FUNCTIONS' GROWTH

Asymptotic Efficiency

- Asymptotic efficiency refers to the analysis of how the performance of an algorithm growth as the size of the input data increases (towards infinity)
- To describe asymptotic efficiency, we use asymptotic notations:
  - Big O notation: upper-bound, worst-case scenario
  - Big Ω notation: lower-bound, best-case scenario
  - Big Θ notation: tight-bound, where both upper and lower are defined.

Comparing 2 algorithms asymptotic efficiency

Consider 2 sorting algorithms A and B on the same set of data of size n. Denote $f(n)$ and $g(n)$ to be the upper bounds of A and B respectively. Which is the better upper bond?

We can compare by solving

$$\boxed{L = \lim_{n \to \infty} \frac{f(n)}{g(n)}}$$

The result can either be:

- $L > 0$: $f(n)$ and $g(n)$ grows at the same rate

  or $L < \infty$ $\Leftrightarrow$ $f(n)$ is $\Theta(g(n))$ or $g(n)$ is $\Theta(f(n))$

  $\begin{cases} f(n) \text{ is } O(g(n)) \\ f(n) \text{ is } \Omega(g(n)) \end{cases}$ $\qquad$ $\begin{cases} g(n) \text{ is } O(f(n)) \\ g(n) \text{ is } \Omega(f(n)) \end{cases}$

  $\Rightarrow$ both functions are tightly bound to each other

- $L = 0$: $f(n)$ grows significantly slower than $g(n)$

  $\Leftrightarrow$ $f(n)$ is $o(g(n))$ or $g(n)$ is $\omega(f(n))$

  $f(n)$ is upper-bounded by $g(n)$ $\qquad$ $g(n)$ is lower-bounded by $f(n)$

  $\Leftrightarrow$ algorithm A is more efficient than B

- $L = \infty$: $f(n)$ grows significantly faster than $g(n)$
  $\Leftrightarrow$ $g(n)$ is $o(f(n))$ or $f(n)$ is $\omega(g(n))$
  $g(n)$ is upper-bounded by $f(n)$ — $f(n)$ is lower-bounded by $g(n)$

  $\Leftrightarrow$ algorithm B is more efficient than A

Big O

We can write: $f(n) = O(g(n))$
If there exist $n_0 > 0$ and $c > 0$ such that:
$$f(n) \leq c \cdot g(n), \quad n \geq n_0$$

We say $g(n)$ is the "fuzzy" upper-bound of $f(n)$. "Fuzzy" because there is a possibility that $f(n)$ grows the same rate as $g(n)$

Little O

We can write: $f(n) = o(g(n))$
if there exist $n_0 > 0$ and $c > 0$ such that
$$f(n) < c \cdot g(n), \quad n \geq n_0$$

The only difference between big O and little O is the "strictness". In little O, there is no possibility of $f(n)$ ever match the grow rate of $g(n)$.

Big Omega

We can write: $f(n) = \Omega(g(n))$
If there exist $n_0 > 0$ and $c > 0$ such that:
$$f(n) \geq c \cdot g(n), \quad n \geq n_0$$

Little Omega

definition can be derived using same logic as Big O / Little O

Big Theta

We can write: $f(n) = \Theta(g(n))$

If there exist $n_0 > 0$ and $c_1, c_2 > 0$ such that:

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \quad n \geq n_0$$

$$f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n))$$

Proof:

$$f(n) = \Theta(g(n)) \Rightarrow \begin{cases} c_1 \cdot g(n) \leq f(n) \\ f(n) \leq c_2 \cdot g(n) \end{cases}$$

$$\iff \begin{cases} g(n) \geq \frac{1}{c_1} f(n) \\ \frac{1}{c_2} f(n) \geq g(n) \end{cases} \Rightarrow \frac{1}{c_1} f \leq g \leq \frac{1}{c_2} f$$

So $g(n) = \Theta(f(n))$

The "$\sim$" symbol

When we say: $f(n) \sim g(n)$

$f(n)$ is asymptotic equivalence to $g(n)$

$f(n) \sim g(n)$ when:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$$

Can think of "$\sim$" symbol as a "stricter" form of Theta Θ notation, similar to how Big O / Little o, Big Omega / Little Omega works.

$$f(n) \sim g(n) \Rightarrow f(n) = (1 + o(1)) \cdot g(n)$$

Proof:

$$f(n) = (1 + o(1)) \cdot g(n)$$

$$\iff \frac{f(n)}{g(n)} = 1 + o(1) \quad (1)$$

Because o(1) refers to a function that approaches 0 as $n \to \infty$.

In other words, if $h(n) = o(1)$, then:

$$\lim_{n \to \infty} h(n) = 0$$

Denote $k(n) = 1 + h(n)$, then as $\lim_{n \to \infty} h(n) = 0$:

$$\lim_{n \to \infty} k(n) = \lim_{n \to \infty} (1 + h(n)) = 1$$

Substitute $k(n) = 1 + h(n)$ backs to equation (1), we can say that:

$$\lim_{n \to \infty} k(n) = \lim_{n \to \infty} 1 + h(n)$$

$$= \boxed{\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1}$$

Definition of $f(n) \sim g(n)$

<u>Example:</u>

The Door-in-Wall Puzzle

You are facing a wall that extends infinitely in both directions. There is a gap somewhere in the wall in one of the direction. Comes up with an algorithm that guarantee to find the gap within a finite number of steps. In other words, what is Θ-bound on the worst case number of steps taken by this algorithm?

<u>Solution:</u>

- Algorithm:

```
for i : 1 -> ∞ :        # number of steps
    for each direction:
        for pos: 0 -> i :
            if see gap
                return pos
```

- Θ-bound on worst case number of steps:
    - Let $T(n)$ describe time complexity of the algorithm, where n is distance from starting point to the gap
    - Total steps taken after k iterations is:
    $$T(k) = 2(1 + 2 + \dots + \ )$$
    $$= k^2 + k$$
    - If gap is located at step n, then it will takes $k = n$ of exploration, substitute this in $T(k)$ we have:
    $$T(n) = n^2 + n \sim n^2$$

<u>Conclusion:</u> $T(n) = O(n^2)$

Better Solution:

- Algorithm:

```
for i : 1 → ∞ :
    for each direction:
        for pos: 0 → 2^i :
            if see gap
                return pos
```

- Θ -bound worst case number of steps
  - Let $T(n)$ describe the time complexity of the algorithm
  - Total steps taken after k iterations:

$$T(k) = 2\left(2^0 + 2^1 + 2^2 + \dots + 2^k\right)$$
$$= 2\left(\frac{2^{k+1} - 1}{2 - 1}\right)$$
$$= 2\left(2^{k+1} - 1\right)$$
$$= 2^{k+2} - 2$$

  - If gap is located at step n, then it will take $k = \log_2 n$ of exploration. Substitute this in $T(k)$ we have:

$$T(n) = 2^{\log_2 n + 2} - 2$$
$$\sim 2^{\log_2 n + 2}$$
$$= 2^{\log_2 n} \cdot 2^2$$
$$= n \cdot 2^2 \qquad \langle a^{\log_a b} = b \rangle$$
$$= 4n$$

Conclusion: $T(n) = O(n)$

$\Rightarrow$ Better than $O(n^2)$

Recap on logarithmic properties

Definition: $\log_a b = k$ < divides b by a k times to reach 1 >

$\Leftrightarrow \quad a^k = b$ < a times itself k times to reach b >

Another definition:

$$a^{\log_a b} = b \qquad \text{or} \qquad b^{1/\log_a b} = a$$

$\Rightarrow$ Intepret as " taking $\log_a b$ steps in base a to get to b ".

Important properties:

- Product property:

$$\log_a (mn) = \log_a (m) + \log_a (n)$$

- Quotient property:

$$\log_a \left(\frac{m}{n}\right) = \log_a (m) - \log_a (n)$$

- Power property:

$$\log_a (m^n) = n \cdot \log_a (m)$$

- Change of Base formula:

$$\log_a b = \frac{\log_k b}{\log_k a}$$

- Special values:

$$\log_a 1 = 0 \qquad \log_a a = 1 \quad \text{for} \begin{cases} \forall a > 0 \\ a \neq 1 \end{cases}$$

$$\log_a 0 = 1$$

- Cancel out:

$$\log_b b^a = a \quad \text{if } b > 0 \text{ and } b \neq 1$$

- Monotonicity:

The logarithm function is monotonically increasing. If:

$$a < b \quad \text{for any } a, b \geq 0$$

Then: $\log(a) < \log(b)$

- Inequality:

If $a \geq 2^b$, then:

$$b \leq \log_2 a$$

Recap on exponent properties

- Positive number expression:

$$2 = e^{\ln(2)}$$

or more generally, $a > 0$:

$$a = e^{\log_e(a)}$$

- Product:
$$a^m \cdot a^n = a^{(m+n)}$$

- Quotient:
$$\frac{a^m}{a^n} = a^{(m-n)}$$

- Power of power:
$$(a^m)^n = a^{m \cdot n}$$

- Power of product:
$$(ab)^n = a^n \cdot b^n$$

- Power of quotient:
$$\left(\frac{a}{b}\right)^n = \frac{a^n}{b^n}$$

- Negative exponent:
$$a^{-n} = \frac{1}{a^n}$$