CSC165H1: Problem Set 4

Due Friday March 27, 2020 before 4pm

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 that 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.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(s) should not be larger than 9MB. You might exceed this limit if you use a word processor like Microsoft Word to create a PDF; 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 Homework page for details on using grace tokens.
- The work you submit must be that of your group; you may not use or copy from the work of other groups, or external sources like websites or textbooks.

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 (so $\mathcal{O}(\log n)$, not $\mathcal{O}(\lceil \log n \rceil)$).
- For algorithm analysis questions, you can jump immediately from a closed-form step count expression to an asymptotic bound without proof (e.g., write "the number of steps is $3n+\log n$, which is $\Theta(n)$"). This applies to upper and lower bounds (Big-Oh and Omega) as well.
- However, you must evaluate all summations *before* jumping to an asymptotic bound.

- Unless specified otherwise in the question, you should use floor/ceiling to ensure you are counting steps exactly as natural numbers.
- Unless specified otherwise in the question, you should count the cost of all lines of code in an algorithm, including constant-time steps.

1. **[8 marks] Analyzing nested loops.** Our goal for this question is to analyse the running time of the following function:

```
1 def print_threes(n: int) -> None:
2     """Precondition: n > 1"""
3     for i in range(1, n + 1):  # Loop 1
4         j = 1
5         while j < i:           # Loop 2
6             print(i)
7             j = j * 3
```

   (a) **[Do not hand in—this question part is not graded.]** 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) $g(n) \in \Theta(f(n))$

   (ii) $f(n)$ and $g(n)$ are eventually $\geq b$ (refer to Problem Set 1 for the definition of *eventually*)

   (iii) $b > 1$

   then $\log_b(g(n)) \in \Theta(\log_b(f(n)))$.

   (b) Find, with proof, an exact expression for the running time of `print_threes` in terms of its input n, assuming $n > 1$. Your final expression may contain summation notation (e.g., $\sum_{i=1}^{n} i$); do **not** simplify to an asymptotic (Big-Oh/Omega/Theta) expression here.

   To simplify your analysis, ignore the cost of Line 4 (`j = 1`).

   (c) Analyze the running time of `print_threes` in terms of its input n, concluding with a Theta bound on the running time.

   You may use the statement from part (a) and your answer to part (b), and the following external facts:[1]

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

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

$$\text{the exponent of } e \text{ in Fact 2 is eventually} \geq 1 \qquad \text{(Fact 3)}$$

[1] 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 $\ln n = \log_e n$.

2. **[9 marks] Odds and evens.**

Consider the following Python function:

```
1  def longest_even_prefix(nums: List[int]) -> int:
2      """Return the length of the longest prefix of nums that contains only even numbers.
3
4      (A prefix of length 0 is considered to contain only even numbers.)
5      """
6      n = len(nums)
7      for i in range(n, -1, -1):        # Loop 1: i = n, n-1, ..., 0
8          found_odd = False
9          for j in range(i):            # Loop 2: j = 0, 1, ..., i - 1
10             if nums[j] % 2 == 1:
11                 found_odd = True
12                 break                 # Breaks out of Loop 2 (goes to Line 14).
13
14         if not found_odd:
15             return i
```

You may find the following formula helpful (valid for all $n \in \mathbb{N}$):

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

(a) Find, with proof, a *tight asymptotic upper bound* (Big-Oh expression) on the worst-case running time of `longest_even_prefix`.[2]

(b) Prove an *aymptotic lower bound* (Omega expression) on the worst-case running time of this algorithm that matches the upper bound you proved in part (a).

(c) Prove that for all $n \in \mathbb{N}$ and every list `nums` of integers of length n, the running time of `longest_even_prefix(nums)` is $\Omega(n)$.[3]

HINT: do a proof by cases based on which elements of `nums` are even/odd.

[2] Remember that *tight* means that it should be possible to prove a matching asymptotic lower bound (Omega) on the worst-case running time, but you shouldn't do that in this part.

[3] If you compare this to the definition of upper bound on the worst-case running time of a function, you might notice that what we're asking you to prove here is a lower bound on the *best-case* (i.e., minimum) running time of `longest_even_prefix`.

3. **[10 marks] Unpredictable loop variables.**

Consider the following function:

```
1  def func(lst: List[int]) -> None:
2      n = len(lst)
3      i = 0
4      j = 1
5      while i < n:
6          if lst[i] >= 0:
7              i = i + j
8          else:
9              lst[i] = abs(lst[i])
10             i = 0
11             j = j * 2
```

(a) Find, with proof, an input family for `func` whose running time is $\Theta(\log n)$.

(b) Find, with proof, an input family for `func` whose running time is $\Theta(n)$ *and* for which the `else` branch (Lines 9–11) executes $\Omega(\log n)$ times.

To simplify your analysis, you may assume n (the length of the list) is a power of 2.[4]

You may use the following formula, valid for all $n \in \mathbb{N}$ and $r \in \mathbb{R}$ where $r \neq 1$:

$$\sum_{i=0}^{n-1} r^i = \frac{1 - r^n}{1 - r}$$

(c) Prove that the worst-case running time for `func` is $\mathcal{O}(n)$.

[4] Your idea will likely generalize for non-powers of 2, but this makes the analysis messier.

4. **[11 marks] An average-case analysis.**

Consider the following algorithm.

```
1  def convert_to_binary(n: int) -> str:
2      """Return the binary representation of n with no leading zeros (as a string).
3
4      Precondition: n > 0.
5      """
6      s = ''
7      x = n
8      while x > 0:
9          r = x % 2
10         x = x // 2          # Same as floor(x/2)
11         s = str(r) + s      # Note: treat + with strings as a constant-time operation
12     return s
```

(a) First, for all $k \in \mathbb{N}$, we let x_k denote the value of variable `x` in this algorithm after k loop iterations, or 0 if fewer than k iterations occur. Note that this depends on the input n. For example, when $n = 9$, we have the sequence $x_0 = 9$, $x_1 = 4$, $x_2 = 2$, $x_3 = 1$, and $x_4 = x_5 = x_6 = \cdots = 0$.

Prove by induction that $\forall n \in \mathbb{Z}^+,\ \forall k \in \mathbb{N},\ \frac{n}{2^k} - \frac{2^k - 1}{2^k} \leq x_k \leq \frac{n}{2^k}$.

You may use the fact that $\forall x \in \mathbb{Z},\ \frac{x-1}{2} \leq \left\lfloor \frac{x}{2} \right\rfloor \leq \frac{x}{2}$.

HINT: you should do induction on k, not n, in the above formula. Introduce n first just as an arbitrary positive integer.

(b) Using part (a), fill in the blanks of the statement below, and prove the resulting statement.

$$\forall n \in \mathbb{Z}^+,\ \forall k \in \mathbb{N},\ (\texttt{convert\_to\_binary(n)} \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow \_\_\_\_\_ \leq n \leq \_\_\_\_\_$$

(c) Now we define the following inputs for this function: for every $n \in \mathbb{Z}^+$, let $\mathcal{I}_n = \{1, \ldots, 2^n - 1\}$.

Using parts (a) and/or (b), find an exact, closed-form expression for the average running time of `convert_to_binary` on $\mathcal{I}_n$ (in terms of n). Do not convert to a Theta expression—we're looking for the exact average number of steps here.

To simplify your solution, you may ignore the costs of lines 6, 7, and 12 in your calculation, and treat the loop body as a single step.

You may use the following formula, valid for all $m \in \mathbb{N}$ and $r \in \mathbb{R}$ where $r \neq 1$:

$$\sum_{i=1}^{m} i r^{i-1} = \frac{1 - r^{m+1}}{(1-r)^2} - \frac{(m+1)r^m}{1-r}$$