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

Problem Set 3

Harvard SEAS - Spring 2026 *Due: Thu Feb. 26, 2026 (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, problem sets and solutions sets from earlier in the 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 OR collaborators not noted.

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

The purpose of this problem set is to solidify your understanding of the RAM model (and variants), and the relations between the RAM model, the Word-RAM model, Python programs, and variants. In particular, you will build skills in simulating one computational model by another and in evaluating the runtime of the simulations (both in theory and in practice).

1. (Simulation in practice: RAMs on Python) In the Github repository, we have given you a partially written Python implementation of a universal RAM Model simulator. Your task is to fill in the missing parts of the code to obtain a complete universal RAM simulator. Your simulator should take as input a RAM Program P and an input x, and simulate the execution of P on x, and return whatever output P produces (if it halts). The RAM Program P is given as a Python list $[v, C_0, C_1, \ldots, C_{\ell-1}]$, where v is the number of variables used by P. For simplicity, we assume that the variables are named $0, \ldots, v-1$ (rather than having names like "tmpptr" and "insertvalue"), but you can introduce constants to give names to the variables. The 0^{th} variable will always be `input_len`, the 1^{st} variable `output_ptr`, and the 2^{nd} variable `output_len`. A command C is given in the form of a list of the form $[\texttt{cmd}]$, $[\texttt{cmd}, i]$, $[\texttt{cmd}, i, j]$, or $[\texttt{cmd}, i, j, k]$, where `cmd` is the name of the command and i, j, k are the indices of the variables or line numbers used in the command. For example, the command $\texttt{var}_i = M[\texttt{var}_j]$ would be represented as $(\text{“}\texttt{read}\text{”}, i, j)$. See the Github repository for the precise syntax as well as some RAM programs you can use to test your simulator.

2. (Empirically evaluating simulation runtimes and explaining them theoretically)

Consider the following two RAM programs:

```
   Input      : A single natural number N (as an array of length 1)
   Output     : 14^(2^N+1) (as an array of length 1)
   Variables  : input_len, output_ptr, output_len, counter, result
 0 zero = 0;
 1 one = 1;
 2 fourteen = 14;
 3 output_len = 1;
 4 output_ptr = 0;
 5 result = 14;
 6 counter = M[zero];
 7     IF counter == 0 GOTO 11;
 8     result = result × result;
 9     counter = counter − one;
10     IF zero == 0 GOTO 7;
11 result = result×fourteen;
12 M[output_ptr] = result;
```

```
   Input      : A single natural number N (as an array of length 1)
   Output     : 14^(2^N+1) mod 2^32 (as an array of length 1)
   Variables  : input_len, output_ptr, output_len, counter, result, temp, W
 0 zero = 0;
 1 one = 1;
 2 fourteen = 14;
 3 output_len = 1;
 4 output_ptr = 0;
 5 result = 14;
 6 W = 2^32;
 7 counter = M[zero];
 8     IF counter == 0 GOTO 15;
 9     result = result × result;
10     temp = result/W;
11     temp = temp × W;
12     result = result − temp;
13     counter = counter − one;
14     IF zero == 0 GOTO 8;
15 result = result × fourteen;
16 temp = result/W;
17 temp = temp × W;
18 result = result − temp;
19 M[output_ptr] = result;
```

(a) Exactly calculate (without asymptotic notation) the RAM-model running times of the above algorithms as a function of N. Which one is faster?

(b) Using your RAM Simulator, run both RAM programs on inputs $N = 0, 1, 2, \ldots, 15$ and graph the actual running times (in clock time, not RAM steps). (We have provided you with some timing and graphing code in the Github repository.) Which one is faster?

(c) Explain the discrepancies you see between Parts 2a and 2b. (Hint: What do we know about the relationship between the RAM and Word-RAM models, and why is it relevant to how efficiently the Python simulation runs?)

(d) (optional[1]) Give a theoretical explanation of the shapes of the runtime curves you see in Part 2b, by providing explicit formulas for the asymptotic runtimes of the two programs (in clock time). You may need to do some research online and/or make guesses about how Python operations are implemented to come up with your estimates.

3. (Simulating Word-RAM by RAM) For every Word-RAM program P, there is a RAM program R that simulates P in the following sense. For every $n \in \mathbb{N}$, input array x of length n, word-length w, and time bound t, $R(x, 2^w, \max_i x[i], t)$ should do the following:

   - If $P[w](x)$ halts without crashing within t steps, then R should halt with the same as the output of $P[w](x)$.
   - If $P[w](x)$ crashes or fails to halt within its first t steps, then the R should indicate so by halting with `output_ptr` $=$ `output_len` $= 2^w$.
   - The runtime of R should be $O(t)$ and the largest memory location accessed by R should be at most $n + t$.
   - All of the values that R's variables and memory cells hold during the computation should have bitlength at most $O(w + \log \max_i x[i] + \log n)$.

   Your proof should use an *implementation-level* description, similar to the proof that RAM programs can be simulated by ones with at most c registers in Theorem 6.4 from the Hesterberg-Vadhan textbook.

   Recall that Word-RAM programs have a finite but changing memory size S; you may want to start your simulation by initializing S. Then think about how each operation of a Word-RAM program P can be simulated in a RAM program R, taking care of any differences between their semantics in the Word-RAM model vs. the RAM model. Don't forget MALLOC!

   Your proof should use an *implementation-level* description, similar to the proof that RAM programs can be simulated by ones with at most c registers in Theorem 6.4 from the Hesterberg-Vadhan textbook. Your answer should include - at least - the following aspects:

   - Initialization: How the memory in the RAM program R will be initialized? Be mindful of the additional "default variables" that a Word-RAM program has, in comparison to the RAM program. Recall that Word-RAM programs have a finite but changing memory size S; you may want to start your simulation by initializing S.
   - Operations: For each operation in the Word-RAM program P, how will the corresponding operation in the RAM program R be performed? Keep in mind that the RAM program "may not by default know" of nuances of the arithmetic in Word-RAM.

[1] This problem won't make a difference between N, L, R-, and R grades.

- Memory: How will the read and write into the memory locations take place? Keep in mind that the RAM program "may not by default know" of nuances of memory reads or writes in the Word-RAM.
- Output: How will the output of the Word-RAM program P be simulated in R? Again, keep the differences between Word-RAM and RAM program in mind.
- Runtime: Justify how your simulation maintains the runtime stated above.
- Bitlength: justify how your simulation maintains the bitlength stated above.
- Don't forget MALLOC and updates to the GOTO command!

4. (reflection) In what ways do LLMs impact your learning outcomes for this course? If you use LLMs in learning course material, give some example(s) of how they helped you learn something that lectures/sections etc did not. If you do not use LLMs, discuss reasons for why you do not use them. All opinions are valid.

   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 psets, 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!