Comp 160: Algorithms, **Recitation 0**

0. State the formal definitions of O, Ω, and Θ notation.

1. Let $f(n) = 2n^2 + n$. Prove the following statements:

   (a) $f(n) = O(n^3)$

   (b) $f(n) = O(n^2)$

   (c) If you combine the last two results we get:

   $$O(n^3) = f(n) = O(n^2) \Rightarrow O(n^3) = O(n^2)$$

   That is, we have shown that $O(n^3) = O(n^2)$. Is this correct? Does this make sense? Justify your answer

2. Let's find matching upper and lower bounds for the function $f(n) = \Sigma_{i=1}^{n} i^2$.

   (a) Before even attempting to answer any such question, the first step is to understand the expression. Start by computing $f(3)$, and $f(5)$. Computing $f(300)$ would probably take you a lot of time, but do you feel confident you can do it?

   (b) Good. Now for a generic $f(n)$, we want to upper bound all elements in the sum by the same number. What number could we use? Apply the upper bound and add up all of the numbers. What do we get? Does this give a big O bound for $f(n)$?

   (c) Now we want to do the same idea but for lower bound. Let's lower bound the smallest element, and apply that to all elements. Add up all the numbers. What do we get?

   (d) That approach was not very useful (there is a big gap between upper and lower bounds). Rather than lower bounding the smallest element, we will lower bound the *middle* element. Can you apply that bound to the middle element and all elements that are larger? How many are there? Add up all of the numbers. What do we get? Note that we are ignoring elements that are smaller than the middle element. Is that ok?

   (e) If you followed the previous steps, you probably found matching upper and lower bounds and thus we got a Θ bound. The method for obtaining an upper bound is called *exaggerate* and the one for the lower bound is called *simplify*. Apply the exaggerate and simplify techniques to the following function: $g(n) = \sum_{i=1}^{n} i \log i$

3. Compare each of the following pairs of functions asymptotically. To compare $f(n)$ and $g(n)$ you should prove that $f(n)$ is $O(g(n))$ or $\Omega(g(n))$ or $\Theta(g(n))$.

   (a) 2^{n+1} vs. 2^n

   (b) $\log(\Theta(1) \cdot n)$ vs. $\log n$

   (c) $\log n^{\Theta(1)}$ vs. $\log n$

Note: the questions below are additional extra questions for practice. Do them on your own time as practice. For some of these questions we do not have solutions written (want to be a TA next semester? why don't you try writing some really good answers so that we can share with your fellow students?).

4. You probably heard that an algorithm that runs in $\Theta(\log n)$ runtime is faster than one that runs in $\Theta(n)$ time. In this exercise, we will actually *prove* such a statement (remember, no matter how trivial a statement is, we should always have a proof of it). Let's start with some easy cases:

   (a) *Prove by induction that* $n < 2^n$:

   (b) *Using the result from (a), prove that* $\log_2 n < n$:

   (c) *Using the result from (b), prove that* $\log_2 n = O(\sqrt{n})$

5. Say that $f(n)$, $g(n)$, $h(n)$ are three functions such that $f(n) = O(g(n))$ and $g(n) = O(h(n))$. Show that $f(n) = O(h(n))$.

6. Let $f(x) = O(x)$ and $g(x) = O(x)$. Let c be a positive constant. Prove or disprove that $f(x) + c \cdot g(y) = O(x + y)$.

7. Now it is time to practice proof structuring. Consider the following problem: A democratic genie wants to grant one wish to each citizen in the world. The genie will start at a city, grant wishes to people in that city until everyone has had one wish. Then, it moves to the next city (in instant time, we are talking about a genie after all), grants wishes to people in the second city, and so on until all humans in the World have been granted a wish[1]. The genie loves procrastinating: each day it will grant a wish to half of the population of the city that have not been granted a wish yet (rounding up).

   As the genie's assistant, your task is the following: you are given a list $c_1, \ldots, c_n$ of *cities*. Each city consists of a string (the name of the city) and an integer (the population of the city). For simplicity you can assume that population of the cities is static along time. The genie will visit the cities in order (first the genie will spend however many days needed in c_1 until everyone is granted a wish. Then move to c_2 and so on). Design an algorithm that reports the largest city that will have all of its citizens granted a wish in the each year (that is, your output should be something like *in the first year the biggest city whose population has been fully granted wishes is* c_4*, for the second year it is* c_7*, and so on.*

   How would you solve the above problem? Try to structure your solution as best as possible.

[1] There is only one rule, though: the genie cannot grant you a high grade in Comp 160. You have to earn that yourself.