CS 161
Summer 2017

Problem Set 2
Due: July 14, 2017, 11:59 p.m.

---

1. **Asymptotics redux** (5 points) For the following pairs of functions $f(n)$ and $g(n)$, indicate whether: (i) $f(n) = O(g(n))$, (ii) $f(n) = \Omega(g(n))$, or (iii) $f(n) = \Theta(g(n))$.

   If you believe $f(n) = O(g(n))$, show constants c and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. If you believe $f(n) = \Omega(g(n))$, show constants c and n_0 such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$.

   [We are expecting selection of (i), (ii), or (iii) and a choice of c and n_0.]

   (a) (1 point) $f(n) = 2n^4 - 3n^2 + 7$, $g(n) = n^5$

   (b) (1 point) $f(n) = \frac{\log_2 n}{n}$, $g(n) = \frac{1}{n}$

   (c) (1 point) $f(n) = 2^n$, $g(n) = 2^{2n}$

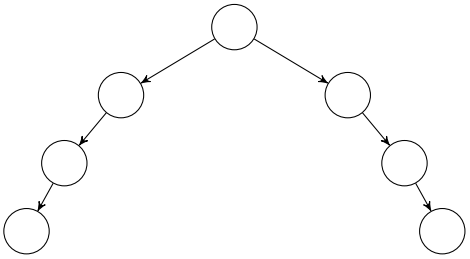
   (d) (1 point) $f(n) = \binom{n}{2}$, $g(n) = 4^{\log_2 n}$

   (e) (1 point) $f(n) = 2^{\sqrt{\log_2 n}}$, $g(n) = (\log_2 n)^{100}$

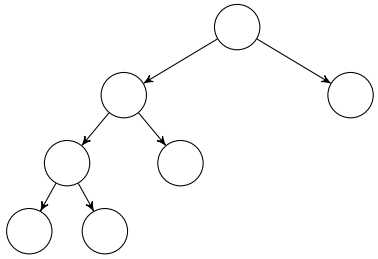
2. **Red, black, or neither?** (4 points) For each of the unlabeled binary trees, state whether or not it can be the structure of a red-black tree. If so, color the vertices red or black. If not, state which of the red-black tree invariants (1 to 5) cannot be satisfied and provide an explanation.

   [We are expecting a YES/NO, and either (1) a valid coloring or (2) a set of violated invariants and a brief explanation. For the colorings, feel free to redraw the trees by hand and upload the image.]

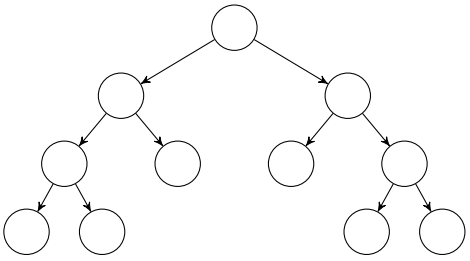
   (a) (1 point)

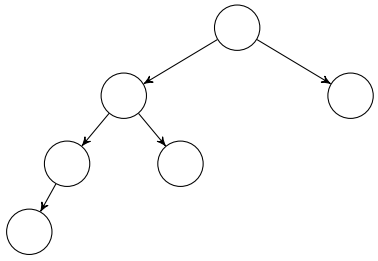
   (c) (1 point)

   (b) (1 point)

   (d) (1 point)

3. **Finding k occurrences.** (7 points) Let L be an unsorted sequence of n numbers, where the numbers are not necessarily distinct. Assume you have an existing an $O(n \log n)$-time and $O(1)$-space sorting algorithm (i.e. heapsort) and the linear-time selection algorithm, which can may be used as subroutines. We want to find all elements in L that occur k or more times with using $O(1)$ extra space.

   (a) (1 point) Let $k = \lceil n/2 \rceil$; describe an $O(n)$ algorithm.
   [We are expecting a description of the algorithm (pseudocode optional), and a brief justification for its runtime.]

   (b) (1 point) Describe an $O(n \log n)$ algorithm that works for any k.
   [We are expecting a description of the algorithm (pseudocode optional), and a brief justification for its runtime.]

   (c) (3 points) Using divide-and-conquer, describe an algorithm with runtime $T(n, k) \leq O(n \log(n/k))$.
   [We are expecting a description of the algorithm, pseudocode, and a brief justification for its runtime.]

   (d) (2 points) Prove the lower bound $T(n, 2) \geq \Omega(n \log n)$.
   [We are expecting a brief, but convincing proof.]

4. **Word representations.** (5 points) As described in class, radix sort runs in $O(d(n + k))$ if the stable sort it uses runs in $O(n + k)$ time. For this problem, assume that radix sort uses bucket sort, which does in fact run in $O(n + k)$ time.

   (a) (1 point) Pretend we want to use radix sort to sort a list of words W in lexicogrpahical (alphabetical) order. All words are padded with *space* characters (assume *space* comes before 'a' lexicographically) to make them the same length. For the following W, describe what the three variables d, n, and k refer to (your explanation should include some reference to W or its elements) and what their values would be for this problem. (For example, if the formula included a variable x that referred to the length of the first word in the list of items being sorted, you might say "x is the length of the first word in the list, and is equal to 3 for list W.")

   W = [the, quick, brown, fox, jumps, over, the, lazy, dog]

   [We are expecting values and descriptions for d, n, and k.]

   (b) (1 point) Now suppose we convert each of the strings in W to their ASCII representations (8-bit binary strings, with *space* mapping to 00100000, so the word 'the' becomes 40 digits long–8 digits per character plus 8 digits per padding space to make it as long as the longest word in W). We still want to use radix sort, and we want to treat these bit strings as literal strings (i.e., do not try to interpret the 8 bit strings into decimal numbers). Now what are the values for d, n, and k?
   [We are expecting values and descriptions for d, n, and k.]

(c) (1 point) Now we're back to using the character strings from part (a), but you happen to have the date (day, month, year) that each word was first published in an English dictionary. You want to sort first by date, then use lexicographical ordering to break ties. You will do this by converting each of the original words in W into words with date information (digits) prepended, appended, or inserted somewhere in the string. (Assume the digits 0-9 come before the *space* character and a-z). Write the string you would use to represent the word "jumps" (first published November 19, 1562) so that it will be correctly sorted by radix sort for the given objective.

[We are expecting a string.]

(d) (1 point) You decide that because you only ever use the words in a certain list V in everyday speech, you would like to save space and simply represent the first word in V with the binary value '0', the second word with '1', the third with '10', etc., continuing to increment by one in binary (and no longer including date information). All subsequent occurences of a particular word w receive the same binary assignment as the first occurence of w, all strings are padded with '0's to make them equal length. V has n words in it, where $n > 2$. Give the time complexity of radix sort on the list V with all words converted to their 0-padded binary strings and explain (informally) why that is correct. Simplify your answer as much as possible where values of d, n, or k are known.

[We are expecting a runtime, and a brief justification.]

(e) (1 point) Not wanting to mess with binary conversions, you decide instead to represent the words in your vocabulary V with "one-hot" vectors (vectors of length n with all 0's except for a single '1' in a position corresponding to a particular word. For example, in W, the word 'the' would be represented as '10000000', since there are eight unique words in the list). Give the new worst-case time complexity of radix sort on the list V, again simplifying as much as possible and explaining (informally) why that is the correct complexity.

[We are expecting a runtime, and a brief justification.]

5. **Centroid trees.** (7 points) We will now shift our attention to trees. Given a tree with N nodes, its centroid is defined by the node whose removal splits the tree into a forest of trees such that each of the resulting trees contains at most $N/2$ nodes.

(a) (2 points) Prove that, for any given tree, there exists a centroid.

[We are expecting a rigorous proof.]

(b) (2 points) Give an algorithm to find the centroid of a tree. Assume that you have an algorithm that can compute the size of a tree given its root in linear time (We will actually learn about this later in the class. If you are curious, please check out the Depth-First-Search algorithm). Also, provide the runtime of this algorithm.

[We are expecting a description of the algorithm (pseudocode optional) and a brief justification for its runtime.]

(c) (2 points) Building upon the concept of a centroid, let us now explore centroid trees (not to be confused with the *centroid of a tree*. In a centroid tree, every node is

the centroid of its own subtree (with this node as the root of the subtree). Give an algorithm to construct a centroid tree from a tree with N nodes. (Hint: use divide-and-conquer) with its runtime.

[We are expecting a description of the algorithm (pseudocode optional) and a brief justification for its runtime.]

(d) (1 point) What is the height (number of levels) of this centroid tree? Please support your claim with a brief explanation.

[We are expecting a height and a brief justification.]

6. **20-questions?** (8 points) Suppose you want to sort an array A of n numbers (not necessarily distinct), and you are guaranteed that all the numbers in the array are in the set $\{1, \ldots, k\}$. A **"20-question sorting algorithm"** is any deterministic algorithm that asks a series of YES/NO questions (not necessarily 20 of them, that's just a name) about A, and then *writes down* the elements of A in sorted order. (Specifically, the algorithm does not need to rearrange the elements of A, it can just write down the sorted numbers in a separate location).

   Note that there are many YES/NO questions beyond just comparison-questions—for example, the following are also valid YES/NO questions: "If I ignored A[3] and A[17] would the array be sorted?" and "Did it rain today?"

   (a) (3 points) Describe a 20-question sorting algorithm that, for every input, asks only $O(k \log n)$ questions. For now, assume that the algorithm accepts n and k as input.

   [We are expecting a description of the algorithm and a brief justification for its runtime.]

   (b) (1 point) Now suppose the algorithm does not accept n or k as input. Describe an algorithm that determines these values and still asks only $O(k \log n)$ questions.

   [We are expecting a description of the algorithm and a brief justification for its runtime.]

   (c) (4 points) Prove that for *every* 20-question sorting algorithm, there exists some array A consisting of n integers between 1 and k that will require $\Omega(k \log \frac{n}{k})$ questions, provided $k \leq n$.

   [We are expecting a mathematically rigorous proof (which does NOT necessarily mean something long and tedious).]

7. **How is the course so far?** (0.5 points extra credit)

   Please complete the poll at `https://goo.gl/forms/LnGR9BvTQj8MuRyH2`.