CSE374 – Spring 2021 – Assignment 3

This assignment is done individually. You can search for information online, use your textbook, ask the instructor or Teaching Assistant for help. Working with anyone else is a non-authorized collaboration, that is, an act of academic dishonesty. If you have any doubt or question about what qualifies as academic dishonesty, please contact the instructor to make an informed decision.

Objective: Reflect on your strategy for success and practice theoretical algorithm analysis.
Due date: Tuesday March 9th at 11:59pm. Assignments do not receive a grade after midnight.
How to submit: A ZIP file, named "*Firstname_Lastname*_Assignment3.zip"
Grading (9 pts): Each box shows the number of points in a question.

I. Overview

You have practiced empirical evaluations in your previous assignment. The current one thus complements your practice by focusing on theoretical evaluations. It also gives you an incentive to take the time to reflect on what worked for you and how you may improve.

0.5 pt each

II. Reflection

You've now had several weeks of the course, so you have gotten feedback on your performance and you are familiar with the expectations as well as the format. This is a good time to reflect on what worked for you and what may need some improvement going forward. To help you contextualize the quantitative feedback (=grades), the grade for each activity in the class is provided on the next page.

1) *What has worked.* What are you currently doing that is helping you to prepare for this course? Please detail at least two activities, over at least one paragraph.
2) *What needs to be addressed*. What do you see as the main challenges in this course? Please detail at least two challenges, over at least one paragraph. In case you don't see any challenge, think of what may be most challenging for others.
3) *How you may address your challenges*. For each of the challenges listed in #2, explain how you plan to address it. You may identify resources that could be helpful but that you haven't used yet.
4) *Class design*. Although this is the 4th semester in a row that the class is taught, there are always a chance to learn from your feedback and adjust for next semester or, when it is feasible, changes to the current term. Think of the ways in which the class is taught and then identify what you find most helpful and/or what could change. Note that questions 1-3 are about yourself but this question is about the class design, so it affects everyone.

1 pt each

III. Tricky little loopses

A common assumption when doing a Θ analysis is that you can just look at loops. One may assume that a single loop would be Θ(n), a nested loop Θ(n^2) and so for. However, there is *much more to it* than that: we need to precisely understand what the loops *do*! This function will serve as example→

```
public static boolean mystery(int[] array){
        int index=1;
        boolean answer = true;
        for(int i=0;i<array.length-1;) {
                if(index==array.length) {
                        i = i + 1;
                        index=i+1;
                }
                else {
                   if(array[i]>=array[index])
                      answer=false;
                   index = index + 1;
                }
        }
        return answer;
}
```

1) Copy `mystery` into Eclipse. Edit its code so that it returns *both* the answer and the number of times that the loop runs. *(Hint: use a Pair)*
In your `main` function, call `mystery` on arrays of different sizes and show how many times the loop runs for each array size.
In comments below your `main` function, interpret the results and conclude about the Θ of `mystery`

2) The function `mystery` had a single loop, yet it did not have the complexity that you may have expected from a single loop. Explain which tricks were used in the function such that its complexity was not as it may have seemed from a cursory look.
3) Use the same tricks as you learned from `mystery` to write a function `trickyhobbitses` that has a single for loop but runs in $\Theta(n^3)$.

Include your entire Eclipse project (*not* just the src files) into your zip file for submission.

0.5 pt each

IV. Various levels of analyses

1) Detail the Θ complexity of addStuff.
2) Detail the ~ complexity of addStuff.
3) Imagine that you had to choose between addStuff and another solution with a complexity of $\Theta(n^3)$. Explain how you would arrive at a decision.
4) Imagine that you had to choose between addStuff and another solution with a complexity of $\Theta(n^2)$. Explain how you would arrive at a decision.

```
public static int addStuff(int[] array){
    int res = 0;
    for(int i=0; i<array.length; i++)
        for(int j=i; j<array.length; j++)
            res += 2;
    return res;
}
```

Provide your answers in a PDF and include it into your zip file for submission.

1 pt each

V. Unusual times

The two questions in this section can be challenging at first, because they will get you to think. Although you have all it takes to answer them, you may not immediately think of the solution. And that's ok. Many algorithms will take time to come up with, and sometimes a bit of inspiration. Look at these questions early enough so you give yourself time. A hint is provided to help you.

1) The most common categories of time complexity are listed on the next page, which you can print as a cheat sheet for your midterm (when the time comes). However, there are other categories. They may be rare, but they still exist. One such category is Θ(log log n). In other words, the time complexity grows as a function of the log *of the log* of n.

Consider a function that takes as argument an array list representing consecutive numbers from 1 to n (e.g., 1, 2, 3, 4, …). Explain which algorithm you may create such that the function runs in Θ(log log n). Your algorithm does not need to do something useful, but it has to do it in Θ(log log n).

Hint: Θ(log log n) arises when one function over the data results in data of size log n, which is further processed through another function in log n.

2) Consider a function that takes as arguments two arrays of the same length, which represent consecutive numbers from 1 to n. Explain which algorithm you may create such that the function runs in $\Theta(\log^2 n)$. Your algorithm does not need to do something useful, but it has to do it in $\Theta(\log^2 n)$.

*Hint: $\Theta(\log^2 n)$ is the product of two log functions, that is, log n * log n.*

Provide your answers in a PDF and include it into your zip file for submission.

Algorithms Cheat Sheet

Core concepts, in short:

- The 'efficiency' of your solution is always a trade-off, at least between time and space. Prioritizing one over the other depends on the context (e.g., embedded systems, real-time systems).

- Knowing the time complexity by reading the code is very limited (i.e. theoretical), because the code may intricate, and because it requires a top-notch knowledge of what a language does.

- Knowing the time complexity by running the code (i.e. empirical) will tell us exactly how long it takes on one specific hardware, given one specific use of this hardware, and given the versions of the libraries used to run the code. It's specific but also 'polluted' by all these factors.

- The (theoretical) time complexity is often expressed as a function of the size of the input. The first input is denoted *n*, the next one *m*.

- The size of the input cannot always be assumed to be arbitrarily large. In many applications, it is known and fixed. What may vary could be the amount of missing data or its properties.

- Counting the theoretical time complexity is done through two assumptions: uniform cost model (we assumed that each operation takes a constant amount of time *although we do know* that adding two big numbers is longer than two small ones) and random access machine model (we lump all operations together by assuming they all take the same time *although we do know* that a division is longer than a multiplication).

- We have four notations for the theoretical time complexity:
    - Exact one. Count everything, sort it by decreasing term at the end.
      E.g.: $\frac{1}{6} N^3 + 20N + 16$
    - ~ ("tilde"). Drop all terms but the leading one.
      E.g.: $\frac{1}{6} N^3 + 20N + 16 \quad \sim \frac{1}{6} N^3$
    - Θ ("theta"). Drop all terms *and constants* but the leading one.
      E.g.: $\frac{1}{6} \boldsymbol{N^3} + 20N + 16$ is in $\boldsymbol{\Theta}(N^3)$
    - O ("big O"). Take any function either of the same class or above. There are multiple answers to this one whereas there is a single answer to ~ or Θ.
      E.g.: $\frac{1}{6} \boldsymbol{N^3} + 20N + 16$ is in $\boldsymbol{O}(N^3)$ but also $\boldsymbol{O}(N^4)$, $\boldsymbol{O}(N^5)$, $\boldsymbol{O}(N!)$ or $\boldsymbol{O}(2^N)$ since all of these functions are eventually bigger than $\frac{1}{6} \boldsymbol{N^3} + 20N + 16$ from some N onward.

Θ	Name	Typical code framework	Description	Example
1	constant	a = b + c;	statement	Add two numbers
log N	logarithm	while (N > 1) { n = N/2; … }	Divide in half	Binary search
N	linear	for (int i=0; i<N; i++) { … }	(Single) loop	Find the maximum
N log N	linearithmic	mergesort or quicksort	Divide and conquer	Mergesort or quicksort

N^2	quadratic	for (int i=0; i<N; i++)
		for(int j=0; j<N; j++)
		{ … }
			Double loop	Check all pairs

N^3	cubic	for (int i=0; i<N; i++)
		for(int j=0; j<N; j++)
		for(int k=0; j<N; k++)
		{ … }
			Triple loop	Check all triples

2^N	exponential	[discussed later in the course]	Exhaustive search	Check all subsets