CS2210 Assignment 1

Adam Gale

October 23, 2023

this is my first time using LaTeX so please have mercy on my formatting

Question 1

Use the definition of "big Oh" to prove that $\frac{1}{2n-1}$ is $O(\frac{1}{n})$.

We need to find a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

$$\begin{aligned} \frac{1}{2n-1} &\leq c\frac{1}{n} && \text{for all } n \geq n_0 \\ \frac{1}{2n-1} \cdot (2n-1) &\leq \frac{c}{n} \cdot (2n-1) && \text{for all } n \geq n_0 \\ 1 &\leq 2c - \frac{c}{n} && \text{for all } n \geq n_0 \end{aligned}$$

Choose the value 1 for c.

$$\begin{aligned} 1 &\leq 2(1) - \frac{1}{n} && \text{for all } n \geq n_0 \\ 1 &\leq 2 - \frac{1}{n} && \text{for all } n \geq n_0 \\ -1 &\leq -\frac{1}{n} && \text{for all } n \geq n_0 \\ 1 &\geq \frac{1}{n} && \text{for all } n \geq n_0 \\ n &\geq 1 && \text{for all } n \geq n_0 \end{aligned}$$

We can see that any positive integer will work; choosing 1 to be n_0 satisfies the equation as well as the condition $n_0 \geq 1$. As n increases, this inequality will continue to be true.

We have found a constant value $c = 1$ and integer constant $n_0 = 1$ that satisfy our conditions and make our initial inequality true, and have proven that $\frac{1}{2n-1}$ is $O(\frac{1}{n})$.

Question 2

Let $f(n)$ and $g(n)$ be positive functions such that $f(n)$ is $O(g(n))$ and $g(n) \geq 1$ for all $n \geq 1$. Using the definition of "big Oh" show that $f(n) + k$ is $O(g(n))$, where $k > 0$ is constant.

We need to find a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) + k \leq cg(n)$ for all $n \geq n_0$.

Since $f(n)$ is $O(g(n))$, we know there is a constant $c' > 0$ and integer $n_0' \geq 1$ such that $f(n) \leq c'g(n)$ for all $n \geq n_0'$.

$$\begin{aligned} f(n) &\leq c'g(n) && \text{for all } n \geq n_0 \\ f(n) + k &\leq c'g(n) + k && \text{for all } n \geq n_0 \\ f(n) + k &\leq (c' + k)g(n)^* && \text{for all } n \geq n_0 \end{aligned}$$

*this inequality operation is only possible bceause $g(n) \geq 1$ for all $n \geq 1$

We have found a constant value $c = (c' + k)$ and integer constant $n_0 = n_0'$ that satisfy our conditions and make our intial inequality true, and have proved that $f(n) + k$ is $O(g(n))$.

Question 3

Use the definition of "big Oh" to prove that $\frac{1}{n}$ is not $O(\frac{1}{n^2})$.

We will use **proof by contradiction**. Assume that $\frac{1}{n}$ is $O(\frac{1}{n^2})$.

If $\frac{1}{n}$ is $O(\frac{1}{n^2})$, there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

$$\begin{aligned} \frac{1}{n} &\leq c\frac{1}{n^2} && \text{for all } n \geq n_0 \\ \frac{1}{n} \cdot n^2 &\leq c\frac{1}{n^2} \cdot n^2 && \text{for all } n \geq n_0 \\ n &\leq c && \text{for all } n \geq n_0 \end{aligned}$$

This inequality is valid only for values of n that are at most c, so this inequality cannot be true for all values of $n \geq n_0$. Any value of $n \geq c + n_0$ contradicts our $n \leq c$ inequality. We have shown that there are no constant values $c > 0$ and $n_0 \geq 1$ such that $\frac{1}{n} \leq c\frac{1}{n^2}$ for all $n \geq n_0$. Consequently, $\frac{1}{n}$ is not $O(\frac{1}{n^2})$.

Question 4

Regarding the *triSearch* algorithm: prove that either (i) the algorithm terminates, or (ii) the algorithm does not terminate.

By analyzing the code, we can identify a case in which the algorithm gets stuck in an infinite loop and **does not terminate**.

Suppose we run *triSeach* ([2],0,0,1), where we are searching through the size-1 array [2] for the value 1.

if $first > last$ **then return** -1	$first \not> last$
else {	
$numValues \leftarrow last - first + 1$	$numValues$ is 1
$third \leftarrow first + \lfloor \frac{numValues}{3} \rfloor$	$third$ is $0 + \lfloor \frac{1}{3} \rfloor = 0$
if $x = L[third]$ **then return** $third$	$x \neq L[third]$
else if $x < L[third]$ **then return** triSearch$(L, first, third, x)$	$x < L[third]$

At this point, the function calls itself with $triSearch([2], 0, 0, 1)$. This is the same as our original function call. This function call will again run as shown above, and will once again call itself with identical parameters. This leads to an infinite loop and the **algorithm will not terminate**.

Question 5

Regarding the *copies* algorithm: prove that either (i) the algorithm outputs the correct answer, or (ii) the algorithm does not output the correct answer.

By analyzing the code, we can identify a case in which the algorithm returns the **incorrect output**.

One case in which the algorithm would return the incorrect result would be $copies([1, 1, 0], 3, 1)$. In this case, the algorithm is looking over the array $[1, 1, 0]$ of length 3 and counting the instances of

the number 1. We expect the algorithm to return the value 2, as 1 appears twice in our array.

if $x = L[0]$ **then** $c \leftarrow 1$	here, c is assigned 1 as $L[0] = x$
else $c \leftarrow 1$	
for $i \leftarrow 0$ **to** $n - 1$ **do**	loop executes 3 times, for $L[0], L[1], L[2]$
if $x = L[i]$ **then** $c \leftarrow c + 1$	c incremented **twice**, $L[0] = x$, $L[1] = x$
return c	c is returned with the value 3

As noted earlier, the correct output for this input is 2, but the algorithm returns a value of 3. This is caused by index 0 being assessed twice, once before the loop and once inside the loop. We have shown that the algorithm **does not output the correct answer**.

Question 6

Regarding the *duplicated* algorithm: compute the time complexity of this algorithm in the worst case. You must explain and give the order of the complexity.

$$c_1 = \begin{cases} duplicateFound \leftarrow \text{true} \\ i \leftarrow 0 \end{cases}$$

$$c_2 = \begin{cases} \textbf{while } duplicateFound = \text{true } \textbf{do } \{ \\ \quad \textbf{if } L[i] = L[i+1] \textbf{ then return } \text{true} \\ \quad \textbf{else } duplicateFound = \text{false} \\ \quad \textbf{if } i < n - 1 \textbf{ then } i \leftarrow i + 1 \\ \} \end{cases}$$

$$c_3 = \begin{cases} \textbf{return } duplicateFound \end{cases}$$

c_1 and c_3 are constants. c_2 is a while loop containing a constant amount of operations. Inside the while loop, the first if/then statement will **return true and end the function call immediately** if the first two values compared are equal. If the first two values are *not* equal, the else statement will set *duplicateFound* to false, and the while loop condition will no longer be met. **The loop will not repeat and** *duplicateFound* **(false) will be returned.** We can see that both possible cases cause the while loop not to repeat, so we can also treat c_2 as a constant.

This puts the algorithm's worst case time complexity at $f(n) = c_1 + c_2 + c_3$. We can combine the constants and restate this as $f(n) = c$.

As the time complexity is constant and does not depend the input, **we can say that** $f(n)$ **is** $O(1)$**.**

Question 7

Modify the given *Search.java* so that it prints the worst case running times of the linear, quadratic, and factorial search functions.

LinearSearch	
n	time
5	193 ns
10	278 ns
100	782 ns
1000	6787 ns
10000	15022 ns
100000	43438 ns

QuadraticSearch	
n	time
5	433 ns
10	886 ns
100	11861 ns
1000	256767 ns
10000	16281415 ns

FactorialSearch	
n	time
7	2726100 ns
8	17338799 ns
9	85774399 ns
10	1001354200 ns
11	13126053799 ns
12	155907374801 ns*

*I was absolutely convinced that my IDE had crashed