CSE 20 Homework 1
Due: Friday October 6, 11pm
via Gradescope

Topics Algorithms and Integer representations

Reading

- Rosen Section 3.1: pages 191-194 (up to but not including Searching Algorithms) and pages 198-201 (except proofs; we'll come back to the proofs in future weeks), Appendix 3 pages A11-A16;
- Section 4.1: pages 237-240 (up to but not including Modular Arithmetic);
- Section 4.2: pages 245-253 (up to but not including Modular Exponentiation).

Key Concepts Algorithm, input of algorithm, output of algorithm, tracing pseudocode given input, higher-level function of an algorithm, definiteness of algorithms, finiteness (termination) of algorithms, correctness of algorithms, optimization problem, greedy approach. Base expansion, decimal expansion, binary expansion, hexadecimal expansion, octal expansion, bit shift, DIV, MOD, integer algorithm for multiplication.

---

1. (**12 points**) Consider the following algorithm:

```
procedure alg1($a, b$: positive integers)
  $x := a$
  $y := b$
  while $y \neq 0$
    $y := y - 1$
    $x := x + a$
  return $x$
```

```
procedure alg2($a, b$: positive integers)
  $x := a$
  $y := b$
  while $y \neq 0$
    $r := x$ mod $y$
    $x := y$
    $y := r$
  return $x$
```

Before you start: trace each of these algorithms with a few possible inputs.
Work to understand what function each implements. You do not need to hand this work in.

(a) You are told that algorithm *alg*1 is finite. *You do not need to prove this claim; but it's good practice to think about how you would.* **Define** pseudocode for algorithm *alg*3 that is identical to *alg*1 except for **one line** and which describes an algorithm that **is not** finite. **Prove** that your example works by specifying sample input where the algorithm never returns output because it goes into an infinite loop *trace the algorithm on this input to justify your answer.*

(b) Is algorithm *alg*2 finite? That is, is its computation guaranteed to terminate no matter which positive integers are chosen for a and b? *Note: for full credit, give your answer (yes or no) and justify it. If your answer is yes, the justification should explain why the algorithm can't possibly go into an infinite loop no matter what inputs are chosen (think carefully about the type of the inputs and the loop condition). If your answer is no, you need to find an example where the algorithm never returns output because it goes into an infinite loop.)*

2. (**6 points**) Show that if there were a coin worth 17 cents, the greedy algorithm using quarters, 17-cent coins, dimes, nickels, and pennies would **not always** produce change using the fewest coins possible.

Note: for full credit, you need to find an example where the greedy algorithm does not produce change using the fewest coins possible, and then describe why this example works: trace the greedy algorithm to explain how many coins it would produce in change? what's a better way to make change using fewer coins?

Second note: for worked examples related to this question, refer to Week 1 discussion section worksheet.

3. (**9 points**) In computations over multiple processors, the load of a processor is the sum of the times of all loads assigned to that processor. To balance the computation, the goal is to minimize the maximum load across the processors. For example, if the loads of three processors are $3, 10, 2$ then the maximum load is 10. This assignment is less balanced (not as good) as one where the loads are $5, 5, 5$ even though both assignments handle the same total load (15).

For this question, assume that the processors are labelled P_1, P_2, P_3. The input to the **load balancing problem** is a list of jobs, each labelled by the time it would take to complete that job. The output of the problem is an assignment of a processor to each job.

The greedy algorithm for load balancing goes through the list of jobs in the order given and assigns each job to the processor with smallest current load (if two processors have the same load, the algorithm chooses the one with smaller index; at the start of the algorithm, all processors have zero load).

(a) If the list of jobs is $j_1 = 4, j_2 = 6, j_3 = 7, j_4 = 8, j_5 = 5, j_6 = 2$, what jobs are assigned to each processor by the greedy algorithm?

P_1 :

P_2 :

P_3 :

(b) Is the greedy algorithm correct for this problem? Explain your answer.

4. (**18 points**) In class, we discussed the Russian Peasant Multiplication (RPM) algorithm. The pseudocode implementation of this algorithm is:

```
procedure $RPM(m:$ real number; $n:$ positive integer)
1. $total := 0$
2. $a := m$
3. $b := n$
4. while $b > 1$
5.     if ($b$ mod $2 = 1$) then $total := total + a$
6.     $a := 2 \cdot a$
7.     $b := b$ div $2$
8. return total + a
```

(a) Trace (i.e. walk through) an example of the operation of this algorithm when $m = 714$ and $n = 64$.

Note: for full credit, Include enough work to trace the execution of the algorithm on these inputs. Clearly label the **output** *of the algorithm in your answer.)*

(b) Convert $m = 714$ and $n = 64$ to binary notation.

Note: include the calculations you use in support of your answer. You need not write an explanation of the conversion for this question.

(c) Multiply $m = 714$ and $n = 64$ using the usual school algorithm for long multiplication (but in base 2 instead of decimal).

Note: include the calculations you use in support of your answer.

(d) Convert your answer from part (c) back to decimal, and compare to part (a).

Note: include the calculations you use in support of your answer. You need not write an explanation of the conversion for this question.

(e) Explain how the steps in parts (a) and parts (c) correspond to one another.

For full credit, the explanation must use grammatically correct, complete sentences but may be short. Make specific reference to RPM (either its English-language description from class or the pseudocode above) and to the usual-school-algorithm for long multiplication.

(f) Trace the RPM algorithm but now with $m = 64$ and $n = 714$. Discuss the difference between this computation and part (a). Generalize your observation to come up with a general principle for choosing which number to pick as m and which number to pick as n when you want to combine the product of two positive integers x and y.