① What is Time Complexity?

Old Computer

data: 1,000,000 elements in an array

Algorithm, Linear search for target that does not exist in array

Time taken: 10 secs

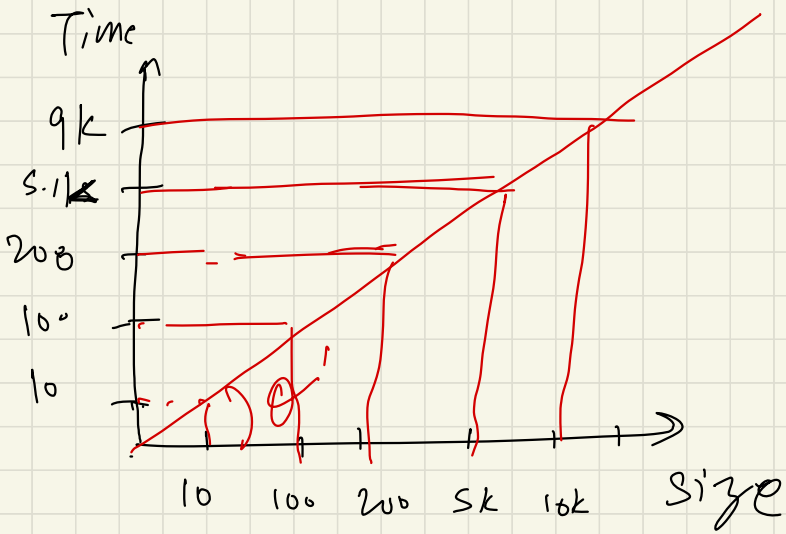
M1 Macbook (very fast)

Time: 1sec

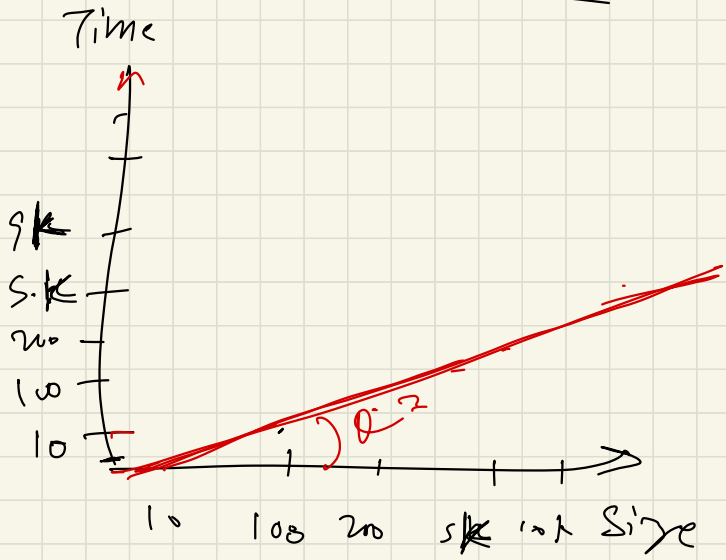
* Both machines have same time complexity.

Time Complexity != Time taken

Old machine

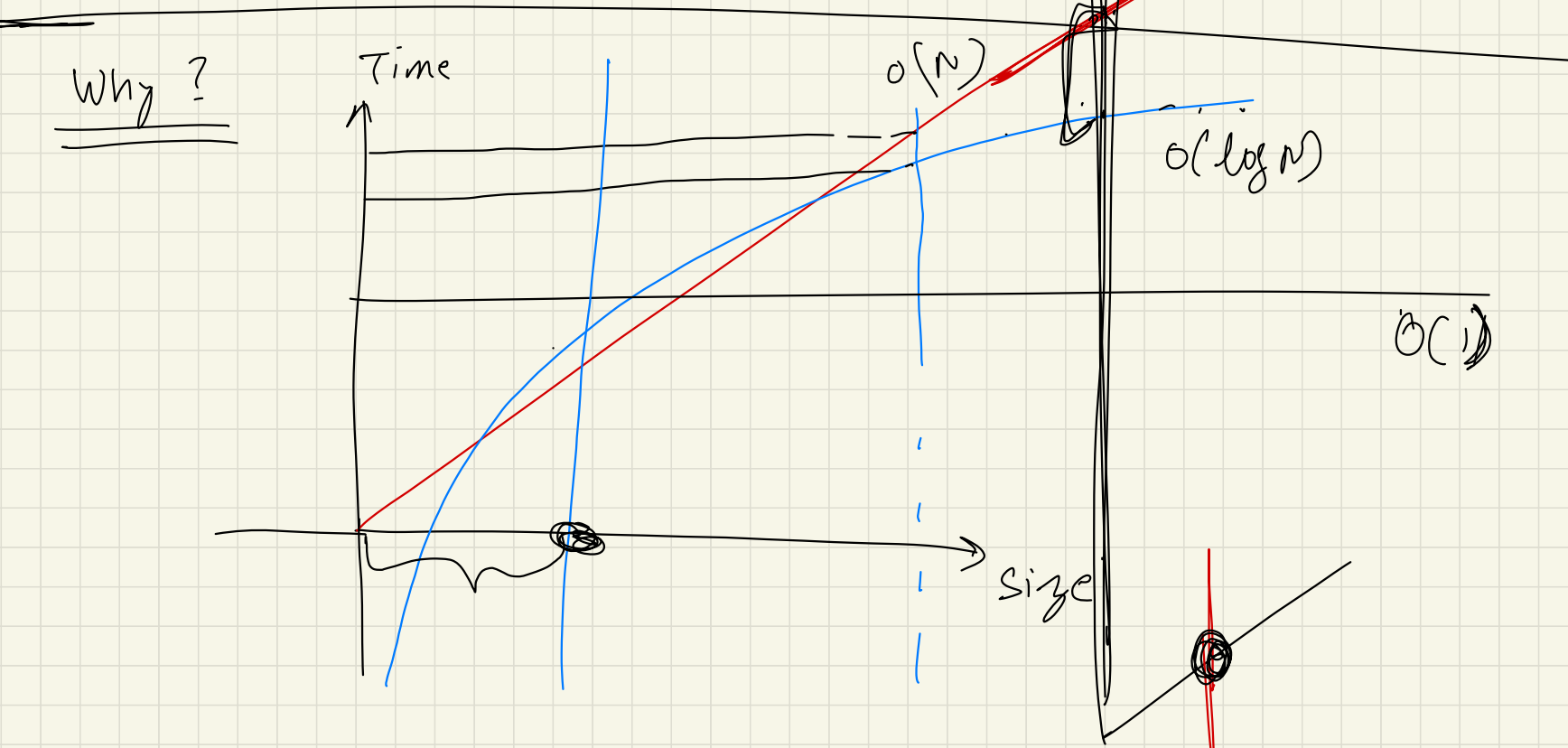


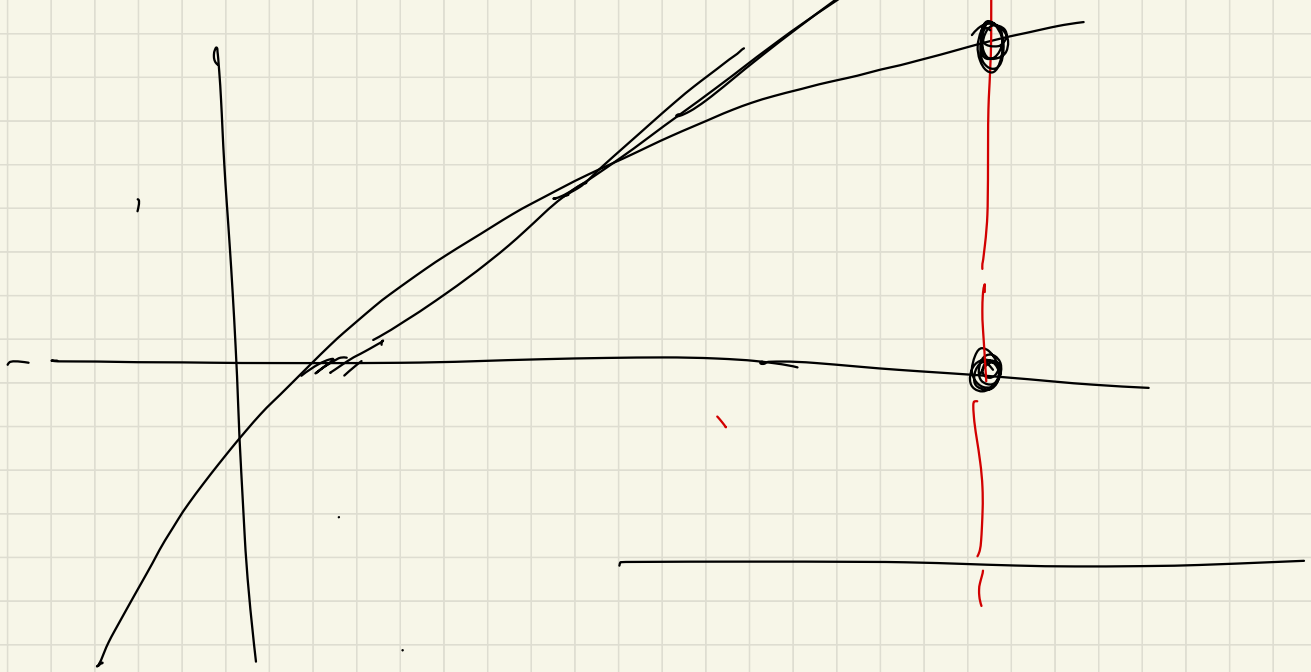
M1 Macbook



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

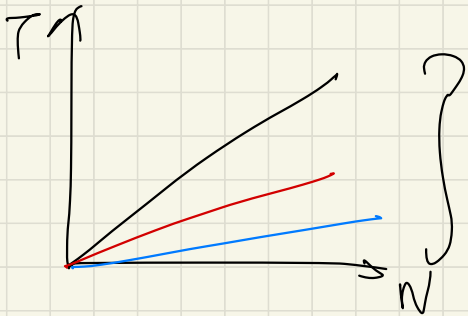
Why ?





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

What do we consider when thinking about complexity:

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

$O(N)$

$y = x$

$y = 2x$

$y = 4x$

$y = 3x + 5$

- Even tho value of actual time is diff they are all growing linearly.
- We don't care about actual time

✱ This is why, we ignore all constants.

④ $O(N^3 + \log N)$

✱ from point no. ②

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

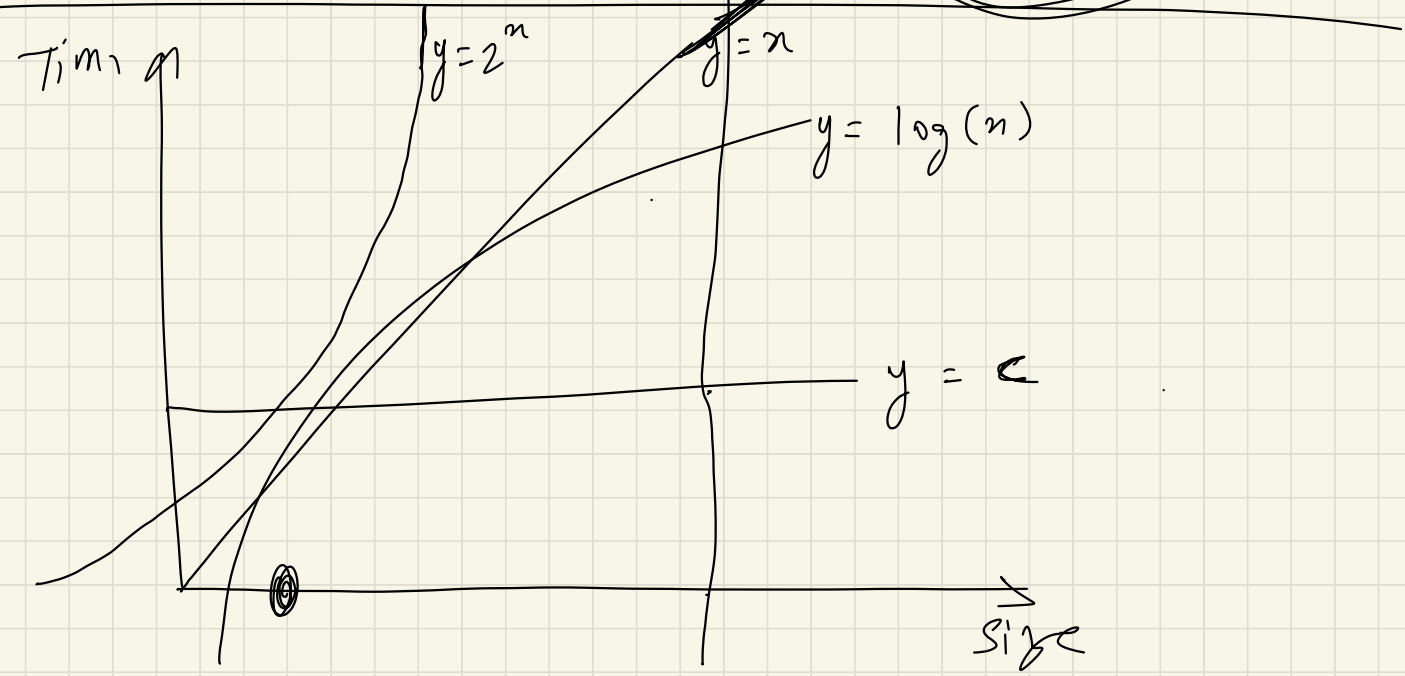
$= (1\text{ mil})^3 + \underbrace{\text{less}}_{\text{very small}}$

very small
hence ignore.

Always ignore less dominating terms.

Ex: $O(3N^3 + 4N^2 + 5N + 6)$

$= (N^3 + N^2 + N) = O(N^3)$



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

$O(N \log N)$

Big-Oh Notation:

Word definition:

$O(N^3)$ → Upper bound.

✱ maths :

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

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

$$O(N^3) = O(6N^3 + 3N + 5)$$

$g(N)$ $\quad$ $f(N)$

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

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

$$= 6 + 0 + 0$$

$$= 6 < \infty$$

finite value

Big Omega : opposite of Big Oh

Words: $\Omega(N^3)$ (lower bound)

Maths:

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

Q: What if an algo has UB & LB as (N^2).

= $O(N^2)$ & $\Omega(N^2)$

Theta Notation: (Combining both)

$\Theta(N^2) \Rightarrow$ words: Both UB & LB is = N^2

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

little o notation:

* This is also giving up.

Words: lower up.

Big Oh

$f = O(g)$

$f \leq g$

little o

(stronger statement)

$f = o(g)$

$f < g$

strictly slower

Maths:

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

Ex: $f = N^2$ $\quad g = N^3$

$$\lim_{N \to \infty} \frac{N^2}{N^3} = \lim_{N \to \infty} \frac{1}{N} = 0$$

little omega:

words

Big Ω

$f = \Omega(g)$

$f \geq g$

little ω

$f = \omega(g)$

$f > g$

maths:

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

Ex:

$$\lim_{n \to \infty} \frac{n^3}{n^2} = \lim_{n \to \infty} n = \infty$$

Space Complexity or Auxiliary Space?

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

Space Complexity of an algorithm is total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input.

For example, if we want to compare standard sorting algorithms on the basis of space, then Auxiliary Space would be a better criteria than Space Complexity. Merge Sort uses O(n) auxiliary space, Insertion sort and Heap Sort use O(1) auxiliary space. Space complexity of all these sorting algorithms is O(n) though.

Q:

```
for ( i = 1 ; i <= n ; ) {
    for ( j = 1 ; j <= k ; j++ ) {
        // some operation
        that takes time t
    }
    i = i + k
}
```

Inner loop: $O(kt)$ time

Ans = $O($ kt * times outer loop is running $)$

?

$i = 1, 1+k, 1+2k, 1+3k, 1+4k \ldots, 1+nk$

$1 + nk \leq N$

Times the outer loop is running

$nk \leq N - 1$

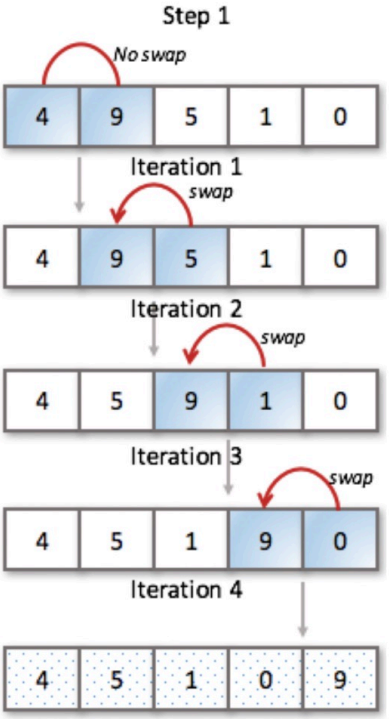
$n = \frac{N-1}{K}$

$O\left(kt * \frac{(N-1)}{k}\right)$

$= O(Nt)$

Ans

Bubble Sort



Worst and Average Case Time Complexity: O(n*n). Worst case occurs when array is reverse sorted.

Best Case Time Complexity: O(n). Best case occurs when array is already sorted.

Auxiliary Space: O(1)

Boundary Cases: Bubble sort takes minimum time (Order of n) when elements are already sorted.

Sorting In Place: Yes

Stable: Yes

Selection Sort

Worst complexity: n^2

Average complexity: n^2

Best complexity: n^2

Space complexity: 1

Method: Selection

Stable: No

The good thing about selection sort is it never makes more than O(n) swaps and can be useful when memory write is a costly operation.

Insertion Sort

Insertion Sort Execution Example

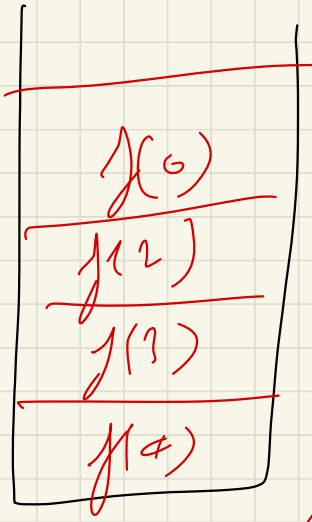
Time Complexity: O(n*2)

Auxiliary Space: O(1)

Boundary Cases: Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.

Sorting In Place: Yes

Recursive Algorithms:



O(N)

Trick: Only call that are interlinked will be in the stack at same time.

Space Complexity = Height of tree
(path)

---

2 types of recursions:

(1) Linear

$F(N) = F(N-1) + F(N-2)$

(2) Divide & Conquer

$F(N) = F(\frac{N}{2}) + O(1)$

① Divide & Conquer Recurrences:

Form:

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

for $n \geq n_0$

Simplest.

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

$a_1 = 1$

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

$\varepsilon_1(n) = 0$

$g(n) = C$

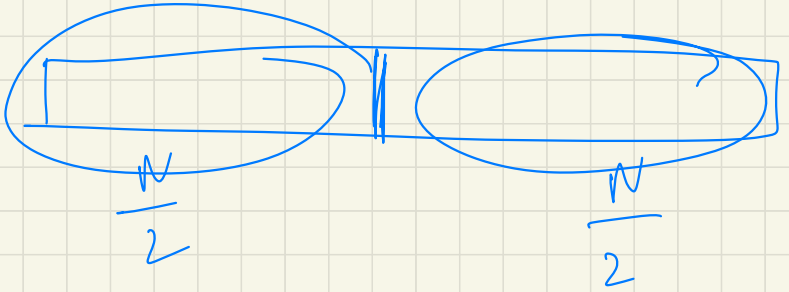
$$T(N) = 9\,T\left(\frac{N}{3}\right) + \frac{4}{3}\,T\left(\frac{5}{6}N\right) + \underbrace{4N^3}_{g(N)}$$

a_1 b_1 a_2 b_2

$$T(N) = 2\,T\left(\frac{N}{2}\right) + \underbrace{(N-1)}_{g(N)}$$

a_1 b_1

when you get ans from this + what you are doing takes max max time.



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

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

// R.H.S of MS

How to actually solve to get complexity:

(1) Plug & chug.

$$F(N) = F\left(\frac{N}{2}\right) + k$$

(2) Master's Theorem

(3) Akra Bazzi (1996)

Akra Bazzi :

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

What is p?

$$a_1 b_1^p + a_2 b_2^p + \dots = 1$$

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

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

$a_1 = 2$

$g(n) = n - 1$

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

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

$p = 1$

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

$O(1)$

$O(C)$

$O((2 * 1 \in) * 1)$

$O(1)$

Plot ρ in formula:

$$T(n) = \theta\left(n' + n' \int_1^n \frac{u-1}{u^2}\, du\right)$$

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

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

$$= \Theta\left(n + n\left[\int_1^n \log u + \frac{1}{u}\right]\right)$$

$$u^{-2}$$

$$\frac{u^{-1}}{-1} = \frac{-1}{u}$$

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

$$= \Theta\left(n + n \log n + 1 - n \right)$$

$$= \Theta\left(n \log n + 1 \right)$$

$$= \Theta\left(n \log n \right)$$ // Time complexity

for array of size N: MS complexity $= O(N \log N)$

Q:

$$T(n) = 2T\left(\frac{n}{2}\right) + \frac{8}{9}T\left(\frac{3n}{4}\right) + n^2$$

a_1 b_1 a_2 b_2

$$2 \times \left(\frac{1}{2}\right)^p + \frac{8}{9} \times \left(\frac{3}{4}\right)^p = 1$$

$$2 \times \frac{1}{4^2} + \frac{8}{9} \times \frac{9}{16} = 1$$

$$p = 2$$

$$T(n) = \Theta\left(n^2 + n^2 \int_1^n \frac{u^2}{u^3} \, du \right)$$

$$= \Theta\left(n^2 + n^2 \ln n \right)$$

$$= \Theta(n^2 \ln n)$$

If you can't find value of β:

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

Let's try $p = 1$

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

$2 > 1 \Rightarrow$ Increase the denominator

$\therefore p > 1$

$p = 2$

$$= 3 * \frac{1}{9} + 4 * \frac{1}{16}$$

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

$\therefore$ $p < 2$

NOTE: When $p <$ power of $(g(n))$

then ans = $g(n)$.

Here, $g(n) = n^2$

$p < 2$ (i.e. power of $g(n)$)

Hence, ans = $O(g(n))$

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

$$= \Theta\left(n^p + n^p \int_1^n u^{1-p} \, du \right)$$

$$= \Theta\left(n^p + n^2 \right)$$

$\therefore \; p < 2$

less dominating term hence ignore

Also $\Theta(n^2)$

Solving Linear Recurrences:

Homogenous eq^n

Ex: $F(N) = F(N-1) + F(N-2)$

Form:

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

$$f(n) = \sum_{i=1}^{n} a_i f(n-i), \text{ for } a_i,\ n \text{ is fixed}$$

n: order of recurrence.

Solution for Fibonacci no:

Steps

$$f(n) = f(n-1) + f(n-2) \quad \text{---(1)}$$

(1) Put $f(n) = \alpha^n$ for some constant α

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

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

divide by α^{n-2}

$$\Rightarrow \boxed{\alpha^2 - \alpha - 1 = 0}$$

$$\frac{\alpha^n}{\alpha^{n-2}} = \frac{\alpha^n}{\alpha^n \times \alpha^2} = \frac{1}{\alpha^{-1}} = \alpha$$

$\Rightarrow$ roots of this quad eqn

Characteristic Eq^n of recurrence.

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

$$\therefore \quad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\alpha_1 = \frac{1+\sqrt{5}}{2}, \qquad \alpha_2 = \frac{1-\sqrt{5}}{2}$$

(2) $f(n) = c_1 \alpha_1^{\,n} + c_2 \alpha_2^{\,n}$ is a soln for

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

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

——— (2)

(3) Fact: No. of roots = No. of ans you have already.

Here we have 2 roots α_1 & α_2.

Hence we should have 2 ans already.

$\therefore$ $f(0) = 0$ & $f(1) = 1$

$$y(0) = 0 = c_1 + c_2 \implies c_1 = -c_2 \quad -(3)$$

$$y(1) = 1 = c_1\left(\frac{1+\sqrt{5}}{2}\right) + c_2\left(\frac{1-\sqrt{5}}{2}\right)$$

from (3)

$$\implies 1 = c_1\left(\frac{1+\sqrt{5}}{2}\right) - c_1\left(\frac{1-\sqrt{5}}{2}\right)$$

$$\boxed{c_1 = \frac{1}{\sqrt{5}}} \qquad \boxed{c_2 = \frac{-1}{\sqrt{5}}}$$

Putting this in eq^{n} no. (2)

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

→ Formula for n^{th} fibo no.

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

as $n \to \infty$

→ this will be close to 0

Hence, this is the dominating term.

Hence ignore.

Time complexity:

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

→ Golden ratio

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

Q: Equal roots:

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

① $f(n) = \alpha^n$

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

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

α^{n-2} both LHS & RHS

$$\Rightarrow \quad \alpha^2 - 2\alpha + 1 = 0 \quad \Longrightarrow \quad \boxed{\alpha = 1} \text{ double root}$$

* General case: If α is repeated r times.

then, $\alpha^n, \; n\alpha^n, \; n^2\alpha^n, \ldots, n^{r-1}\alpha^n$

are all soln to the recurrence.

Hence I can take 2 roots as :

1, $n\alpha^n$

$$f(n) = c_1(\alpha)^n + c_2 n\alpha^n$$

$$f(n) = c_1 + c_2 n$$

$f(0) = 0$ & $f(1) = 1$

$f(0) = \boxed{0 = c_1}$

$f(1) = 1 = c_1 + c_2$

$\boxed{c_2 = 1}$

Ans : $f(n) = n \implies$ Time complexity $= O(n)$

Non- homogenous linear recurrences:

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + a_3 f(n-3) + \cdots + a_d f(n-d) + g(n)$$

↳ When this extra function is there, it is non-hg fr.

How to solve:

① Replace $g(n)$ by 0 & solve usually.

$$f(n) = 4f(n-1) + 3^n, \quad f(0) = 1$$

$$f(n) = 4 \cdot f(n-1)$$

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

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

$$\alpha - 4 = 0 \qquad \alpha = 4$$

Homogeneous Sol^{n} $\Rightarrow$ $y(n) = C_1 \alpha^{n}$

$$y(n) = C_1 4^{n}$$

(2) Take $g(n)$ on one side and find particular Sol^{n}:

$$y(n) - 4y(n-1) = 3^{n}$$

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

If $g(n) = n^2$, guess a polynomial of degree 2

My guess: $y(n) = C\,3^{n}$ Put C over here

$$c3^n - 4c3^{n-1} = 3^n \Longrightarrow c = -3$$

particular sol^n $\Rightarrow$ $y(n) = -3^{n+1}$

(3) Add both sol^n together:

$$y(n) = c_1 4^n + \left(-3^{n+1}\right)$$

$y(1) = 1$ $\Longrightarrow$ this already provided

$$c_1 4 - 3^2 = 1$$

$$\Longrightarrow c_1 = \frac{5}{2}$$

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

How do we guess particular soln?

* If $g(n)$ is exponential, guess of some type. Ex: $g(n) = 2^n + 3^n$

Guess: $y(n) = a2^n + b3^n$

* If it is polynomial $(g(n))$, sum of same degree.

Ex: $g(n) = n^2 - 1 \Rightarrow$ sum of same degree $\Rightarrow$ 2 deg

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

$g(n) = 2^n \neq n$

sum $f(n) = a2^n + (bn + c)$

Let say you guessed, $f(n) = a\,2^n$ and it fails, then try $(an+b)2^n$, if this also fails increase the degree, $(an^2+bn+c)2^n$

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

① put $2^n = 0$

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

$$y(n) = \alpha^n$$

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

$$\alpha = 2$$

(2) Given particular soln:

$$y(n) = 2^n$$

Given: $y(n) = a2^n$

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

$$a = a + 1$$ ✗ wrong.

Hence gives another one from our rules.

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

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

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

$a = 1$; b is could be

$$f(n) = n\,2^n$$

// particular solⁿ

(3) General ans:

$$y(n) = c_1 2^n + n 2^n$$

$$y(0) = 1 = c_1 + 0$$

$$\boxed{c_1 = 1}$$

$$\Rightarrow y(n) = 2^n + n 2^n \quad \text{Ans}$$

Complexity $= O(n 2^n)$