5/12/2021

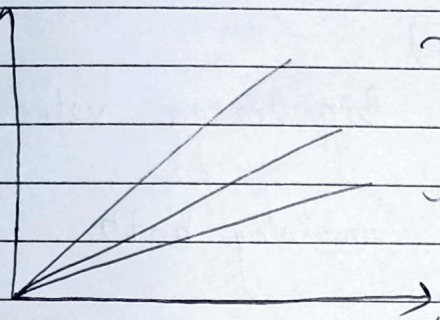
Time and Space Complexity

Time Complexity != Time taken

☆ Function that gives us the relationship about how the time will grow as the input grows.

What do we consider when thinking about complexity:

① Always look for worst case complexity.
② Always look at complexity for large / ∞ data
③



☆ Even two value of actual time is different they are all growing linearly.
☆ We don't care about actual time.
☆ This is why, we ignore all constraints

④ $O(N^3 + \log N)$

1 million ⇒ $((1\text{mil})^3 + \log(1\text{mil}))$

$(1\text{mil})^3 + \underbrace{60}_{\text{very small, hence ignore}}$

Always ignore less dominating terms.

$O(1) < O(\log(N)) < O(N) < O(N\log(N)) < O(2^n)$

Big - Oh Notation

Defⁿ :- Upper bound.

Algo can be solved in less complexity but will not exceed Big-Oh value

Big Omega : Opposite of Big Oh

Defⁿ : $\Omega(N^3)$

Lower bond

Never Less than Big Omega value.

Theta Notation: Combining both

$O(N^2)$ ⇓

Both upper bound & Lb is N^2

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

little o notation :

- ☆ This is also giving upper bound.
- ☆ Lose upper bound.

Big Oh	little Oh
$f = O(g)$	$f = O(g)$
$f \leq g$	$f < g$ (strictly slower)

little Omega :

$$f = \omega(g) \Rightarrow$$

Big Omega	Little Omega
$f = \Omega(g)$	$f = \omega(g)$
$f \geq g$	$f > g$

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} = \infty.$$

Eg.

- ```
  for (i = n , i > 0 , i = i - c) {
      // some const work }
  ```

$n=10$ $c=2$	$n=11$, $c=2$	$\therefore TC = O(n/c + 1)$
$i = 10, 8, 6, 4, 2$	$i = 11, 9, 7, 5, 3, 1$	$= O(n)$

- ```
  for (i = 1, i < n, i = i * c ) {
      O(1) work
  }
  ```

$n = 32$, $c = 2$	$n = 33$, $c = 2$
$i = 1, 2, 4, 8, 16$	$i = 1, 2, 4, 8, 16, 32$

$1, c, c^2, c^3 \ldots\ldots c^{k-1}$

$$c^{k-1} < n$$

$$k - 1 < \log n$$

$$k < \log n + 1$$

$$k = O(\log n)$$

```
for (i = n, i > 1, i = i / c) {
    O(1) work
}
```

n = 32, c = 2
32, 16, 8, 4, 2

T.C = O(log(n))

```
for (i = 2, i < n, i = Math.pow(i, c)) {
    O(1) work
}
```

$2, 2^c, (2^c)^c$

$2, 2^c, 2^{c^2} \cdots 2^{c^{k-1}}$

$2^{c^{k-1}} < n$

$c^{k-1} < \log(n)$

$k - 1 < \log_c(\log(n))$

$T.C = O(\log(\log n))$

```
for (i = 0, i < n, i++) {
    for (j = 1, j < n, j = i * 2) {
        O(1) work
    }
}
```

T.C = O(n * log(n))