Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dmytro Chibisov
Sandeep Sadanandan

Winter Semester 2007/08
Solution Sheet 3
November 08, 2007

Fundamental Algorithms

Problem 1 (5 Points)

Consider the definitions of o and ω given below.

$f(n) = o(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$f(n) = \omega(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

From these definitions, derive the definitions of o and ω which were given in the class. (Just give an intuitive explanation)

Solution

$f(n) = o(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

This intuitively says that, as n approaches infinity, the function $f(n)$ becomes relatively insignificant, compared to the value of $g(n)$. This means, there exists a certain number n_0, after which even a small fraction of $g(n)$ is very large compared to $f(n)$. More formally: For a positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $f(n) < c \cdot g(n)$ for all $n \geq n_0$.
The above definition happens to be the one given in class. A similar arguement is valid for ω also.

Problem 2 (10 Points)

Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove the following.

1. $f(n) = O(g(n))$ implies $g(n) = O(f(n))$
2. $f(n) = O(g(n))$ implies $\lg f(n) = O(\lg g(n))$. Assume $\lg g(n) > 0$ and $f(n) \geq 1$ for all sufficiently large n.
3. $f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$
4. $f(n) = O(g(n))$ imples $g(n) = \Omega(f(n))$

Solution

1. $f(n) = O(g(n))$ implies $g(n) = O(f(n))$

   $f(n) = O(g(n))$ means $g(n)$ grows faster than $f(n)$. It cannot imply that $f(n)$ grows faster than $g(n)$. Hence not true.

2. $f(n) = O(g(n))$ implies $\lg f(n) = O(\lg g(n))$. Assume $\lg g(n) > 0$ and $f(n) \geq 1$ for all sufficiently large n.

   True. If $g(n)$ grows faster than $f(n)$, $\lg g(n)$ grows faster than $f(n)$.

3. $f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$

   True. Explanation for the previous one is applicable here too.

4. $f(n) = O(g(n))$ imples $g(n) = \Omega(f(n))$

   True. $f(n) = O(g(n))$ means $g(n)$ grows faster than $f(n)$. It implies that $g(n)$ grows faster than $f(n)$. Hence true.

Problem 3 (10 Points)

Prove or disprove the following

1. $o(f(n)) \cap \omega(f(n)) = \phi$
2. $O(f(n)) - O(f(n)) = 0$

Solution

1. $o(f(n)) \cap \omega(f(n)) = \phi$

   $o(f(n)) = \{$all functions which grow strictly slower than $f(n)\}$.

   $\omega(f(n)) = \{$all functions which grow strictly faster than $f(n)\}$.

   Their intersection must be all the functions which grows faster and slower than $f(n)$ simultaneously. Since there are no such functions in $o(f(n))$ and $\omega(f(n))$, it is ϕ.

2. $O(f(n)) - O(f(n)) = 0$

   Each O symbol represents a different approximate quantity. Since the LHS may be $(f(n) - (-f(n)) = 2f(n)$, the best we can say is $O(f(n)) - O(f(n)) = O(f(n))$.

Problem 4 (15 Points)

Fill in the cells of the following table with "yes" or "no", depending on the relationships of functions $f(n)$ and $g(n)$. Other variables: $k \geq 1$, $\epsilon > 0$, $c > 1$ and $m > 1$ are constants.

$f(n)$	$g(n)$	O	o	Ω	ω	Θ
$\lg^k n$	n^ϵ					
n^k	c^n					
2^n	$2^{\frac{n}{2}}$					
$n^{\lg m}$	m^{lgn}					
$\lg(\lg^* n)$	$\lg^*(\lg n)$					

Note: $\lg^*$ is called the iterative logarithm function. It is defined as the number of successive applications of lg function on a given positive number n, until it reduces to 1.
Example: $\lg^* 16$.

$$\lg 16 = 4$$
$$\lg 4 = 2$$
$$\lg 2 = 1$$

Here 3 calls of the function lg were possible before n reduced to 1. Hence, $\lg^* 16 = 3$.

Solution

$f(n)$	$g(n)$	O	o	Ω	ω	Θ	Hint
$\lg^k n$	n^ϵ	yes	yes				$\frac{\mathrm{d}}{\mathrm{d}x} \ln u = \frac{1}{u}\frac{\mathrm{d}u}{\mathrm{d}x}$
n^k	c^n	yes	yes				$\frac{\mathrm{d}}{\mathrm{d}x} c^u = c^u \frac{\mathrm{d}u}{\mathrm{d}x}$
2^n	$2^{\frac{n}{2}}$			yes	yes		
$n^{\lg m}$	m^{lgn}					yes	$2^{\lg n \cdot \lg m} = 2^{\lg m \cdot \lg n}$
$\lg(\lg^* n)$	$\lg^*(\lg n)$	yes	yes				if $\lg^* n = k$, $\lg k = O(k-1)$

Problem 5 (10 Points)

Write down the contents of the following arrays after every step of selection sort until they are completely sorted. Assume that the arrays given represent their initial arrangement of the numbers. Also compute the number of operations needed. (Comparison and Swapping are the operations)

1. | 12 | 8 | -2 | 23 | 5 | 0 |
2. | 31 | 17 | 29 | 11 | 7 | 5 | 3 |

Solution

Selection Sort: In selection sort, one finds out the smallest element of the array and swaps that one with the first element of the array. After this step, as the smallest is already in it's correct position, one focuses on the rest of the array. The next smallest element is found out and then swapped with the second element of the array. Now, as the second element is in it's correct postion, the process continues with the rest of the array. And so on, until the whole array is sorted.
For finding out the smallest element, we need to do $O(n)$ comparisons, and we need to do this n times, which gives us a total complexity of $O(n^2)$.

1. | 12 | 8 | -2 | 23 | 5 | 0 |

   The steps are

   (a) Initial State | 12 | 8 | -2 | 23 | 5 | 0 |

   (b) After 5 comparisons and 1 swap | -2 | 8 | 12 | 23 | 5 | 0 |

   (c) After 4 comparisons and 1 swap | -2 | 0 | 12 | 23 | 5 | 8 |

(d) After 3 comparisons and 1 swap

-2	0	5	23	12	8

(e) After 2 comparisons and 1 swap

-2	0	5	8	12	23

(f) After 1 comparison and 0 swaps

-2	0	5	8	12	23

2.

31	17	29	11	7	5	3

The steps are

1. Initial State

31	17	29	11	7	5	3

2. After 6 comparisons and 1 swap

3	17	29	11	7	5	31

3. After 5 comparisons and 1 swap

3	5	29	11	7	17	31

4. After 4 comparisons and 1 swap

3	5	7	11	29	17	31

5. After 3 comparisons and 0 swap

3	5	7	11	29	17	31

6. After 2 comparisons and 1 swap

3	5	7	11	17	29	31

7. After 1 comparison and 0 swaps

3	5	7	11	17	29	31