CS2223 Algorithms B Term 2013

Exam 1. November 15, 2013 - SOLUTIONS

By Prof. Carolina Ruiz
Dept. of Computer Science
Worcester Polytechnic Institute

Instructions:

- Show your work and justify your answers
- Use the space provided to write your answers
- Ask in case of doubt

Problem I. [15 Points] Asymptotic Growth of Functions.

Prove in detail that for any constant $a \geq 0$: $n^{a} log(n) = O(n^{a+1})$. **Show your work.**

Solution: For illustration purposes, we include two alternative solutions.

Solution 1: using the definition of Big-Oh:

We need to find constants $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0$, $n^{a} log(n) \leq cn^{a+1}$.

Note that since $a \geq 0$ and $n > 0$ then $n^{a} > 0$. Also, we know that for all $n > 0$, $log(n) \leq n$. Hence, $n^{a} log(n) \leq n^a\, n = n^{a+1}$. Therefore for constants $c = 1$ and $n_0 = 1$, we have that for all $n \geq n_0$, $n^{a} log(n) \leq cn^{a+1}$. Hence, $n^{a} log(n) = O(n^{a+1})$. □

Solution 2: using the limit rule:

$$\lim_{n\to\infty} \frac{n^{a} \log(n)}{n^{a+1}} = \lim_{n\to\infty} \frac{\log(n)}{n} = \lim_{n\to\infty} \frac{1/n}{1} \text{ (using de L'Hôpital's rule)} = 0$$

Since this limit exists and is equal to 0, then $n^{a} log(n) = O(n^{a+1})$. □

Problem II. [30 points] Runtime Analysis

The bubleSort algorithm receives a list as its input and returns this list sorted in increasing order.
(Algorithm below adapted from http://interactivepython.org/courselib/static/pythonds/SortSearch/sorting.html)

def bubbleSort(alist):	Cost per instruction	Number of repetitions
1. n = length(alist)	...... $\boldsymbol{c_1}$	.... **1** ....
2. for j in [n-1, n-2, …, 1]:	...... c_2	..... $n-1$. ....
3. for i in [0, 1, 2, …, j-1]:	...... c_3	... $n(n-1)/2$ ......
4. if alist[i] > alist[i+1]:	...... c_4	... $n(n-1)/2$ ......
5. temp = alist[i]	...... c_5	... $n(n-1)/2$ ......
6. alist[i] = alist[i+1]	...... c_6	... $n(n-1)/2$ ......
7. alist[i+1] = temp	...... c_7	... $n(n-1)/2$ ......
8. return(alist)	...... $\boldsymbol{c_8}$	.... **1**. ....

1. [20 Points] Use worst case analysis to construct a function $T(n)$ that gives the runtime of this algorithm as a function of n, the length of the input list. Notes:
   - Instructions 1 and 8: Assume that they are executed in constant time (as shown above).
   - Java's equivalent of instruction 2 is: for (int j = n-1; j >= 1; j--)
   - Java's equivalent of instruction 3 is: for (int i = 0; i < j; i++)

   Show your work step by step, and justify your answer.

   <u>Solution</u>: We describe below our runtime analysis instruction by instruction:

   1. Provided in the problem statement: constant time.

   2. j varies from n-1 to 1, so the condition of this loop is executed n-1 times. If we follow the textbook convention we'd add 1, for a total of n, to include the final check of the loop condition.

   3-7 i varies from 0 to j-1. So the number of times that this loop is executed is:

j = n-1:	i = 0,	1, …,	n-4,	n-3,	n-2:	n-1 times	(= j times)
j = n-2:	i = 0,	1, …,	n-4,	n-3:		n-2 times	(= j times)
j = n-3:	i = 0,	1, …,	n-4:			n-3 times	(= j times)
…	…					…	…
j = 1:	i = 0:					1 time	(= j times)

   Hence, the total number of times that each of these instructions is executed is $\sum_{j=1}^{n-1} j = \frac{(n-1)n}{2}$.

   Note: If we follow the convention in the textbook that the condition of the loop is executed 1 more time that the body of the loop (and hence each row count in the tabulation above will be incremented by 1) the number of times that instruction 3 would be executed is:
   $\sum_{j=2}^{n} j = \frac{(n+1)n}{2} - 1 = \frac{(n+2)(n-1)}{2}$.

   8. Provided in the problem statement: constant time.

Hence, $T(n) = c_1 + c_2(n-1) + (c_3 + c_4 + c_5 + c_6 + c_7)\frac{(n-1)n}{2} + c_8 = k_2n^2 + k_1n + k_0$

for constants $k_0 = c_1 - c_2 + c_8$; $k_1 = c_2 - \frac{c_3 + c_4 + c_5 + c_6 + c_7}{2}$; $k_2 = \frac{c_3 + c_4 + c_5 + c_6 + c_7}{2}$.

2. [10 points] Provide an asymptotic upper bound $g(n)$ as tight as possible for your $T(n)$ function above and prove in detail that $T(n) = O(g(n))$.

Solution: Let $g(n) = n^2$.

Claim: $T(n) = k_2n^2 + k_1n + k_0 = O(n^2)$

Proof: We need to find constants $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0$, $T(n) \leq c\, g(n)$. Note that $k_1n \leq k_1n^2$ and $k_0 \leq k_0n^2$ for all $n \geq 1$. Take $c = k_0 + k_1 + k_2$ and $n_0 = 1$. Then,

for all $n \geq n_0$, $T(n) \leq c\, g(n)$, and so $T(n) = O(g(n)) = O(n^2)$. □

Problem III. [25 points] Transpose Symmetry of Big-O and Big-Omega

Let $f(n)$ and $g(n)$ be asymptotically positive functions. Use the definition of Big-O and Big-Omega to prove in detail that

$$f(n) = O(g(n)) \text{ if and only if } g(n) = \Omega(f(n))$$

1. [10 points] Prove that if $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$.

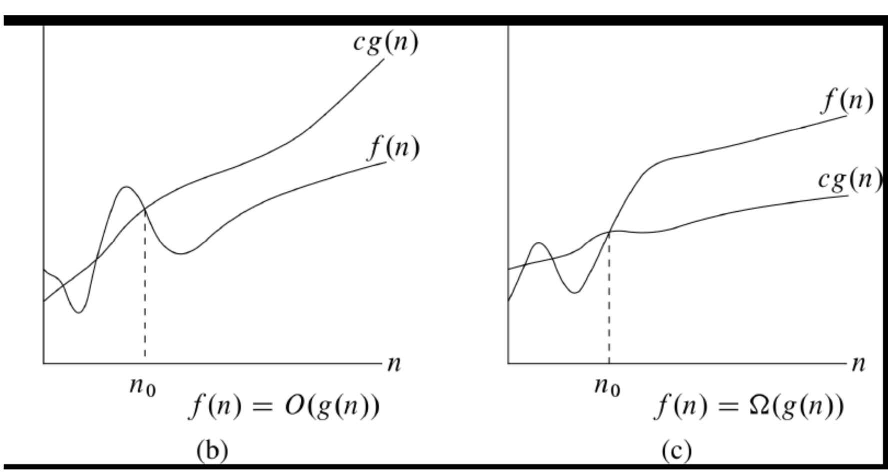
   Solution:
   If $f(n) = O(g(n))$, then there exist constants $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0$, $f(n) \leq c\, g(n)$. Take $k = \frac{1}{c}$. Since $c > 0$ then $k > 0$. Note that for all $n \geq n_0$, $kf(n) \leq g(n)$ and so $g(n) = \Omega(f(n))$.

2. [10 points] Prove that if $g(n) = \Omega(f(n))$ then $f(n) = O(g(n))$.

   Solution:
   If $g(n) = \Omega(f(n))$, then there exist constants $k > 0$ and $n_0 > 0$ such that for all $n \geq n_0$, $g(n) \geq k\, f(n)$. Take $c = \frac{1}{k}$. Since $k > 0$ then $c > 0$. Note that for all $n \geq n_0$, $f(n) \leq cg(n)$ and so, $f(n) = O(g(n))$.

3. [5 points] Explain in words and with plots what it means intuitively for a function $f(n)$ to be $O(g(n))$ or for $f(n)$ to be $\Omega(g(n))$.

Solution: Graphs taken for the textbook: T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction to Algorithms (Third Edition). MIT Press. 2009. $f(n) = O(g(n))$ means that for large enough n's, a constant multiple of $g(n)$ is an upper bound for $f(n)$. In other words, the growth rate of $g(n)$ is greater than or equal to that of $f(n)$ as n goes to infinite. Similarly, $f(n) = \Omega(g(n))$ means that for large enough n's, a constant multiple of $g(n)$ is a lower bound for $f(n)$. In other words, the growth rate of $g(n)$ is smaller than or equal to that of $f(n)$ when n goes to infinite.



Problem IV. [35 Points] Asymptotic Growth of Functions.

Consider the following functions:

$f_1(n) = 3^n$

$f_2(n) = 7$ (that is, $f_2(n)$ is a constant function that always returns 7).

$f_3(n) = n^5$

1. [5 points] List the above functions in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n) = O\big(g(n)\big)$.

   Your list: **Solution:** $7 < n^5 < 3^n$

2. [20 points] Prove in detail that your list is correct. That is, prove that $f(n) = O\big(g(n)\big)$ for every pair of functions $f(n)$ and $g(n)$, where $g(n)$ immediately follows $f(n)$ on your list above.

Solutions:

We provide 2 alternative proofs of each result, but one proof suffices.

Proof: $7 = O(n^5)$:

Proof using the definition of Big-Oh:

Take $c = 7, n_0 = 1$. Then, for all $n \geq n_0$: $7 \leq c\, n^5$, and so $7 = O(n^5)$.

Note also that $n^5 \neq O(7)$ since there are no constants $c > 0,\ n_0 > 0$ such that $n^5 \leq 7c$ for all $n \geq n_{0.}$ As a consequence of this, $n^5 \neq \Theta(7)$.

Proof using the limit rule:

$\lim_{n\to\infty} \frac{7}{n^5} = 0$. Hence, $7 = O(n^5)$ and $n^5 \neq O(7)$.

Proof: $n^5 = O(3^n)$:

Proof using the limit rule:

$\lim_{n\to\infty} \frac{n^5}{3^n} = \lim_{n\to\infty} \frac{5n^4}{\ln(3) * 3^n} \text{(using de L'Hôpital's rule)} =$
$\lim_{n\to\infty} \frac{5*4*3*2*1}{\ln(3)^5 * 3^n} \text{(using de L'Hôpital's rule 4 more times)} = 0.$

Hence, $n^5 = O(3^n)$ and $3^n \neq O(n^5)$. As a consequence of this, $3^n \neq \Theta(n^5)$.

Proof using the definition of Big-Oh:

We need to find constants $c > 0,\ n_0 > 0$ such that $n^5 \le c * 3^n$ for all $n \ge n_{0.}$

Take $c = 1$ and $n_0 = 27.$ Note that for all $n \ge n_0,\ 5 * log_3(n) \le n$. Therefore $3^{5*log_3(n)} \le\ 3^n$, and so $3^{log_3(n^5)} \le\ 3^n$ which implies that $n^5 \le\ 3^n$ as we wanted.

3. [10 points] Are there any pairs of functions $f(n)$ and $g(n)$ from your list above that satisfy $f(n)\ =\ \Theta\ (g(n))$? Prove your answer.

Solution: No, there are no functions $f(n)$ and $g(n)$ from the list above that satisfy $f(n)\ =\ \Theta\ (g(n))$. We have already proven this explicitly when using the Limit Rule in our proofs in part 2 above.