**Assignment 2 – PS11 - [Hospital Emergency]**

1. Which is the shortest route to reach the airport from the hospital? The hospital and airport nodes should be taken as an input and is not fixed to the nodes mentioned in the graph above.

Ans: Shortest route from the hospital 'a' to reach the airport 'o' is [a, b, c, j, k, m, o] and it has minimum travel distance 23Km

2. How long would it take for the ambulance to reach the airport if the ambulance travels at an average speed of 80 km/hr.

Ans: it will take 17:25 minutes for the ambulance to reach the airport if the ambulance travels at an average speed of 80 km/hr.

**Algorithm to perform the above task**:

We have used a variant of the well-known Dijkstra’s algorithm to find efficient shortest path from Manipal Hospital to the Airport.

And we used Binary Heap for Priority Queue implementation and here are the steps involved,

**1)**Create a Min Heap of size V where V is the number of vertices in the given graph. Every node of min heap contains vertex number and distance value of the vertex.  
**2)** Initialize Min Heap with source vertex as root (the distance value assigned to source vertex a is 0). The distance value assigned to all other vertices is INF (infinite).  
**3)**While Min Heap is not empty, do following

* Extract the vertex with minimum distance value node from Min Heap. Let the extracted vertex be u.
* For every adjacent vertex v of u, check if v is in Min Heap. If v is in Min Heap and distance value is more than weight of u-v plus distance value of u, then update the distance value of v.

**Pseudocode:**

1) Initialize distances of all vertices from source as infinite.

2) Create an empty priorityQueue PQ. Every item

of PQ is a pair (weight, vertex). Weight (or

distance) is used as first item of pair

as first item is by default used to compare

two pairs

3) Insert source vertex into PQ and make its

distance as 0.

4) While either PQ doesn't become empty

a) Extract minimum distance vertex from PQ.

Let the extracted vertex be U.

b) Loop through all adjacent of U and do

following for every vertex V.

// If there is a shorter path to V through U.

If distance[V] > distance[U] + weight(U, V)

(i) Update distance of V, i.e., do

distance [V] = distance [U] + weight(U, V)

(ii) Insert v into the PQ (Even if V is

already there)

5) Print distance array distance [] to print all shortest

paths.

**Time complexity:**

heapifyTopToBottom(): O(ElogV) where E is number of edges and V is the number of vertices of the graph

initialize(): O(V)

getShortestPath(): O(V)

So overall time complexity of the algorithm is O(ElogV)

By using tree set or any other algorithm it would have taken O(E\*V) or O(V\*V). However, by using the heapify algorithm, the time taken for the algorithm = time taken to heapify all the vertices i.e. O(ElogV).

Hence this is an efficient algorithm in comparison with any other algorithm.