ALGORITHM

```
       /* displayPath = (TextView) findViewById(R.id.my_shortest_path);*/
        int number_of_nodes = length;
        int adjacency_matrix[][] = new int[number_of_nodes + 1][number_of_nodes + 1];
            for (int i = 1; i <= number_of_nodes; i++)
            {
            for (int j = 1; j <= number_of_nodes; j++)
           {
               adjacency_matrix[i][j] = cost[i-1][j-1];
           }
        }
        tsp(adjacency_matrix);
}

    public ShortestPath()
    {
        stack = new Stack<Integer>();
    }

    public void tsp(int adjacencyMatrix[][])
    {
        numberOfNodes = adjacencyMatrix[1].length - 1;
        int[] visited = new int[numberOfNodes + 1];
        visited[1] = 1;
        stack.push(1);
        int element, dst = 0, i;
        int min = Integer.MAX_VALUE;
        boolean minFlag = false;
       displayPath.append("1"+"\n\n");+
       while (!stack.isEmpty())
        {
```

```
        element = stack.peek();


i = 1;
        min = Integer.MAX_VALUE;
        while (i <= numberOfNodes)
        {
            if (adjacencyMatrix[element][i] > 1 && visited[i] == 0)
            {
                if (min > adjacencyMatrix[element][i])
                {
                    min = adjacencyMatrix[element][i];
                    dst = i;
                    minFlag = true;
                }
            }
            i++;
        }
        if (minFlag)
        {
            visited[dst] = 1;
            stack.push(dst);
            displayPath.append(String.valueOf(dst)+" - ");
            displayPath.append(getCompleteAddressString(mlocation[dst-1].latitude,mlocation[dst-1].longitude) + "\n\n");
            minFlag = false;
            continue;
        }
        stack.pop();
    }
```

ANALYSIS:

There are at the most 2^nn2n.n sub-problems and each one takes linear time to solve. Therefore, the total running time is $O(2^n n^2)$