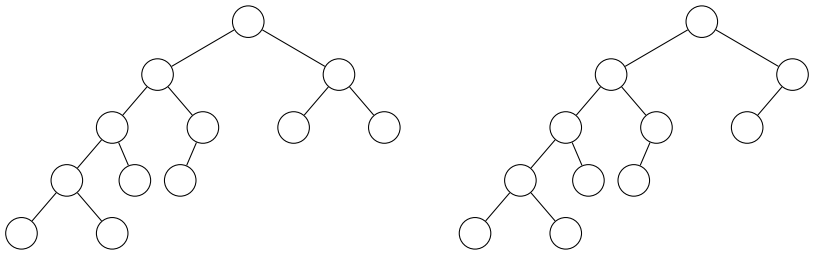
CS 161 (Stanford, Winter 2022) Homework 4

Style guide and expectations: Please see the "Homework" part of the "Resources" section on the webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards. You should cite all sources you used outside of the course material.
What we expect: Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

Exercises. The following questions are exercises. We suggest you do these on your own. As with any homework question, though, you may ask the course staff for help.

1 Exercise: Coloring Red-Black Trees (6 pt.)

For each of the following examples, if the nodes can be colored red or black to make a legitimate red-black tree, then give such a coloring. If not, then say that they cannot.



[We are expecting: For each tree, either an image of a colored-in red-black tree or a statement "No such red-black tree." No justification is required.**]**

2 Word Representations

As described in class, radix sort runs in $O(d(n+r))$ where n is the number of elements, r is the base, and d is the max length of the elements. For this problem, assume that radix sort uses counting sort, which does in fact run in $O(n+r)$ time. [**Note**: The question looks long but it is in fact very straightforward.]

1. **(2 pt.)** Pretend we want to use radix sort to sort a list of words W in lexicographical (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 r refer to (your explanation should include some reference to W or its elements) and what their values would be for this problem.
   W = [the, quick, brown, fox, jumps, over, the, lazy, dog]
   [We are expecting: values and descriptions for d, n, and r.**]**

2. **(2 pt.)** 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 r? **[We are expecting:** values and descriptions for d, n, and r.**]**

3. **(2 pt.)** 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. 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.**]**

4. **(2 pt.)** 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 r are known. **[We are expecting:** a runtime, and a brief justification.**]**

5. **(2 pt.)** 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.**]**

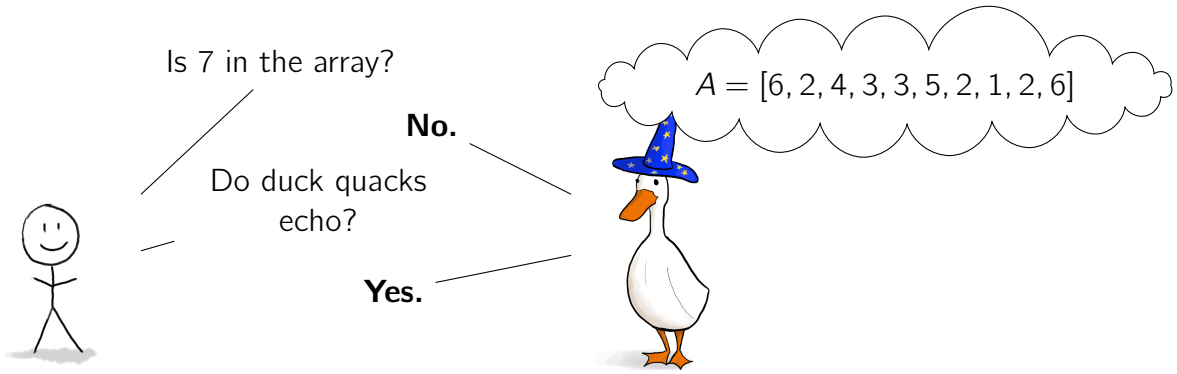
Problems. The following questions are problems. You may talk with your fellow CS 161-ers about the problems. However:

- Try the problems on your own *before* collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

---

3 Wise Duck

A wise duck has knowledge of an array A of length n, so that $A[i] \in \{1, \ldots, k\}$ for all i (note that the elements of A are not necessarily distinct). You don't have direct access to the list, but you can ask the wise duck *any* yes/no questions about it. For example, you could ask "If I remove $A[5]$ and swap $A[7]$ with $A[8]$, would the array be sorted?" or "are ducks related to grebes?" This time you did bring a paper and pencil, and your job is to write down all of the elements of A in sorted order.[1] You are allowed to take all the time you need to do any computations on paper with the wise duck's answers, but the wise duck charges one ice cream cone per question.



1. **(4 pt.)** Design an algorithm which outputs a sorted version of A which uses $O(k \log n)$ ice cream cones. You may assume that you know n and k, although this is not necessary. **[We are expecting:** Pseudocode and a clear English explanation of what it is doing. An explanation of why the algorithm uses $O(k \log n)$ ice cream cones. You do not need to prove that your algorithm is correct.**]** [**HINT:** *One approach is to think first about what you would do for $k = 2$: that is, when A contains only the numbers 1 and 2. What information do you need to write down a sorted version of A in this case?*]

2. **(0 pt.)** Prove that any procedure to solve this problem must use $\Omega(k \log \frac{n}{k})$ ice cream cones. [**HINT:** *What happens when $n < k$? What about when $n = k$ and $n > k$?*] **[We are expecting:** Nothing. It's not required.**]**

4 Goose Tree (4 pt.)

A goose comes to you with the following claim. They say that they have come up with a new kind of binary search tree, called `gooseTree`, even better than red-black trees! More precisely, `gooseTree` is a data structure that stores comparable elements in a binary search tree. It might also store other auxiliary information, but the goose won't tell you how it works. The goose claims that `gooseTree` supports the following operations:

- `gooseInsert(k)` inserts an item with key k into the `gooseTree`, maintaining the BST property. It does not return anything. It runs in time $O(1)$.

[1] Note that you don't have any ability to change the array A itself, you can only ask the wise duck about it.

- `gooseSearch(k)` finds and returns a pointer to node with key k, if it exists in the tree. It runs in time $O(\log(n))$.
- `gooseDelete(k)` removes and returns a pointer to an item with key k, if it exists in the tree, maintaining the BST property. It runs in time $O(\log(n))$.

Above, n is the number of items stored in the `gooseTree`. The goose says that all these operations are deterministic, and that `gooseTree` can handle arbitrary comparable objects. You think the goose's logic is a bit loosey-goosey. How do you know the goose is wrong?
Notes:

- You may use results or algorithms that we have seen in class without further justification.
- Since the gooseTree is still a kind of binary search tree, you can access the root of gooseTree by calling gooseTree.root().
- Proofs that just say "since it's impossible to do these operations that fast in BST, this data structure can't exist" is not sufficient.

[We are expecting: Formally prove that the goose is wrong by showing that we can solve an algorithmic problem that we know the lower bound for with this data structure.**]**

5 Merging Two Sorted Lists

Prove that any comparison-based algorithm for merging two sorted lists of length m and n ($m \geq n$), requires $\Omega(n \log(1 + \frac{m}{n}))$ comparisons. We will be using the decision tree method to prove this lower bound.

1. **(4 pt.)** What is the number of leaves in the decision tree? (i.e. How many possible ways are there to merge two lists of size m, n?) **[We are expecting:** A value in terms of m, n, **AND** a brief justification of how you derived this number**]**
2. **(4 pt.)** Prove that merging requires $\Omega(n \log(1 + \frac{m}{n}))$ comparisons. **[We are expecting:** Mathematical derivation of the final result. Explicitly write all the steps.**]**

6 Ternary Search Tree

In this problem, we will be talking about ternary search trees (TST). Ternary search trees are similar to binary search trees but rather than having just 2 pointers (left and right), they have 3 pointers (left, middle, and right). A TST T is a ternary tree in which each node contains a point of the form (x, y) for some $x, y \in \mathbb{Z}$. Moreover, no two points in T can share the same x value and no two points can share the same y value. Then the following properties hold for every node u with key (x, y) in any TST:

- Any point (x_L, y_L) in the left-subtree of u has $x_L < x$ and $y_L < y$.
- Any point (x_M, y_M) in the middle-subtree of u has $x_M > x$ and $y_M < y$.

- Any point (x_R, y_R) in the right-subtree of u has $x_R > x$ and $y_R > y$.

Clarification: You can assume that for any node v in TST T, you can compute the number of nodes that belong to the subtree rooted at v in time $O(1)$ (e.g., it is stored as part of TST).

1. **(6 pt.)** For any set of n distinct points (where no two points share the same x-coordinate or y-coordinate), there exists a ternary search tree T on these n points with height, $h(T)$.

   (a) Prove: $h(T) = O(n)$

   (b) Prove or disprove: $h(T) = O(\log(n))$ [**HINT:** *Is there an arrangement of points in which you can only make a tree of height $O(n)$?*]

   [We are expecting: If the claim is true, do not just provide an example tree. Formally show that any construction works for a general set of n points. If the claim is false, show that an arrangement of n points does not fit the constraint.**]**

2. **(8 pt.)** Suppose that you are given a point (x', y') and a TST T which contains n points and has height h. You wish to determine whether (x', y') is contained in T or not. Design an efficient algorithm to solve this problem. Prove the correctness of the algorithm. Analyze its running time in terms of n and also in terms of h. **[We are expecting:** Pseudocode AND a short English description of your algorithm. A formal proof of correctness and a brief justification of your runtime that is clear and convincing to the grader.**]**