Stamati Morellas
COM S 311 - Homework 5
Recitation: Tuesday @ 1:10 pm
Due: 11/3/19

[1]

Traverse the set of edges E in graph G and evaluate the weight of each edge e to see if that same edge exists in T, the MST of G. If it does exist, skip. If not, subtract S from the weight of edge e and add edge e to the set of edges in E' in G'.Repeat this until no more edges remain. Next, we must compute the MST of G'. We will do this using *Kruskal's Algorithm*. Sort all the edges E' in G' in order of increasing weight. Then, pick the smallest edge and check if it forms a cycle with the spanning tree T' formed up to this point. If the cycle is not formed, include this edge. If else, discard it. Repeat this until there are (V - 1) remaining in T'. This algorithm should take approximately $O(mlogn)$ time.

[2]

We can compute the MST of G' using the *Reverse-Delete Algorithm.* Assume that the edges connecting nodes in S to other nodes in S have the highest weight and that the edges connecting nodes in set S to the rest of the nodes in V, but not in S, have the next highest weight. We then start deleting edges in order of decreasing cost while considering if deleting edge e will disconnect the current node from the rest of the graph as the Reverse-Delete Algorithm states. Once completed, each node in set S should have 1 edge connecting it to another node. When looking at the results, if every node in set S has 1 edge, then $MST(G, S)$ succeeded. Otherwise, $MST(G, S)$ does not exist. This algorithm should take approximately $O(mlogn)$ time.

[3]

To create this algorithm, we must first evaluate the sum of the nodes connected to d[v] with the weight of the incoming edges from those nodes. Each vertex d[v] is equivalent to a sum of the weight of an edge from some vertex a to v, and the cost of the nodes going to d[a] Once we have this, we loop through each vertex v in d[v] and for each incoming edge for vertex v from a given node, evaluate whether or not this is the shortest path to vertex v. Repeat this until every vertex v has been evaluated. If the algorithm makes it through all the vertices in d[v], the program will return true. This algorithm will take $O(m) + O(m)$ time, which is approximately equal to $O(m)$ time.

[4]

This algorithm can be constructed by using a *Greedy Algorithm for Interval Scheduling*. Start by computing the total baking time for each cake, which can be calculated with the given equation $C_i = b_i + d_i$ where b_i is the time it takes in minutes for each cake to bake and d_i is the time in minutes it takes to decorate each cake. This will take *O(n)* time. Next, order the cakes by increasing baking times. Create an array to keep track of the sequence of scheduled activities. Then for each cake, if the time it takes to make is shorter than that of the previous cake, add it to the schedule and update the previous finish time. Repeat this until there are no more cakes left. The running time of this algorithm is *O(nlogn)* time.