**Q1) Write an algorithm to find the shortest path from the source vertex of a graph to all the vertices and print the paths all well in a graph. What is the runtime of the algorithm?**

```
public class FirstQuestion {   public static void
dijkstra(int[][] graph, int source) {     int count =
graph.length;     boolean[] visitedVertex = new
boolean[count];     int[] distance = new int[count];
   for (int i = 0; i < count; i++) {
visitedVertex[i] = false;
     distance[i] = Integer.MAX_VALUE;
   }
   distance[source] = 0;
   for (int i = 0; i < count; i++) {

     int u = findMinDistance(distance, visitedVertex);
visitedVertex[u] = true;

     for (int v = 0; v < count; v++) {        if (!visitedVertex[v] && graph[u][v] != 0 &&
(distance[u] + graph[u][v] < distance[v])) {          distance[v] = distance[u] + graph[u][v];
       }
     }
   }
   for (int i = 0; i < distance.length; i++) {
     System.out.println(String.format("Distance from %s to %s is %s", source, i, distance[i]));
   }

 }
```

```
  private static int MinDistance(int[] distance, boolean[] visitedVertex) {
int minDistance = Integer.MAX_VALUE;

   int minDistanceVrtx = -1;     for (int i = 0; i <
distance.length; i++) {      if (!visitedVertex[i] &&
distance[i] < minDistance) {        minDistance =
distance[i];        minDistanceVrtx = i;
     }
   }
   return DistanceVertex;
 }


  public static void main(String[] args) {
    int graph[][] = new int[][] { { 0, 0,7, 2, 0, 0, 0 }, { 0, 0, 4, 0, 0, 3, 0 }, { 7, 4, 0, 7, 3, 0, 0 },
       { 4, 0, 7, 0, 0, 0, 7}, { 0, 0, 3, 0, 0, 2, 0 }, { 0, 3, 0, 0, 4, 0, 7 }, { 0, 0, 0, 7, 0, 7, 0 } };
   Dijkstra T = new Dijkstra();
   T.dijkstra(graph, 0);
  }
}
```

**Runtime = O(E Log V)**

**Q2) Given a Huffman coding tree, which algorithm would you use to get the codewords for all the symbols? What is its time-efficiency class as a function of the alphabet size?**

Greedy Algorithm will be the best for this;

The algorithm is work like that;

1) The Huffman Coding Tree is traversed from the root.

2) Create an array A which will store the codeword.

3) Append 0 to the array A for the left child and 1 for the right child.

4.) If we reach the leaf node in the tree, print the contents of the array A.

5) Follow these steps recursively.

The overall time efficiency or the time complexity of this algorithm is O(n) where n = input size or alphabet size.

But if we need to build the tree, then the time complexity will be O(nlogn) because of the involvement of min-heap data structure.

## Codes

```
public void decode(Node root, String code, boolean[] coding) {
   int i = 0;
if(root == null) {
      throw new IllegalArgumentException("Given tree is empty");
   }
   if(i  !=  coding.length)  {              if  (root
instanceof            InternalNode)           {
InternalNode t = (InternalNode)root;
         if(coding[i] == false) {
            t.left = (InternalNode)root;
            i++;
            decode(t.left, code, coding);
         }
         if(coding[i] == true) {
            t.right = (InternalNode)root;
            i++;
            decode(n.right, code, coding);
         }
      }
```

```
      else if (root instanceof LeafNode) {
LeafNode l = (LeafNode)root;
          code += l.data;
          i++;
          decode(root, code, coding);
       }
    }
}
```

**Q3) Give an algorithm for determining if a graph is two-colorable. if it is possible to color every vertex red or blue so that no two vertices of the same color have an edge between them. Your algorithm should run in time O(V + E), where V is the number of vertices and E is the number of edges in the graph. You should assume that the graph is undirected and that the input is presented in adjacency-list form.**

There is a simple algorithm for determine to if graph is tow-colorable and assigning colors to its vertices: do a breadth-first search, assigning red to the first layer, blue to the second layer, red to the third layer,and goes like that. Then go over all the edges and checks if the two endpoints of this edge have different colors. This algorithm is O(|V|+|E|) and the last step ensures its correctness.