Spring 2023 Q-exam — CSCE 750 (Algorithms) — Solutions

1. **(Solving a Recurrence)** Let $T(n)$ be any positive-valued function defined for all integers $n \geq 0$ by the following recurrence, which holds for all sufficiently large n:

$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n \text{ is even,} \\ T(n-1) + n & \text{if } n \text{ is odd.} \end{cases}$$

   Find tight asymptotic bounds on $T(n)$, that is, find a function $f(n)$, as simple as possible, such that $T(n) = \Theta(f(n))$ as $n \to \infty$. Justify your answer using the substitution method.

   Answer: $T(n) = \Theta(n \lg n)$. Key fact: if n is odd, then $n-1$ is even. So

$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n \text{ is even,} \\ 2T((n-1)/2) + 2n - 1 & \text{if } n \text{ is odd.} \end{cases}$$

   In both cases, $T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n)$, which matches the standard Mergesort recurrence.

2. **(Longest Welded Rod)** You are supplied with a sequence $r_1, \ldots, r_n$ of $n > 0$ rods of various positive integer lengths (in inches, say). Your job is to weld (i.e., fused end-to-end) rods to form the longest possible single welded rod. There are two constraints, however:

   (a) The order of the rods cannot be swapped. That is, if $i < j$ and r_i and r_j both appear in the welded rod, then r_i must be somewhere to the left of r_j.

   (b) It may or may not be possible to weld two given rods together.

   Design an algorithm for doing this. Your algorithm takes as input: (1) an array $L[1 \ldots n]$ of positive integers where $L[i]$ is the length (in inches) of rod r_i; (2) an array $W[1 \ldots n, 1 \ldots, n]$ of Booleans, where $W[i, j] =$ TRUE iff it is possible to weld r_i directly with r_j. Your algorithm should return the length of the longest possible welded rod. (You are not required to determine which rods make up the optimal rod.) **Explain** your algorithm well enough so that an intelligent reader (who has taken CSCE 750) with no specialized knowledge can implement it.

   Your algorithm must run in time $O(n^2)$. As usual, you may assume that all arithmetic and comparison operations on integers take $O(1)$ time each.

   Answer:

```
LongestRod(L, W)
    Allocate an array R[1...n] of integers
    // R[i] is to be the longest possible length of a welded rod ending with r_i.
    for i := 1 to n do
        R[i] := L[i] // Just know about r_i by itself, initially
        // Now try to weld r_i to a previous rod
        foreach j such that 1 ≤ j < i and W[j,i] do
            if R[j] + L[i] > R[i] then
                R[i] := R[j] + L[i] // Get a longer rod if r_i is welded to r_j
    // R-table complete. Now find the optimal length (the max value in R).
```

```
m := 0
for i := 1 to n do
    if R[i] > m then
        m := R[i]
return m
```

3. **(Shortest Path)** Dijkstra's algorithm (famously) may fail on a digraph that has negative edge weights. Let $G := (V, E, w)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$ that may have *at most one* edge with negative weight. **Design** an algorithm that takes G and two vertices $s, t \in V$ as input and returns the minimum weight of an $s \to t$ path. **Describe** your algorithm with enough precision so that an intelligent reader (who has taken CSCE 750) with no specialized knowledge can implement it.

   Your algorithm must run in time $O((n + m) \lg m)$, where $n = |V|$ and $m = |E|$. As usual, you may assume that G is represented by adjacency lists, and all arithmetic and comparison operations on weights take $O(1)$ time each. You may also assume (as usual) that G has no negative-weight cycles. For full credit, **explain briefly** why your algorithm is correct. [Note: The Bellman-Ford algorithm computes shortest paths when weights can be negative, but you cannot simply invoke it because it takes too long to run.]

   Answer: High-level description:

   (a) Look to see if G has a negative edge weight. (This takes time $O(n+m)$ to search through the edges of G.)

   (b) If G has no negative-weight edge, then run Dijkstra's algorithm with source s and return $t.d$.

   (c) Otherwise, let $(u, v) \in E$ be such that $w(u, v) < 0$.

      i. Remove (u, v) from E. Let G' be the resulting graph.

      ii. Run Dijkstra's algorithm on G' with source s, and set $d_1 := t.d$ and $d_2 := u.d$.

      iii. Run Dijkstra's algorithm on G' again, this time with source v, and set $d_3 := t.d$.

      iv. Return $\min(d_1, d_2 + w(u, v) + d_3)$.

   Explanation: Part (b) works because Dijkstra's algo works when there are no negative edge weights. Otherwise, let $e = (u, v)$ be the only negative-weight edge. A shortest $s \to t$ path (if one exists) either uses e once or avoids it. If e is not used, then Dijkstra on G' gives the shortest $s \to t$ distance. Otherwise, the shortest path consists of a shortest $s \to u$ path, followed by e, followed by a shortest $v \to t$ path. We find which of these two possibilities gives the shorter distance and return it.

   The algorithm runs Dijkstra's algorithm at most twice, plus $O(n+m)$ extra work, so the total time is asymptotically the same as for Dijkstra, which is $O((n + m) \lg m)$ (without bothering to use a Fibonacci heap).