Algorithms[1] Lab 1

Due: Tuesday, 24 September 2019 (afternoon), *at the beginning of class*

Notes on Grading

Please write your answer to each problem on a separate sheet of paper, since, to improve consistency in grading, a given problem will be graded by the same grader. If you use more than one sheet of paper for a given problem, staple *just* those sheets together. Paper clip all your answers together and turn that in. **Don't forget to write your name on each sheet.** Note that it is just as important to write in a style that is easy to understand and convincing as it is to get the correct answer. So, your assignment will be evaluated based not only on the correctness of your final answer, but also on clarity, neatness and attention to details.

Topics Covered For This Lab

- Analyzing worst-case and best-case running times
- Rates of growth, including rates of growth of standard functions

Review of Big-O, Big-Omega, and Big-Theta

Big-O

Big-O can be thought of as an upper bound, i.e. the running time of an $O(n^3)$ algorithm is no worse than n^3. So, to show that $f(n)$ is $O(g(n))$, we need to find a constant $c > 0$ and an integer constant $n_0 > 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

Big-Omega

As noted above, Big-O can be thought of as an upper bound, i.e. the running time of an $O(n^3)$ algorithm is no worse than n^3. But, this may not be true for all inputs. Sometimes we want to talk about a *lower bound* on the running time of an algorithm. For this, we use Big-Omega (Ω) notation. For example, saying that the running time of an algorithm is $\Omega(n)$ means that the running time is at least n. So an algorithm could be both $O(n^3)$ and $\Omega(n)$. A Big-Omega proof is exactly the same as a Big-O proof except that the inequality is reversed. In other words, to show that $f(n)$ is $\Omega(g(n))$, we need to find a constant $c > 0$ and an integer constant $n_0 > 0$ such that $f(n) \geq cg(n)$ for all $n \geq n_0$.

Big-Theta

If an algorithm is Big-O and Big-Omega of the same function, it is said to be Big-Theta (Θ) of that function. For example, if the running time of an algorithm is $O(n^2)$ *and* $\Omega(n^2)$, then it is also $\Theta(n^2)$. In other words, the running time of that algorithm is no better and no worse than n^2. To prove that $f(n)$ is $\Theta(g(n))$, we need to show that it is both $O(g(n))$ *and* $\Omega(g(n))$. Note that while the c in the Big-O proof can be different from the c in the Big-Omega proof, the n_0 must be the same, i.e. they are both true from the same point onward. But, this is not a problem – we have an n_0 in each of the two proofs and, if they are not the same, we just take the maximum.

[1] CSCI 2200, Laura Toma and Stephen Majercik, Bowdoin College

In Lab Exercises (COLLABORATION LEVEL 0)

1. Which of the following are true? **For the last three only**, use asymptotic analysis to show why or why not.
   (a) $\frac{n^2}{3} + 3n = O(n)$
   (b) $\frac{n^2}{3} + 3n = \Omega(n)$
   (c) $\frac{n^2}{3} + 3n = \Theta(n)$
   (d) $\frac{n^2}{3} + 3n = \Theta(n^2)$
   (e) $2^{n+1} = O(2^n)$
   (f) $2^{2n} = O(2^n)$
   (g) $3^{\lg n} = O(n^2)$

2. Given $3n^2 + 10n \lg n$, is it:
   (a) $O(1)$, $\Omega(1)$, $\Theta(1)$?
   (b) $O(n)$, $\Omega(n)$, $\Theta(n)$?
   (c) $O(n \lg n)$, $\Omega(n \lg n)$, $\Theta(n \lg n)$?
   (d) $O(n^2)$, $\Omega(n^2)$, $\Theta(n^2)$?

3. Given $5n \lg n + \frac{n}{2}$, is it:
   (a) $O(\lg n)$, $\Omega(\lg n)$, $\Theta(\lg n)$?
   (b) $O(n)$, $\Omega(n)$, $\Theta(n)$?
   (c) $O(n \lg n)$, $\Omega(n \lg n)$, $\Theta(n \lg n)$?

4. Given $100\sqrt{n} + 50 \lg n$, is it:
   (a) $O(\lg n)$, $\Omega(\lg n)$, $\Theta(\lg n)$?
   (b) $O(\sqrt{n})$, $\Omega(\sqrt{n})$, $\Theta(\sqrt{n})$?
   (c) $O(n)$, $\Omega(n)$, $\Theta(n)$?

5. Which grows faster: $n^{n \lg n}$ or n^{n^2}? (Hint: The two have the same base, i.e. n.)

6. What is the order of growth of $n^2\sqrt{n} \lg n + n^3$? Can you give a Big-Theta bound?

7. Prove or disprove: if $f(n)$ is $O(g(n))$, this implies that $g(n)$ is $\Omega(f(n))$.

8. (Fun Interview Question) You are presented with 9 marbles. All of the marbles look identical i.e. same shape, color, and dimensions. However, 8 of the 9 marbles have exactly the same weight; the last marble is heavier. The only tool you have to measure weights is an old fashioned balance scale. You are only allowed to use the scale 2 times. How do you find the one marble that is not the same weight as the others?

Homework Problems (COLLABORATION LEVEL 1)

You are encouraged to collaborate at Level 1 for these problems. Please refer to the class website for a description of this collaroration level. List the people with whom you discussed the problems.

1. Find the order of growth of the following functions. Note:
   - the term that grows the fastest is the one that determines the order of growth,
   - there are several useful slides from the lecture with rules and relative growth rates, and
   - there is a helpful list of rules of exponents and logarithms at the end of this lab.

   (a) $n \lg \lg n + n \lg n + \sqrt{n} \lg^2 n$
   (b) $\sqrt{n} \lg n + n$
   (c) $n^2 + \sqrt{n} \lg^3 n$
   (d) $3^{\lg n} + n^2 + n \lg n$
   (e) $\sqrt{3}^{\lg n} + n^2 + n \lg n$
   (f) $2^n + 2^{2n}$
   (g) $2^{\lg n} + \lg n^2$
   (h) $(\lg n)^{\lg n} + n^3$

2. Give an example of a *continuous* (not piece-wise) positive function $f(n)$ such that $f(n)$ is neither $O(n)$ nor $\Omega(n)$. Hint: Think about an oscillating function that never stays within fixed bounds.

3. Arrange the following functions in ascending order of growth rate. For each pair of consecutive functions, give a brief justification for why they are in this order. For example, if you ordered A, B, C, you need to justify that $A = O(B)$ and $B = O(C)$.

   $2^{\sqrt{\lg n}}$ $\quad 2^n$ $\quad n^{4/3}$ $\quad n(\lg n)^3$ $\quad n^{\lg n}$ $\quad 2^{2^n}$ $\quad 2^{n^2}$

4. Suppose you have algorithms with these five running times:

   (a) n^2
   (b) $100n^2$
   (c) n^3
   (d) $n \lg n$
   (e) 2^n

   How does the running time of these algorithms change when you double the input size?

5. Describe a method for finding both the minimum and the maximum of n numbers with fewer than $3n/2$ comparisons. Be sure to cover the case when n is even *and* the case when n is odd.

6. Suppose each row of an $n \times n$ array A consists of 1's and 0's such that, in any row i of A, all the 1's come before any 0's. Assuming A is already in memory, describe a method running in $O(n)$ time (*not* $O(n^2)$ time) for finding the row of A that contains the most 1's. You must explain why your algorithm is $O(n)$.

Rules of Exponents and Logarithms

- Exponents:
  1) $n^{(a+b)} = n^a n^b$
  2) $n^{(a-b)} = \frac{n^a}{n^b}$
  3) $n^{ab} = (n^a)^b = (n^b)^a$
- Logarithms:
  1) $\lg^k n = (\lg n)^k$
  2) $\lg \lg n = \lg(\lg n)$
  3) $a^{\log_b c} = c^{\log_b a}$
  4) $a^{\log_a b} = b$ (in particular, $2^{\lg b} = b$)
  5) $\log_a n = \frac{\log_b n}{\log_b a}$
  6) $\lg n^b = b \lg n$ (so taking the lg of something can be a way to get rid of exponents)
  7) $\lg xy = \lg x + \lg y$
  8) $\lg \frac{x}{y} = \lg x - \lg y$
  9) $\log_a b = \frac{1}{\log_b a}$