15-351 / 15-650 / 02-613: Exercises 1a

1. Let G be a graph where every edge has a different weight (as we assumed in class). Show that the edge with the smallest edge weight overall is in the minimum spanning tree.

2. Consider the following problem:

   ```
   Let G be a graph where every edge has a different weight
   (as we assumed in class). Show that there is only one,
   unique minimum spanning tree in this case.
   ```

   For each of the following potential solutions, determine if it is correct, and if it is not correct, explain why.

   (a) Let $G = (V, E)$ be a graph where every edge has a distinct weight. Suppose, for the sake of contradiction, that there are two distinct minimum spanning trees, T_1 and T_2. Since T_1 and T_2 are distinct trees, their edge sets must be different. Let $W(T_1)$ be the sum of the edge weights in T_1, and $W(T_2)$ be the sum of the edge weights in T_2.

   Because all edge weights in G are distinct, no two edges share the same weight. Since T_1 and T_2 are composed of different edges drawn from a set of strictly distinct numbers, the toal sums of their weights must be different. Therefore, $W(T_1) \neq W(T_2)$.

   However, by definition, both T_1 and T_2 are minimium spanning trees, which emeans they must have the exact same minimum total weight. This leads to a contradiction. Therefore, our initial assumption must be false, and there can only be one unique MST.

   (b) Suppose there is another, different, MST T_2 from our original tree T_1. It must contain at least one differing edge d. If this edge is added to the original MST, then a cycle is created where some edge m is the maximum edge. By the cycle property, m can't belong to any MST. Since m must belong to either T_1 or T_2 which are both MSTs, this leads to a contradiction.

   (c) We can prove this by looking at Kruskal's ALgorithm, which correctly finds the MST of a any graph. In Kruskal's, the first step is to sort all edges in increasing order of weight. Because every edge weight is unique, there are no ties to break. Therefore, there is only one strictly unique sorted order of edges. The algorithm then iterates through this unique list one by one, adding an edge to the tree if and only if it does not create a cycle. Because the input order is locked and unique, and the cycle-checking logic is completely deterministic, there are no arbitrary choices to make, so every time you run Kruskal, it will output the same MST. Thus, the MST is unique.

3. Let G be a graph of n nodes where every edge weight is either 1 or 2 (now edges can have the same weight). Let G' be G with edges of weight 2 removed. Show that the weight of the minimum spanning tree of G is $n-2+\text{cc}(G')$, where $\text{cc}(G')$ is the number of connected components of G'.

4. Let G be a graph with positive, integer edge weights where several edges may have the same weights. Show that, for all integers i, the number of edges of weight i is the same in every minimum spanning tree of G. *Hint: Start your answer with: "Suppose not. Let T_1 and T_2 be two minimum spanning trees that do not have the same number of edges of some weights."*

5. You are given an undirected graph $G = (V, E)$ with edge weights $d(u, v)$, which you can assume are all distinct, and you are given a minimum spanning tree T on G. You expect one of the edges of G to disappear at some time in the future, but you don't know which edge it will be, and when the edge does disappear, you'll need to find another minimum spanning tree very quickly. For example, the MST represents a communication network for stock traders, and if a link fails, you have to fix it as fast as possible.

   For this problem, you may assume that the graph is densely enough connected that there the disappearing edge doesn't cause the graph to become disconnected. Also, of course, if the edge that disappears is not actually in T, then the MST won't change.

   Give an efficient algorithm to preprocess T and G to label each edge e in T with another edge $r(e)$ of G so that if e disappears, adding $r(e)$ to the tree creates a new minimum spanning tree in the modified graph.For this problem, try to be as efficient as possible, but so long as you do not brute force the solution, you will get credit—more specifically, there exists an $O(|V||E|)$ solution, but you will get full credit so long as your solution is $O(|V||E| \log |V|)$.

6. As you'll recall, Prim's algorithm for finding a minimum spanning tree (MST) works through repeatedly choosing the minimum weight edge on the frontier to grow the tree from an arbitrary starting point. A naive implementation of Prim's might loop over all edges to find the minimum weight next edge, which would result in an $O(|E||V|)$ runtime. On the other hand, using a heap, we can improve the runtime to $O(|E| \log |V|)$.

   What if someone produced a better version of a heap (let's call it an *improved heap*) where the extract-min operation still takes $O(\log |V|)$ time, but the decrease-key operation takes only constant $O(1)$ time? What would be the runtime of Prim's using such an *improved heap*? Justify your answer.

7. Consider the following proposed divide-and-conquer algorithm for solving the minimum spanning tree problem on a connected graph G with unique weights.

```
Maybe-MST(G=(V,E)):
    If |V| <= 2:
        return G
```

```
Partition V into any balanced connected sets V1 and V2.
Let E1 be edges entirely between nodes in V1.
Let E2 be edges entirely between nodes in V2.
Let E3 be edges crossing between V1 and V2.
Let G1 = (V1, E1).
Let G2 = (V2, E2).
M1 = Maybe-MST(G1)
M2 = Maybe-MST(G2)
Let e be the minimum weight edge in E3.
return M1 + e + M2
```

Basically, Maybe-MST divides the graph into two connected halves of the same (or almost the same) size, and runs itself recursively on the halves. When you get down to the small case of a graph that's just 1 or 2 nodes, you return the entire graph as the MST of that piece.

Either prove that this algorithm works to compute an MST, and give runtime bounds, or show that Maybe-MST doesn't actually work.