5b. Dijkstra’s algorithm and Analyze its Time Complexity

```
public class DijkstraAlgorithm {

 public void dijkstraAlgorithm(int[][] graph, int
source) {
  int nodes = graph.length;
  boolean[] visited_vertex = new
boolean[nodes];
  int[] dist = new int[nodes];
  for (int i = 0; i < nodes; i++) {
   visited_vertex[i] = false;
   dist[i] = Integer.MAX_VALUE;
  }
  dist[source] = 0;
  for (int i = 0; i < nodes; i++) {
   int u = find_min_distance(dist, visited_vertex);
   visited_vertex[u] = true;
   for (int v = 0; v < nodes; v++) {
    if (!visited_vertex[v] && graph[u][v] != 0
&& (dist[u] + graph[u][v] < dist[v])) {
     dist[v] = dist[u] + graph[u][v];
    }
   }
  }
  for (int i = 0; i < dist.length; i++) {
   System.out.println(String.format("Distance
from Vertex %s to Vertex %s is %s", source, i,
dist[i]));
  }
 }
 private static int find_min_distance(int[] dist,
boolean[] visited_vertex) {
  int minimum_distance =
Integer.MAX_VALUE;
  int minimum_distance_vertex = -1;
  for (int i = 0; i < dist.length; i++) {
   if (!visited_vertex[i] && dist[i] <
minimum_distance) {
    minimum_distance = dist[i];
    minimum_distance_vertex = i;
   }
  }
  return minimum_distance_vertex;
 }

 public static void main(String[] args) {
  int graph[][] = new int[][] {
   { 0, 1, 1, 2, 0, 0, 0 },
   { 0, 0, 2, 0, 0, 3, 0 },
   { 1, 2, 0, 1, 3, 0, 0 },
   { 2, 0, 1, 0, 2, 0, 1 },
   { 0, 0, 3, 0, 0, 2, 0 },
   { 0, 3, 0, 0, 2, 0, 1 },
   { 0, 2, 0, 1, 0, 1, 0 }
  };
  DijkstraAlgorithm Test = new
DijkstraAlgorithm();
  Test.dijkstraAlgorithm(graph, 0);
 }
}
```

<u>Pgm6 a &b :</u>

Implement Warshall and Floyd’s algorithm

```
import java.lang.*;
public class AllPairShortestPath {
 final static int INF = 99999, V = 4;

 void floydWarshall(int dist[][]) {
  int i, j, k;
  for (k = 0; k < V; k++) {
   for (i = 0; i < V; i++) {
    for (j = 0; j < V; j++) {
     if (dist[i][k] + dist[k][j] < dist[i][j])
      dist[i][j] = dist[i][k] + dist[k][j];
    }
   }
  }
  printSolution(dist);
 }

 void printSolution(int dist[][]) {
  System.out.println("The following matrix shows the shortest
distances between every pair of vertices");
  for (int i = 0; i < V; ++i) {
   for (int j = 0; j < V; ++j) {
    if (dist[i][j] == INF)
     System.out.print("INF ");
    else
     System.out.print(dist[i][j] + " ");
   }
   System.out.println();
  }
 }

 public static void main(String[] args) {
  int graph[][] = { { 0, 5, INF, 10 },
   { INF, 0, 3, INF },
   { INF, INF, 0, 1 },
   { INF, INF, INF, 0 } };
  AllPairShortestPath a = new AllPairShortestPath();
  a.floydWarshall(graph);
 }
}
```

Pgm 8 :Implement LCM algorithm

Transform and Conquer Approach

```
public class LCMCalculator {
 private static int gcd(int a, int b) {
  if (b == 0)
   return a;
  return gcd(b, a % b);
 }

 private static int lcm(int a, int b) {
  return (a * b) / gcd(a, b);
 }

 public static int lcmArray(int[] arr) {
  int result = arr[0];
  for (int i = 1; i < arr.length; i++) {
   result = lcm(result, arr[i]);
  }
  return result;
 }

 public static void main(String[] args) {
  int[] numbers = {12, 15, 20, 25};
  int result = lcmArray(numbers);
  System.out.println("LCM of the array is: " + result);
 }
}
```