# CS208 Theory Assignment 2

12312110 李轩然

**DDL: Mar.18 20:50**

## Chapter 2 Exercise 1

### Description

Suppose you have algorithms with the five running times listed below. (Assume these are the exact running times.) How much slower do each of these algorithms get when you (a) double the input size, or (b) increase the input size by one?

**(a)** $n^2$
**(b)** $n^3$
**(c)** $100n^2$
**(d)** $nlogn$
**(e)** $2^n$

### Analysis

- **(a)**
  Suppose $N = n^2$.
  (a) double the input size, then $N_a = (2n)^2 = 4n^2$, thus the algorithm will get slower by $4$ times.
  (b) increase the input size by one, then $N_b = (n+1)^2 = n^2 + 2n + 1$, thus the algorithm will get slower by an addition of $2n + 1$.

- **(b)**
  Suppose $N = n^3$.
  (a) double the input size, then $N_a = (2n)^3 = 8n^3$, thus the algorithm will get slower by $8$ times.
  (b) increase the input size by one, then $N_b = (n+1)^3 = n^3 + 3n^2 + 3n + 1$, thus the algorithm will get slower by an addition of $3n^2 + 3n + 1$.

- **(c)**

Suppose $N = 100n^2$.
(a) double the input size, then $N_a = 100(2n)^2 = 400n^2$, thus the algorithm will get slower by $4$ times.
(b) increase the input size by one, then
$N_b = 100(n+1)^2 = 100n^2 + 200n + 100$, thus the algorithm will get slower by an addition of $200n + 100$.

- **(d)**
  Suppose $N = nlogn$.
  (a) double the input size, then $N_a = 2nlog(2n) = 2nlog2 + 2nlogn$. When $n$ is very large, $2nlog2$ can be ignored compared to $2nlogn$, thus the algorithm will get slower by $2$ times.
  (b) increase the input size by one, then $N_b = (n+1)log(n+1)$. It seems that $N_b$ can not be simplified, thus the algorithm will get slower by an addition of $(n+1)log(n+1) - nlogn$.

- **(e)**
  Suppose $N = 2^n$.
  (a) double the input size, then $N_a = 2^{2n}$, thus the algorithm will get slower by its squared, $2^n$.
  (b) increase the input size by one, then $N_b = 2^{n+1} = 2 * 2^n$, thus the algorithm will get slower by $2$ times.

# Chapter 2 Exercise 5

## Description

Assume you have functions $f$ and $g$ such that $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

**(a)** $log_2 f(n)$ is $O(log_2 g(n))$.
**(b)** $2^{f(n)}$ is $O(2^{g(n)})$.
**(c)** $f(n)^2$ is $O(g(n)^2)$.

## Analysis

**(a) False**

Because $f(n)$ is $O(g(n))$, there exists constants $c > 0$ and $n_0$, such that for all $n > n_0$, we have:

$$f(n) \leq c \cdot g(n)$$

$$log_2 f(n) \leq log_2 c + log_2 g(n)$$

It seems that we lost a constant before $log_2 g(n)$, thus (a) might be false. Here is a **counterexample**:

$$f(n) = 4, g(n) = 1$$

then

$$log_2 f(n) = 2, O(log_2 g(n)) = O(0)$$

At this time, $4$ is $O(1)$, but we can not find a constant $c$ such that $2 < c \cdot 0$, thus (a) is false.

**(b) False**

Because $f(n)$ is $O(g(n))$, there exists constants $c > 0$ and $n_0$, such that for all $n > n_0$, we have:

$$f(n) \leq c * g(n)$$

$$2^{f(n)} \leq 2^{c \cdot g(n)} = 2^c \cdot 2^{g(n)}$$

$$2^{f(n)} \leq C \cdot 2^{g(n)}$$

$C$ is another constant. Thus we can easily give a **counterexample**:

$$f(n) = 2n, g(n) = n$$

then

$$2^{f(n)} = 4^n, O(2^{g(n)}) = O(2^n)$$

When $n$ is very large, we can not find a constant $C$ such that $4^n \leq C \cdot 2^n$ (or $2^n \leq C$, more clearly), thus (b) is false.

**(c) True**

Because $f(n)$ is $O(g(n))$, there exists constants $c > 0$ and $n_0$, such that for all $n > n_0$, we have:

$$f(n) \leq c \cdot g(n)$$

$g(n)$ must be positive, thus we can square both sides:

$$f(n)^2 \leq c^2 \cdot g(n)^2$$

$c^2$ is still a positive constant, and the inequality holds for every $n > n_0$, thus $f(n)^2$ is $O(g(n)^2)$, (c) is true.