A) Find Cheapest Path

Assuming we have already had a minHeap that is sorted by minCost

```
Given a start and destination City
ChangeKey of start, set minCost as 0
While minHeap is not empty
    Extract the min of the heap (First iteration would be the start)
    For every neighbor that the extract min has
        Check how far away the neighbor City is away from the extracted min
        If the existing neighbor minCost is greater than the weight of the edge plus the minCost
        of the extracted node, then update minCost of the extract to the sum of the edge cost
        and minCost of the extracted node.

Get the minCost of the destination city (the minimum distance away from the start city) and call it
distance
While distance is not zero
    Check neighbors of the node (first iteration would be destination node)
        If the minCost of the neighbor is equal to the difference of its weight edge from distance
            Add to path list
            Update the node to the neighbor
Return path list
```

Since we are finding the distance of each City in the minHeap away from the start node, this will give us a time complexity of O(n+m)

Then, since we are iterating from the destination node all the way back to the beginning of the start node, we will at worst take O(n) time to get our result. Thus, this will take O(n*(n+m)) runtime. At worst, it will take O(n^2) runtime.

B) Find Minimum Total Cost

Assuming the minHeap is sorted

```
While the minHeap is not empty
    Extract the min
    Check the neighbors of the extracted min
        Get the weight of the edge connecting the extracted min and the neighbor
        If the weight is lower than the recorded edge weight
            Replace with the lowest edge weight into minCost
Iterate through every city and get its minCost which represents the lowest edge weight, add them, and return the total cost
```

Finding the lowest edge of each City will take O(n+m) runtime, then iterating through all of it and getting the total lowest cost will take O(n) runtime. Thus, the total time complexity would be O(n*(n+m)) to find the lowest cost to maintain the cities connected. Worst case scenario would be O(n^2) runtime.

C) Adjacency List vs Adjacency Matrix

In terms of space complexity and time complexity, it is better to use adjacency list since it will be O(n+m) runtime, where n is nodes and m is the edges. If it was a matrix, then it would have been a runtime of O(n^2) in an undirected tree. However, if you were to try and find the edge between two nodes, a matrix would take O(1) runtime rather the linear runtime of an adjacency list.