1. (a) Red street in the city Shampoo-Banana can be modeled as a straight line starting at 0. The street has n houses at locations $x_1, x_2, \ldots, x_n$ on the line. The local cable company wants to install some new fiber optic equipment at several locations such that every house is within distance r from one of the equipment locations. The city has granted permits to install the equipment, but only at some m locations on the street given y locations $y_1, y_2, \ldots, y_m$. For simplicity assume that all the x and y values are distinct. You can also assume that $x_1 < x_2 < \ldots < x_n$ and that $y_1 < y_2 < \ldots < y_m$.

   - Describe a greedy algorithm that finds the minimum number of equipment locations that the cable company can build to satisfy the desired constraint that every house is within distance r from one of them. Your algorithm has to detect if a feasible solution does not exist. Prove the correctness of the algorithm. One way to do this by arguing that there is an optimum solution that agrees with the first choice of your greedy algorithm.
   - **Not to submit:** The cable company has realized subsequently that not all locations are equal in terms of the cost of installing equipment. Assume that c_j is the cost at location y_j. Describe a dynamic programming algorithm that minimizes the total cost of installing equipment under the same constraint as before. Do you see why a greedy algorithm may not work for this cost version?

Solution: The greedy algorithm's strategy is to repeatedly choose the rightmost y_j that covers the leftmost uncovered x_i until all the houses are covered. The algorithm will return the set of fiber optic cable equipment locations.

Greedy Algorithm:
- $S \leftarrow \emptyset$, $i \leftarrow 1$, $j \leftarrow 1$
- While $i \leq n$
  - If $j > m$ or $x_i < y_j - D$, return IMPOSSIBLE
  - While $j \leq m$ and $x_i \geq y_j - D$, increment j
  - Add y_{j-1} to S
  - While $i \leq n$ and $x_i \leq y_{j-1} + D$, increment i
- Return S

The preceding algorithm runs in $\boldsymbol{O(n+m)}$ time. Note that we are assuming here that the locations are given in a sorted order. Otherwise we would need to spend $O(n \log n + m \log m)$ time to sort them first. In order to prove the correctness via induction it is convenient to rephrase the greedy algorithm in a recursive fashion described below. Here we assume that the array $X[1..n]$ stores the locations $x_1, \ldots, x_n$ and $Y[1..m]$ stores the locations $y_1, \ldots, y_m$. The algorithm below will be called in the main program with $a = 1, b = n$ and $c = 1, d = m$.

RecursiveGreedy $(X[a..b], Y[c..d])$:
- If $a > b$ return $\emptyset$.
- If none of $y_c, ..., y_d$ covers x_a, return IMPOSSIBLE.
- Let j be the largest index in $c, c+1, \ldots, d$ such that y_j covers x_a.
- Let i be the largest index in $a, a+1, \ldots, b$ such that x_i is not covered by y_j
- $S' =$ **RecursiveGreedy**$(X[i..b], Y[j+1..d])$
- Output $S' \cup \{y_j\}$.

The correctness of the algorithm can then be proved by induction.

Proof of correctness:

We shall first prove that there is an optimum solution that contains the first choice of y_j that the greedy algorithm would pick. Suppose *OPT* is any optimum solution. The first location Greedy would choose to include, g_1, is the rightmost y_j that covers x_1, which is the leftmost x_i and therefore the one Greedy would seek to cover first by definition. We'll show that *OPT* could also have been constructed using g_1 without any loss of optimum quality, so if there are any optimum solutions to the problem, g_1 is part of at least one such solution.

Let o_1 be the leftmost position that *OPT* chose. If $o_1 = g_1$, the proof is complete. Otherwise we consider other possible positions for o_1. Suppose o_1 is to the right of g_1; then o_1 does not cover x_1, because g_1 was the rightmost position that could; also, because there is no other element in *OPT* to the left of o_1 by definition, nothing in *OPT* can cover x_1, so *OPT* is not a solution; this is a contradiction, so o_1 is not to the right of g_1. So suppose o_1 is to the left of g_1. Then we know that o_1 can't be so far to the left that it fails to cover x_1 at all, because then o_1 could be removed from *OPT* to form a solution *OPT*$'$ that is smaller in size and thus better; this would make *OPT* not optimum, a contradiction. So o_1 must cover x_1, and o_1 is to the left of g_1. We also know that if o_1 covers any houses to the right of x_1, then g_1 also covers them, since g_1 at least covers x_1 like o_1 does, and g_1 is situated further to the right than o_1. This means *OPT* could replace o_1 with g_1 without losing any coverage or losing optimality.

We can then use induction to prove that the greedy algorithm in its entirety provides an optimal solution to the problem. It is convenient here to use the recursive version of the algorithm. We omit the formal inductive argument. ■

(b) **Solution:** Why Greedy might fail: Let $D = 1, x_1 = 1, y_1 = 1, y_2 = 2, c_1 = 1, c_2 = 2$. The greedy algorithm will pick the equipment location at y_2 as it is the rightmost equipment location that covers the only (and therefore leftmost uncovered) house at x_1, thus the greedy solution has cost 2. However, we can cover the only house with y_1 which only has cost 1. Thus greedy does not work for all inputs.

In the following we will assume that the x_i's and $y_i's$ are sorted by taking $O(n \log n + m \log m)$ time. Let *Next*(j) be the smallest index i such that $x_i > D + y_j$ or $n + 1$ if no such index exists. We can preprocess this in $O(m \log n)$ time via binary search (even though linear searching would be sufficient for the same overall runtime) for each $j \in [1, m]$ and store it in an array $N[1..m]$. This allows us to find the index of the rightmost house more than D distance away from y_j if one exists in constant time. Let *MCFS*(i, j) be the minimum total cost of building a subset of the rightmost $(m - j + 1)$ cable locations such that the rightmost $(n - i + 1)$ houses are within D distance of an equipment location. The recurrence below satisfies this definition.

$$MCFS(i,j) = \begin{cases} 0 & \text{if } i > n \\ \infty & \text{if } j > m \text{ or } y_j - x_i > D \\ MCFS(i, j+1) & \text{if } x_i - y_j > D \\ \min\{c_j + MCFS(N[j], j+1), MCFS(i, j+1)\} & \text{otherwise} \end{cases}$$

Thus the call *MCFS*$(1, 1)$ will return the minimum cost to cover all n houses with cable equipment locations chosen such that all houses are within D distance of an equipment location. Since this algorithm has $O(mn)$ distinct calls and its arguments are integers, we will memoize using a 2d array $MCFS[1..n+1][1..m+1]$. The total work is constant for all cases due to the preprocessing of *Next*, thus it runs in $O(nm)$ time. Overall, including the sorting and time spent constructing $N[1..m]$, it takes $\boldsymbol{O(n \log n + m \log m + m \log n + mn)}$ time. ■

Rubric: Out of 10 points, we would allocate:
- 5 points for the greedy algorithm (standard DP algorithm rubric scaled)
- 5 points for proof. 4 points for the main claim regarding the the existence of an optimum solution containing the first location opened by Greedy. 1 point for discussing inductive proof based on the main claim.

2. Let $G = (V, E)$ be an edge-weighted undirected graph. We are interested in computing a minimum spanning tree T of G to find a cheapest subgraph that ensures connectivity. However, some of the nodes in G are unreliable and may fail. If a node fails it can disconnect the tree T unless it is a leaf. Thus, you want to find a cheapest spanning tree in G in which all the unreliable nodes (which is a given subset $U \subset V$) are leaves. Describe an efficient for this problem. Note that your algorithm should also check wither a feasible spanning tree satisfying the given constraint exists in G.

Solution: We first note that if T is a spanning tree of G such that all unreliable nodes are leaves in T, then removing the nodes in U from T will leave a tree T' on $V \setminus U$, since each node of U is a leaf in T. Moreover, if $u \in U$ and (u, v) is the unique edge in T incident to u then $v \notin U$, as otherwise, v would have to have degree 2 in T (recall that U is a proper subset of V, so T must contain at least one node not in U; if $v \in U$, then v must be adjacent to both u and one such node $u' \notin U$). Therefore edges connecting two U-nodes must not appear in T.

From the above observations we derive the following algorithm.

```
CalculateReliableMST(G, U):
  Remove from G all edges (u, v) where both u, v ∈ U
  Let G′ be the graph obtained by removing U from G
  If G′ is not connected, output that no feasible solution exists
  Compute an MST T in G′
  For each u ∈ U:
      If u has no edges incident to it in G, output that no feasible solution exists
      Else, let (u, v) be the lowest-weight edge incident to u in G. Add (u, v) to T.
  Output T
```

As we loop through all edges of G once during the first step and the for loop iterates through each vertex and edge in G at most once, the algorithm can be implemented in $O(m + n + A(m, n))$ time where $A(m, n)$ is the time required to find an MST in a graph with m edges and n nodes. Any MST algorithm could be used; in particular, we have an $O(m + n \log n)$ algorithm for MST computation by running Prim's algorithm using a Fibonacci heap, so we get an overall running time of $O(m + n \log n)$ where m is the number of edges in G and n is the number of nodes.

Note: It is possible to modify Prim's and/or Kruskal's algorithm to obtain a correct solution to this problem. While that is a reasonable approach, it is often better to step back and understand the problem structure first before thinking of a specific algorithm. ■

Rubric: 10 points:

- 3 points for correctly removing all nodes from U and calculating an initial MST
- 2 points for noting that all edges (u, v), with $u, v \in U$, cannot be included in the resulting MST
- 2 points for properly adding the lowest-weight edge incident to each $u \in U$ to the tree
- 1 point for noting that no feasible solution exists if a node $u \in U$ has no edges incident to it in G after removing all edges (u, v) from G with $u, v \in U$
- 2 points for correct runtime analysis

3. Consider the language $L_{\text{OH}} = \{\langle M\rangle \mid M \text{ halts on at least one input string}\}$. Note that for $\langle M\rangle \in L_{\text{OH}}$, it is not necessary for M to *accept* any string; it is sufficient for it to *halt* on (and possibly rejects) some string. Prove that L_{OH} is undecidable.

Solution: For the sake of argument, suppose there is an algorithm DecideOH that correctly decides the language L_{OH}. Then we can solve the halting problem as follows:

```
DecideHalt(⟨M,w⟩):
  Encode the following Turing machine M′:
      M′(x):
        run M on input w
        return True or False // Does not really matter what we return
  if DecideOH(⟨M′⟩)
      return True
  else
      return False
```

We prove this reduction correct as follows:

$\Longrightarrow$ Suppose M halts on input w.
Then M' halts on *every* input string x.
Thus, M' halts on at least one input string.
So DecideOH accepts the encoding $\langle M'\rangle$.
So DecideHalt correctly accepts the encoding $\langle M, w\rangle$.

$\Longleftarrow$ Suppose M does not halt on input w.
Then M' diverges on *every* input string x.
That is, M' does not halt on any input string.
So DecideOH rejects the encoding $\langle M'\rangle$.
So DecideHalt correctly rejects the encoding $\langle M, w\rangle$.

In both cases, DecideHalt is correct. But that's impossible, because Halt is undecidable. We conclude that the algorithm DecideOH does not exist. ∎

Rubric: 10 points:
- 4 points for correct reduction.
- 3 points for "if" proof.
- 3 points for "only if" proof.