CS 6363.500 Homework 1

Due Tuesday February 5th on eLearning

January 22, 2019

Please answer each of the following questions.

Some important homework policies

- Each student must write their solutions in their own words and submit their solutions to eLearning. Clearly print your name, the homework number (Homework 1), and the problem number at the top of every page in case we print anything. Start each numbered homework problem on a new page.
- Unless the problem states otherwise, you must justify (prove) that your solution is correct.
- We strongly suggest you use LaTeX to typeset your solutions. Any illegible solutions will be considered incorrect. The announcement for this homework links to a template for writing solutions in LaTeX.
- If you use outside sources or write solutions in close collaboration with others, then you may cite that source or person and still receive full credit for the solution. Material from the lecture, the textbooks, or prerequisite courses need not be cited. Failure to cite other sources or failure to provide solutions in your own words, even if quoting a source, is considered an act of academic dishonesty.

See https://utdallas.edu/~kyle.fox/courses/cs6363.005.19s/about.shtml and https://utdallas.edu/~kyle.fox/courses/cs6363.005.19s/writing.shtml for more detailed policies. If you have any questions about these policies, please do not hesitate to ask in class, in office hours, or through email.

1. A duck, a robin, and a chick decide to make up a song that they can sing to their friends. Being birds, their vocabulary is limited. Also, the chick is somewhat shy but agreeable, and he will sing his 'lyrics' only when specifically asked to by the audience.

   The lyrics to their song are best described in the pseudocode below. BIRDSONG($peep[1 .. n]$) takes as its only parameter an array of booleans $peep[1 .. n]$ where $peep[i] =$ TRUE if and only if the chick should sing during the ith verse.

```
BIRDSONG(peep[1 .. n]):
  for i ← 1 to n
      The duck sings "quack!"
      for j ← 1 to i
          The robin sings "chirp!"
          if peep[i]
              for k ← 1 to j
                  The chick sings "peep!"

  The duck sings "quack!", the robin sings "chirp!", and the chick sings "peep!".
```

   (a) Truthfully write the phrase ***"I have read and understand the course policies."***

   (b) Give as tight an *upper bound* as you can for the amount of time it takes to perform BIRDSONG($peep[1 .. n]$). Your bound should be given as $O(g(n))$ for some simple function $g(n)$, and it should be true regardless of which bits in $peep[1 .. n]$ are set to TRUE. Don't forget to give a brief justification for your bound.

   (c) Give as tight a *lower bound* as possible for the amount of time it takes to perform BIRDSONG($peep[1 .. n]$). Your bound should be given as $\Omega(g(n))$ for some simple function $g(n)$. Again, justify your bound.

   (d) If possible, give an asymptotically *tight* bound for the amount of time it takes to perform BIRDSONG($peep[1 .. n]$), stating it as $\Theta(g(n))$ for some simple function $g(n)$. If not possible, give a brief explanation for why not.

2. A *binomial tree of order k* is defined recursively as follows:

   - A binomial tree of order 0 is a single node.
   - For all $k > 0$, a binomial tree of order k consists of two binomial trees of order $k-1$, with the root of one tree connected as a new child of the root of the other. (See the figure below.)

   Prove the following claims (preferably using induction):

   (a) For all non-negative integers k, a binomial tree of order k has exactly 2^k nodes.

   (b) For all positive integers k, attaching a new leaf to every node in a binomial tree of order $k-1$ results in a binomial tree of order k.

   (c) For all non-negative integers k and d, a binomial tree of order k has exactly $\binom{k}{d}$ nodes with depth d. (Hence the name!) *[Hint: For any positive integer r and non-negative integer t, we have $\binom{r}{t} = \binom{r-1}{t-1} + \binom{r-1}{t}$.]*

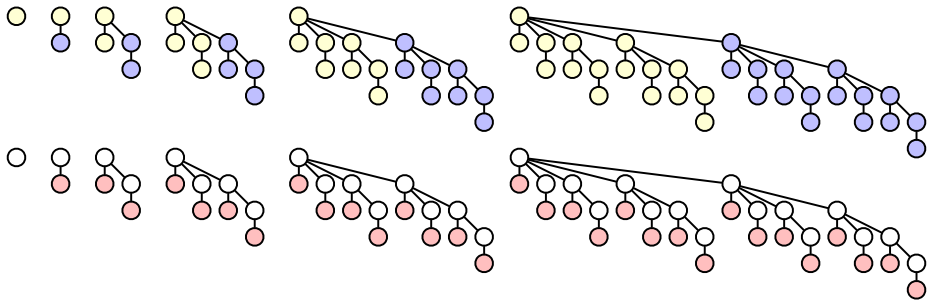

Figure 1. Binomial trees of order 0 through 5. Top row: The recursive definition. Bottom row: The property claimed in part (b).

3. Using Θ-notation, provide asymptotically tight bounds in terms of n for the solution to each of the following recurrences. Assume each recurrence has a non-trivial base case of $T(n) = \Theta(1)$ for all $n < n_0$ where n_0 is a suitably large constant. For example, if asked to solve $T(n) = 2T(n/2) + n$, then your answer should be $\Theta(n \log n)$. **Give a brief explanation for each solution.** [*Hint: Only some of these recurrences can be solved using the master method, but they can all be solved using recursion trees.*]

   (a) $T(n) = 8T(n/4) + n^{1.5}$

   (b) $T(n) = 7T(n/2) + n^3$

   (c) $T(n) = 5T(n/3) + n$

   (d) $T(n) = 3T(n/5) + T(2n/5) + n$

   (e) $T(n) = 3T(n/3) + n \lg n$

4. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children and every leaf has exactly the same depth. The *depth* of a complete binary tree is the depth of each leaf (by convention, a binary tree with exactly one node has depth 0). Our goal for this problem is to design an algorithm to compute the depth of *largest complete subtree* of a given binary tree.

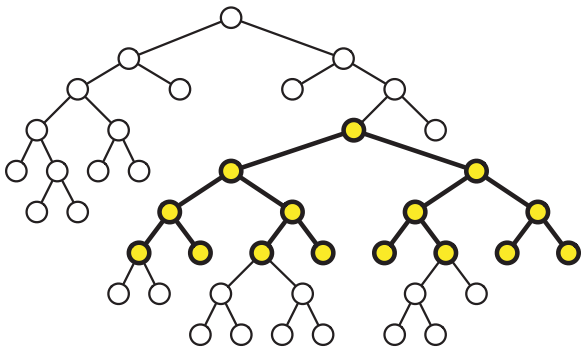

Figure 2. The largest complete subtree of this binary tree has depth 3.

(a) Suppose you are given a binary tree T. Let $\ell_\uparrow$ be the depth of the largest complete subtree *rooted on the left child of T's root*. Let $r_\uparrow$ be the depth of the largest complete subtree *rooted on the right child of T's root*. Finally, let ℓ and r denote the depth of the largest complete subtrees rooted anywhere in the left and right subtrees of T, respectively.

What is the depth of the largest complete subtree of T in terms of $\ell_\uparrow$, $r_\uparrow$, ℓ, and r? Be sure to justify your answer.

(b) Describe a recursive algorithm that computes the depth of the largest complete subtree of a given binary tree T. If the input to your algorithm should be a subtree, then it suffices to pass in the root of that subtree. Don't analyze the running time yet, that's for part (c). *[Hint: Use part (a).]*

(c) What is the running time of your algorithm in terms of n, the number of nodes in T? *[Hint: How much time do you spend in each subproblem outside the recursive calls? How many subproblems are there?]*

5. You are a visitor at a political convention with n delegates; each delegate is a member of exactly one political party. It is impossible to tell which political party any delegate belongs to; in particular, you will be summarily ejected from the convention if you ask. However, you can determine whether any pair of delegates belong to the *same* party by introducing them to each other. Members of the same political party always greet each other with smiles and friendly handshakes; members of different political parties always greet each other with angry stares and insults.

   Suppose more than half the delegates belong to the same political party. Describe an efficient algorithm that identifies all members of this majority party. If you write pseudocode (which we highly recommend) be sure to formally define its input parameters and any subroutines you use. *[Hint: Write a divide-and-conquer algorithm. Any correct algorithm with a running time of $O(n \log n)$ is worth full credit.]*