Name:

Student ID #:

- You are expected to work on this assignment on your own
- Use pseudocode, Python-like or English to describe your algorithms. Absolutely no C++/C/Java
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your algorithms
- Homework has to be submitted electronically on Gradescope by the deadline. No late assignments will be accepted
Problem 1. (10 points)

Let $G = (V, E)$ be a weighted undirected graph. We define the bottleneck of a path $p$ in $G$ as the minimum weight of any edge on $p$. We define the maximum bottleneck of any $s, t$-path as the maximum, over all paths $p$ from $s$ to $t$, of the bottleneck of $p$.

Prove or disprove: given any connected, undirected, edge-weighted graph, algorithm \texttt{Bottleneck} below produces a tree $T$ such that the bottleneck of the path $p$ from $s$ to $t$ in $T$ is the maximum bottleneck of any $s, t$-path in the original graph.

\textbf{Algorithm} \texttt{Bottleneck} ($G(V, E) : graph$)
\begin{itemize}
  \item \texttt{sort} the edges $e_1, e_2, \ldots, e_m$ in order of decreasing cost
  \item $T \leftarrow \emptyset$
  \item \textbf{for} $i \leftarrow 1, 2, \ldots, m$ \textbf{do}
    \begin{itemize}
      \item Add $e_i$ to $T$ if doing so does not create a cycle in $T$
    \end{itemize}
  \item return $T$
\end{itemize}

Answer:
Problem 2. (10 points)

Consider a variation of the UNION-FIND data structure, with the union-by-rank heuristic but without path compression. That is, we implement `MAKE-SET` and `UNION` as usual, but we do not reset the parent pointers in `FIND-SET`. Show that there is some sequence of \( n \) calls to `MAKE-SET`, some number (at most \( n \)) of calls to `UNION`, and \( m \) calls to `FIND-SET` that require \( \Omega(m \log n) \) time from this suboptimal implementation.

Answer:
Problem 3. (10 points)

In the United States, coins are minted with denominations of 1, 5, 10, 25, and 50 cents. Now consider a country whose coins are minted with denominations of \( \{d_1, \ldots, d_k\} \) units. We want to devise an algorithm that will enable us to make change of \( n \) units using the minimum number of coins.

1. The greedy algorithm for making change repeatedly uses the biggest coin smaller than the amount to be changed until it is zero. Provide a greedy algorithm for making change of \( n \) units using US denominations. Prove its correctness and analyze its time complexity.

2. Show that the greedy algorithm does not always give the minimum number of coins in a country whose denominations are \( \{1, 6, 10\} \).

3. Give an efficient dynamic programming algorithm that correctly determines the minimum number of coins needed to make change of \( n \) units using denominations \( \{d_1, \ldots, d_k\} \). Analyze its running time.

Answer: