EL-GY 9343 Homework 1

Yihao Wang
yw7486@nyu.edu

1. Prove the following properties of asymptotic notation:

(a) $n = \omega(\sqrt{n})$

For any given constant $c > 0$, there always exists a $n_0 = c^2$ such that for any $n \geq n_0$, we have $n > c \cdot \sqrt{n}$. Thus $n = \omega(\sqrt{n})$. ■

(b) If $f(n) = \Omega(g(n))$, and $h(n) = \Theta(g(n))$, then $f(n) = \Omega(h(n))$

With the definition of Θ notation, we have $h(n) = \Theta(g(n)) \implies g(n) = \Theta(h(n)) \implies g(n) = \Omega(h(n))$ and $g(n) = O(h(n))$.

Using the transitivity of Ω notation, we have $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n)) \implies f(n) = \Omega(h(n))$. ■

(c) $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$ (Transpose Symmetry property)

If $g(n) = \Omega(f(n))$, there exist constants $c > 0$ and $n_0 \geq 0$ such that $g(n) \geq c_0 \cdot f(n)$ for all $n \geq n_0$, which also indicates that $\forall n \geq n_0$, $f(n) \leq 1/c_0 \cdot g(n) = c_1 \cdot g(n)$, where $c_1 = 1/c_0 > 0$. Thus we have $f(n) = O(g(n))$. ■

2. Indicate, for each pair of expressions (A, B) in the table below, whether A is O, o, Ω, ω, or Θ of B. Assume that $k \geq 1$, $\epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with "yes" or "no" written in each box.

	A	B	O	o	Ω	ω	Θ
a	$lg^k n$	n^ϵ	yes	yes	no	no	no
b	n^k	c^n	yes	yes	no	no	no
c	$\sqrt{n}$	$n^{\sin n}$	no	no	no	no	no
d	2^n	$2^{n/2}$	no	no	yes	yes	no
e	$n^{\lg c}$	$c^{\lg n}$	yes	no	yes	no	yes
f	$\lg(n!)$	$\lg(n^n)$	yes	no	yes	no	yes

3. For each algorithm check all the possible formulas that could be associated with it in the following table.

(a) $4(5^{3\ \log_5 n}) + 12n + 9527 \sim \Theta(n^3 + n)$

(b) $\sqrt[5]{3n!} \sim \Theta(n^{n/5})$

(c) $\frac{1}{6}(5^{\log_{16} n})^2 + 4n + 17 \sim \Theta(n^2 + n)$

(d) $3n\ \log_3 n + (\log_2 n)^3 \sim \Theta(n\log n + (\log n)^3)$

(e) $\log_4 \log_2 n + 61 \sim \Theta(\log\log n)$

(f) $2^{5\ \log_4 n} \sim \Theta(n^{5/2})$

(g) $(\log_2 n)^2 + \log_3 \log_3 n \sim \Theta((\log n)^2 + \log\log n)$

	a	b	c	d	e	f	g
A1					✓		✓
A2				✓			
A3	✓		✓			✓	
A4			✓	✓	✓	✓	✓
A5	✓		✓		✓		

4. We want to check if there is an element occurs more than $n/2$ times in an array containing n elements, assuming only equality checks are allowed.

(a) Algo.1 is part of the required algorithm. What is the time complexity now?

It is $O(n)$ now.

(b) Make the algorithm complete by adding a few more lines to substitute the underlined text. Your modification should NOT change the time complexity. Be sure to return things as indicated.

Algorithm 1 Find majority element in an array

Input: $L[1, ..., n]$ as input list containing n real numbers

Output: True or False. If true, also returning the majority element

```
c = 0, v = L[1]
for i = 1, 2, ..., n do
    if c == 0 then
        v = L[i]
    end if
    if v == L[i] then
        c = c + 1
    else
        c = c - 1
    end if
end for
c = 0
for i = 1, 2, ..., n do
    if L[i] == v then
        c = c + 1
    end if
    if c == n//2 + 1 then
        return True, v
    end if
end for
return False
```