CS1200: Intro. to Algorithms and their Limitations Prof. Anurag Anshu

Problem Set 1

Harvard SEAS - Spring 2026 *Due: Sat 2026-02-07 (11:59pm)*

Please see the syllabus for the full collaboration and generative AI policy, as well as information on grading, late days, and revisions.

All sources of ideas, including (but not restricted to) any collaborators, AI tools, people outside of the course, websites, ARC tutors, and textbooks other than Hesterberg–Vadhan must be listed on your submitted homework along with a brief description of how they influenced your work. You need not cite core course resources, which are lectures, the Hesterberg–Vadhan textbook, sections, SREs, earlier problem sets and earlier solutions sets in this semester. If you use any concepts, terminology, or problem-solving approaches not covered in the course material by that point in the semester, you must describe the source of that idea. If you credit an AI tool for a particular idea, then you should also provide a primary source that corroborates it. Github Copilot and similar tools should be turned off when working on programming assignments.

If you did not have any collaborators or external resources, please write 'none.' Please remember to select pages when you submit on Gradescope. A problem set on the border between two letter grades cannot be rounded up if pages are not selected.

Your name:
Collaborators and External Resources:
No. of late days used on previous psets:
No. of late days used after including this pset:

1. (Asymptotic Notation)

   (a) (practice using asymptotic notation) Fill in the table below with "T" (for True) or "F" (for False) to indicate the relationship between f and g. For example, if f is $\Omega(g)$, the first cell of the row should be "T." No justification necessary. Notice that some of the functions are the same as in Problem Set 0. Recall that, throughout CS1200, all logarithms are base 2 unless otherwise specified.

f	g	Ω	ω	Θ
$3(\log_2 n)^{3/2}$	$2(\ln n) \cdot (\ln n + 3)$			
$3n^3$	$\lvert\{S \subseteq [n] : \lvert S\rvert \leq 4\}\rvert$			
$(n+1)^{n+1}$	$(n+1) \times n!$			
4^n	$(4 + (-1)^n)^n$			

   (b) (runtimes: $T^=$ vs. T) Let $g : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ be a nondecreasing function, i.e. if $x_0 \geq x_1$, then $g(x_0) \geq g(x_1)$. (For example $g(n) = n^2$, $g(n) = n \log n$, or $g(n) = 2^n$.) Let $T^= : \mathbb{N} \to \mathbb{N}$ and $T : \mathbb{R}^{\geq 0} \to \mathbb{N}$ be the runtimes of an algorithm A, as defined in Lecture 2.

      i. Consider the two statements:

$$\text{“}T^= = \Omega(g) \implies T = \Omega(g)''$$

      and

$$\text{“}T = \Omega(g) \implies T^= = \Omega(g)''.$$

      One of these statements is true and one is false. Prove the correct statement. For the false statement, give an example of a potential runtime $T^=$ and a function g to demonstrate. (Hint: one of the pairs of functions in the table above may be helpful.)

      ii. (Optional question[1]) Prove that $T^= = O(g)$ iff $T = O(g)$. Thus the distinction between T and $T^=$ does not matter when we are interested in "big-oh" runtime bounds, which are nondecreasing in most "natural" cases.

[1] This question will be graded only if you have seriously attempted all the other questions. It can only be used to increase your grade from an R to an R^+.

2. (Understanding computational problems and mathematical notation)

Recall the definition of a *computational problem* from Lecture Notes 2.

Consider the following computational problem $\Pi = (\mathcal{I}, \mathcal{O}, f)$:

- $\mathcal{I} = \mathbb{N} \times \mathbb{N}^{\geq 2} \times \mathbb{N}$, where $\mathbb{N}^{\geq 2} = \{2, 3, 4, \ldots\}$.
- $\mathcal{O} = \{(c_0, c_1, \ldots, c_{k-1}) : k, c_0, \ldots, c_{k-1} \in \mathbb{N}\}$
- $f(n, b, k) = \{(c_0, c_1, \ldots, c_{k-1}) : n = c_0 + c_1 b + c_2 b^2 + \cdots + c_{k-1} b^{k-1}, \forall i \ 0 \leq c_i < b\}$.

Here is an algorithm `BC` to solve Π:

```
1 BC(n, b, k)
2 foreach i = 0, ..., k − 1 do
3 |   c_i = n mod b;
4 |   n = (n − c_i)/b;
5 if n == 0 then return (c_0, c_1, ..., c_{k−1});
6 else return ⊥;
```

(a) If the input is $(n, b, k) = (57, 12, 3)$, what does the algorithm `BC` return? Is `BC`'s output a valid answer for Π with input $(57, 12, 3)$?

(b) Describe the computational problem Π in words. (You may find it useful to try some more examples with $b = 10$.)

(c) Is there any $x \in \mathcal{I}$ for which $f(x) = \emptyset$? If so, give an example; if not, explain why.

(d) For each possible input $x \in \mathcal{I}$, what is $|f(x)|$? ($|A|$ is the size of a set A.) Justify your answer(s) in one or two sentences.

(e) Let $\Pi' = (\mathcal{I}, \mathcal{O}, f')$ be the problem with the same $\mathcal{I}$ and $\mathcal{O}$ as Π, but $f'(n, b, k) = f(n, b, k) \cup \{(0, 1, \ldots, k-1)\}$. Does every algorithm A that solves Π also solve Π'? (Hint: any differences between inputs that were relevant in the previous subproblem are worth considering here.) Justify your answer with a proof or a counterexample.

3. (Radix Sort) In the Sender–Receiver Exercise associated with lecture 3, you studied the sorting algorithm `SingletonBucketSort`, generalized to arrays of key–value pairs, and proved that it has running time $O(n+U)$ when the keys are drawn from a universe of size U. In this problem you'll study `RadixSort`, which improves the dependence on the universe size U from linear to logarithmic. Specifically, `RadixSort` can achieve runtime $O(n + n \cdot (\log U)/(\log n))$, so it achieves runtime $O(n)$ whenever $U = n^{O(1)}$.

   `RadixSort` is constructed by using `SingletonBucketSort` as a subroutine several times, but on a smaller universe size b. Specifically, it turns each key from $[U]$ into an array of k subkeys from $[b]$ using the algorithm `BC` from Problem 2 above as a subroutine, and then iteratively sorts on each of the k subkeys, Crucially, `RadixSort` uses the fact that `SingletonBucketSort` can be implemented in a way that is *stable* in the sense that it preserves the order in the input array when the same key appears multiple times. (See the "Food for Thought" section in the SRE notes.) Here is pseudocode for `RadixSort`:

```
1  RadixSort(U, b, A)
   Input  : A universe size U ∈ ℕ, a base b ∈ ℕ with b ≥ 2, and an array
            A = ((K_0, V_0), ..., (K_{n-1}, V_{n-1})), where each K_i ∈ [U]
   Output : A valid sorting of A
2  k = ⌈log_b U⌉;
3  foreach i = 0, ..., n − 1 do
4  |  V'_i = BC(K_i, b, k) ;                          /* V'_i is an array of length k */
5  foreach j = 0, ..., k − 1 do
6  |  foreach i = 0, ..., n − 1 do
7  |  |  K'_i = V'_i[j]
8  |  ((K'_0, (V_0, V'_0)), ..., (K'_{n-1}, (V_{n-1}, V'_{n-1}))) =
   |   SingletonBucketSort(b, ((K'_0, (V_0, V'_0)), ..., (K'_{n-1}, (V_{n-1}, V'_{n-1}))));
9  foreach i = 0, ..., n − 1 do
10 |  K_i = V'_i[0] + V'_i[1] · b + V'_i[2] · b^2 + ··· + V'_i[k − 1] · b^{k−1}
11 return ((K_0, V_0), ..., (K_{n-1}, V_{n-1}))
```

Algorithm 0.1: Radix Sort

(You can also read a description of Radix Sort in CLRS Section 8.3 for the case of sorting arrays of keys (without attached items) when U and b are powers of 2, albeit using different notation than us.)

(a) (proving correctness of algorithms) Prove the correctness of `RadixSort` (i.e. that it correctly solves the SortingOnFiniteUniverse problem defined in SRE 1).

Hint: You will need to use the stability of `SingletonBucketSort` in your argument. If it were replaced with an instable implementation (or any other unstable sorting algorithm, such as `ExhaustiveSearchSort` with an unfortunate ordering on permutations), then the resulting algorithm would not be a correct sorting algorithm. For intuition, you may want to think about what happens when you sort a spreadsheet by one column at a time.

(b) (analyzing runtime) Show that `RadixSort` has runtime $O((n+b) \cdot \lceil \log_b U \rceil)$. Set $b = \min\{n, U\}$ to obtain our desired runtime of $O(n + n \cdot (\log U)/(\log n))$. (This runtime

analysis is outlined in CLRS, but you'd need to adapt it to our notation and slightly more general setting.)

(c) (implementing algorithms) Implement `RadixSort` using the implementations of `SingletonBucketSort` and `BC` that we provide you in the GitHub repository.

(d) (experimentally evaluating algorithms) In `ps1_experiments.py`, we've provided code for running experiments to evaluate the runtime of sorting algorithms on random arrays (with $b = \min\{n, U\}$ in the case of `RadixSort`) and for graphing the results. Run this code and attach the resulting graph (you should see that each sorting algorithm dominates in some region of the graph – if you want better results you can try increasing the number of trials in the experiments file).

Note: Your implementation of RadixSort, as well as any code you write for experimentation and graphing need not be submitted. Depending on your implementation, running the experiments could take anywhere from 15 minutes to a couple of hours, so don't leave them to the last minute!

(e) Do the shapes of the transition curves found in Part 3d match what we'd expect from the asymptotic runtime formulas we have for the algorithms? Explain. For a most thorough answer, try setting the asymptotic runtimes of `SingletonBucketSort` and `RadixSort` to be equal to each other (ignoring the hidden constant in $O(\cdot)$) and see what $\log U$ vs. $\log n$ relationship follows, and similarly for comparing `RadixSort` and `MergeSort`.

4. (Reflection Question) There are a number of resources to support your learning in CS1200, such as Lecture, Ed, Office Hours, Sections, Hesterberg-Vadhan book, Recommended Readings, Collaboration with Classmates, Sender-Receiver Exercises. Which of these (or any others that come to mind) have you found most helpful so far and why? Are there ones that you should take more advantage of going forward? Do you have suggestions for how the course can make these more helpful to you?

   Quick note on grading : Good responses are usually about a paragraph, with something like 7 or 8 sentences. Most importantly, please make sure your answer is specific to this class and your experiences in it. If your answer could have been edited lightly to apply to another class at Harvard, points will be taken off.

   Note: As with the previous pset, you may include your answer in your PDF submission, but the answer should ultimately go into a separate Gradescope submission form.

5. Once you're done with this problem set, please fill out this survey so that we can gather students' thoughts on the problem set, and the class in general. It's not required, but we really appreciate all responses!