Homework Assignment #2

Due: Monday Jan 31 (11:55pm)

(Total 40 marks)

CMPUT 204
Department of Computing Science
University of Alberta

Problem 1. (10 marks) True or False? Justify your answer briefly.

a. $n^3 + 2n^2 \in \omega(0.1n^{3.1} + n)$

Solution: This is false. $0.1n^{3.1} + n$ grows faster than $n^3 + 2n^2$.

b. $n \log^5 n \in \Theta(n^{\frac{5}{4}})$

Solution: This is false. We can show $n \log^5 n \in o(n^{\frac{5}{4}})$. Since $\log n \in o(n^\epsilon)$ for any $\epsilon > 0$, thus $\log n \in o(n^{\frac{1}{20}})$, and therefore $\log^5 n \in o((n^{\frac{1}{20}})^5) = o(n^{\frac{1}{4}})$ and $n \log^5 n \in o(n^{\frac{5}{4}})$.

c. $2^{n \log n} \in \Theta(4^n)$

Solution: This is false. Applying log to both sides we get $(n \log n)$ vs $(n \log 4)$. The former is clearly asymptotically larger.

d. For any constant $a > 1$: $a^{n+1} \in o(n!)$.

Solution: This is true. The factorial sequence starts with small numbers, but when n is sufficiently large, $n \times (n-1) \cdots \times 1$ will be greater than $a \times \cdots \times a$ ($n+1$ of a's multiplied), and the gap grows bigger along with increase of n.

e. $\sqrt{\log n} \in o(\log(\sqrt{n}))$

Solution: This is true. $\sqrt{\log n} = 2^{\log(\sqrt{\log n})} = 2^{0.5 \log\log(n)}$ while $\log \sqrt{n} = 0.5 \log n = 2^{\log(0.5 \log n)} = 2^{\log(0.5) + \log\log n} = 2^{\log\log(n) - 1}$. The latter function grows faster.

Problem 2. (12 marks) Order the following list of functions by increasing big-Oh growth rate. Group together those functions that are big-Theta of one another.

$3^{\sqrt{n}}$	$2^{2+1/n}$	$\log n / \log\log n$	$\log n$	$e^{\ln n}$
2^{2^n}	4^n	n^{100}	$1/\log n$	$4n^{3/2}$
$3^{\sqrt{\log n}}$	$5(n + 1/n)$	$n \log^3 n$	$2^{n^2 \log n}$	$\log(n!)$
$n!$	n^3	$n^2 \log n$	$4^{\log n}$	$\sqrt{\log\log n}$

Solution: $1/\log n$, $2^{2+1/n}$, $\sqrt{\log\log n}$, $\log n / \log\log n$, $\log n$, $3^{\sqrt{\log n}}$, $\{e^{\ln n}, 5(n+1/n)\}$, $\log(n!)$, $n \log^3 n$, $4n^{3/2}$, $4^{\log n}$, $n^2 \log n$, n^3, n^{100}, $3^{\sqrt{n}}$, 4^n, $n!$, $2^{n^2 \log n}$, 2^{2^n}

Problem 3. (10 marks) Prove or disprove each of the following statements:

a. For any functions $f, g : \mathbb{N} \longrightarrow R^+$, either $f \in O(g)$ or $g \in O(f)$.

Solution: This is false. Suppose that

$$f(n) = \begin{cases} 0 & \text{if } n \text{ is odd} \\ n & \text{if } n \text{ is even} \end{cases} \qquad g(n) = \begin{cases} n & \text{if } n \text{ is odd} \\ 0 & \text{if } n \text{ is even} \end{cases}$$

It is easy to see that for any constants $c, d > 0$, even if $n > d$, always $f(n) > c.g(n)$ if n is even and $g(n) > c.f(n)$ if n is odd.

b. For any functions $f, g : \mathbb{N} \longrightarrow R^+$, if $f(n) > g(n)$ for all $n > 0$ then $f(n) + g(n) \in \Theta(f(n))$.

Solution: This is true. It follows that for all $n > 0$, $f(n) + g(n) \leq 2f(n)$ and therefore, with $n_0 = 1$ and $c = 2$ $f(n) + g(n) \in O(f(n))$. Also, we know that $f(n) + g(n) >= f(n)$ therefore, with $n_1 = 1$ and $c = 1$, $f(n) + g(n) \in \Omega(f(n))$; together we get it is in $\Theta(f(n))$.

c. For any functions $f, g : \mathbb{N} \longrightarrow R^+$, if $f(n) \in \Theta(n)$ and $g(n) \in \Theta(n)$ then $2^{f(n)} \in \Theta(2^{g(n)})$.

Solution: This is not true. For instance, let $f(n) = 2n$ and $g(n) = n$. Then $f(n), g(n) \in \Theta(n)$. But $2^{f(n)} = 2^{2n} = (4^n)$ and $2^{g(n)} = 2^n$, and we know $2^n \in o(4^n)$.

d. For any function $f : \mathbb{N} \longrightarrow R^+$, $f(n) \in \Theta((f(n))^2)$.

Solution: This is not true, e.g., if $f(n) = \frac{1}{n}$.

Problem 4. (10 marks) Suppose you are given a set of small boxes, numbered 1 to n, identical in every aspect except that each of the first i contains a pearl whereas the remaining $n - i$ are empty. In the first part of this question, you are given two magic wands that can each test if a box is empty or not in a single touch, except that a wand disappears if you test it on a box that is empty. Show that, without knowing the value of i, you can use the two wands to determine all the boxes containing pearls using no more than $O(\sqrt{n})$ wand touches.

a. **We here sketch a solution for this part of the question.**

   We will fix number k later. The idea is first use one wand on boxes $1, k, 2k, 3k, \ldots$ The smallest i for which the wand burns on box ik indicates that the first empty box is among $(i-1)k + 1, \ldots, ik$. Now we use the second wand sequentially from $(i-1)k + 1$ to $ik - 1$ to find it. The total number of touches will be at most: $n/k + k$; n/k is the number of boxes for the first wand and k for the second one.

   Now, you are to complete this solution by describing how you choose k and explaining why your choice leads to an algorithm that has the desired efficiency (i.e., to determine all the boxes containing pearls using no more than $O(\sqrt{n})$ wand touches).

   Answer: If we choose k to be (about) $\sqrt{n}$ then we have $n/\sqrt{n} + \sqrt{n} = 2\sqrt{n} \in O(\sqrt{n})$ touches. For other choices, say we let $k = n^{0.4}$, then $n/n^{0.4} + n^{0.4} \in \Theta(n^{0.6})$, but we have $\sqrt{n} \in o(n^{0.6})$, so this choice doesn't work.

b. Now suppose you are given three magic wands. How would you extend the above algorithm to obtain an even more efficient algorithm? Describe your algorithm, give an upper bound in running time (in terms of the number of wand touches) using the big-O notation, and provide a brief analysis.

   Answer: Apply the same idea to the use of the second wand. E.g., for the use of the first wand, we can choose $k_0 = n^{0.6}$, for the use of the second wand, let 's choose $k_1 = n^{0.3}$, then the running time is $n/n^{0.6} + n^{0.6}/n^{0.3} + n^{0.3} \in O(n^{0.4})$. Can you do better than this?

c. We should all be familiar with binary search: Given a sorted list of items, it works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one. In this question, the given list of boxes can be considered sorted: non-empty boxes proceed empty boxes. Give an algorithm to find the first empty box in $O(\log n)$ time independent of how many magic wards may be used. Then, answer the following questions: by your algorithm, in the worst-case how many magic wards is required (express your answer using the big-O notation)? What about the best-case? Justify your answers briefly.

Answer: Just apply binary search in a straightforward way. Worst-case: $O(\log n)$, best-case: $O(1)$.