CS 161 **Problem Set 1**
Fall 2025 **Due:** Friday October 3 at 11:59pm on Gradescope.

Style guide and expectations: Please see the top of the "Homework" page on the course webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards.

Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

Collaboration policy: You may do the HW in groups of size up to three. Please submit one HW for your whole group on Gradescope. (Note that there is an option to submit as a group). See the "Policies" section of the course website for more on the collaboration policy.

LLM policy: Check out the course webpage for best practices on how to productively use LLMs on homework, if you use them at all.

Exercises

We recommend you do the exercises on your own before collaborating with your group. The point is to check your understanding.

---

1. **(1 pt.)** See the IPython notebook HW1.ipynb for Exercise 1. Modify the code to generate a plot that convinces you that $T(x) = O(g(x))$. **Note:** There are instructions for installing Jupyter notebooks in the pre-lecture exercise for Lecture 2.

   [**We are expecting:** *Your choice of c, n_0, the plot that you created after modifying the code in Exercise 1, and a short explanation of why this plot should convince a viewer that $T(x) = O(g(x))$.*]

   SOLUTION:

2. **(10 pt.)** For each row, indicate whether or not the quantity in column **A** is O, Ω, or Θ of the quantity in column **B**. Put a ✓ if your answer is "yes", and leave blank if your answer is "no." We've filled in the first two rows for you. Notice that it's possible for multiple spaces per row to have a ✓ in them. All logarithms are base 2 unless otherwise stated.

   [**We are expecting:** *Some of the blanks to be marked with a ✓ (or X, or ☺, or your favorite symbol). No explanation is required.*]

   Note: The LaTeX template for this problem set has the code for the table.

	A	**B**	O	Ω	Θ
1	$\log n$	n	✓		
2	$n/2023$	$2023 \cdot n$	✓	✓	✓
3	n^3	n^2			
4	3^n	2^n			
5	n^2	2^n			
6	$n^{0.5}$	$(0.5)^n$			
7	$\ln n$	$\log_{10} n$			
8	n^3	$8^{\log_2 n}$			
9	$n^{1/\ln n}$	1			
10	$n^{1/3}$	$(\log n)^3$			
11	$\log \log n$	$\sqrt{\log n}$			

SOLUTION:

3. **(4 pt.)**

   (a) **(2 pt.)** Prove that n is $O(n \log_2 n)$.
   [**We are expecting:** *A formal proof, using the definition of* $O(\cdot)$ *that we saw in class.*]

   (b) **(2 pt.)** Prove that n is *not* $\Omega(n \log_2 n)$.
   [**We are expecting:** *A formal proof, using the definition of* $\Omega(\cdot)$ *that we saw in class.*]

SOLUTION:

Problems

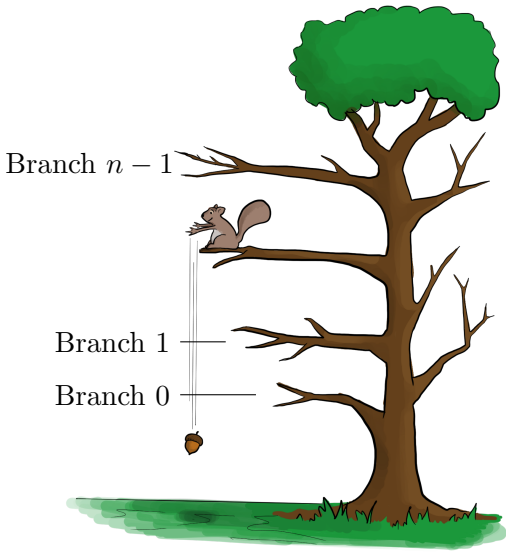
---

4. **[Nuts!] (14 pt.)**

*[**Meta problem-solving skills:** Another version of this problem could jump straight to part (d). The structure of this problem, particularly (b) and (c), is supposed to give you an example of how to solve a harder problem by building up from simpler problems. In the future, you'll have to do more and more of this on your own!]*

Socrates the Scientific Squirrel is conducting some experiments. Socrates lives in a very tall tree with n branches, and she wants to find out what is the lowest branch i so that an acorn will break open when dropped from branch i. (If an acorn breaks open when dropped from branch i, then an acorn will also break open when dropped from branch j for any $j \geq i$.)

The catch is that, once an acorn is broken open, Socrates will eat it immediately and it can't be dropped again.



(a) **(2 pt.)** Suppose that Socrates has $\lceil \log(n) \rceil + 1$ acorns. Give a procedure so that she can identify the correct branch using $O(\log(n))$ drops.

[**We are expecting:** *Very clear pseudocode **or** a short English description of your algorithm. You do not need to justify the number of drops. If it helps you may assume that n is a power of* 2.]

(b) **(2 pt.)** Suppose that Socrates has only one acorn. Give a procedure so that she can identify the correct branch using $O(n)$ drops, and explain why your $O(\log(n))$-drop solution from part (a) won't work.

[**We are expecting:** *Very clear pseudocode **or** a short English description of your algorithm, and one sentence about why your algorithm from part (a) does not apply. You do not need to justify the number of drops. If it helps you may assume that the acorn breaks when dropped from the top branch (in all parts of this problem).*]

(c) **(3 pt.)** Suppose that Socrates has two acorns. Give a procedure so that she can identify the correct branch using $O(\sqrt{n})$ drops.

[**We are expecting:** *Pseudocode **AND** a short English description of your algorithm, and a justification of the number of drops. If it helps you may assume that n is a perfect square.*]

(d) **(5 pt.)** Suppose that Socrates has $k = O(1)$ acorns. Give a procedure so that she can identify the correct branch using $O(n^{1/k})$ drops.

[**We are expecting:** *Pseudocode **AND** a short English description of your algorithm, and a justification of the number of drops. If it helps you many assume that n is of the form $n = m^k$ for some integer m.*]

(e) **(2 pt.)** What happens to the runtime of your algorithm in part (d) when $k = \lceil \log(n) \rceil + 1$? Is it $O(\log(n))$, like in part (a)? Is it $O(n^{1/k})$ when $k = \lceil \log(n) \rceil + 1$, like in part (d)?

[**We are expecting:** *A sentence of the form "the number of drops of my algorithm in part (d) when $k = \lceil \log(n) \rceil + 1$ is $O(\_\_\_\_)$", along with justification.* ***Also****, we are expecting two yes/no answers to the two yes/no questions (you should justify your answers but do not need to include a formal proof).*]

(f) **(NOT REQUIRED. 1 BONUS pt.)** Is $\Theta(n^{1/k})$ drops is the best that Socrates can do with k acorns, for $k = O(1)$? Either give a proof that she can't do better, or give an algorithm with asymptotically fewer drops.

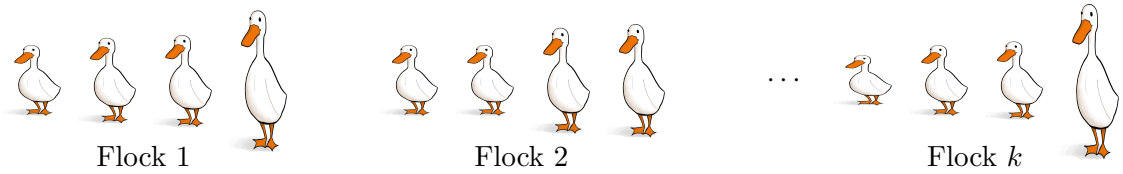
[**We are expecting:** *Nothing. This part is not required.*]

SOLUTION:

5. **[Ducks in a row.] (11 pt.)**

*[**Meta problem-solving skills:** As the hint in part (b) suggests, you might want to consider an algorithm you've already seen to solve this problem. In general, taking inspiration from algorithms you already know is a good problem-solving technique!]*

There are k flocks of ducks, and each flock has n ducks each. The k flocks are coming together for a mixer, and for a particular event, they would like to sort all kn ducks by height.[1] Each flock submits a height-ordered list of its ducks, and you (the organizer) are presented with k ordered lists, $A_1, \ldots, A_k$, each of length n.



(a) **(1 pt.)** Your assistant, having just seen Lecture 2, is excited to try MergeSort on the ducks. They suggest concatenating the lists (to get a big list of length nk), and then running MergeSort on this big list. How long will this take?

[**We are expecting:** *A single big-Oh expression. No justification is needed.*]

(b) **(6 pt.)** Suppose that k is significantly smaller than n. (For example, say that $k = O(\log n)$). Design an algorithm that is *asymptotically faster* than the algorithm suggested in part (a) [2].

[**Hint:** *Take inspiration from MergeSort.*]

[**We are expecting:** *Pseudocode **AND** a short English description of your algorithm; **AND** a clear statement of the running time; **AND** an explanation for why this is asymptotically faster than your answer in part (a) when $k = \log(n)$.*

You do not need justify the running time, but you might want to do so for your own confidence and/or for partial credit. You do not need to formally prove that your running time is asymptotically faster than part (a), but you should give a clear explanation, don't just write down two ugly expressions and assert that one is smaller than the other.]

(c) **(4 pt.)** Rigorously prove by induction that your algorithm is correct. If it's relevant, you may assume that the MERGE algorithm that we saw in class is correct. If it helps, you may assume that k is a power of 2.

[**We are expecting:** *A rigorous proof by induction. Make sure to clearly label your inductive hypothesis, base case, inductive step and conclusion.*]

SOLUTION:

[1] Bonus points for coming up with the most creative description of this event. The course staff is going with an elaborate duck line dance.

[2] Here, "asymptotically faster" means that if the running time of your algorithm is $T(n)$, and if $T_0(n)$ is your running time from part (a), then $T(n) = O(T_0(n))$ but $T_0(n)$ is *not* $O(T(n))$.