Analysis of Algorithms
CS 375, Fall 2022
Problem Set 1
Due **AT THE BEGINNING OF CLASS** Wednesday, October 5

- For this assignment, standard file naming conventions apply: Please submit your typewritten answers in a PDF file named `CS375_PS1_<userid>.pdf` where `<userid>` is replaced by your full Colby userid, and submit it to your `SubmittedWork` folder. Please reach out to me right away with any questions or concerns about this!

- From your textbook (CLRS), please read: Chapters 2.1, 2.2, and 3. **NOTE:** Some material there is more mathematically intense than we need—don't worry about those parts!—but please do read over the parts that give basic ideas behind content covered in lecture (e.g., O, Ω, Θ notation, functions' orders of growth, loop invariants).

- Ex. 3 below mentions *brute force* algorithms; we'll talk more about them on Sept. 28.

- A few essential, general style guidelines for writing algorithms in CS375:

  – Unless explicitly instructed otherwise, avoid using `break` statements (or `continue` or `pass`) statements in algorithms for CS375—as we will discuss in class, employing them can interfere with common approaches to reasoning about code correctness (e.g., using loop invariants). Instead, please find other approaches, perhaps using boolean variables to indicate loop termination conditions.

    Answers employing `break` (or `continue`, `pass`, ...) may not receive full credit. Please feel free to talk with me about any particular situations that might come up—I'm happy to talk with you about how to avoid `break`s in your code!

  – When presenting an algorithm, avoid using language-specific commands/routines (without comments) in pseudocode. Pseudocode, by definition, is supposed to be language-independent, higher-level than particular programming languages.

  – Please be sure to initialize variables or declare inputs/ouptuts, and be sure the purpose of **every variable**—including the names of functions or algorithms—can be quickly and fully understood by a reader; typically, this is done either by using descriptive variable names or by describing variables in English. Algorithms in which variables cannot be promptly understood may not receive full credit, so please feel free to ask me about variable names and descriptions—I'm happy to talk with you about particular situations that might come up!

  – Often, the clearest way to present an algorithm is with both pseudocode and English description. Pseudocode must be accompanied by some kind of clarifying English description to count as fully explained.

- *A general note for CS375:* When writing up your homework, please present your answers neatly and **explain your answers clearly**, giving all details needed to make your answers easy to understand. Graders may not award full credit to incomplete or hard to understand solutions. Clear communication *is* the point, on every assignment.

  In general in CS375, unless explicitly specified otherwise, answers should be accompanied by explanations. Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Exercises

1. Prof. E. Nigma's colleagues in the Department of Insufficient Specification at the Portland Institute of Technology (which continues to not exist!) were working on the following pseudocode algorithm, which takes an integer `n` as input:

```
DIS-algo(n):
===========
    r = 0
    i = n * n
    while i > 1 do
      for j = 1 to i do
        r = r + 1
      i = i/2
    return r
```

In the above, `i/2` is integer division, i.e., it returns an integer, truncating any decimal part of the result of the division——so, as one concrete example (not necessarily one that comes up in this exercise!), `5/2` would return 2, not 2.5.

(a) What is the final value of `r` returned by this function? Express it as a function of `n`.

(b) Express the running time (as a function of `n`) of `DIS-algo` using Θ-notation.

(c) Express the space complexity (as a function of `n`) of `DIS-algo` using Θ-notation.

As usual, be sure to give concise but convincing explanations for your answers.

NOTE: Here are some hints, which might be especially helpful for part 1a of this exercise:

- You are welcome to assume that the input value `n` is a power of 2, if it simplifies your analysis.
- Using summation notation may help.
- There's a common "trick" in algorithm analysis involving summation—note that for some exponent x, $\frac{1}{2^x} = (\frac{1}{2})^x$. Here's a quick derivation of that:

$$\begin{aligned} \frac{1}{2^x} &= \frac{1^x}{2^x} && \text{because } 1 = 1^x \text{ for any } x \\ &= (\frac{1}{2})^x && \text{by properties of exponents} \end{aligned}$$

  This "trick" might be useful in your analysis of this algorithm.

- Please be careful of off-by-one errors!

2. Prof. Sue Persmart in the CS Department at Portland Institute of Technology likes to tell a story about the invention of chess.

   (a) According to legend, the game of chess we invented long ago in India by a certain sage. When he took the invention to his king, the king liked the game so much that he offered the inventor any reward he wanted. The inventor asked for some grain to be obtained as follows: Just one grain of wheat was to be placed on the first square of the chessboard, then two grains on the second square, four grains on the third square, eight grains on the fourth, etc., until all 64 squares had been filled.

   If it took 1 second to count each grain of wheat, how long (in seconds) would it take to count all the grains of wheat due to the sage?

   (b) What if, instead of doubling the number of grains for each square of the chessboard, the inventor asked for adding two grains. Then (assuming again that it took 1 second to count each grain) how long (in seconds) would it take to count all the grains of wheat due to the sage?

3. The Department of Redundancy Department at the Portland Institute of Technology (which continues to not exist—I know I said that previously in this problem set, but this is the Department of *Redundancy* Department, after all . . .) has asked you for help!

   They work with arrays $A[0..n]$ of integers, in which every integer is between 1 and n; there's no constraint on the ordering of numbers in the array. In addition, they happen to know that all of the integers between 1 and n are necessarily in each array. And that means that exactly one number is repeated in each array. (Do you see why?)

   They want you to come up with an algorithm that, for any such input array, outputs the number that's repeated in the array. You'll come up with several such algorithms—but no two will be the same!

   (a) First, give a $\Theta(n^2)$-time brute force algorithm to solve this problem.

   (b) Then, give a $\Theta(n \lg n)$-time algorithm to solve this problem.

   (c) Then, give a $\Theta(n)$-time algorithm to solve this problem that has $O(n)$ space complexity.

   (d) Finally, give a $\Theta(n)$-time, $O(1)$-space complexity algorithm to solve this problem.

   As is conventional, we're asking about worst case time complexity unless otherwise specified.

   To earn full credit, be sure to include all of the following for each of the four algorithms:

   - Pseudocode, accompanied by a brief English explanation of what the algorithm does
   - A concise but convincing explanation of correctness
   - A concise but convincing explanation of its time complexity
   - A concise but convincing explanation of its space complexity