CSCI 270 Homework #1 Solutions

1. Let T be a binary tree with n nodes. Each node will either have 2 children, 1 child, or it will be a leaf node. T has L leaf nodes, P nodes with 2 children, and $(n - L - P)$ nodes with 1 child. Use induction on n to prove that $L = P + 1$.
   Solution:
   Base Case: $n = 1$, then $L = 1$ and $P = 0$, which checks out.
   Inductive Hypothesis: The claim is true for all $n < N$.

   Consider a graph with N nodes. This graph must have at least one leaf node x.

   Remove x and its incoming edge from the graph. What we have left is a binary tree with $N-1$ nodes, which falls under the inductive hypothesis. Thus the reduced graph has P nodes with 2 children and $P+1$ leaf nodes.

   Now add x back in. There are two cases, depending on whether x's parent has 2 children or 1 child. If x is an only child, then the removal of x did not change the size of P or L: we removed one leaf node (x), and turned x's parent from a node with 1 child into a leaf node. Therefore the original graph has P nodes with 2 children and $P + 1$ leaf nodes, proving the claim. On the other hand, if x has a sibling, then the removal of x reduced L by one (x is no longer there) and reduced P by one (x's parent now only has 1 child). Therefore the original graph has $P + 1$ nodes with 2 children and $P + 2$ leaf nodes, proving the claim.

   By induction, the claim is true.

2. Suppose $f(n) = O(s(n))$, and $g(n) = O(r(n))$. Prove or disprove (by giving a counterexample) the following claims:

   (a) $f(n) - g(n) = O(f(n) - r(n))$
   Solution:
   False. Let $f(n) = n^2 + n, g(n) = n, r(n) = n^2$. $f(n) - g(n) = n^2$, and $f(n) - r(n) = n$. The claim is therefore false.

   (b) $\frac{f(n)}{g(n)} = O(\frac{s(n)}{g(n)})$
   Solution:
   True. We know $f(n) \leq cs(n), \forall n > n_0$. Therefore, $\frac{f(n)}{g(n)} \leq \frac{cs(n)}{g(n)} = c\frac{s(n)}{g(n)}, \forall n > n_0$, so $\frac{f(n)}{g(n)} = O(\frac{s(n)}{g(n)})$.

   (c) if $f(n) = O(g(n))$, then $f(n) + g(n) = O(s(n))$
   Solution:
   False. Let $f(n) = s(n) = n, g(n) = n^2$. Then $f(n) + g(n) = n^2 + n$. The claim is therefore false.

   (d) if $s(n) = O(r(n))$, then $f(n) = O(g(n))$
   Solution:
   False. Let $f(n) = s(n) = r(n) = n^2, g(n) = n$. The claim is therefore false.

3. Order the following functions from smallest asymptotic running time to greatest. Additionally, identify the two functions i and j where $f_i(n) = \theta(f_j(n))$. Justify your answers. **In this class, when I ask for a justification, it is sufficient to give an intuitive explanation.**

(a) $f_a(n) = \frac{n^2}{\log n}$

(b) $f_b(n) = n^2$

(c) $f_c(n) = \log^2 n$

(d) $f_d(n) = \sum_{i=1}^{n} i$

(e) $f_e(n) = \log_{1.5} n^2$

(f) $f_f(n) = 2^{\log n}$

(g) $f_g(n) = n^{\frac{8}{9}}$

(h) $f_h(n) = 1.001^n$

Solution: $f_e < f_c < f_g < f_f < f_a < f_d = f_b < f_h$.
f_e, f_c are polylogs. The change in base for f_e is a constant factor difference. $\log n^2 = 2 \log n$, so taking n^2 is also a constant factor difference. Therefore f_e is the smallest.
f_h is an exponential, and thus the largest. Everything else is a polynomial.
$f_g < n$, and thus the smallest polynomial. f_f is exactly n, and comes next. f_a is slightly smaller than n^2. f_d is a constant factor difference from f_b, so these are the two functions with the same growth rate.

4. You are given a unweighted directed acyclic graph G, and a topological ordering for G. The topological ordering is represented as a function f which takes as input an integer between 1 and n and outputs the node with that index in the topological ordering in constant time. Give an efficient algorithm to determine the length of the longest path in G. You should aim for a linear run-time algorithm.
Solution:
Keep n different variables, indicating the length of the longest path which ends at node i. Initialize these values to 0. This step takes $\theta(n)$ time.

Now iterate through all nodes, in the order of the topological ordering. That is, run the following code first for $f(1)$, then $f(2)$, etc.

For the current node x, check the lengths of the longest paths ending at each node with an incoming edge to x. Take the max of these, add 1, and assign this as the length of the longest path ending at x. This step works because we will have calculated all previous nodes already since we are tackling them in topological order. Over all iterations of this step, we will check each edge once, and each node once, so this step takes $\theta(m + n)$ time.

Now walk through each node to find the length of the longest path ending at that node. Take the max of these values: this is the desired length. This step takes $\theta(n)$ time.
The total runtime for this algorithm is linear: $\theta(m + n)$.