CSC165H1: Problem Set 4

Due Thursday March 28, 2019 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.
- For algorithm analysis questions, you can jump immediately from a step count 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.

1. **[9 marks] Printing multiples.** Consider the following algorithm:

```
1  def print_multiples(n: int) -> None:
2      """Precondition: n > 0."""
3      for d in range(1, n + 1):            # Loop 1
4          for multiple in range(0, n, d):  # Loop 2 (also, see Python Note below)
5              print(multiple)
```

PYTHON NOTE: `range(0, n, d)` consists of the multiples of $d \geq 0$ and $< n$: $0, d, \ldots, d(\lceil n/d \rceil - 1)$.

(a) Find, with proof, a **Theta** bound for $\sum_{i=1}^{n} \left\lceil \frac{n}{i} \right\rceil$ in terms of elementary functions (e.g., n, $\log n$, 2^n, etc.). You may not use any properties of Big-Oh/Omega/Theta; instead, only use their definitions, as well as the following Facts (clearly state where you use them):

$$\sum_{i=1}^{n} \frac{1}{i} \in \Theta(\log n) \qquad \text{(Fact 1)}$$

$$\forall x \in \mathbb{R},\ x \leq \lceil x \rceil < x + 1 \qquad \text{(Fact 2)}$$

(b) Using your answer to part (a), analyse the running time of `print_multiples` to determine a Theta bound on the running time.

(c) Now consider the following variation on the above algorithm.

```
1  def print_multiples2(n: int) -> None:
2      """Precondition: n > 0."""
3      for d in range(1, n + 1):              # Loop 1
4          for multiple in range(0, n, d):    # Loop 2
5              print(multiple)
6
7          if d % 5 == 0:
8            for i in range(d):               # Loop 3
9                print(i)
```

Analyse the running time of `print_multiples2` to determine a Theta bound on the running time. To simplify your analysis, you can focus on just counting the steps taken by Lines 5 and 9.

You may use your work from previous parts of this question, and the summation formula $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

2. **[9 marks] Varying running times, input families, and worst-case analysis.**

```
1  def alg(lst: List[int]) -> None:
2      n = len(lst)
3      i = 1
4      j = 1
5      while i < n:                  # Loop 1
6          if lst[i] % 2 == 0:
7              i = i + 1
8              j = j * 2
9          else:
10             i = i * 2
11             for k in range(j):    # Loop 2
12                 print(k)
```

To simplify your analyses for this question, ignore the costs of Lines 2–4 and Line 10. (Of course, you'll have to understand what these lines do to analyse the running time, but you don't need to count them as "steps.")

(a) Find, with proof, an input family for `alg` whose running time is $\Theta(2^n)$.

(b) Find, with proof, an input family for `alg` whose running time is $\Theta(\log n \times 2^{\sqrt{n}})$, where the `else` branch of Loop 1 runs $\Theta(\log n)$ times.

You can use the fact that $\log n - \log(\log n) \in \Theta(\log n)$ in your proof.

(c) Prove that the worst-case running time of `alg` is $\mathcal{O}(2^n)$ (note that this matches the worst-case lower bound we can derive from part (a)).

HINT: unlike other examples we've looked at in class, there are no "early returns" here. Instead, use what you've learned about analysing loops with non-standard loop variable changes, and remember that you don't need *exactly* 2^n steps, just an *at most* $\mathcal{O}(2^n)$.

3. **[8 marks] Rearrangements, best-case analysis.**

We define the **best-case running time** of an algorithm `func` as follows:

$$BC_{func}(n) = \min\{\text{running time of executing } \texttt{func(x)} \mid x \in \mathcal{I}_n\}$$

Note that this is analogous to the definition of worst-case running time, except we use min instead of max.

(a) Review the definitions of what it means for a function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ to be an upper bound or lower bound on the worst-case running time of an algorithm (Definitions 5.11 and 5.12 in the Course Notes).

Then, let $\mathcal{I}_n$ denote the set of all inputs of size n for `rearrange`, and write two symbolic definitions:

(i) What it means for a function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ to be an *upper bound* on the best-case running time of an algorithm.

(ii) What it means for a function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ to be a *lower bound* on the best-case running time of an algorithm.

Clearly state which definition is which in your answer. Your definitions should be very similar to the definitions we mentioned above from the Course Notes.

HINT: it is easier to first translate the simpler statements "M is an upper bound on the minimum of set S" and "M is a lower bound on the minimum of set S" to make sure you have the right idea.

(b) Consider the following Python function.

```
1  def rearrange(lst: List[int]) -> None:
2      for i in range(2, len(lst)):                    # Loop 1
3          if i % 2 == 0:
4              j = i - 2
5              while j >= 0 and lst[j+2] < lst[j]:     # Loop 2
6                  lst[j+2], lst[j] = lst[j], lst[j+2]  # Swap lst[j] and lst[j+2]
7                  j = j - 2
8          else:
9              j = i - 2
10             while j >= 0 and lst[j+2] > lst[j]:     # Loop 3
11                 lst[j+2], lst[j] = lst[j], lst[j+2]  # Swap lst[j] and lst[j+2]
12                 j = j - 2
```

Analyse the best-case running time of `rearrange` to find a Theta bound for it. Your analysis should consist of **two** parts, proving matching upper and lower bounds on the best-case running time separately, and then using them to conclude a Theta bound.

To simplify your analysis, you only need to count the total number of steps taken inside Loops 2 and 3, and not the other operations (e.g., Lines 4 and 9).

4. **[0 marks] An average-case analysis.**

Updated March 21: because of the lecture scheduling change on March 18, we have decided not to grade this question, and instead will only be counting the first three questions on this problem set. We encourage you to complete this question as extra practice and preparation for the final exam, but please do not include it in your problem set submission.

~~**Note**: we'll introduce this type of analysis with an example similar to this problem on Monday, March 18. We recommend prioritizing the other problems on this problem set until then.~~

Consider the following algorithm.

```
1 def find_one_or_two(lst: List[int]) -> Optional[int]:
2     """Return the index of the first 1 or 2 in lst, or None if neither of them appear."""
3     for i in range(len(lst)):
4         if lst[i] == 1 or lst[i] == 2:
5             return i
6 
7     return None
```

We consider the following inputs for this function: for every $n \in \mathbb{N}$, we define the set $\mathcal{I}_n$ to be the set of lists of length n that are permutations of the numbers $\{1, 2, \ldots, n\}$. We know that $|\mathcal{I}_n| = n!$ (where $n! = n \times (n-1) \times \cdots \times 2 \times 1$).

Note that if $n \geq 1$, then every list in $\mathcal{I}_n$ contains a 1 or 2, and so the algorithm `find_one_or_two` will always return an index.

For this question, you can use your answer for each part in all subsequent parts.

(a) Let $n \in \mathbb{N}$, and assume $n \geq 2$. Let $i, j \in \mathbb{N}$, and assume $i \neq j$, and that both are between 0 and $n-1$, inclusive.

Find a formula, in terms of n, i, and/or j, for the exact number of lists `lst` in $\mathcal{I}_n$ where `lst`$[i]$ equals 1 and `lst`$[j]$ equals 2. Prove that your formula is correct.

(b) Let $n \in \mathbb{N}$, and assume $n \geq 2$. Let $i \in \mathbb{N}$, and assume i is between 0 and $n-1$, inclusive.

Find a formula, in terms of n and/or i, for the exact number of lists `lst` in $\mathcal{I}_n$ where `lst`$[i]$ equals 1, and 2 appears in `lst` at an index *greater* than i.

(c) Let $n \in \mathbb{N}$, and assume $n \geq 2$. Find an exact expression for the average running time of `find_one_or_two` on $\mathcal{I}_n$.

Count each iteration of the loop as just 1 step, and include the last partial iteration (the one that executes `return i`) in your count. Note that for this set of inputs, the loop will always return early, so the final `return None` will never execute.

You may use the following formulas in your calculation, valid for all $m \in \mathbb{N}$:

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

$$\sum_{i=1}^{m} i^2 = \frac{m(m+1)(2m+1)}{6}$$

HINT: you can use your work from part (b), but make sure in your analysis here to consider the cases where 2 appears before 1 in the list.