FirstName LastName
StudentID acct@ucsc.edu

CMPS 102 — Winter 2019 – Homework 1 revised 1/12

Four problems, 36 points, due Wednesday Jan. 23rd, see the *Homework Guidelines*

1. (10 pts) A grad student comes up with the following algorithm to sort an array $A[1..n]$ that works by first sorting the first 2/3rds of the array, then sorting the last 2/3rds of the (resulting) array, and finally sorting the first 2/3rds of the new array.

```
1:  function G-SORT(A, n)                          ▷ takes as input an array of n numbers, A[1..n]
2:      G-sort-recurse(A, 1, n)
3:  end function
4:  function G-SORT-RECURSE(A, ℓ, u)
5:      if u − ℓ ≤ 0 then
6:          return                                  ▷ 1 or fewer elements already sorted
7:      else if u − ℓ = 1 then                      ▷ 2 elements
8:          if A[u] < A[ℓ] then                     ▷ swap values
9:              temp ← A[u]
10:             A[u] ← A[ℓ]
11:             A[ℓ] ← temp
12:         end if
13:     else                                        ▷ 3 or more elements
14:         size ← u − ℓ + 1
15:         twothirds ← ⌈(2 ∗ size)/3⌉
16:         G-sort-recurse(A, ℓ, ℓ + twothirds − 1)
17:         G-sort-recurse(A, u − twothirds + 1, u)
18:         G-sort-recurse(A, ℓ, ℓ + twothirds − 1)
19:     end if
20: end function
```

   First (5 pts), prove that the algorithm correctly sorts the numbers in the array (in increasing order). After showing that it correctly sorts 1 and 2 element intervals, you may make the (incorrect) assumption that the number of elements being passed to *G-sort-recurse* is always a multiple of 3 to simplify the notation (and drop the floors/ceilings).

   Next (1 pts), Derive a recurrence for the algorithm's running time (or number of comparisons made).

   Finally (4 pts), obtain a good asymptotic upper bound (big-O) for your recurrence.

2. Induction Proof correctness (10 pts).

   Recall that a full binary tree contains (A) just a single leaf node, or (B) is an internal node (the root) connected to two disjoint subtrees, which are themselves full binary trees.

   First consider the following claim and proof. Think about if the theorem is true or not, and if the proof is correct or not. (These preliminary thoughts need not be included in your answer.)

   Claim 1. *In any full binary tree, the number of leaf nodes is one greater than the number of internal nodes.*

Proof. (??) By induction on number of internal nodes.

For all $n \geq 0$, let IH(n) be the statement: "all full binary trees having exactly n internal nodes have $n + 1$ leaf nodes."

Base Case: Show IH(0). Every full binary tree with zero internal nodes is formed by case (A) of the definition, and thus consists of just a single leaf node. Therefore, every full binary tree with 0 internal nodes has exactly 1 leaf node, and IH(0) is true.

Inductive Step: Assume $k > 0$ and IH(k) holds to show that IH$(k + 1)$ also holds.

Consider an arbitrary full binary tree T with k internal nodes. By the inductive hypothesis T has $k + 1$ external nodes. Create a $k + 1$ internal node tree T' by removing a bottom leaf node in T and replacing it with an internal node connected to two children that are leaves. T' has one more internal node than T, and $2 - 1 = 1$ more external node than T. Therefore T' has $k + 1$ internal nodes and $(k + 1) + 1 = k + 2$ external nodes, proving $P(k + 1)$. □

Now consider the following claim. Give a counter-example showing that the claim is false (1 pt). (Recall that the height of a binary tree is the length of the longest root-to-leaf path).

Claim 2. *For all n, all full binary trees with n internal nodes have height $n - 1$.*

We have the following "proof" of the claim.

Proof. ?? By induction on n. For each $n \geq 1$, let IH(n) be the statement "all full binary trees with n internal nodes have height $n - 1$".

Base Case: Show IH(0). Every full binary tree with zero internal nodes is formed by case (A) of the definition, and thus consists of just a single leaf node. Therefore, every full binary tree with 0 internal nodes has only one leaf (which is also the root). Thus the longest root-to-leaf path has length 0, and IH(0) is true.

Inductive Step: Let $n \geq 1$ and show that IH(n) implies IH$(n + 1)$. Let T be an arbitrary full binary tree with n internal nodes. Create the binary tree T' having $n+1$ internal nodes by removing a bottom leaf node in T and replacing it with an internal node connected to two children that are leaves. The longest root-to-leaf path in T' is thus one greater than the longest root-to-leaf path in T, so the height of T' is the height of T plus 1. Furthermore, by the inductive hypothesis IH(n), the height of T is $n - 1$. Therefore T' has $n + 1$ internal nodes and height $n - 1 + 1 = n$, showing IH$(n + 1)$. □

Identify and clearly describe the flaw in this proof (4 pts).

Now go back to the proof of the first claim. Does it have a flaw (1 pt)? Clearly describe the flaw, or argue that the proof is correct (4 pts).

3. Asymptotic notation (6 pts): Exercise 5 of Chaper 2 (prove or disprove 3 asymptotic implications).

4. Submarine Hiding (10 pts).

   A submarine is ordered to patrol between two ports. Assume the locations along the patrol route are identified by consecutive integers between 0 (for the first port) and n (for the other port). One way for the submarine to hide is to settle on the bottom in a "sea valley" where the depth at that location is greater than the depths of both adjacent locations. The submarine may determine the depth at its

current location by performing a "depth survey" that is somewhat expensive. The returned depth for the location will be a non-negative number, with larger numbers indicating deeper depths. Since the ports have adjacent locations on only one side, they do not qualify as "sea valleys". Furthermore, assume:

- the depth at both ports is 0
- the depth at the other locations is positive
- adjacent locations always have different depths

First (9 pts): Create a divide and conquer algorithm for finding a "sea valley" (a location whose depth is greater than the adjacent locations) that requires only $O(\log n)$ depth surveys in the worst case (any sea valley will do, it need not be the first or deepest). State your algorithm, argue that it is correct, derive the recurrence for the number of depth surveys needed, and then bound the recurrence. Measure the cost of your algorithm in "depth surveys" and ignore the time to sail between locations.

Second (1 pt): Assume that the submarine starts out at location 0, and runs your algorithm by sailing to each requested depth survey location in turn, and the locations are evenly spaced at one unit apart. Get a good asymptotic bound on how far the submarine might have to sail (as a function of n) when using your algorithm.

Recommended exercises (not to be turned in)

1. The solved exercises in Chapters 1, 2, and 5 (Divide and Conquer).
2. Exercises 1 and 2 in Chapter 1 of the text.
3. Exercises 3 and 4 in Chapter 2 of the text.