CSC317 Fall 2022
Instructor: Odelia Schwartz
Assignment 2

Sept 6, 2022 (due Sept 13 2022 by midnight, on Blackboard). Note: Some of the questions are based on the Cormen textbook. Points listed below are for relative weighting of the questions in this assignment. Each assignment will in the end be weighted equally.

1. **Express the function $2n^3 + 100n^2 + 6$ in terms of big Theta θ notation (in the simplest possible form). Explain your answer (you do not need to prove, but explain your logic). (5 points)**

Asymptotically, we can ignore lower order terms and only consider the leading term (the term with the greatest factor of n). Considering this, the function is bounded above by n^3, making it O(n^3), and bounded below by n^3, making it Ω(n^3). By the definitions of big O and big Omega, the function could never run any faster or slower than n^3, and so it is tightly bound by the function n^3, making it Θ(n^3) complexity.

2. **List in order from fastest run time to slowest run time the following functions (for instance n^2 is faster than n^3, so you would list n^2 first and below it you would list n^3). If two functions are asymptotically similar in their run time (for instance, n^2 and $10n^2$ are asymptotically similar; they are both $\Theta(n^2)$), list them together. Explain your answer (no need to formally prove; just explain the logic that you followed in the ordering). (10 points)**

a. **1000** (this function does not grow with n at all, it will always be 1000 (constant time), making it the fastest out of all of them for big enough n)
b. $5log_2n$ (ignoring the constant 5 in front, this function is log(n) or logarithmic, which is slower than constant time but faster than linear)
c. $n + 10$, $3n + 2log_2n$ (asymptotically, these two functions are similar since they both have a leading term of n after we ignore constants and lower order terms, making them both linear time which is slower than logarithmic but faster than linearithmic)
d. $nlog_2n$ (this function is linearithmic (the form nlog(n), a logarithm times n), which is slower than linear but faster than quadratic)
e. $8n^{2.5}$ (ignoring constants we have $n^{2.5}$, which is in between quadratic (n^2) and cubic (n^3); since its slower than quadratic it must also be slower than the previous linearithmic function, and since $2.5 < 3$ it must be faster than cubic fuctions)
f. $2n^3, n^3 + 2n^2 + 10$ (ignoring constants and lower order terms, we have a cubic (n^3) leading term in both of these functions, which is slower than our previous $n^{2.5}$ but definitely faster than a function n^x where x > 3)

g. $n^{3.1}$ (here we have a function n^x where x > 3, so it is slower than cubic but faster than a function n^x where x > 3.1, and also faster than exponential or higher complexity classes)
h. 2^n (finally, this is an example of an exponential function, which is slower than any n^x functions where x is a constant, making it the slowest function in our list; if we had any factorial (n!) or super-exponential (like n^n) functions then they would be even slower, but we don't have any of these so 2^n is the slowest function in our list)

3. **Explain why the statement, "The running time of algorithm A is at least O(n^3)," is meaningless. (5 points)**

Saying an algorithm is at least O(n^3) doesn't really give us any useful information because if we are deciding whether to use this algorithm over another one and speed is important to us, we need to know information such as the slowest runtime and the average runtime. If all we know is that it runs in at least O(n^3), the algorithm could have a worst case of something infeasible like O(n!), in which case we likely don't want to implement this algorithm for anything serious because it is too slow. It is much more useful to know the worst or average time complexity of an algorithm so that you can anticipate how fast it will actually run on your system(s).

4. **Why do we usually care more about the worst case run time? Does it make a difference if we say the worst case run time is "Θ(n^2)" instead of "O(n^2)"? (4 points)**

Big O is an upper bound measurement of the worst time complexity our algorithm could have, while big Theta is a tight bound above and below. So it makes a difference when you say Θ(n^2) as compared to O(n^2), because Θ(n^2) means that our algorithm will at worst run with complexity n^2, and at best run with complexity n^2. O(n^2), on the other hand, means that it will run at worst with complexity n^2, but sometimes it may run faster. An example of this is insertion sort, which is O(n^2) since this is the worst case complexity, but in the best case it can run proportional to n.

5. **Which of the following functions are Ω(n^2). Explain your answer (you do not need to prove, but explain your logic). (6 points)**

a. $5n + 5$
b. $3n^2 - 7$
c. $6n^4 + 5$

Functions (b) and (c) are Ω(n^2) because big Omega describes the lower bound of a function, or the best case time complexity. We can see that it is not possible for functions (b) and (c) to perform faster than n^2 since they both have an n^2 or greater as their leading terms. Function (b) has a $3n^2$, so it could not possibly run faster than n^2, and function (c) has a $6n^4$, which means it is also not possible for it

to run faster than n^2, so we know that both of these functions are lower bounded by n^2. Function (a), on the other hand, has $5n$ as its leading term, which is less than n^2, so it is $\Omega(n)$ instead. Therefore, the only functions that are $\Omega(n^2)$ are (b) and (c).