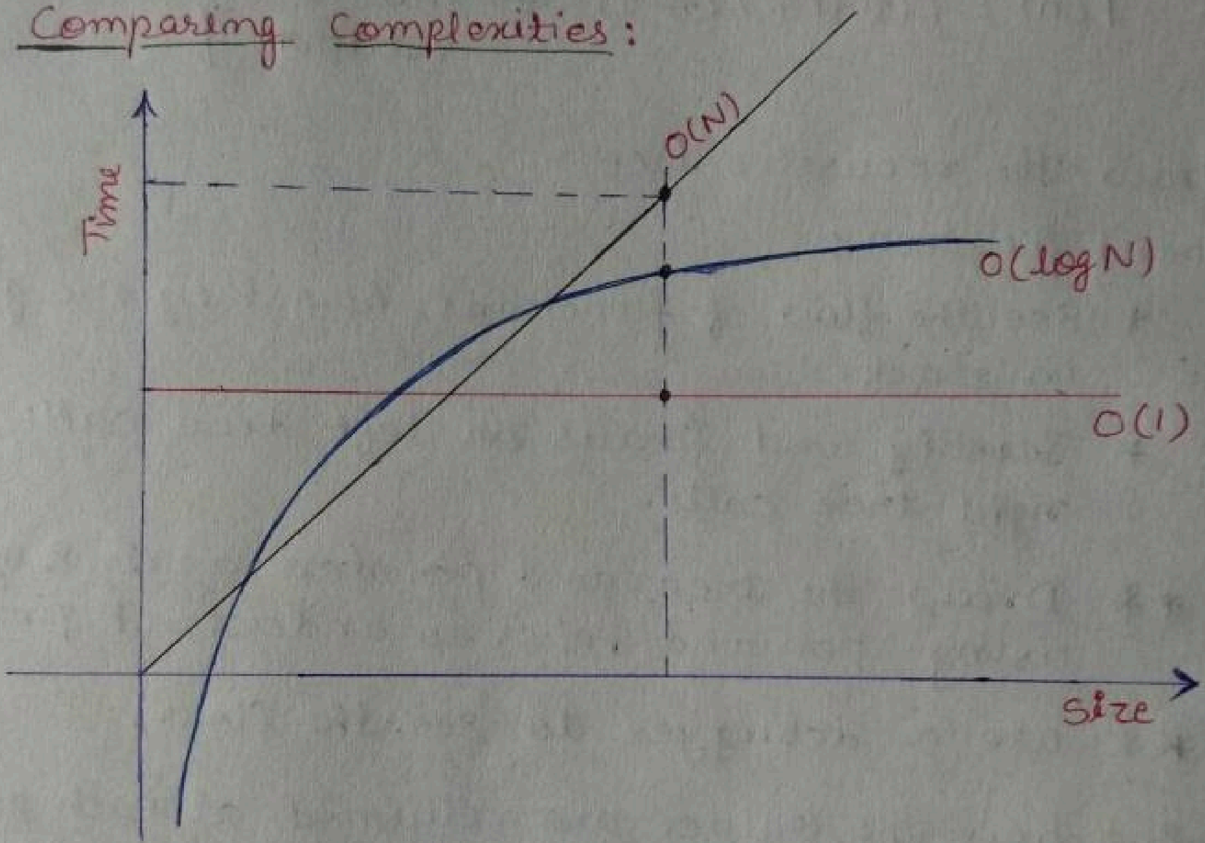
Time and Space Complexity ①

* What is Time Complexity?

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

* Comparing complexities:

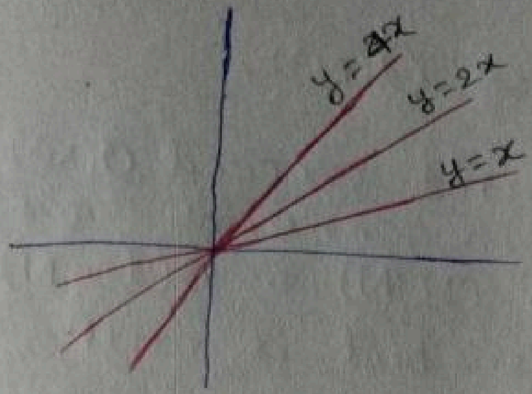


$\boxed{O(1) < O(\log N) < O(N)}$ ← when the input size is larger

* What do we consider when thinking about Complexity (Procedure for Analysing Complexity) :—

1.) Always look for worst case complexity.

2.) Always look at complexity for large/infine data.

3.) 

All these have O(N) complexity (Linear graph)

* Even though value of actual time is different they are all growing linerly.
* We don't care about actual time.
* This is why, we ignore all constants.

4.) Always ignore less dominating terms:

Let's say, we have a complexity like:

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

• from point no. 2 :- Always look at complexity for large data.

$\Rightarrow$ Let's say N = 1 million (sec)

$\Rightarrow$ ((1 million)^3 + log(1 million))

= [(1 million)^3 + 6 secs]

very small as compared to (1 million)^3

// Hence ignore

* Big O Notation:

This is upper bound i.e, it tells about the maximum complexity any algorithm can have

Let's say,

An algo has a complexity of $O(N^3)$.

Means ⇒ the complexity can't exceed $O(N^3)$. It may be solved in constant time, may be solved in $O(N)$ or $O(\log N)$ time or in $O(N^2)$ time, but it will never exceed $O(N^3)$.

Mathematical Definition :

$$f(N) = O(g(N))$$

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

* Big Omega Notation :-

This is lower bound, i.e, it will take atleast a certain amount of time.

Let's say,

An algo has a complexity of $\Omega(N^3)$.

Means :→ The complexity can be more that $\Omega(N^3)$. It will take atleast $\Omega(N^3)$.

Mathematical Definition :

$$\boxed{\lim_{N \to \infty} \frac{f(N)}{g(N)} > \infty}$$

* Theta (Θ) Notation :-

It represent both upper and lower bound.

Mathematical Definition :

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

* Little - O Notation :-

This is a loose upper bound.

## Big - O	Little - O
$\Rightarrow f = O(g)$	$\Rightarrow f = o(g)$
$\Rightarrow f \leq g$	$\Rightarrow f < g$ (strictly slower)

Mathematical Definition

$$\boxed{\lim_{N \to \infty} \frac{f(N)}{g(N)} = 0}$$

* Little - Omega Notation :-

This is a loose lower bound.

## Big Ω	Little Omega (ω)
$\Rightarrow f = \Omega(g)$	$\Rightarrow f = \omega(g)$
$\Rightarrow f \geq g$	$\Rightarrow f > g$

Mathematical Definition

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

* Space Complexity:

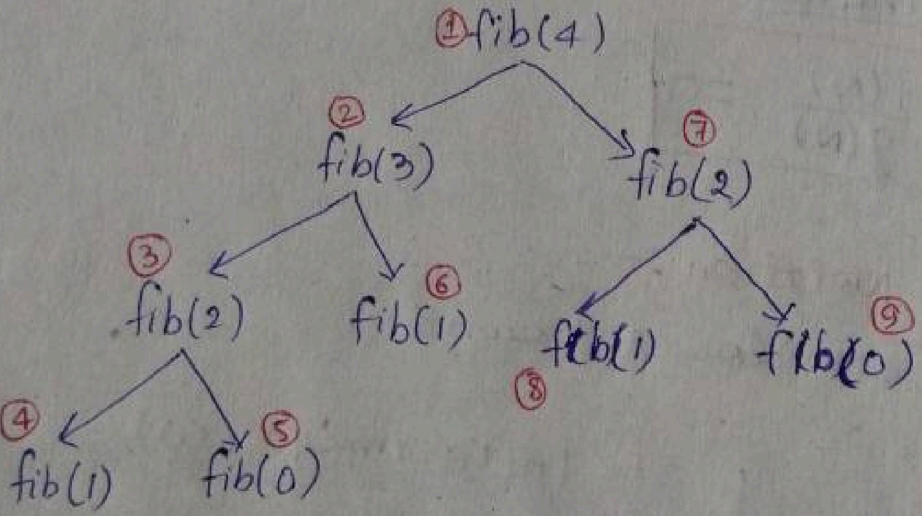
- Total space taken by the algorithm with respect to the input size.
- Space complexity include both Auxiliary space (extra space used by an algo) and space used by input.

* Recursive Algorithms:

In Recursion, we know, functions calls are stored in stack. Hence, recursive programs don't have constant space complexity.

Space complexity of Fibonacci no. –

Let's take fib(4)



flow of execution: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9

NOTE : • At any particular point of time, no two function calls at the same level of recursion will be in the stack at the same time.

• only calls that are interlinked with each other will be in the Stack at the same time.

⇒ At one particular level of tree, there will be only one call that are in stack at a time. It is not possible that all function calls (Here, 9) will be in the stack at the same time.

So, maximum space taken = Height of the tree

$\boxed{\therefore \text{ space complexity} = O(N)}$

* Types of Recursions —

① Linear Recursion , ② Divide and Conquer

* Divide and Conquer Recurrences :

form :-

$$\boxed{T(x) = a_1 T(b_1 x + \Sigma_1(x)) + a_2 T(b_2 x + \Sigma_2(x)) + \cdots + a_k T(b_k x + \Sigma_k(x)) + g(x)}$$

for $\forall\ x \geq x_0$
constant

$T(x)$ = Time complexity of x

Here, $a_1, a_2, \ldots a_k$ = constants
$b_1, b_2, \ldots b_k$ = constant

and $\Sigma_1(x)$ = some function of x

Ex:- $T(N) = T\left(\frac{N}{2}\right) + C$

Here, $a_1 = 1$ and $g(x) = C$

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

$\Sigma_1(x) = 0$

* Best way to get complexity

** Akra - Bazzi formula **

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

Here, $T(x)$ - Time complexity of x

what is P ?

$a_1 b_1^p + a_2 b_2^p + \text{------} = 1$

or

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

Example: $T(N) = 2T\left(\frac{N}{2}\right) + (N-1)$

Here, $a_1 = 2$ $g(N) = N-1$

$b_1 = 1/2$

Put in the formula of P, & find P

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

$\boxed{P = 1}$.

Put P in Akra-Bazzi formula —

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

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

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

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

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

$$= \theta\left(\cancel{x} + x\log x + 1 - \cancel{x}\right)$$

$= \theta(x\log x + 1)$ // Ignore constant

$= \theta(x\log x)$

Hence, for array of size N:

Merge Sort complexity $= \theta(N\log N)$

Case: If you can't find value of P

• Let's take an example

Q. $T(x) = 3T\left(\frac{x}{3}\right) + 4T\left(\frac{x}{4}\right) + x^2$

Sol:→ Let's try $\boxed{P=1}$

$$3\times\left(\frac{1}{3}\right)^P + 4\times\left(\frac{1}{4}\right)^P = 1$$

when $P = 1$

$3 \times \left(\frac{1}{3}\right)^1 + 4 \times \left(\frac{1}{4}\right)^1 = 1$

Here, we get 2 which is greater than 1

$\boxed{2 > 1}$ $\Rightarrow$ Now we have to increase the denominator

i.e increase the value of P

when $P = 2$

$3 \times \left(\frac{1}{3}\right)^2 + 4 \times \left(\frac{1}{4}\right)^2$

$= \frac{1}{3} + \frac{1}{4} = \frac{4+3}{12} = \frac{7}{12} < 1$

$\therefore \boxed{1 < P < 2}$

NOTE :-

when, $P <$ Power of $g(x)$

then, ans $= g(x)$

VVI

Here

$g(x) = x^2$

and $P < 2$ (ie, Power of $g(x)$)

Hence, $\boxed{\text{ans} = O(g(x))} = O(x^2)$

** Linear Recurrences **

Form of Homogeneous Linear Recurrences

$$f(x) = a_1 f(x-1) + a_2 f(x-2) + a_3 f(x-3) + \cdots$$
$$\cdots + a_n f(x-n)$$

or

$$f(x) = \sum_{i=1}^{n} a_i f(x-i)$$, for a_i & n = fixed
n = order of recurrence

Example :- $F(n) = F(n-1) + F(n-2)$ ——— ①

Solution :-

Step 1: Put $F(n) = \alpha^n$ for some constant α in eqⁿ ①

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

$\Rightarrow \boxed{\alpha^n - \alpha^{n-1} - \alpha^{n-2} = 0}$ ← characteristic eqⁿ of recurrence

divide by α^{n-2}

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

roots are :— $\alpha = \frac{1+\sqrt{5}}{2}$

$\alpha_1 = \frac{1+\sqrt{5}}{2}$, $\alpha = \frac{1-\sqrt{5}}{2}$

Step 2:

$F(n) = c_1 \alpha_1^n + c_2 \alpha_2^n$ // is also a solⁿ for fibonacci

$$F(n) = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$$ ——— ②

step 3 : Fact

$\boxed{\text{** no. of roots = no. of answer we have already}}$

Here, we have 2 roots α_1 and α_2

Hence, we should have 2 answer already.

$\therefore \; F(0) = 0$ and $F(1) = 1$

$F(0) = 0$

$\Rightarrow C_1 + C_2 = 0 \quad \Rightarrow \boxed{C_1 = -C_2}$ —— 3

$F(1) = 1$

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

from 3

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

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

Putting in eqn 2

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

formula for 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 $n \to \infty$ this will be close to 0.

Hence, this is less denominating term

$\Rightarrow$ Hence ignore while taking complexity

∴ Time complexity for nth fibonacci number

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

When we get Equal roots —

→ If α is repeated r times
then $\alpha^n, n\alpha^n, n^2\alpha^n, \text{-------}, n^{r-1}\alpha^n$
are all solution to the recurrence.

* Non-homogenous linear Recurrences

Form

$$F(n) = a_1 f(n-1) + a_2 f(n-2) + a_3 F(n-3) + \text{------} -- + a_d F(n-d) + \underbrace{g(n)}$$

↳ when this extra function is there, it is non-homogeneous L.R.

Solution

Replace $g(n)$ by 0 and solve usually.

Example:

$$F(n) = 4F(n-1) + 3^n \quad \text{———} \quad (1)$$

Sol ⇒ Put $g(n) = 0 \quad \Rightarrow 3^n = 0$
Step 1 →

$F(n) = 4F(n-1)$

Put $F(n) = \alpha^n$

$\Rightarrow \alpha^n = 4\alpha^{n-1}$

$\Rightarrow \alpha^n - 4\alpha^{n-1} = 0 \quad \Rightarrow \alpha - 4 = 0 \quad \Rightarrow \boxed{\alpha = 4}$

→ Homogeneous $sol^n \Rightarrow F(n) = c_1 \alpha^n$ // Put $\alpha = 4$

$$\boxed{F(n) = c_1 4^n}$$

Step 2 : Take $g(n)$ on one side and find particular sol^n —

$\therefore F(n) - 4F(n-1) = 3^n$ —— (2)

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

My Guess : $F(n) = c\,3^n$ —— (4)

Put in eq^n (2) —

$\Rightarrow c\,3^n - 4c\,3^{n-1} = 3^n \Longrightarrow \boxed{c = -3}$

Find particular sol^n —

Put $c = -3$ in eq^n (4) —

$\Rightarrow \boxed{F(n) = -3 \times 3^n = -3^{n+1}}$

Step (3) :- Find general sol^n by adding both the solution

$$\boxed{F(n) = c_1 4^n + (-3^{n+1})}$$ —— (5)

We already have 2 answer
i.e, $F(0) = 0$ and $F(1) = 1$

$\Rightarrow F(1) = 1 \Rightarrow c_1 4 - 3^2 = 1 \Rightarrow \boxed{c_1 = 5/2}$

Put the value of c_1 in eq^n (5) —

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

$\therefore$ Time complexity $= O\left(\frac{5}{2} 4^n - 3^{n+1}\right)$

* How do we guess particular solution?

If $g(n)$ is exponential :
then guess of same type.

Ex :- $g(n) = 2^n + 3^n$

Guess : $\boxed{F(n) = a2^n + b3^n}$

If $g(n)$ is polynomial :
then guess of same degree.

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

Guess of same degree (Here, degree = 2).

Guess : $F(n) = an^2 + bn + c$

If $g(n)$ is combination of exponential & polynomial :

Ex :- $g(n) = 2^n + n$

Guess : $F(n) = a2^n + (bn + c)$

$\boxed{\text{NOTE}}$: let say you guessed, $F(n) = a2^n$ and it fails. Then, try $(an+b)2^n$. If this also fails, increase the degree i.e, $(a^2n + bn + c)2^n$.