CS35 Lab 4 Big-O

Kenneth Barkdoll and Yael Borger

September 2021

1 Proofs

A function is $f(n)$ is $O(g(n))$ provided that $\exists c > 0$ and $n_0 \geq 1$ such that $\forall n > n_0$, $f(n) \leq c \times g(n)$.

1. Prove that $8n^3 + 7n^2 - 12$ is $O(n^3)$.

   Proof. Let $n \geq 1$. For any value n, $8n^3 \leq 8n^3$, $7n^2 \leq 7n^3$, and $-12 \leq 0$. By summing these equations it is clear that:

   $$8n^3 + 7n^2 - 12 \leq 8n^3 + 7n^3 - 0 \tag{1}$$
   $$8n^3 + 7n^2 - 12 \leq 15n^3 \tag{2}$$

   Thus, because these equations are true for all $n > 1$, $8n^3 + 7n^2 - 12$ is $O(n^3)$, as long as $c \geq 15$ and $n_0 \geq 1$. □

2. Prove that $6n^2 - n + 4$ is $O(n^2)$.

   Proof. Let $n \geq 1$. For any value n, $6n^2 \leq 6n^2$, $-n \leq 0$, and $4 \leq 4n^2$. By summing these equations it is clear that:

   $$6n^2 - n + 4 \leq 6n^2 - 0 + 4n^2 \tag{3}$$
   $$6n^2 - n + 4 \leq 10n^2 \tag{4}$$

   Thus, because these equations are true for all $n > 1$, $6n^2 - n + 4$ is $O(n^2)$, as long as $c \geq 10$ and $n_0 \geq 1$. □

2 Pseudo-code Analysis

1. Function fnA(n)

   This function is $0(n)$. This is because as the size of n doubles, the function will only have to do twice as many assignments, and when the size of n triples, the function will only have to do three times as many assignments. This means the function operates in linear time.

2. Function fnB(n)

   This function is $O(n^2)$. The nested loops mean that there are n operations done for every value of n. This represents a quadratic expansion of the number of operations done for each value of n, as doubling the size of n quadruples the number of operations.

3. Function fnC(n)

   This function is $O(n)$. For each value of n, the program sets four variable assignments. This means that when n doubles, so does the run time of the program, since n is only iterated through once. Thus, this program runs in linear time.

4. Function fnD(n)

   This function is $0(n^3)$. The function iterates through n^3 values in the for loop. This means that a doubling in the size of n would increase the run time of the program eight-fold. This is indicative of cubic time.

5. Function fnE(n)

   This function is $O(n \log n)$. The for loop runs n times, making this at least a linear function, however the inside of the loop operates in logarithmic time. When the size of n doubles, the number of iterations of the while loop only increases by 1. When the size of quadruples, the number of iterations by the while loop only increases by 2. This is indicative of a logarithmic increase in run time within the loop. Combined, this function does a linear number of logarithmic operations, meaning it is $O(n \log n)$.

6. Function fnF(n)

   This function is $O(n^4)$. The nested for loops indicate that the function does n^2 operations n^2 times, which combine to a run time of n^4. Doubling the size of n increases the run time sixteen-fold.

These functions can tentatively be ranked in order from slowest to fastest: F, D, B, E, C, A.

3 Pseudo-Code Matching

The functions from Function Timer can be ranked from slowest to fastest in the order 1, 3, 2, 4, 6, 5.

Based on the Big-O analysis of the pseudo-code functions it is clear that these are the matchings (where the number is the number from Function Timer, and the name is from the lab report (part 2):

Function 1 is Function fnF. fnF's quadratic run time is the slowest of all of the Big-O analyses, and thus is Function 1 as Function 1's run time dwarfs the other function run times.

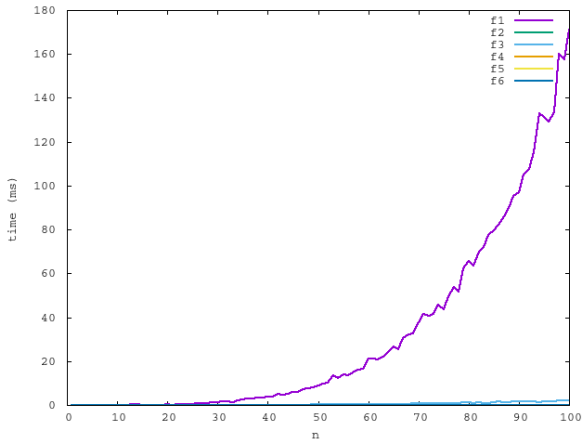

Figure 1: Graph 1

Function 2 is Function fnB. Function fnB is quadratic, and thus will be slower than the linear and n log(n) functions, but still slower than functions 1 and 3, which are $O(n^4)$ and $O(n^3)$ respectively. Thus, because Function 2 is the third slowest function, it must be fnB.

Function 3 is Function fnD. Function fnD is $O(n^3)$, which will never be as slow as $O(n^4)$, but will be much slower than any linear or near linear (n log n) function, as well as being slower than $O(n^2)$ meaning that, fnD will be the second slowest after Function 1 (as seen in Graph 1).

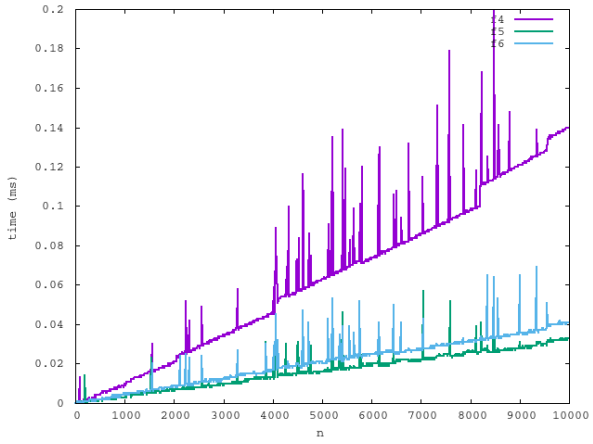

Figure 2: Graph 2

Function 4 is Function fnE. Between functions A, C, and E, there are three remaining functions that do not appear to run in parabolic time. Two (A and C) are linear, and thus must be slower than function E which is $O(n \log n)$. Furthermore, it is just slightly visible from Graph 2 that function 4 increases in slope as n increases, which is a property of an $n \log n$ function but not of a linear time function.

Finally Function 6 is Function fnC and Function 5 is Function fnA. They are both linear time functions, but their exact operations are able to differentiate them. Function 6 (C) is slightly slower, since for every value from 1 to n, it enters a for loop and does four variable assignments, whereas Function 5 (A) only iterates through half the length of n and only assigns one variable each time. This means that for a given n, both Functions C and A run in linear time, but Function C does eight times the number of assignments, giving it a longer run time. Thus, since Function 6 has a higher run time for a given n than Function 5, Function 6 is fnC and Function 5 is fnA.