1. Suppose we are given both an undirected graph G with weighted edges and a minimum spanning tree T of G. In all 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. Of course, we could just recompute the minimum spanning tree from scratch in $O(|E| + |V| \log |V|)$ time, but you can do better.

   (a) Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge $e \in T$ is decreased.

   Solution: In this case, the given MST T is still a MST. While this may seem obvious, we argue it carefully here. Let T be the MST, and W be the weight of T. Let $e = (u, v)$, and let the decrease in weight be d, so that the new cost of T becomes $W - d$. Let T_u and T_v be the subtrees obtained by removing e. Now, if T doesn't remain an MST, then clearly any new MST T' must contain e, for if it did not, then since $w(T') < W - d < W = w(T)$, this contradicts the fact that T was an MST with the original weights. But if T' contains e, then the only way it can have cost lower than $W - d$ is by either connecting the nodes of T_u with less weight than $w(T_u)$, or by connecting the nodes of T_v with less weight than $w(T_v)$. But T_u and T_v are both MSTs for their node sets, otherwise T would not have been minimal to begin with. Thus, T' cannot have less weight than $W - d$. ■

   (b) Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge $e \notin T$ is increased.

   Solution: In this case, the given MST T is still a MST. To see this, consider a run of Kruskal's algorithm which produced T. All of the same decisions would be made when e has a higher weight, and so the same tree will be produced. (This assumes the edge weights are unique. A more subtle argument is needed if the edge weights are not unique.) ■

   (c) Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge $e \in T$ is increased.

   Solution: Let $e = (u, v)$ and let T_u and T_v be the subtrees obtained by removing e. By doing BFS (ignoring edge weights) from u and from v , we can determine which vertices are in T_u and which are in T_v in time $O(|V| + |E|)$. (Actually, we can do this in $O(|V|)$, since we only need to consider edges of the tree T, but the next step will take $O(|E|)$ anyway.) Assume we have marked each node with its membership. Now examine each edge, and keep the minimum weight edge e' with one endpoint in T_u and the other in T_v. This can be done in $O(|E|)$ time. The total time is thus $O(|V| + |E|)$. (Okay, we could perhaps speed this up by examining only those edges adjacent to vertices of T_u, thus ignoring edges with both endpoints in T_v.) Can you argue why the resulting tree is an MST? ■

   (d) Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge $e \notin T$ is decreased.

Solution: Let $e = (u, v)$. Add e to T which will create a unique cycle, which we can find doing BFS in $T \cup \{e\}$ starting from u and ignoring weights. This takes $O(|T|) = O(|V|)$ time. Now remove the maximum weight edge on that cycle ($O(|V|)$). Can you argue why the resulting tree is an MST? ■

2. Let $G = (V, E)$ be an undirected graph where each edge has a weight from the set $\{1, 10, 25\}$. Describe a *linear-time* algorithm to find an MST of G.

   There are two possible solutions, one using a modified Prim's algorithm, and one using meta-graphs of connected components.

   Solution (Modified Prim's): The idea behind this is that Prim's grows the spanning tree by maintaining a tree and adding one vertex to the tree at a time. This works in linear time for us, because the choice of edge is made efficient without the need to sort. First construct three bags containing 1-weight edges, 10-weight edges, and 25-weight edges respectively . It takes $O(E)$ to build the lists, The find operation is $O(1)$, the decrease key operation is $O(1)$. Thus, the algorithm runs in $\boldsymbol{O(V + E)}$ ***time***. ■

   Solution (Connected Components): Find the connected components in the graph with the smallest weight set of edges. Shrink them and find the connected components again with edges of next weight, shrink them and so on. This algorithm also runs in $\boldsymbol{O(V + E)}$ ***time*** since we only have 3 distinct edge weights. ■