CS208 · Algorithm Design and Analysis

何泽安 (He Zean)
12011323

# Assignment 2

Mar. 8, 2022

## Ch.2 - Ex.1

**(a)** $n^2$

**Double the input size** $(2n)^2 = 4n^2$, it becomes as 4 times slow as before.

**Increase the input size by one** $(n+1)^2 = n^2 + 2n + 1$, it uses $2n+1$ more time, or increases the time in a rate of $\frac{2n+1}{n^2}$.

**(b)** $n^3$

**Double the input size** $(2n)^3 = 8n^3$, it becomes as 8 times slow as before.

**Increase the input size by one** $(n+1)^3 = n^3 + 3n^2 + 3n + 1$, it uses $3n^2 + 3n + 1$ more time, or increases the time in a rate of $\frac{3n^2+3n+1}{n^3}$.

**(c)** $100n^2$

**Double the input size** $100(2n)^2 = 400n^2$, it becomes as 4 times slow as before.

**Increase the input size by one** $100(n+1)^2 = 100n^2 + 200n + 100$, it uses $200n + 100$ more time, or increases the time in a rate of $\frac{2n+1}{n^2}$.

**(d)** $n \log n$

**Double the input size** $(2n)\log(2n) = 2(n\log n) + (2\log 2)n$, it doubles the time and add $(2\log 2)n$ more time, a.k.a. it increases the time in a rate of $1 + \frac{2\log 2}{\log n}$.

**Increase the input size by one** $(n+1)\log(n+1)$, it uses $\log(n+1) + n\log\frac{n+1}{n}$ more time, or increase the time in a rate of $(\log(n+1) + n\log\frac{n+1}{n})/(n\log n)$.

**(e)** $2^n$

**Double the input size** $2^{2n} = (2^n)^2$, now its running time can be considered as the square of the previous one.

**Increase the input size by one** $2^{n+1} = 2 \cdot 2^n$, it becomes as 2 times slow as before.

## Ch.2 - Ex.5

**(a) False** Assume that $f(n) = 2$, $g(n) = 1$, here we can find $k = 0, c = 2$ s.t. $\forall x > k, f(x) \le c \cdot g(x)$, a.k.a. $f(n) = O(g(n))$. However, $\log_2 f(n) = 1$, $\log_2 g(n) = 0$ for all $n$, it's trivial that we cannot find a fair $\{k, c\}$ s.t. $\forall x > k, f(x) \le c \cdot g(x)$, which means $\log_2 f(n)$ is not $O(\log_2 g(n))$.

**(b) False** Assume that $f(n) = 2n$, $g(n) = n$, then $2^{f(n)} = 4^n$ while $2^{g(n)} = 2n$, we cannot find a pair of $\{k, c\}$ s.t. $\forall x > k, f(x) \le c \cdot g(x)$, since for a certain $n$, should have $c \ge 2^n$.

**(c) True** We already have $k_1, c_1$ s.t. $\forall x > k_1, f(x) \le c_1 \cdot g(x)$, then for the same pair of $k, c$, we still can say $\forall x > k_1, f(x)^2 \le (c_1 \cdot g(x))^2$. A.k.a. we find a pair $k_2 = k_1, c_2 = c_1^2$ s.t. $\forall x > k_2, f(x)^2 \le c_2 \cdot g(x)^2$, which is the definition of big-O notation, say $f(n)^2 = O(g(n)^2)$.