CS120: Intro. to Algorithms and their Limitations Hesterberg & Vadhan

Problem Set 3

Harvard SEAS - Fall 2022 *Due: Wed Sept. 28, 2022 (11:59pm)*

Your name:
Collaborators:
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 RAM Model simulator. Your task is to fill in the missing parts of the code to obtain a complete 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:

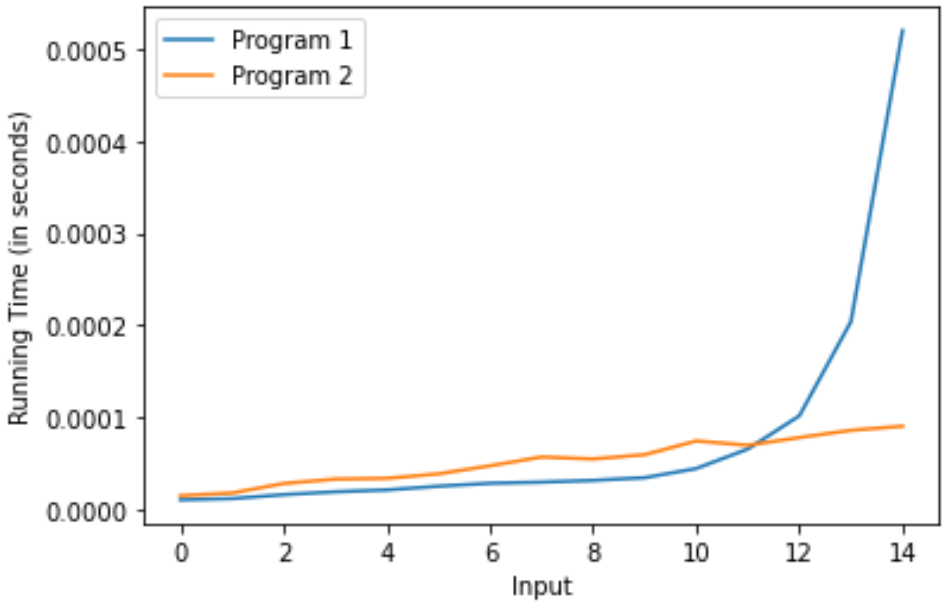
```
   Input     : A single natural number N (as an array of length 1)
   Output    : 11^(2^N+1) (as an array of length 1)
   Variables: input_len, output_ptr, output_len, counter, result
 0 zero = 0;
 1 one = 1;
 2 eleven = 11;
 3 output_len = 1;
 4 output_ptr = 0;
 5 result = 11;
 6 counter = M[zero];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

```
   Input     : A single natural number N (as an array of length 1)
   Output    : 11^(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 eleven = 11;
 3 output_len = 1;
 4 output_ptr = 0;
 5 result = 11;
 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 * eleven;
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?

Solution. Here is the plot.



It looks like the first program is slightly faster until $N = 12$, at which point it becomes dramatically slower than the second program. □

(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 (using asymptotic estimates) of the shapes of the runtime curves you see in Part 2b. 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) Show that for every Word-RAM program P, there is a RAM program P' such that P' halts on x iff P halts on x, and if they halt, then $P'(x) = P(x)$ and

$$\text{Time}_{P'}(x) = O\left(\text{Time}_P(x) + n + w_0\right),$$

where n is the length of x and w_0 is the initial word size of P on input x. (This was stated without proof in Lecture Notes 7.)

Your proof should use an *implementation-level* description, similar to our proof that RAM programs can be simulated by ones with at most c registers. Recall that Word-RAM programs have read-only variables `mem_size` and `word_len`; you may want to start your simulation by calculating these variables as well as another variable representing $2^{\texttt{word\_len}}$. Then think

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

about how each operation of a Word-RAM program P can be simulated in a RAM program P', taking care of any differences between their semantics in the Word-RAM model vs. the RAM model. Don't forget MALLOC!