CSC165H1: Problem Set 4 – UPDATED March 16

Due March 22, 2017 before 10pm

General instructions

Please read the following instructions carefully before starting the problem set. They contain important information about general problem set expectations, problem set submission instructions, and reminders of course policies.

- Your problem sets are graded on both correctness and clarity of communication. Solutions which are technically correct but poorly written will not receive full marks. Please read over your solutions carefully before submitting them.
- Each problem set may be completed in groups of up to three. If you are working in a group for this problem set, please consult `https://github.com/MarkUsProject/Markus/wiki/Student_Groups` for a brief explanation of how to create a group on MarkUs.

  Exception: Problem Set 0 must be completed individually.
- Solutions must be typeset electronically, and submitted as a PDF with the correct filename. **Handwritten submissions will receive a grade of ZERO.**

  The required filename for this problem set is **problem_set4 – UPDATED March 16.pdf**.
- Problem sets must be submitted online through MarkUs. If you haven't used MarkUs before, give yourself plenty of time to figure it out, and ask for help if you need it! If you are working with a partner, you must form a group on MarkUs, and make one submission per group. "I didn't know how to use MarkUs" is not a valid excuse for submitting late work.
- Your submitted file should not be larger than 9MB. This may happen if you are using a word processing software like Microsoft Word; if it does, you should look into PDF compression tools to make your PDF smaller, although please make sure that your PDF is still legible before submitting!
- Submissions must be made *before* the due date on MarkUs. You may use *grace tokens* to extend the deadline; please see the Problem Set page for details on using grace tokens.
- The work you submit must be that of your group; you may not refer to or copy from the work of other groups, or external sources like websites or textbooks. You may, however, refer to any text from the Course Notes (or posted lecture notes), except when explicitly asked not to.

Additional instructions

- All final Big-Oh, Omega, and Theta expressions should be fully simplified according to three rules: don't include constant factors (so $\mathcal{O}(n)$, not $\mathcal{O}(3n)$), don't include slower-growing terms (so $\mathcal{O}(n^2)$, not $\mathcal{O}(n^2 + n)$), and don't include floor or ceiling functions.
- You may use common algebraic properties of logarithms (change of base, log of a product, etc.), and the fact that for all $x, y \in \mathbb{R}^+$, $x > y \Leftrightarrow \log x > \log y$.
- For questions in which you're analysing algorithms, you may use (without proving them) all of the theorems given in the **Properties of Big-Oh, Omega, and Theta** handout, as long as you clearly state which theorems you are using.

1. **[8 marks] Little-Oh.** Recall the definition of Big-Oh:

$$\exists c \in \mathbb{R}^+,\ \exists n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow g(n) \leq cf(n)$$

Here is one variation of this definition.

Definition 1 (little-oh)**.** Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. We say that g is **little-oh** of f, and write $g \in o(f)$, when:

$$\forall c \in \mathbb{R}^+,\ \exists n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow g(n) \leq cf(n)$$

This is a *stronger* property than Big-Oh, in the sense that if $g \in o(f)$, then $g \in \mathcal{O}(f)$.[1] While $g \in \mathcal{O}(f)$ says (colloquially) that "it is possible to scale up f so that it eventually dominates g," $g \in o(f)$ says that "no matter how you scale f (up or down), it will always eventually dominate g." Or, in terms of rates of growth, $g \in \mathcal{O}(f)$ means that g grows at most as quickly as f, while $g \in o(f)$ means that g grows strictly slower than f.

Prove the following statements about little-oh, using only the definitions of little-oh and Big-Oh. You may not use any external properties of Big-Oh in this question.

(a) Prove that for all positive real numbers a and b, if $a < b$ then $n^a \in o(n^b)$.

(b) Prove that for all functions $f, g : \mathbb{N} \to \mathbb{R}^+$, if $g \in o(f)$ then $f \notin O(g)$.
Note: we've restricted the range of f and g to exclude 0.

2. **[12 marks] A tricky nested loop.** Our goal for this question is to analyse the running time of the following function:

```
1 def nested(n):
2     """Assume n is an integer that is greater than 1."""
3     b = 1
4     while b <= n:          # Loop 1
5         i = 1
6         while i < b:       # Loop 2
7             print(i)
8             i = i * 3
9         b = b + 1
```

(a) First, prove that for all functions $f, g : \mathbb{N} \to \mathbb{R}^+$ and $b \in \mathbb{R}^+$, if all three of the following conditions are true:

(i) $g(n) \in \mathcal{O}(f(n))$

(ii) $f(n)$ and $g(n)$ are eventually $\geq b$ [**Correction**: this used to say ≥ 1]

(iii) $b > 1$

then $\log_b(g(n)) \in \mathcal{O}(\log_b f(n))$.

You may *not* use any external properties about Big-Oh; we're looking for you to prove this statement using the definition of Big-Oh.

Note: review the Extra Instructions for hints about working with logarithms.

[1] The only difference between these two definitions is in the quantification of c, with the switch to universal quantification leading to a more powerful claim. In general, if the statement $\forall c \in \mathbb{R}^+,\ P(c)$ is true, then $\exists c \in \mathbb{R}^+,\ P(c)$ is also true.

(b) Find an **exact** expression for the *total number of iterations* of the inner loop of `nested`, in terms of the function's input n. Your expression may contain summation notation (e.g., $\sum_{i=1}^{n} i$); you do not need to simplify it here. Make sure to use the floor and/or ceiling functions to ensure that you're counting with natural numbers.

(c) Determine, with proof, a good *upper bound* (Big-Oh only) on the running time of `nested`, in terms of the value of its input n. Note that by "good" we mean that it should be possible to prove the matching Omega lower bound, even though we are not requiring it here.

You can use the statement you proved in part (a), and your solution to part (b), and the following external facts:[2]

$$\forall x \in \mathbb{R},\ x \leq \lceil x \rceil < x + 1 \qquad \text{(Fact 1)}$$

$$n! \in \Theta(e^{n \ln n - n + \frac{1}{2} \ln n}) \qquad \text{(Fact 2)}$$

the exponent of e in Fact 2 is eventually ≥ 1 (Fact 3)

3. **[10 marks] Algorithm analysis.** Consider the following function, which takes in a list of integers.

```
1  def my_sum(L):
2      x = 0
3      y = 1
4      n = len(L)
5      i = 0
6      while x + y < n:
7          if L[i] % 2 == 0:
8              y = y + 1
9          else:
10             x = x + y
11         i = i + 1
```

Let $WC(n)$ and $BC(n)$ represent the worst-case and best-case runtime functions of `my_sum`, respectively.

(a) Prove that $WC(n) \in \mathcal{O}(n)$, where n is the length of the input list to `my_sum`.

(b) Prove that $WC(n) \in \Omega(n)$, where n is the length of the input list to `my_sum`.

(c) Prove that $BC(n) \notin \Theta(n)$, where n is the length of the input list to `my_sum`. You can use the definition of Theta that says $g \in \Theta(f)$ if and only if $g \in \mathcal{O}(f) \wedge g \in \Omega(f)$.

[2] Some reminders of notation: $n!$ is the *factorial function* defined as $n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (n-1) \cdot n$. $e \approx 2.71828\ldots$ is Euler's number; the natural logarithm is the function $\ln n = \log_e n$.