**I collaborated with:**

**Problem**
In many situations, you might find yourself dealing with a dynamically changing data structure. One such simple example is that of dynamically changing edge weights in a graph. This problem asks you to consider how you would maintain a minimum-cost spanning tree in such an environment.
Suppose you are given a graph $G$ with weighted edges and a minimum spanning tree $T$ of $G$.

**(a)** Describe and analyze an algorithm to update the minimum spanning tree when the weight of a single edge $e$ is decreased.

**(b)** Describe and analyze an algorithm to update the minimum spanning tree when the weight of a single edge $e$ is increased.

In both cases, the input to your algorithm is the edge $e$ and its new weight; your algorithms should modify $T$ so that it is still a minimum spanning tree. *Hint: consider $e \in T$ and $e \notin T$ separately.*
**Solution**

A This algorithm will perform a BFS on the tree until we find one of two things. The edge $e = \{u, v\} \in T$ or $e \notin T$. If $e$ is in the graph then we just update its weight and we are done. Otherwise, we add $e$ to $T$, which adds a cycle, perform another BFS from $v$, marking every node we encounter. If we encountered a node $b$ twice, then we remove the edge of highest weight in this cycle. We now have our updated MCST.
Runtime: $1^{st}$ BFS $O(|V|+|E|)$ * $2^{nd}$ BFS $O(|V|+|E|)$ * walk through cycle $O(|V|+|E|) = O(3(|V|+|E|)) = O(|V| + |E|)$

*Proof.* Performing our first BFS to find the node of destination will tell us where to change $T$. Now we have two cases, $e \in T$ and $e \notin T$:
Case $e \in T$: If this is the case then $e$ will be updated to have a lower weight and $T$ mantains its MCST property.
Case $e \notin T$: In this case, let $e$ connect vertices $u$ to $v$ in $G$. By performing a BFS and finding $v$ we find the potential spot for $e$ and add it to our tree. This forms a cycle and we want to get rid of the edge that costs the most in this cycle. Removing it will end the cycle and reduce the cost of $T$ so that it keeps its MCST property. □

B This algorithm will perform a BFS much like in part $A$ to find one of two things. The edge $e = \{u, v\} \in T$ or if $e \notin T$. If $e \notin T$ then we can increase its weight and be done. If $e \in T$, then we remove it. This disconnects $T$ into two parts, $V$ and $V'$. We perform a BFS from $v$ and mark all of the vertices we visit. Note that these vertices are in $V'$ and not in $V$. We now take a look at our list of edges, $G(E)$. These edges either connect two marked nodes, two unmarked nodes, or one marked node and an unmarked node. We find the lowest cost edge that connects a marked node and an unmarked node and add it to $T$. $T$ is now a MCST.
Runtime: $1^{st}$ BFS $O(|V| + |E|)$ * $2^{nd}$ BFS $O(|V| + |E|)$ * find min edge $O(|E|)$ * add lowest cost edge to $T$ $O(|V| + |E|) = O(3|V| + 4|E|) = O(|V| + |E|)$

*Proof.* Performing the first BFS on $T$ will tell us where node $e$ has the potential to be. We have two cases $e \in T$ and $e \notin T$:
Case $e \notin T$: If this is the case then $e$ was not originally in our MCST, so increasing the cost will not influence $T$.
Case $e \in T$: In this case, we find $e$ by looking at the vertices it connects. We update its weight and remove it which disconnects $T$, as there are no cycles in trees. We now have two sets of vertices, $V$ and $V'$. By marking the vertices of $V'$, we can detect the edges that are in this cut. A set of cut edges are those edges that if removed from $G$, they disconnect the graph into two distinct sets of vertices. By finding the smallest cut edge, we are able to connect $T$ with the smallest edge possible. This maintains $T$'s MCST property. □