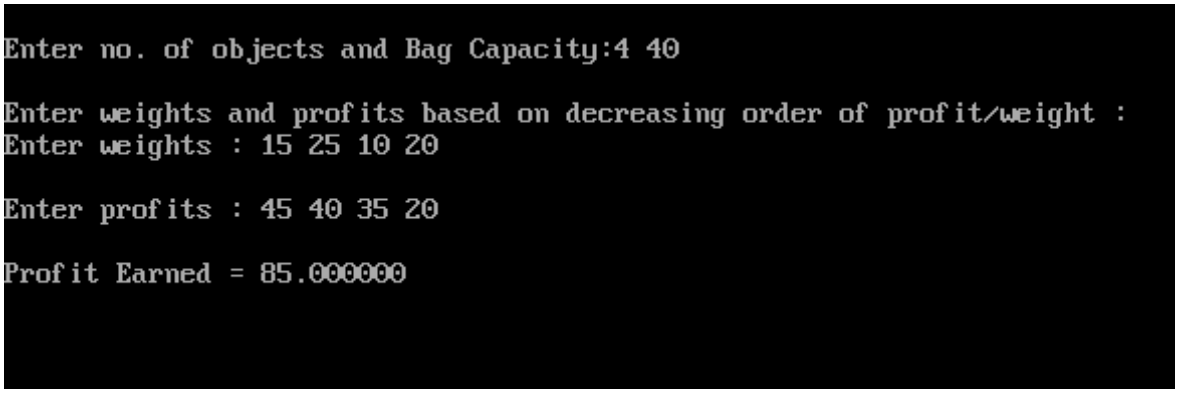


Program 3: Write a C Program to Perform 0/1 Knapsack Problem.

```
#include<stdio.h>
#include<conio.h>
void knapsack(int m,int n,int w[], int p[]);
void main()
  {
        int m,n,w[10],p[10],i;
        clrscr();
        printf("\nEnter no. of objects and Bag Capacity:");
        scanf("%d%d",&n,&m);
        printf("\nEnter weights and profits based on decreasing order of profit/weight : ");
        printf("\nEnter weights : ");
        for(i=0;i<n;i++)
        scanf("%d",&w[i]);

        printf("\nEnter profits : ");
        for(i=0;i<n;i++)
        scanf("%d",&p[i]);

        knapsack(m,n,w,p);
        getch();
  }
  void knapsack(int m,int n,int w[], int p[])
  {
        double profit=0,rc=m,i;
        double x[10];
        for(i=0;i<n;i++)
        x[i]=0;


        for(i=0;i<n;i++)
        {
              if(w[i]<=rc)
```

```
            {
                rc=rc-w[i];
                x[i]=1;
            }
            else
            {
                break;
            }
        }

        for(i=0;i<n;i++)
        profit=profit+x[i]*p[i];
        printf("\nProfit Earned = %lf",profit);
}
```

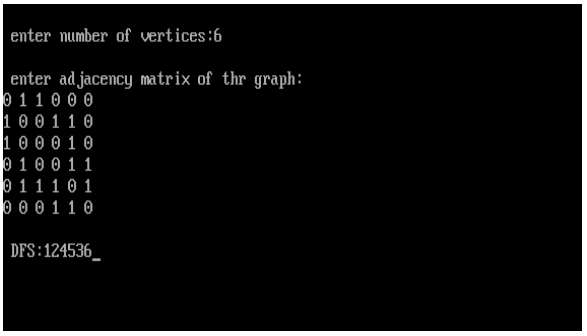
Output of 0/1 Knapsack



Program 4: Write a C Program to implement DFS algorithm.

```
#include<stdio.h>
#include<conio.h>
void  DFS(int);
int g [10][10], visited[10],n;
void main()
{
int i,j;
clrscr();
printf("\n  enter number of vertices:");
scanf("%d", &n);
printf("\n  enter adjacency  matrix  of the  graph:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&g[i][j]);
for(i=1;i<=n;i++)
visited[i]=0;
printf("\n DFS:");
DFS(1);
getch();
}
void  DFS(int  i)
{
int j;
printf("%d", i);
visited[i]=1;
for(j=1;j<=n;j++)
if(!visited[j]&&g[i][j]==1)
DFS(j);
}
```

Output of DFS



Program 5: Write a C Program to implement BFS algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int cost[10][10],i,j,k,n,q[10],front=-1,rear=-1,v,visit[10],visited[10];
void main()
{
int n;
clrscr();
printf("\n enter number of vertices:");
scanf("%d", &n);
printf("\n enter adjacency matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d", &cost[i][j]);
v=1;
printf("\n BFS:");
printf("%d", v);
visited[v]=1;
k=1;
while(k<n)
{
for(j=1;j<=n;j++)
if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
{
visit[j]=1;
q[rear++]=j;
}
v=q[front++];
printf("%d",v);
k++;
visit[v]=0;
```

```
visited[v]=1;

}

getch();

}
```

Output of BFS

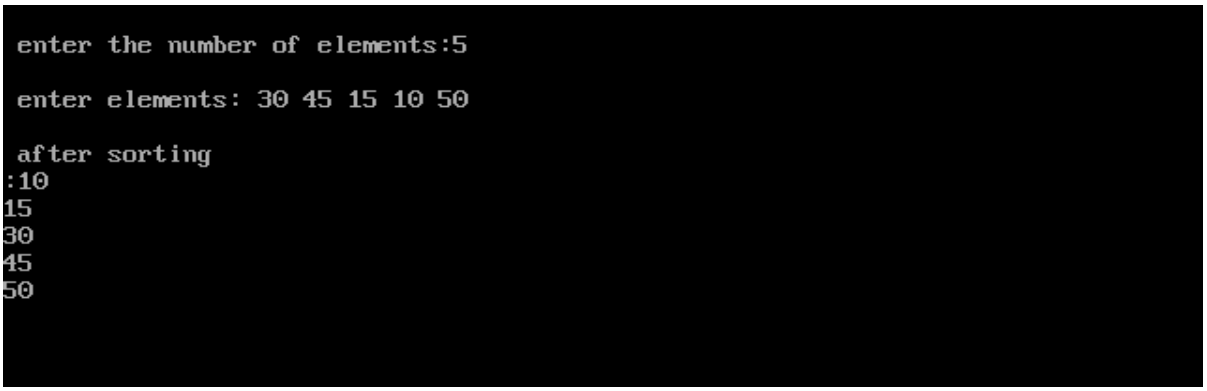
Program 6: Write a C Program to implement Merge sort.

```
#include<stdio.h>
#include<conio.h>
void mergesort(int a[], int low, int high);
void merge(int a[], int low,int mid, int high);
void main()
{
        int a[10],n,i;
        clrscr();

        printf("\n enter the number of elements:");
        scanf("%d",&n);

        printf("\n enter elements:");
        for(i=0;i<n;i++)
        scanf("%d",&a[i]);
 {
        mergesort(a,0,n-1);
        printf("\n after sorting\n:");
        for(i=0;i<n;i++)
        printf("%d\n", a[i]);
}
getch();
}
void mergesort(int a[10], int low, int high)
{
        int mid;
        if(low<high)
        {
        mid=(low+high)/2;
        mergesort(a,low,mid);
```

```
        mergesort(a,mid+1,high);
        merge(a,low,mid,high);
        }
}
void merge(int a[10], int low, int mid, int high)
{
int c[10],i,j,k;
i=low;
j=mid+1;
k=low;
while((i<=mid)&&(j<=high))
{
        if(a[i]<a[j])
        {
                c[k]=a[i];
                i++;
                k++;
        }
        else
        {
                c[k]=a[j];
                k++;
                j++;
        }
}
while(i<=mid)
{
c[k]=a[i];
i++;
k++;
}
```

```
while(j<=high)
{
c[k]=a[j];
j++;
k++;
}
for(i=low;i<=k-1;i++)
a[i]=c[i];
}
```

Output of Merge sort



Program 7: Write a C Program to implement Quicksort.

```
#include<stdio.h>
#include<conio.h>
void quicksort(int a[], int low, int high);
void main()
{
int a[10],n,i;
clrscr();
printf("\n enter number of elements:");
scanf("%d", &n);
printf("\n enter elements:");
```

```
for(i=0;i<n;i++)
scanf("%d", &a[i]);
quicksort(a,0,n-1);
printf("\n after sorting:");
for(i=0;i<n;i++)
printf("%d\n", a[i]);
getch();
}
void quicksort(int a[10], int low, int high)
{
int i,j,piv,temp;
i=low+1;
j=high;
piv=a[low];
if(low<high)
{
while(low<high)
{
while((a[i]<piv)&&(i<=high))
i++;
while((a[j]>piv)&&(j>low))
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;
break;
}
}
quicksort(a,low,j-1);
quicksort(a,j+1,high);
}
}
```

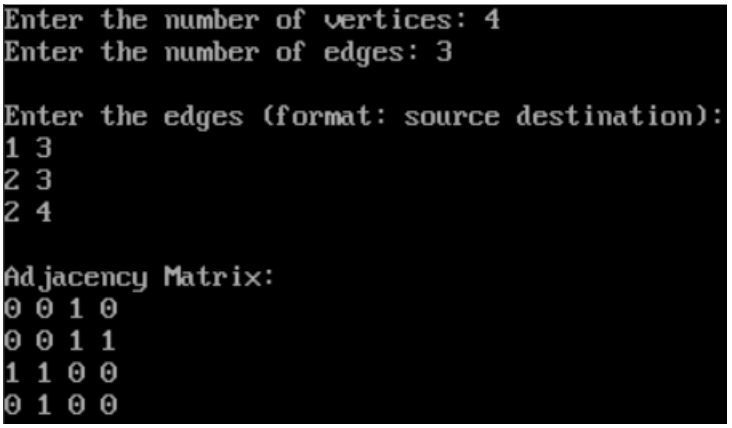
Output of Quick sort

Program 8: Write a C Program that accepts the vertices and edges for a graph and stores it as an adjacency matrix.

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

  int main() {
   int i,j, n, e;
   clrscr();
   int adjacencyMatrix[10][10] = {0};

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

   printf("Enter the number of edges: ");
   scanf("%d", &e);

   printf("\nEnter the edges (format: source destination):\n");
   for ( i = 1; i < =e; i++)
```

```
{
        int src, dest;
        scanf("%d %d", &src, &dest);
        adjacencyMatrix[src][dest] = 1;
        adjacencyMatrix[dest][src] = 1;
    }
    printf("\nAdjacency Matrix:\n");
    for (i=1;i<=n;i++)
     {
        for (j=1;j<=n;j++) {
           printf("%d ", adjacencyMatrix[i][j]);
        }
        printf("\n");
    }
    getch();

    return 0;
}
```

Output



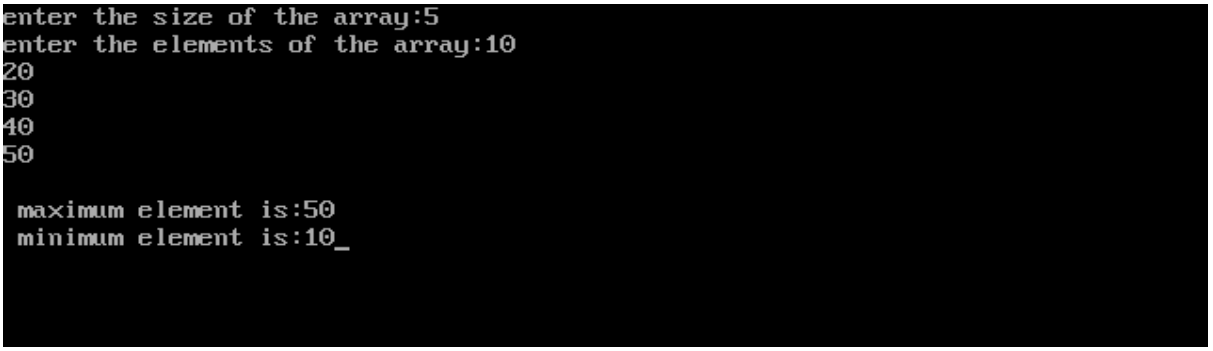
Program 9: Write a C Program to Find a Maximum and minimum value in an array using divide and conquer technique.

```
#include<stdio.h>
#include<conio.h>
int maximum(int a[],int i,int n);
int minimum(int a[],int i,int n);
void main()
{
int a[10],i,n,min,max;
clrscr();
printf("enter the size of the array:");
```

```
scanf("%d",&n);
printf("enter the elements of the array:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
min=a[(minimum(a,1,n))];
max=a[(maximum(a,1,n))];
printf("\n maximum element is:%d",max);
printf("\n minimum element is:%d",min);
getch();
}
int minimum(int a[],int i,int n)
{
static int min=0;
if(i<n)
{
if(a[min]>a[i])
{
min=i;
i++;
minimum(a,i,n);
}
}
return min;
}
int maximum(int a[],int i,int n)
{
static int max=0;
if(i<n)
{
if(a[max]<a[i])
{
max=i;
i++;
```

```
maximum(a,i,n);
}
}
return max;
}
```

Output of Maximum and minimum



Program 10: Write a C Program to implement Binary search tree and tree traversals algorithm.

```
#include<stdio.h>
#include<conio.h>
struct node
{
struct node *lchild;
int info;
struct node *rchild;
};
struct node *root=NULL;
void create();
void preorder(struct node *temp);
void inorder(struct node *temp);
void postorder(struct node *temp);
void main()
{
int ch;
clrscr();
while(1)
{
printf("\n1. CREATE");
printf("\n2. PRE-ORDER");
printf("\n3. IN-ORDER");
printf("\n4. POST-ORDER");
printf("\n5. EXIT");
printf("\nEnter your choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1: create(); break;
case 2: preorder(root); break;
case 3: inorder(root); break;
case 4: postorder(root); break;
case 5: exit(0);
}
}
}
void create()
{
struct node *newn,*temp=root,*parent;
int ele;
printf("\nEnter element to insert: ");
scanf("%d",&ele);
newn=(struct node *)malloc(sizeof(struct node));
newn->lchild=NULL;
newn->info=ele;
newn->rchild=NULL;
if(root==NULL)
root=newn;
else
{
while(temp!=NULL)
{
parent=temp;
if(newn->info<temp->info)
```

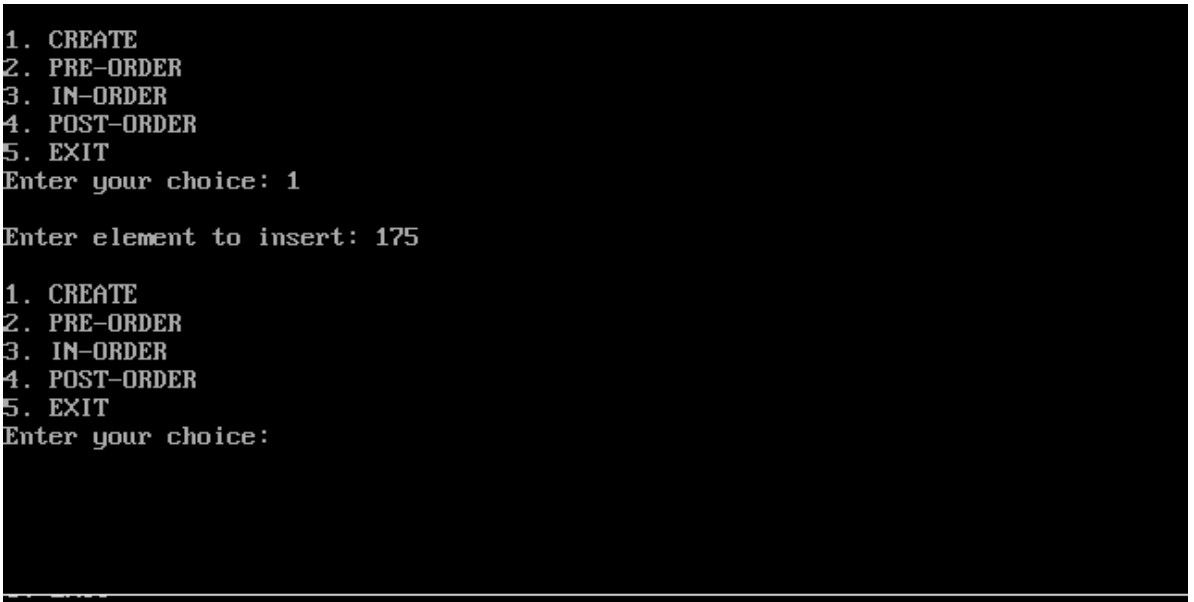
```
{
        temp=temp->lchild;
}
else
{
      temp=temp->rchild;
}
}
if(newn->info<parent->info)
parent->lchild=newn;
else
   parent->rchild=newn;

}
}
void preorder(struct node *temp)
{
if(temp!=NULL)
{
printf("%d ",temp->info);
preorder(temp->lchild);
preorder(temp->rchild);
}
}
void inorder(struct node *temp)
{
if(temp!=NULL)
{
inorder(temp->lchild);
printf("%d ",temp->info);
inorder(temp->rchild);
}
}
void postorder(struct node *temp)
{
if(temp!=NULL)
{
postorder(temp->lchild);
postorder(temp->rchild);
printf("%d ",temp->info);
}
}
```

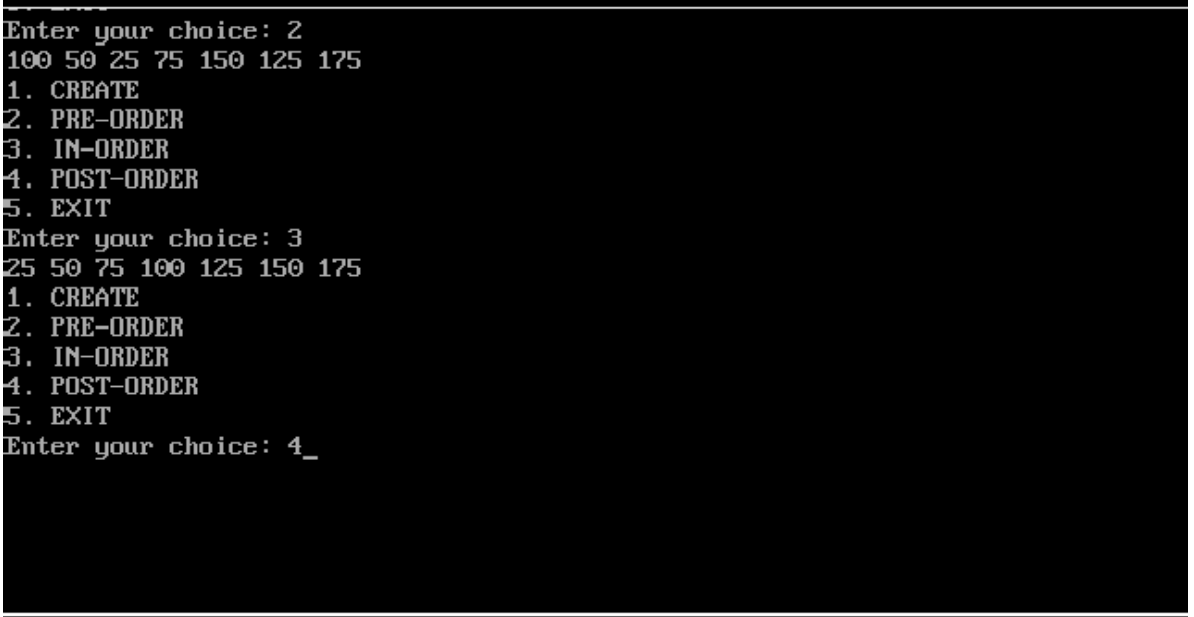
Output of Binary search tree traversals

```
1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice: 1

Enter element to insert: 100

1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice: 1

Enter element to insert: 50

1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice: 1
```

```
Enter element to insert: 150

1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice: 1

Enter element to insert: 25

1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice: 1
```

```
Enter element to insert: 75

1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice: 1

Enter element to insert: 125_
```


1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice: 1
Enter element to insert: 175
1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice:


Enter your choice: 2
100 50 25 75 150 125 175
1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice: 3
25 50 75 100 125 150 175
1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice: 4


Enter your choice: 4
25 75 50 125 175 150 100
1. CREATE
2. PRE-ORDER
3. IN-ORDER
4. POST-ORDER
5. EXIT
Enter your choice: 5

Program 11: Write a C Program to implement 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 un(int,int);
void main()
{
        clrscr();
        printf("\nImplementation 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("\nThe edges of Minimum Cost Spanning Tree are: ");
        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(un(u,v))
        {
                printf("\n%d edge (%d,%d) = %d",ne++,a,b,min);
                mincost =mincost+min;
        }
        cost[a][b]=cost[b][a]=999;
    }
    printf("\nMinimum cost = %d",mincost);
    getch();
}
int find(int i)
{
    while(parent[i])
    i=parent[i];
    return i;
}
int un(int i,int j)
{
    if(i!=j)
    {
        parent[j]=i;
```

```
            return 1;
        }
        return 0;
}
```

Output of Kruskal's

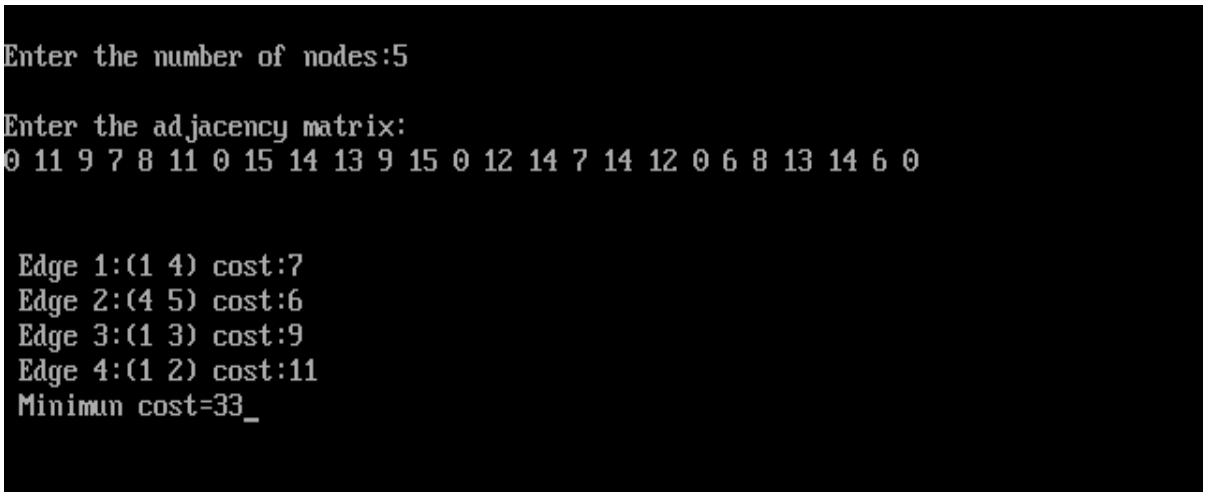
```
Implementation of Kruskal's algorithm

Enter the no. of vertices: 6

Enter the cost adjacency matrix:
0 5 10 0 0 0
5 0 0 45 55 20
10 0 0 0 0 25
0 45 0 0 35 50
0 55 0 35 0 45
0 20 25 50 45 0

The edges of Minimum Cost Spanning Tree are:
1 edge (1,2) = 5
2 edge (1,3) = 10
3 edge (2,6) = 20
4 edge (4,5) = 35
5 edge (2,4) = 45
Minimum cost = 115_
```

Program 12: Write a C Program to implement 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 of Prim's



Program 13: Write a C Program to implement N-Queens problem.

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
int board[20],count;
void queen(int,int);
void main()
{
        int n,i,j;
        clrscr();
        printf("* N-Queens Problem Using Backtracking *");
        printf("\n\nEnter number of Queens(Min. 4) :");
        scanf("%d",&n);
        queen(1,n);
        getch();
}
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)
                {
                        if(board[i]==j)
                                printf("\tQ");
                        else
                                printf("\t-");
                }
        }
}

int place(int row,int column)
{
        int i;
        for(i=1;i<=row-1;++i)
        {
                if(board[i]==column)
                        return 0;
                else
                if(abs(board[i]-column)==abs(i-row))
                        return 0;
        }
        return 1;
}
void queen(int row,int n)
{
        int column;
        for(column=1;column<=n;++column)
        {
                if(place(row,column))
                {
                        board[row]=column;
```

```
if(row==n)
                    print(n);
                else
                    queen(row+1,n);
            }
        }
}
```

Output of N-Queens

```
*** N-Queens Problem Using Backtracking ***

Enter number of Queens(Min. 4) :4_
```

```
Solution 1:
        1       2       3       4
1       -       Q       -       -
2       -       -       -       Q
3       Q       -       -       -
4       -       -       Q       -
Solution 2:
        1       2       3       4
1       -       -       Q       -
2       Q       -       -       -
3       -       -       -       Q
4       -       Q       -       -_
```