Table of Contents

Question 1.

The algorithm to create a graph of the best bottleneck for G is simply making a maximum spanning tree of G. We will do this using Kruskal's algorithm using the disjoint-set data structure.

Algorithm
Max_Spanning_Tree (`graph`: undirected graph $= (V,\ E)$)

```
1: let result = {}
2: let sorted-edges = E sorted by bandwidth decreasing
3: let sorted-edges-cluster = []
4:
5: for edge in sorted-edges do
6:     sorted-edges-cluster.append(MAKE-CLUSTER(edge))
7:
8: for edge = (u, v) in sorted-edges-cluster do
9:     if u.cluster ≠ v.cluster then
10:        result = result ∪ edge
11:        UNION(u.cluster, v.cluster)
12:
13: return result
```

The algorithm `Max_Spanning_Tree()` is a greedy algorithm with the choice of longest edge length that does not create a cycle.

`Max_Spanning_Tree()` gives a feasible solution since we output a spanning tree with no cycles.

The complexity of `Max_Spanning_Tree()` is just $\mathcal{O}(n \log n)$ since we sort which is $\mathcal{O}(n \log n)$. The 2 `for` loops all loop $\mathcal{O}(n)$ times and have $\mathcal{O}(1)$ operations. Every other line is $\mathcal{O}(1)$, so the final time compelxity is $\mathcal{O}(n \log n)$

Theorem: All spanning trees have the same number of edges

Proof:

Let $G = (V,\ E)$ be an arbitrary graph.

We will show that any spanning tree of a connected graph G must have $|V| - 1$ edges.

Base Case:

For a graph of size $|V| = 1$, the a spanning tree would have no edges.

Induction Hypothesis:

Suppose for a graph $G = (V,\ E)$ with $|V| = n$, we can create a spanning tree T with $n - 1$ edges.

Induction Step:

WTS: A spanning tree T' for a graph G' with $|V'| = n + 1$ has n edges.

The spanning tree from [IH] has all nodes except for 1. Since by assumption, the graph is connected, then there is an edge in G' to the new vertex. Then use T from [IH] and connect one edge in G' to the new node to create T'.

We now have a spanning tree T' for G' with n edges. ■

Theorem: Greedy spanning tree T given by `Max_Spanning_Tree()` is optimal

Proof (Greedy stays ahead):

Let $G = \{g_1,\ g_2,\ \cdots,\ g_{n-1},\ g_n\}$ be the spanning tree given by `Max_Spanning_Tree()`, ordered by add order

Let $O = \{o_1,\ o_2,\ \cdots,\ o_{n-1},\ o_n\}$ be a maximum spanning tree that's most similar to G, ordered by add order

We know they have to be the same size by the theorem above.

WTS: We can change O to be more similar to G while keeping optimality.

$$\begin{aligned} \text{Suppose} \quad & g_1 = s_1 \\ & g_2 = o_2 \\ & \vdots \\ & g_{k-2} = o_{k-2} \\ & g_{k-1} = o_{k-1} \\ \text{but} \quad & g_k \neq o_k \qquad \text{for some } k \in [0,\ \cdots,\ n] \end{aligned}$$

since $g_k \neq o_k$, then $b_{o_k} \leq b_{g_k}$ because if $b_{o_k} > b_{g_k}$, then the greedy algorithm would've chosen that

edge.

Let $g_k = (u,\ v)$
Now replace o_k with g_k for O'

We know that $b(P)$ with P going to the verticies u and v in O still has the best bottleneck since `Max_Spanning_Tree()` selected the biggest bandwidth edge going to them.

Repeat until $O = G$
$\therefore O$ becomes more similar to G with every path still having the best bottleneck. ■

Question 2.

To prove that the degrees of a graph are valid, we just need to find one graph that supports those degrees. The following algorithm creates a graph with the most clustering possible, as each node tries to connect to the nodes that come after it in descending order of degrees.

Algorithm
Validate_Degrees (`graph`: undirected graph $= (V,\ E)$)

```
1: let degrees = V sorted in descending order
2: if degrees.[0] > |E| then
3:     return false
4:
5: for degree in degrees do
6:     subtract 1 from the next degree elements in degrees
7:     if any element went below 0 then
8:         return false
9:     resort degrees
10:
11: return true
```

Complexity: The complexity of `Validate_Degrees()` is $\mathcal{O}(n^2 \log n)$.

Firstly we sort the degrees which is $\mathcal{O}(n \log n)$ and then we go over every element, and visit its right neighbours depending on its degree, and finally resorting the array. Since we know that the max degree is bounded by n due to the early return on line 2, the worst case is if its a complete graph, in which case every element will visit every other right element, giving $\mathcal{O}(n+n \log n)$ done n times which is $\mathcal{O}(n^2 \log n)$.

Proof (contradiction):

It is clear that a graph created by `Validate_Degrees()` is valid, so if `Validate_Degrees()` returns `true` then the input is valid. Let's assume that `Validate_Degrees()` returned `false` on a valid graph.

Assumption: Let G be a valid graph and `Validate_Degrees()` be false.

Let `e` be the node with a degree that caused `Validate_Degrees()` to fail (and therefore have a remaining degree greater than 0). That means that when attempting to connect `e` to its right neighbours to complete its degree, we attempted to connect it to a node with a remaining degree of 0. Therefore, there are not enough remaining nodes to for `e` to connect to so that it has its supposed degree. This is because we have already connected `e` to all of its left neighbours according to our algorithm, and our right neighbours are in descending order, so if we hit a neighbour with no degrees remaining, then none of the remaining nodes have any remaining degrees to connect to us, so the G cannot be valid. This goes against our assumption so `Validate_Degrees()` also returns `false` on incorrect graphs.

$\therefore$ `Validate_Degrees()` always gives the correct answer.