Let $G = (V, E, w)$ be an undirected and weighted graph, where $w(e) > 0$ is an injective weight function (i.e. all edge weights are unique). You are given that a particular edge $e \in E$ has its weight modified, and are asked to find the new minimum spanning tree. Of course, you *could* just recompute the minimum spanning tree but that's boring! We want to do something more clever. In all four cases, assume you are given the actual minimum spanning tree T in advance.

(a) Suppose that the edge $e \in T$ has its weight decreased. Explain why T is still the minimum spanning tree.

Hint. *Is this obvious?*

(b) Suppose that the edge $e \notin T$ has its weight increased. Explain why T is still the minimum spanning tree.

Note. *The argument is obvious because all of the edge weights are unique... a more subtle argument needs to be made if the edge weights are not necessarily unique.*

(c) Suppose that the edge $e \in T$ has its weight increased. Describe an $O(m + n)$ algorithm to compute the new minimum spanning tree.

(d) Suppose that the edge $e \notin T$ has its weight decreased. Describe an $O(m)$ algorithm to compute the new minimum spanning tree.

Hint. *Either e belongs in the new MST or e doesn't...*

Solution.

(a) T contains all of the edges with the minimal weights in the tree, which are notably distinct. For any $e \in T$, we know that the value must be in the lowest $n - 1$ edges. Decreasing the weight will not change this fact. So, e will remain in the minimum spanning tree.

(b) Similar to part (a), T contains the edges with the lowest $n - 1$ lowest weights. For an edge $e \notin T$, we have no way on getting it into T by increasing the weight.

(c) Consider removing e from T, breaking it into two smaller trees T_1 and T_2. For each edge e' in the graph that connects T_1 and T_2, check to see if e' is smaller than the original edge we removed. If none are found, then the original edge is still part of the minimal spanning tree. The initial split of the tree and the final update will take $O(n)$ operations and checking each edge would take $O(m)$, so the final time complexity is $O(n + m)$.

(d) We can begin by adding in the given edge to the tree. This will create a cycle, so starting from this edge, we follow the cycle and identify the largest side(s). From there, we simply remove the largest edge. The most time-consuming step is actually checking all edges in the cycle, which is simply $O(m)$.