Question 1.:

Fix MST

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 $O(|V||E| \log|V|)$.

Answer:

Algorithm 1: Fix MST for missing edge $e \in T$

```
1 input: $e = (u, v) \in T$ is the edge that goes missing.
2 output: $r(e)$ is the edge to add to $T$.
  Once $e$ goes missing, $T \setminus e$ becomes two connected components $C_u$ and $C_v$.
3 Apply DPS at $u$ to find all vertices in $C_u$, denoted $V_u$.
4 Apply DPS at $v$ to find all vertices in $C_v$, denoted $V_v$.
  The whole DPS searching process takes $O(|V|)$ time.
5 let $B = \{(l, r) \mid l \in V_u, r \in V_v\}$.
  Cost at most $O(|E|)$ to collect a subset of edges.
6 let $b = \text{argmin}_{(l,r)\in B}\, d(l, r)$
  Cost at most $O(|E|)$ to find a minimum among a set with cardinality $< |E|$
7 return $b$.
  The time complexity of the algorithm is $O(|E|) = O(|V| + 2|E|)$.
```

Apply the Algorithm 1 to each edge in T to obtain $r(e)$ for each $e \in T$. The total time cost is $O(|V||E|)$ since there're totally $|V| - 1$ number of edges in T.

Better Prim

Question 2.: 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 deleteMin operation still takes O(log |V |) time, but the decreaseKey operation takes only constant O(1) time? What would be the runtime of Prim’s using such an improved heap? Justify your answer.

Answer: A careful examination of the Prim algorithm shows that the deleteMin is applied once for each vertex, and the decreaseKey is applied once for each edge. The total time complexity of the algorithm is $O(|V||\text{deleteMin}| + |E||\text{decreaseKey}|)$, which is $O(|V| \log(|V|) + |E|)$, which is a improved result.

MST Divide and Conquer

Answer: **The algorithm is wrong**: consider the following graph with each corner as vertice, suppose the algorithm partition the vertice into two set, the above vertices and the below vertices, then the returned MST would necessarily contain the edge of weight = 99, which is absure.

```
┌─────────────┐
│      1      │
│             │
│1           1│
│             │
│     99      │
└─────────────┘
```