Time and Space complexity

(Time complexity)

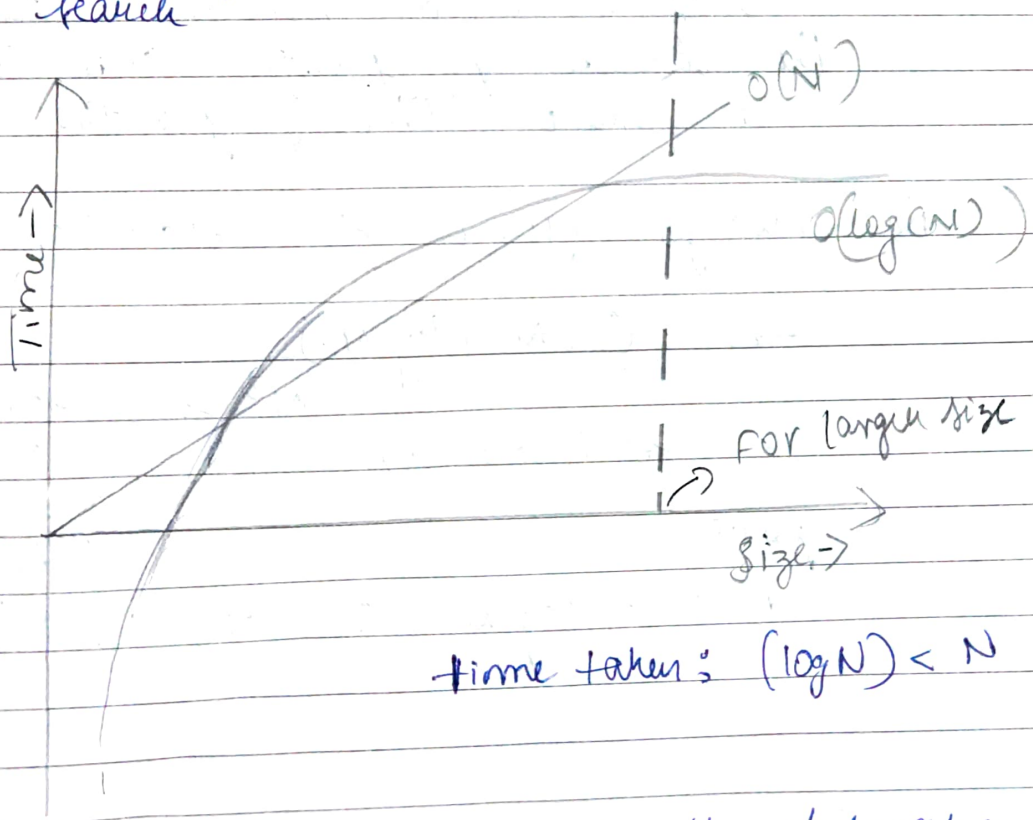
⇒ What is time complexity?

It is a function that gives us the relationship about how the time will grow as the input size grows.

REMEMBER: Time complexity ≠ Time taken

⇒ Why do we need time complexity?

Example of linear search ($O(N)$) and binary search ($O(\log N)$)



Time taken: $(\log N) < N$

We usually ignore smaller data values.

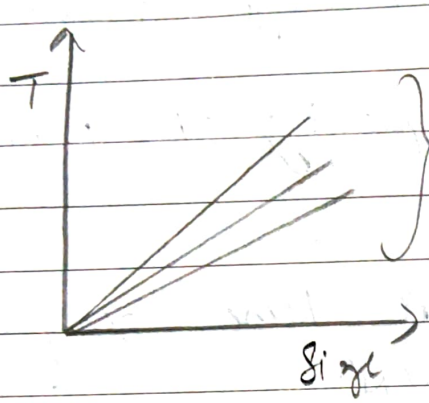
∴ Binary search is better.

→ Considerations

i) Always look for worst case complexities.

ii) Always look at complexity for large / infinite data.

iii)



* Ignore the constants, as they are not required. We just look at the trend of growth and not the exact time of growth.

iv) Always ignore less dominating terms.

ex: $O(N^3 + \log N)$

$N^3 >>> \log N$.

So ignore $\log N$.

less T.C ↑ high T.C ↑

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

→ Big - O Notation

The time complexity specified by a Big-O-Notation, will never exceed a given value.

For ex: $O(N^3)$ → upper bound

The time complexity to run this program may vary from $O(1)$ to $O(N^3)$ but will never exceed $O(N^3)$

Mathematically explaining:

$$f(n) = O(g(n))$$

$$\underbrace{O(n^3)}_{g(n)} = O(6n^3 + 3n + 5) \text{ (example)}$$
$$f(n)$$

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

$$\lim_{n \to \infty} \frac{6n^3 + 3n + 5}{n^3}$$

$$= 6 + \frac{3}{n^2} + \frac{5}{n^3} \Rightarrow 6 + \frac{3}{\infty} + \frac{5}{\infty}$$

$$= 6 //$$

$$6 < \infty$$

→ Big omega Notation:

It is exactly opposite of Big-O.

That is, it takes minimum value given to execute a code.

$\Omega(N^3)$ → lower bound.

→ Theta Notation:

Combining both Big-O and Big omega.

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

→ Little-O notation:

If $f = O(g)$

$f < g$ (not equal to)

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

→ Little - Ω notation :

If $f = \Omega(g)$

$f > g$ (not giving the lower bound)

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

(Space complexity)

⇒ What is auxiliary space ?

Auxiliary space is the extra space or temporary space used by an algorithm.

Space complexity = Input size + Auxiliary space.

(Analysis of recursive programs)

→ Space complexity = Height of tree path

[tree formed by recursive path]

Two types of recursions:

* Linear : Ex → F(N) = F(N-1) + F(N-2) (Fibonacci)
* Divide and conquer : F(N) = F(N/2) + O(1) (Binary search)

→ Divide and conquer :

Form:

$$T(x) = a_1 T(b_1 x + \varepsilon_1(x)) + a_2 T(b_2 x + \varepsilon_2(x)) + \dots + a_k T(b_k x + \varepsilon_k(x)) + g(x)$$

Finding the complexity of a given D&C form :

i) Plug and Chug (not needed)

ii) Master's theorem (not needed)

iii) Akra - Bazzi theorem ✓

$$T(x) = \Theta\left(x^p + x^p \int_1^x \frac{g(u)}{u^{p+1}}\, du\right)$$

where $p = \sum_{i=1}^{k} a_i b_i^p = 1$

Merge sort

Example: $T(N) = 2T(N/2) + c$ $\quad (c = N-1)$

$a_1 = 2$ $\quad g(x) = x - 1$

$b_1 = \frac{1}{2}$

(Trial and error to find p)

$2 \times \left(\frac{1}{2}\right)^p = 1$

$\therefore p = 1$

$$T(x) = \Theta\left(x^1 + x^1 \int_1^x \frac{x-1}{u^2}\, du\right)$$

$$= \Theta\left(x + x \int_1^x \left(\frac{1}{u} - \frac{1}{u^2}\right) du\right)$$

$$= \Theta\left(x + x\left[\int_1^x \frac{du}{u} - \int_1^x \frac{du}{u^2}\right]\right)$$

$$= \Theta\left(x + x\left[\log u + \frac{1}{u}\right]_1^x\right)$$

$$= \Theta\left(x + x\left[\log x + \frac{1}{x} - \Theta 1\right]\right)$$

$$= \Theta(x + x\log x + 1 - 1)$$

$$= \Theta(x\log x + 1)$$

$T(x) = \Theta(x \log x)$ //

NOTE: When solving for P,
check if $p <$ power of $g(x)$.
If yes, the $T(x) = g(x)$ //

→ solving linear homogenous recurrences

Form:

$f(n) = a_1 f(n-1) + a_2 f(n-2) \ldots\ldots a_n f(x-n)$

$f(n) = \sum_{i=1}^{n} a_i f(x-i)$ //

Example $f(n) = f(n-1) + f(n-2)$ ⇒ Fibonacci series

(i) put $f(n) = \alpha^n$ for some constant α

$\alpha^n = \alpha^{n-1} + \alpha^{n-2}$

$\alpha^n - \alpha^{n-1} - \alpha^{n-2} = 0$ (divide by α^n)

$\Rightarrow \alpha^2 - \alpha - 1 = 0$

$$\boxed{\alpha = \frac{1 \pm \sqrt{5}}{2}}$$

(ii) $f(n) = C_1 \alpha_1^n + C_2 \alpha_2^n$

$$f(n) = C_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + C_2 \left(\frac{1-\sqrt{5}}{2}\right)^n \quad -(2)$$

(iii) no of roots = no of answers we already have.

$\therefore f(0) = C_1 + C_2 = 0$

$$C_1 = -C_2$$

$$f(1) = C_1 \left(\frac{1+\sqrt{5}}{2}\right) + C_2 \left(\frac{1-\sqrt{5}}{2}\right)$$

$f(1) = 1$

$$\therefore C_1 \left(\frac{1+\sqrt{5}}{2}\right) - C_1 \left(\frac{1-\sqrt{5}}{2}\right) = 1$$

$$\boxed{C_1 = \frac{1}{\sqrt{5}}} \qquad \boxed{C_2 = \frac{-1}{\sqrt{5}}}$$

putting this in equn (2)

$$\boxed{f(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^n}$$

↳ This is the general formula to find the n^{th} fibonacci number.

$$f(n) = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n \right]$$

↳ as 'n' increases, it tends to zero.

∴ Time complexity = $O\left(\frac{1+\sqrt{5}}{2}\right)^n$

$T(N) = O(1.6180)^n$ //

NOTE: If α is repeated,

then $\alpha^n, n\alpha^n, n^2\alpha^2 \ldots n^{r-1}\alpha^n$ are all solutions.

Non homogenous linear recurrences:

$$f(n) = a_1 f(n-1) + a_2 f(n-2) \cdots + a_k f(n-k) + g(n)$$

(1) Replace $g(n)$ by 0 and solve it like homogenous eqn.

(2) Take $g(n)$ one one side and find particular soln

ex: $f(n) - 4f(n-1) = 3^n$

Guess something that is similar to $g(n)$

ex: $f(n) = C3^n$ //

$$C3^n - 4C3^{n-1} = 3^n$$

$$C = -3$$

particular solution:

$$f(n) = -3^{n+1}$$ //

(3) Add both solutions:

$$f(n) = C_1 4^n + (-3^{n+1})$$

$$f(1) = 1 \Rightarrow C_1 4 - 3^2 = 1$$

$$C_1 = \frac{5}{2}$$

$$f(n) = \frac{5}{2} 4^n - 3^{n+1}$$

How do we guess the particular soln

* if $g(n)$ is exponential, guess of same type.

Ex: $g(n) = 2^n + 5^n$?

$f(n) = a2^n + b3^n$

* If it is polynomial, guess of same degree.

Ex: $g(x) = x^2 - 1$

$f(n) = an^2 + bn + c$ //

ex: $g(n) = 2^n + n$

$\Downarrow$

$f(n) \Rightarrow a2^n + (bn+c)$ //

If the guessing fails,

for ex: $g(n) = a2^n$

then try $\Rightarrow (an+b).2^n$

$\Downarrow$

if this fails

then try $\Rightarrow (an^2+bn+c)2^n$

Question:

$f(n) = 2f(n-1) + 2^n$, $f(0) = 1$.

(1) $f(n) = 2f(n-1)$

$f(n) = \alpha^n$

$\alpha^n - 2\alpha^{n-1} = 0$

$\alpha = 2$

(2) Given particular soln

$g(n) = 2^n$

$f(n) = a2^n$,

$a2^n - 2a2^{n-1} = a2^n$

$a2^n = 2a2^{n-1} + 2^n$ (divide by 2^n)

↓

we would get the soln.

So increase the degree of guessed answer.

$f(n) = (an+b)2^n$

$(an+b)2^n = 2(a(n-1)+b)2^{n-1} + 2^n$

$an+b = an - a + b + 1$

$\boxed{a = 1}$

$\boxed{f(n) = n \times 2^n}$

(3) Assumed.

$f(n) = c_1 2^n + n2^n$

$f(0) = 1, \quad \therefore c_1 \neq 0 = 1$

$c_1 = 1$

$\boxed{f(n) = 2^n + n2^n}$

$T(n) = O(n2^n)$