CS 161	**Problem Set 4**
Winter 2021	**Due:** Wed, Feb 10 at 11:59 pm PST

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!

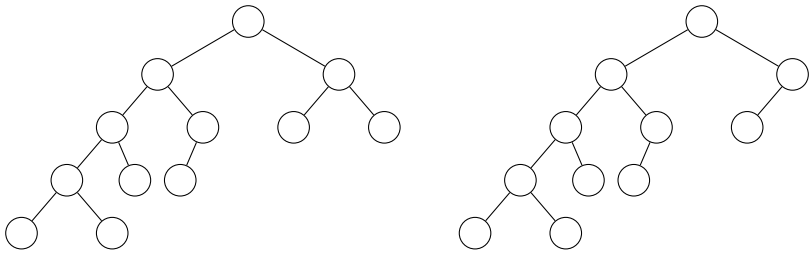
Exercises

We suggest you do these on your own. As with any homework problem, though, you may ask the course staff for help.

---

1. **(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.

   (For reference, the LaTeXcode to make these trees is included at the end of the PSET, as well as instructions about how to color the nodes red or black. If you use this code, make sure you include `\usepackage{tikz}` before the `\begin{document}` command. You are also welcome to take a screenshot and then color the trees in MSPaint, or make a hand-drawn copy and take a photo, or...)

   [**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. (10 pt.) 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 bucket sort, which does in fact run in $O(n+r)$ time. [**Note**: The question looks long but it is in fact very straightforward.]

(a) **(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. (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 r.]

(b) **(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.]

(c) **(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.]

(d) **(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.]

(e) **(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

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.

---

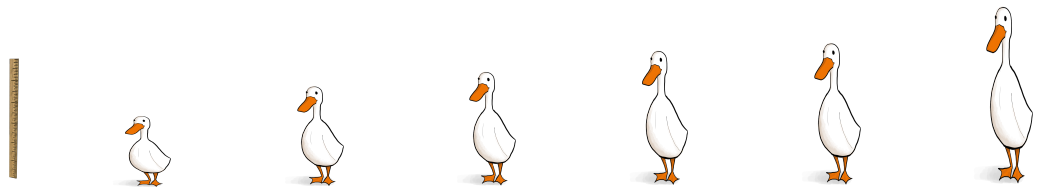
3. [OPTIONAL] The impossible job interview

You're interviewing for your dream job at an ecological ethical tech company with healthy snacks. You already passed 28 stages of interviews, and your final interviewer asks you to design a binary search tree data structure that performs INSERT operations in $O(\sqrt{\log n})$ time using a comparison-based algorithm.

Design such a data structure or prove that this is impossible.

[**We are expecting:** If possible: An English description of the algorithm and a running time analysis. If impossible: A formal proof that this is impossible.]

4. [OPTIONAL] [**Duck.**] Suppose that n ducks are standing in a line, ordered from shortest to tallest (not necessarily of unique height).



You have a measuring stick of a certain height, and you would like to identify a duck which is the same height as the stick, or else report that there is no such duck. The only operation you are allowed to do is `compareToStick(j)`, where $j \in \{0, \ldots, n-1\}$, which returns `taller` if the j'th duck is taller than the stick, `shorter` if the j'th duck is shorter than the stick, and `the same` if the j'th duck is the same height as the stick. You forgot to bring a piece of paper, so you can only remember a constant number of integers in $\{0, \ldots, n-1\}$ at a time.

(a) Give an algorithm which either finds a duck the same height as the stick, or else returns "No such duck," in the model above which uses $O(\log(n))$ comparisons.

[**We are expecting:** Pseudocode **AND** an English description of your algorithm. You do not need to justify the correctness or runtime.]

(b) Prove that any algorithm in this model of computation must use $\Omega(\log(n))$ comparisons.

[**We are expecting:** A short but convincing argument.]

5. (6 pt.) [**Goose!**] 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 $O(1)$.
- `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().

[**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.]

6. **(8 pt.)** 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.

   (a) **(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]

   (b) **(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.]

7. **(1 pt.) Feedback:** Please fill out this Wellness Check Google Form.

   `https://docs.google.com/forms/d/e/1FAIpQLSdgnigmqSm75nDfzId8nhkHDT3SSQQwFMZGAMk04Linlm0ypA/viewform?`

   Please make sure you are accessing the Google Form with your Stanford email/account.

   Did you fill out the poll? [**We are expecting:** "yes"]

Useful Latex Code

Here is some code to generate the red-black trees; we've shown how to color one of the nodes green, you can use this example to color them red or black. You are also welcome to take a screenshot and color in your favorite paint program, or include a picture of a hand-drawn image, or any other way you can think of getting a picture of a colored red-black tree into your PSET.

```
\begin{center}
\begin{tikzpicture}[xscale=0.7]
\begin{scope}
\node[draw,circle,fill=green](b) at (0,0) {};
\node[draw,circle](b0) at (-2,-1) {};
\node[draw,circle](b1) at (2,-1) {};
\node[draw,circle](b00) at (-3,-2) {};
\node[draw,circle](b01) at (-1,-2) {};
\node[draw,circle](b10) at (1,-2) {};
\node[draw,circle](b11) at (3,-2) {};
\node[draw,circle](b000) at (-4,-3) {};
\node[draw,circle](b001) at (-2.5,-3) {};
\node[draw,circle](b010) at (-1.5,-3) {};
\node[draw,circle](b0000) at (-5,-4) {};
\node[draw,circle](b0001) at (-3,-4) {};
\draw (b) -- (b0);
\draw (b) -- (b1);
\draw (b0) -- (b00);
\draw (b0) -- (b01);
\draw (b1) -- (b10);
\draw (b1) -- (b11);
\draw (b00) -- (b000);
\draw (b00) -- (b001);
\draw (b01) -- (b010);
\draw (b000) -- (b0000);
\draw (b000) -- (b0001);
\end{scope}

\begin{scope}[xshift=10cm]
\node[draw,circle](b) at (0,0) {};
\node[draw,circle](b0) at (-2,-1) {};
\node[draw,circle](b1) at (2,-1) {};
\node[draw,circle](b00) at (-3,-2) {};
\node[draw,circle](b01) at (-1,-2) {};
\node[draw,circle](b10) at (1,-2) {};
\node[draw,circle](b000) at (-4,-3) {};
\node[draw,circle](b001) at (-2.5,-3) {};
\node[draw,circle](b010) at (-1.5,-3) {};
\node[draw,circle](b0000) at (-5,-4) {};
\node[draw,circle](b0001) at (-3,-4) {};
\draw (b) -- (b0);
\draw (b) -- (b1);
\draw (b0) -- (b00);
\draw (b0) -- (b01);
```

```
\draw (b1) -- (b10);
\draw (b00) -- (b000);
\draw (b00) -- (b001);
\draw (b01) -- (b010);
\draw (b000) -- (b0000);
\draw (b000) -- (b0001);
\end{scope}

\end{tikzpicture}
\end{center}
```