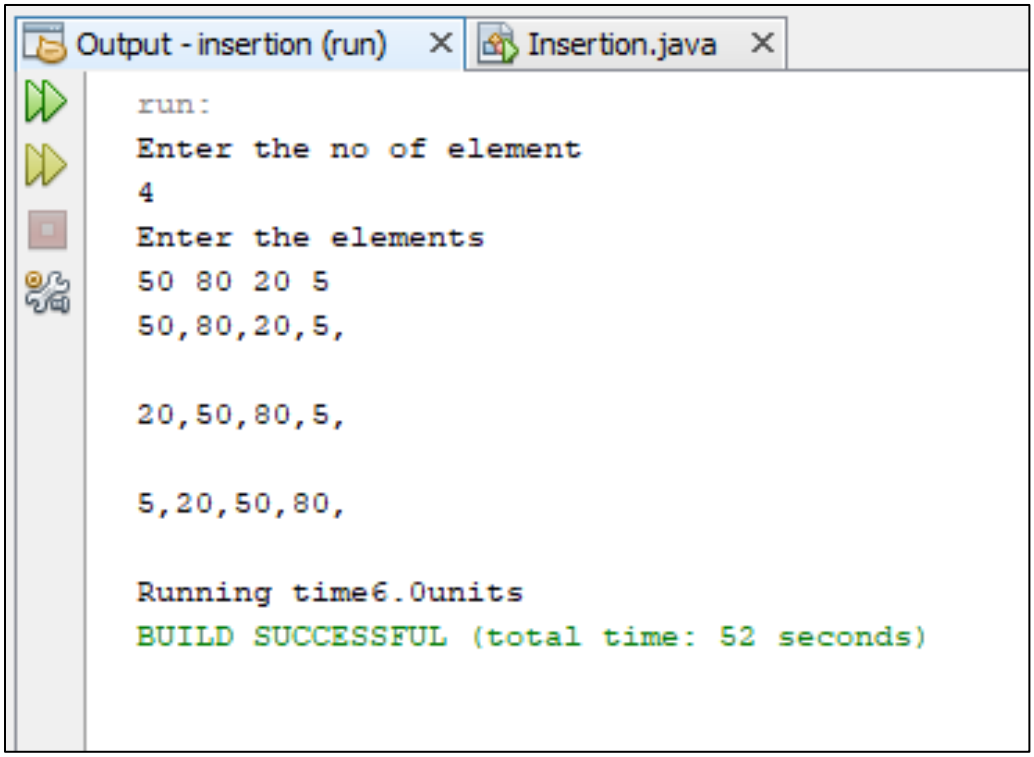
PRACTICAL NO: 1

AIM: Write a program to implement insertion sort and find the running time of the memory complexity.

PROGRAM CODE:

```
package insertion;
import java.util.Scanner;
public class Insertion {
   public static void main(String[] args) {
      System.out.println("Enter number of inputs in array:");
      Scanner a=new Scanner(System.in);
      int n=a.nextInt();
      int b[] = new int[n];
      System.out.println("Enter all elements:");
      for(int i=0;i<n;i++){
         b[i] = a.nextInt();
      }
      System.out.println("\n");
      insertionSort(b);
   }
   private static void printNumbers(int[] input){
      for(int i=0;i<input.length;i++){
         System.out.println(input[i]+",");
      }
      System.out.println("\n");
   }
   private static void insertionSort(int[] array) {
       int n=array.length;
       for(int j=1;j<n;j++){
          int key=array[j];
          int i=j-1;
          while((i>-1) && (array[i]>key)){
             array[i+1]=array[i];
             i--;
          }
          array[i+1]=key;
          printNumbers(array);
       }
      double time=n*(n-1)/2;
      System.out.println("Running Time: "+time+" units");
   }
}
```

OUTPUT:


Output - insertion (run)
Insertion.java
run:
Enter the no of element
4
Enter the elements
50 80 20 5
50,80,20,5,
20,50,80,5,
5,20,50,80,
Running time6.0units
BUILD SUCCESSFUL (total time: 52 seconds)

PRACTICAL NO: 2

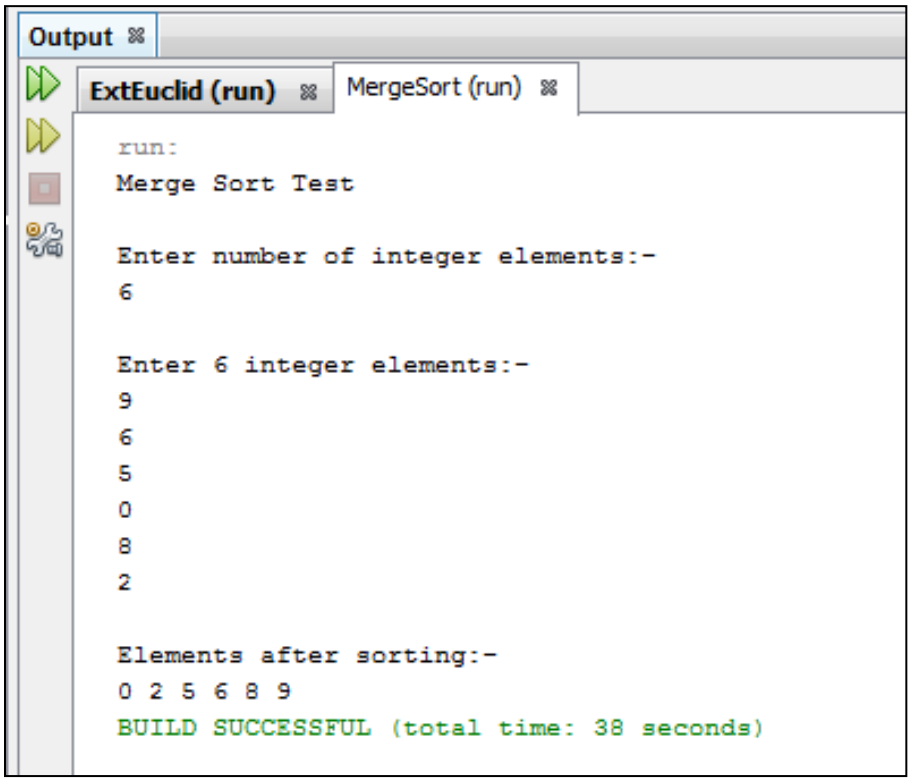
AIM: Write a program to implement merge sort algorithm. Compare the time and memory complexity.

PROGRAM CODE:

```
package mergesort;
import java.util.Scanner;
public class MergeSort
{
   public static void sort(int[] a, int low, int high){
      int N = high - low;
      if (N <= 1)
      return;
      int mid = low + N/2;
      sort(a, low, mid);
      sort(a, mid, high);
      int[] temp = new int[N];
      int i = low, j = mid;
      for (int k = 0; k<N; k++){
      if (i == mid)
         temp[k] = a[j++];
      else if (j == high)
         temp[k] = a[i++];
      else if (a[j]<a[i])
         temp[k] = a[j++];
      else
         temp[k] = a[i++];
      }
      for (int k = 0; k<N; k++)
         a[low + k] = temp[k];
      }
      public static void main(String[] args) {
         Scanner scan = new Scanner( System.in );
         System.out.println("Merge Sort Test\n");
         int n, i;
         System.out.println("Enter number of integer elements:-");
         n = scan.nextInt();
         int arr[] = new int[ n ];
         System.out.println("\nEnter &quot;+ n +&quot; integer elements:-");
```

```
        for (i = 0; i<n; i++)
        arr[i] = scan.nextInt();
        sort(arr, 0, n);
        System.out.println("\nElements after sorting:-");
        for (i = 0; i<n; i++)
        System.out.print(arr[i]+"");
        System.out.println();
        }
  }
```

Output:-



PRACTICAL NO: 3

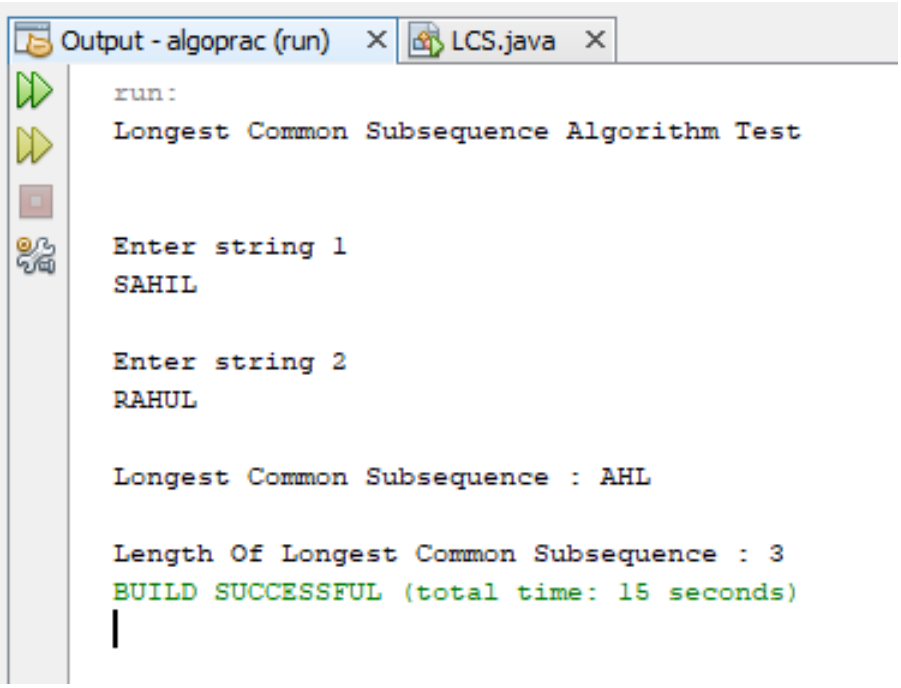
AIM: Write a program to implement Longest Common Subsequence(LCS) algorithm.

PROGRAM CODE:

```
package lcs;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class LCS
{
      int count=0;
      public String lcs(String str1, String str2){
      int l1 = str1.length();
      int l2 = str2.length();
      int[][] arr = new int[l1 + 1][l2 + 1];
      for (int i = l1 - 1; i >= 0; i--){
         for (int j = l2 - 1; j >= 0; j--){
            if (str1.charAt(i) == str2.charAt(j))
            arr[i][j] = arr[i + 1][j + 1] + 1;
         else
            arr[i][j] = Math.max(arr[i + 1][j], arr[i][j + 1]);
         }
      }
      int i = 0, j = 0;
      StringBuffer sb = new StringBuffer();
      while (i < l1 && j < l2){
      if (str1.charAt(i) == str2.charAt(j)){
      sb.append(str1.charAt(i));
      i++;
      j++;
      count++;
      }
      else if (arr[i + 1][j] >= arr[i][j + 1])
      i++;
      else
      j++;
      }
      return sb.toString();
   }
```

```
    public static void main(String[] args) throws IOException
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Longest Common Subsequence Algorithm Test\n");
        System.out.println("\nEnter string 1");
        String str1 = br.readLine();
         System.out.println("\nEnter string 2");
        String str2 = br.readLine();
        LCS obj = new LCS();
        String result = obj.lcs(str1, str2);
        System.out.println("\nLongest Common Subsequence : "+ result);
        System.out.println("\nLength Of Longest Common Subsequence : "+obj.count);
    }
}
```

Output:-



PRACTICAL NO;4

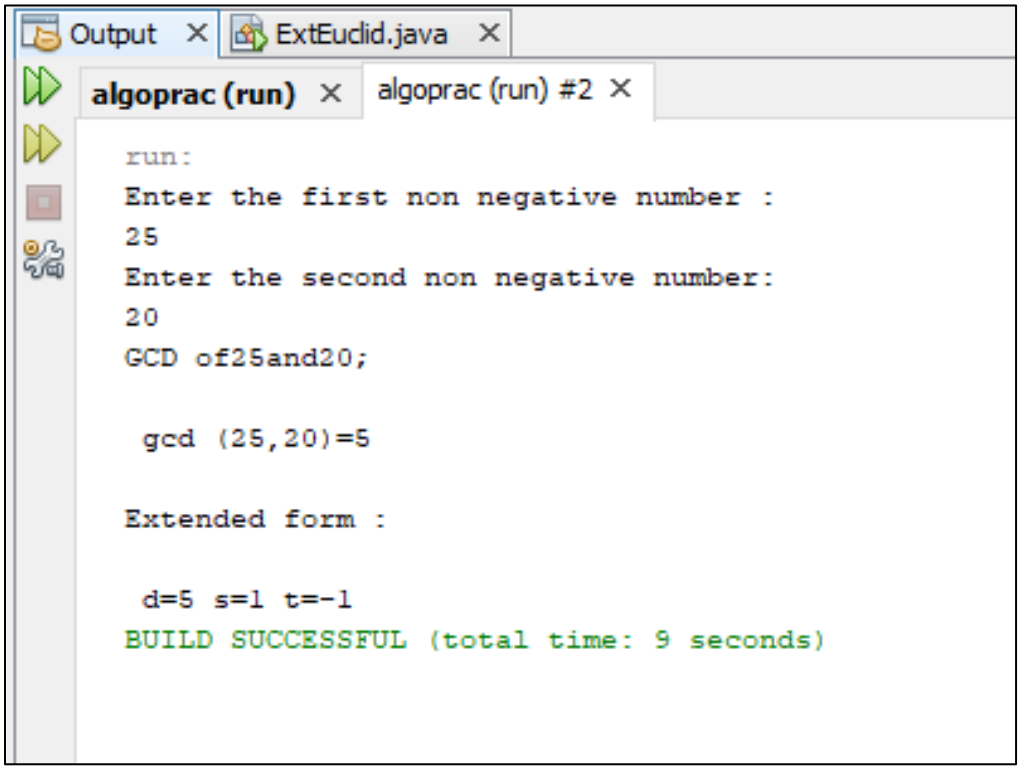
AIM: Write a program to implement Euclid's algorithm to implement gcd of two non negative integers a and b. Extend the algorithm to find x and y such that gcd(a,b) = ax+by. Compare the running time and recursive calls made in each case.

PROGRAM CODE:

```
package exteuclid;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class ExtEuclid {
   public static void main(String[] args) throws IOException {
      int [] ans = new int[3];
      int x,y,a,b;
      BufferedReader br = new BufferedReader (new InputStreamReader(System.in));
      System.out.println("Enter the first non negative number :");
      a=Integer.parseInt(br.readLine());
      System.out.println("Enter the second non negative number:");
      b=Integer.parseInt(br.readLine());
      ans=Euclid(a,b);
      System.out.println("GCD of" +a+ "and" +b+ ";");
      System.out.println("\n gcd ("+a+","+b+")="+ans[0]+"\n");
      System.out.println("Extended form :\n");
      System.out.println(" d="+ans[0]+" s="+ans[1]+" t="+ans[2]+"");
}
   public static int[]Euclid(int a, int b)
   {
      int[]ans=new int[3];
      int q;
      if(b==0)
      {
         ans[0]=a;
         ans[1]=1;
         ans[2]=0;
      }
      else{
         q=a/b;
         ans=Euclid(b,a%b);
         int temp=ans[1]-ans[2]*q;
```

```
            ans[1]=ans[2];
            ans[2]=temp;
        }
        return ans;
    }
}
```

OUTPUT:



PRACTICAL NO: 5

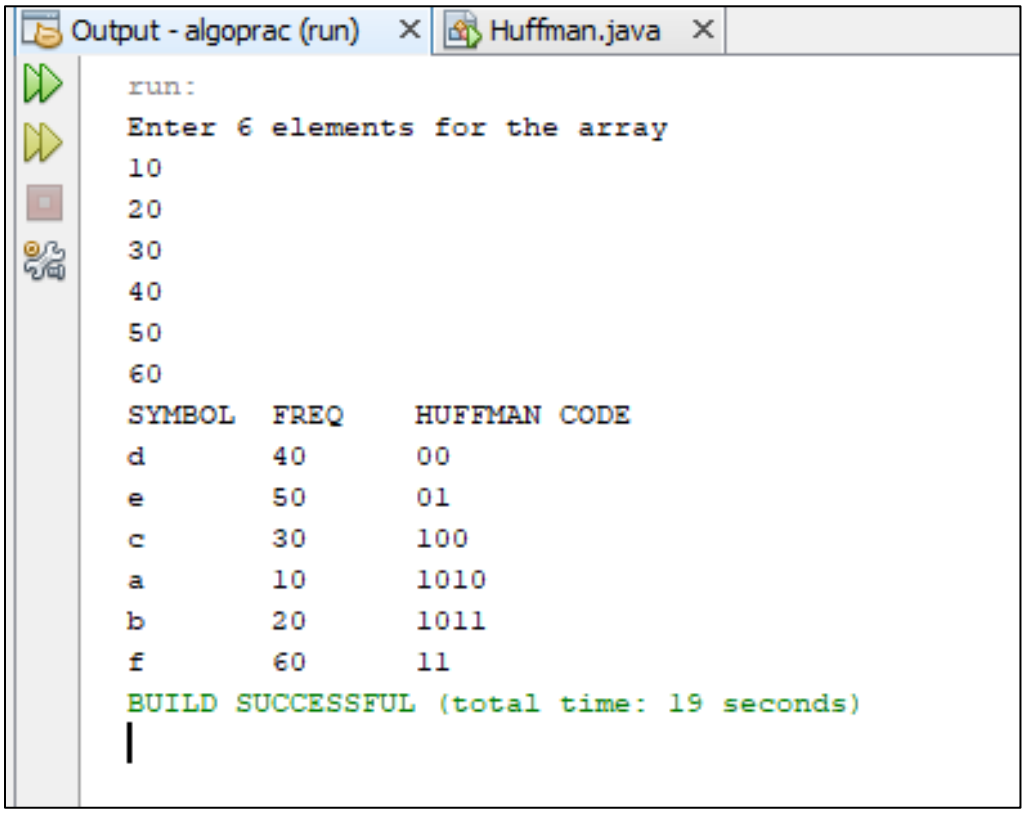
AIM: Write a program to implement Huffman's code algorithm.

PROGRAM CODE:

```
package huffman;
import java.util.*;
import java.io.*;
abstract class HuffmanTree implements Comparable<HuffmanTree> {
   public final int frequency;
   public HuffmanTree(int freq) {
      frequency = freq;
   }
   public int compareTo(HuffmanTree tree) {
      return frequency - tree.frequency;
   }
}
class HuffmanLeaf extends HuffmanTree {
   public final char value;
   public HuffmanLeaf(int freq, char val) {
      super(freq);
      value = val;
   }
}
class HuffmanNode extends HuffmanTree {
   public final HuffmanTree left, right;
   public HuffmanNode(HuffmanTree l, HuffmanTree r) {
      super(l.frequency + r.frequency);
      left = l;
      right = r;
   }
}
public class Huffman {
   public static HuffmanTree buildTree(int[] charFreqs, char[] test2) {
      PriorityQueue<HuffmanTree> trees = new PriorityQueue<HuffmanTree>();
      for (int  i = 0; i < charFreqs.length;i++)
      if (charFreqs[i] > 0) {
         trees.offer(new HuffmanLeaf(charFreqs[i], test2[i]));
      }
      assert trees.size() > 0;
      while (trees.size() > 1) {
         HuffmanTree a = trees.poll();
         HuffmanTree b = trees.poll();
         trees.offer(new HuffmanNode(a, b));
      }
      return trees.poll();
```

```
    }
    public static void printCodes(HuffmanTree tree, StringBuffer prefix) {
        assert tree != null;
        if (tree instanceof HuffmanLeaf) {
            HuffmanLeaf leaf = (HuffmanLeaf) tree;
            System.out.println(leaf.value + "\t" + leaf.frequency + "\t" + prefix);
        } else if (tree instanceof HuffmanNode) {
            HuffmanNode node = (HuffmanNode) tree;
            prefix.append("0");
            printCodes(node.left, prefix);
            prefix.deleteCharAt(prefix.length() - 1);
            prefix.append("1");
            printCodes(node.right, prefix);
            prefix.deleteCharAt(prefix.length() - 1);
        }
    }
    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        System.out.println("Enter 6 elements for the array");
        String str = "abcdef";
        int n = 6;
        char[] test2 = str.toCharArray();
        int charFreqs[] = new int[n];
        for(int i = 0; i < n; i++)
        {
            charFreqs[i] = s.nextInt();
        }
        HuffmanTree tree = buildTree(charFreqs, test2);
        System.out.println("SYMBOL\tFREQ\tHUFFMAN CODE");
        printCodes(tree, new StringBuffer());  }
}
```

OUTPUT:-


Output - algoprac (run)
Huffman.java
run:
Enter 6 elements for the array
10
20
30
40
50
60
SYMBOL FREQ HUFFMAN CODE
d 40 00
e 50 01
c 30 100
a 10 1010
b 20 1011
f 60 11
BUILD SUCCESSFUL (total time: 19 seconds)

PRACTICAL NO: 6

AIM: Write a program to implement Kruskal's algorithm.

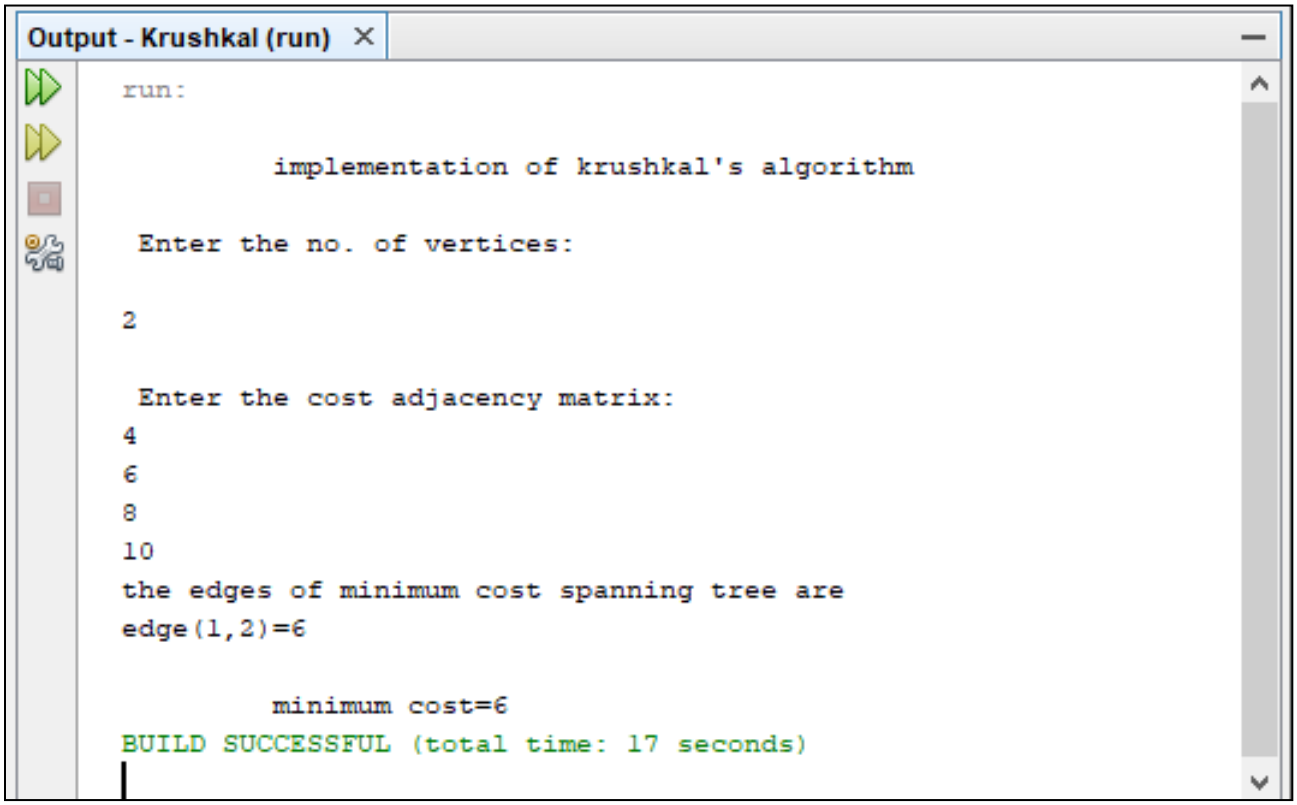
PROGRAM CODE:

```
package krushkal;
import java.util.*;
public class Krushkal
{
   public final static Scanner STDIN_SCANNER = new Scanner(System.in);
   public static int i,j,k,a,b,u,v,n,ne=1;
   public static int min,mincost=0;
   public static int[][]cost = new int[20][20];
   public static int[] parent = new int[20];

   public static void main(String[] args)
   {
      System.out.println("\n\t implementation of krushkal's algorithm");
       System.out.println("\n Enter the no. of vertices:");
       n=STDIN_SCANNER.nextInt();
       System.out.println("\n Enter the cost adjacency matrix:");
       for(i=1;i<=n;i++)
       {
          for(j=1;j<=n;j++)
          {
             cost[i][j]=STDIN_SCANNER.nextInt();
             if(cost[i][j]==0)
             {
                cost[i][j]=999;
             }
          }
       }
       System.out.println("the edges of minimum cost spanning tree are");
       while(ne<n)
       {
          min = 999;
          for(i=1;i<=n;i++)
          {
             for(j=1;j<=n;j++)
             {
               if(cost[i][j]<min)
               {
                  min = cost[i][j];
                  a=u=i;
                  b=v=j;
```

```
            }
          }
        }
        if(v!=u)
        {
          parent[v]=u;
          ne++;
          System.out.println("edge("+a+","+b+")="+ min);
          mincost+=min;
        }
        cost[a][b] = (cost[b][a] = 999);
      }
      System.out.println("\n\t minimum cost=" +mincost);
   }
}
```

OUTPUT:



PRACTICAL NO: 7

AIM: Write a program to Implement Dijkstra's Algorithm.

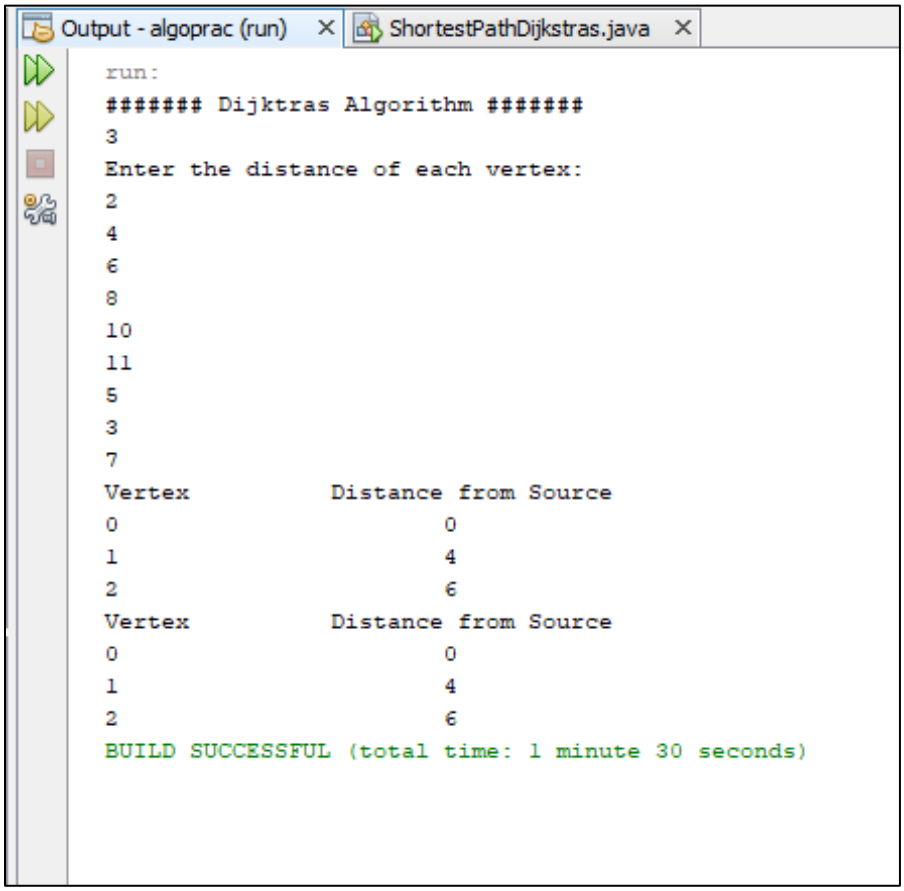
PROGRAM CODE:

```
package hello;
import java.util.*;
import java.lang.*;
import java.io.*;

public class ShortestPathDijkstras
{
   static int V=5;
   int minDistance(int dist[],Boolean sptSet[])
   {
      int min=Integer.MAX_VALUE,min_index=-1;
      for(int v=0;v<V;v++)
      if(sptSet[v]==false && dist[v]<=min)
      {
         min= dist[v];
         min_index=v;
      }
      return min_index;
   }
   void printSolution(int dist[],int n)
   {
      System.out.println("Vertex\t\tDistance from Source");
      for(int i=0;i<V;i++)
      System.out.println(i+"\t\t\t"+dist[i]);
   }
   void dijkstra(int graph[][],int src)
   {
      int dist[]=new int[V];
      Boolean sptSet[]=new Boolean[V];
      for(int i=0;i<V;i++)
      {
         dist[i]=Integer.MAX_VALUE;
         sptSet[i]=false;
      }
      dist[src]=0;
      for(int count=0;count<V-1;count++)
      {
         int u=minDistance(dist,sptSet);
         sptSet[u]=true;
         for(int v=0;v<V;v++)
```

```
        {
            if(!sptSet[v]&& graph[u][v]!=0
                &&dist[u]!=Integer.MAX_VALUE&&dist[u]+graph[u][v]<dist[v])
            {
               dist[v]=dist[u]+graph[u][v];
            }
        }
        printSolution(dist,V);
    }
  }
  public static void main(String[]args)
  {
    Scanner scan=new Scanner(System.in);
    int vertices;
    int[][] graph;
    System.out.println("######## Dijktras Algorithm ########");
    V=scan.nextInt();
    graph=new int[V][V];
    System.out.println("Enter the distance of each vertex:");
    for (int i=0;i<V;i++)
    {
       for(int j=0;j<V;j++)
       {
          graph[i][j]=scan.nextInt();
       }
    }
    ShortestPathDijkstras obj1=new ShortestPathDijkstras();
    obj1.dijkstra(graph, 0);
  }
}
```

OUTPUT:-


Output - algoprac (run)
ShortestPathDijkstras.java
run:
####### Dijktras Algorithm #######
3
Enter the distance of each vertex:
2
4
6
8
10
11
5
3
7
Vertex Distance from Source
0 0
1 4
2 6
Vertex Distance from Source
0 0
1 4
2 6
BUILD SUCCESSFUL (total time: 1 minute 30 seconds)

PRACTICAL NO: 8

AIM: Write a program to implement greedy set cover algorithm to solve set covering problem.

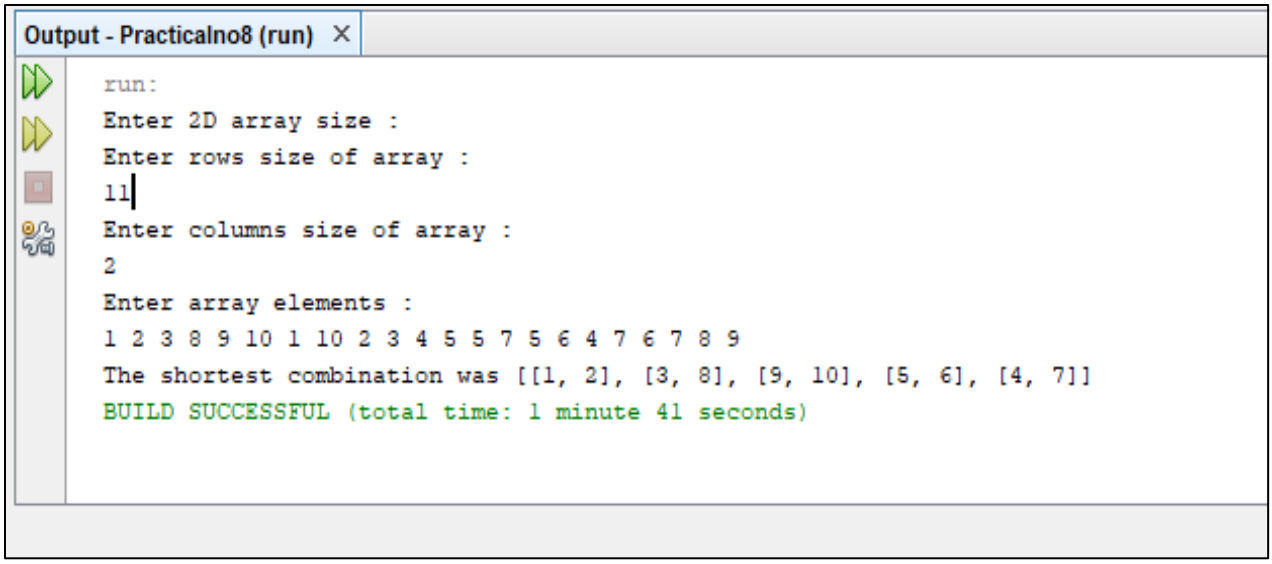
PROGRAM CODE:

```
package practicalno8;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Scanner;
import java.util.Set;

public class Practicalno8
{
   interface Filter<T>
   {
      boolean matches(T t);
   }
   private static <T> Set<T> shortestCombination(Filter<Set<T>> filter,List<T> listOfSets)
   {
      final int size = listOfSets.size();
      if (size > 20)
         throw new IllegalArgumentException("Too many combinations");
         int combinations = 1 << size;
         List<Set<T>> possibleSolutions = new ArrayList<Set<T>>();
      for (int l = 0; l < combinations; l++)
      {
         Set<T> combination = new LinkedHashSet<T>();
         for (int j = 0; j < size; j++)
         {
            if (((l >> j) & 1) != 0)
               combination.add(listOfSets.get(j));
         }
         possibleSolutions.add(combination);
      }
      Collections.sort(possibleSolutions, new Comparator<Set<T>>()
      {
         public int compare(Set<T> o1, Set<T> o2)
         {
            return o1.size() - o2.size();
         }
      }
      );
```

```
        for (Set<T> possibleSolution : possibleSolutions)
        {
            if (filter.matches(possibleSolution))
                return possibleSolution;
        }
        return null;
    }
    public static void main(String[] args){
        System.out.println("Enter 2D array size : ");
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter rows size of array : ");
        int rows=sc.nextInt();
        System.out.println("Enter columns size of array : ");
        int columns=sc.nextInt();
        System.out.println("Enter array elements : ");
        Integer arrayOfSets[][]=new Integer[rows][columns];
        for(int i=0; i<rows;i++)
        {
            for(int j=0; j<columns;j++)
            {
                arrayOfSets[i][j]=sc.nextInt();
            }
        }
        Integer[] solution = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        List<Set<Integer>> listOfSets = new ArrayList<Set<Integer>>();
        for (Integer[] array : arrayOfSets)listOfSets.add(new
LinkedHashSet<Integer>(Arrays.asList(array)));
        final Set<Integer> solutionSet = new LinkedHashSet<Integer>(Arrays.asList(solution));
        Filter<Set<Set<Integer>>> filter = new Filter<Set<Set<Integer>>>()
        {
            public boolean matches(Set<Set<Integer>> integers)
            {
                Set<Integer> union = new LinkedHashSet<Integer>();
                for (Set<Integer> ints : integers)
                    union.addAll(ints);
                return union.equals(solutionSet);
            }
        };
        Set<Set<Integer>> firstSolution = shortestCombination(filter,listOfSets);
        System.out.println("The shortest combination was " + firstSolution);
    }
}
```

OUTPUT:


Output - Practicalno8 (run)
run:
Enter 2D array size :
Enter rows size of array :
11
Enter columns size of array :
2
Enter array elements :
1 2 3 8 9 10 1 10 2 3 4 5 5 7 5 6 4 7 6 7 8 9
The shortest combination was [[1, 2], [3, 8], [9, 10], [5, 6], [4, 7]]
BUILD SUCCESSFUL (total time: 1 minute 41 seconds)