Time & Space Complexity

old computer

data (1,000,000) elements

Algorithm linear search for target that does not exist in the array

Time taken 10 sec

new computer

data (1,000,000) elements

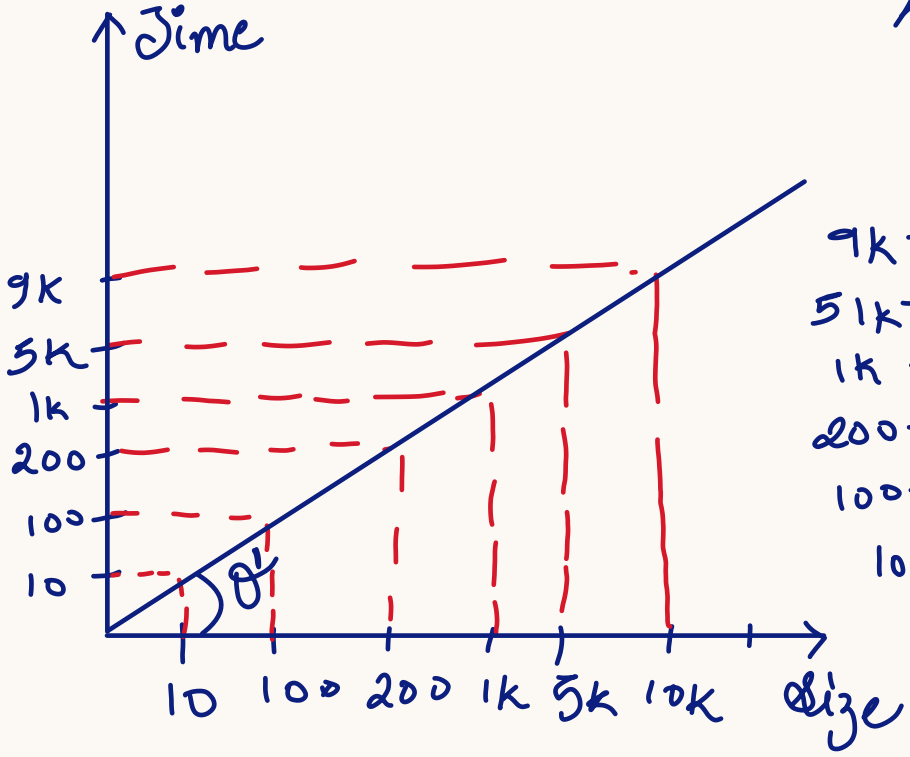
Algorithm, linear search for target that does not exist in Array.

Time taken 1 sec

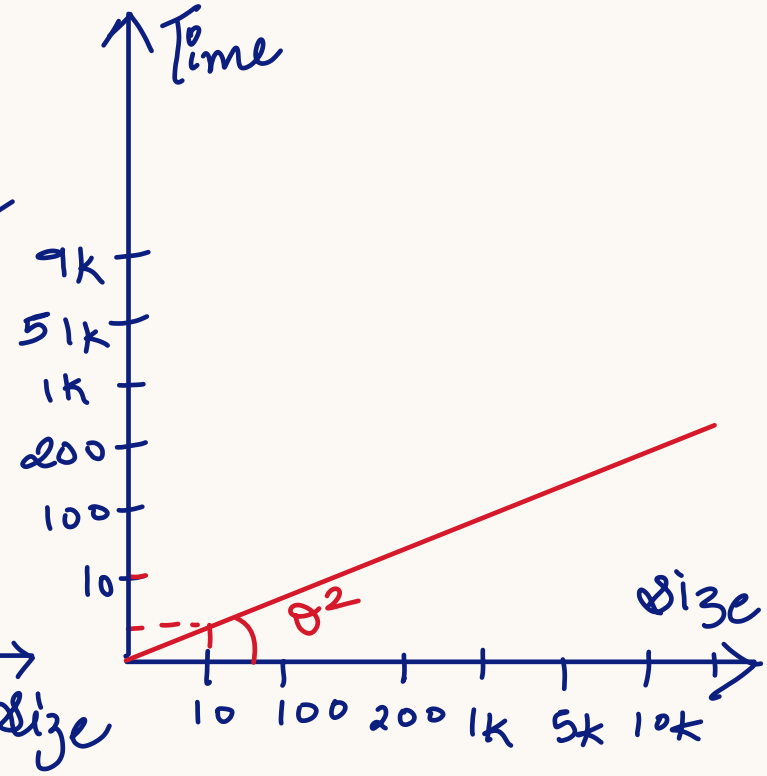
* Both machines have same time complexity

Time complexity != Time taken

old machine

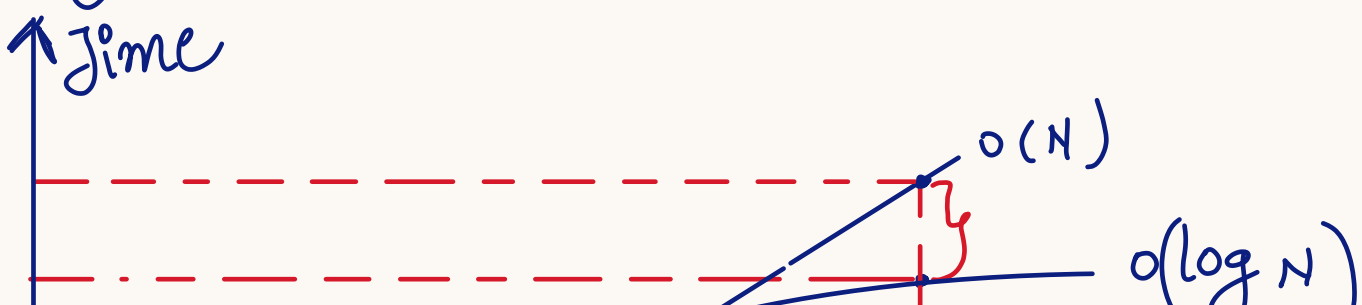


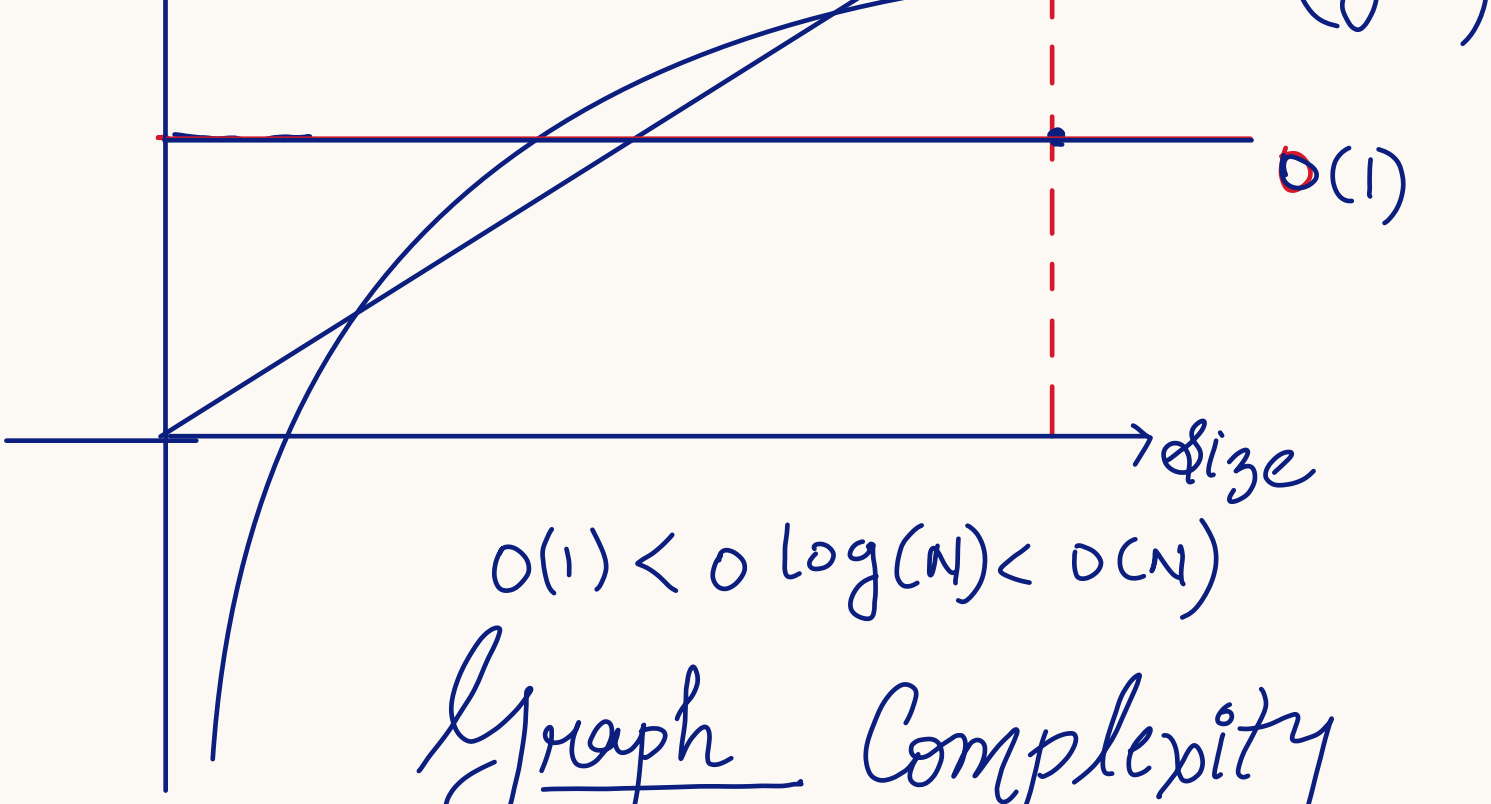
m2 macbook



* Time complexity is a function that gives us the relationship about how the time will grow as the input grows

* Why ?

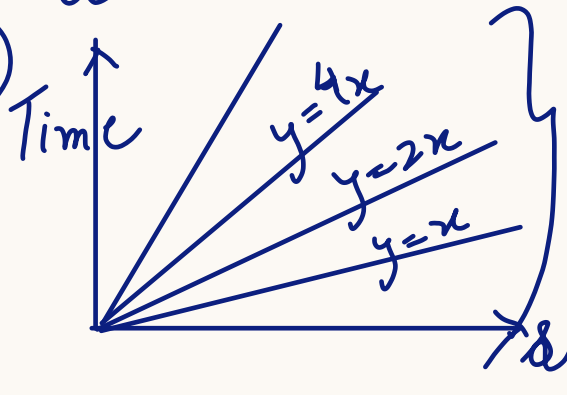




$O(1) < O\log(N) < O(N)$

Graph Complexity

* what do we consider when thinking about complexity ?

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

$O(N)$ * Even though the value of actual time is different 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 number (2) 1 million data

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

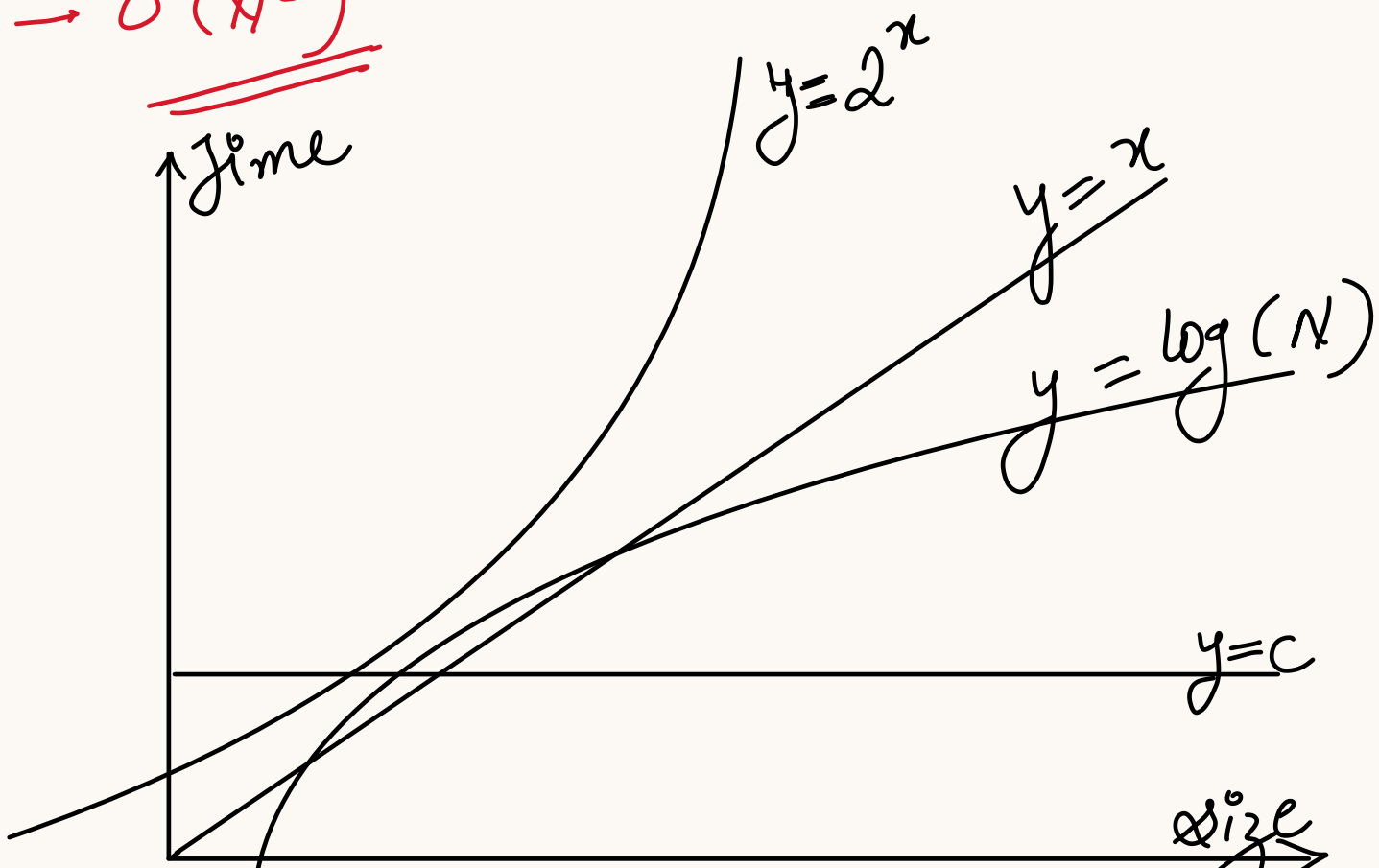
$\Rightarrow (1\text{million})^3 + 6$ secs (very small hence ignore)

Always ignore less dominating terms.

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

$\rightarrow N^3 + N^2 + N$

$\rightarrow O(N^3)$



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

Big - Oh Notation :

* word definition

$O(N^3)$ → upper bound

The algorithm will not exceed N^3

* Maths: $f(n) = O(g(n))$

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

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

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

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

$$= 6 + \frac{3}{N^2} + \frac{5}{N^3} \qquad = 6 + \frac{3}{\infty} + \frac{5}{\infty}$$

$= 6 + 0 + 0$

$= G < \infty$

finite value

Big Omega: Opposite of Big Oh notation.

words ' $\Omega(N^3)$ ⟶ lower bound means the algorithm will never be less than N^3 its complexity starts from N^3. Minimum N^3 time complexity would be required.

Maths $\lim_{N \to \infty} \frac{f(x)}{g(x)} > 0$

Q) what if an algorithm has lowerbound and upperbound as (N^2).

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

Theta Notation

$\theta(N^2) \Rightarrow 0 < \lim_{N \to \infty} \frac{f(x)}{g(x)} < \infty$

→ Both upper bound and lower bound is N^2.

Little oh Notation

words: loose upper bound

Big oh

$f = o(g)$

$f \leq g$

More stronger little oh statement

$f = o(g)$

$f < g$

strictly lower than g

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

Ex. $f = N^2$ $g = N^3$ (say)

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

Little Omega: loosely lower bound.

(ω)

Big Ω

$f = \Omega(g)$

$f \geq g$

little omega (ω)

$f = \omega(g)$

$f > g$

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

$f(N) \to N^3$, $g(N) \to N^2$

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 : $O(kt * \text{times outer loop is running})$

$i = 1, 1+k, 1+2k, 1+3k, \ldots, 1+xk$

$1 + xk \leq N$

$xk \leq N-1$

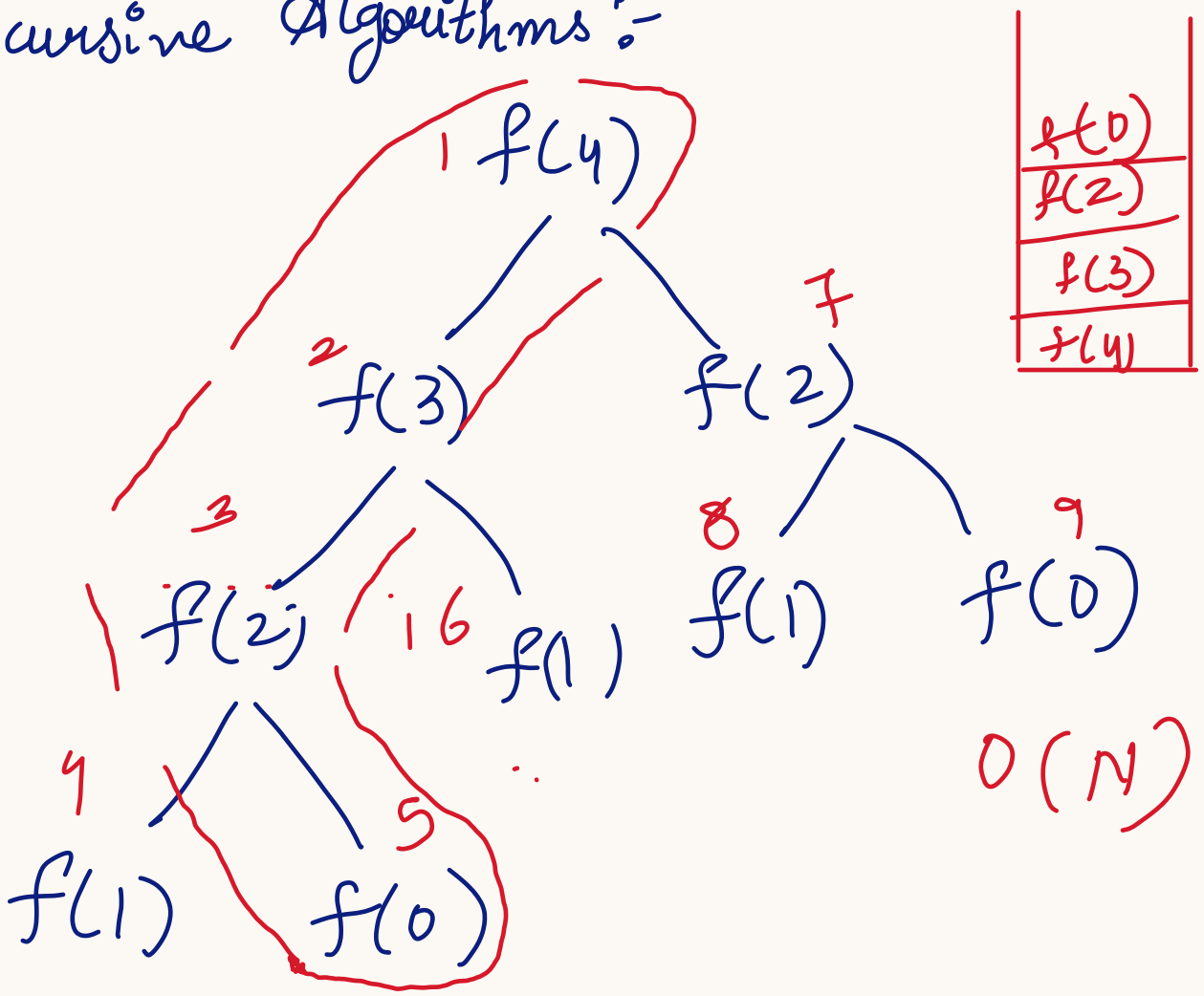
$$x = \frac{N-1}{k}$$

x times the outer loop is running.

$O(k t * (\frac{N-1}{k}))$ → constants are removed.

$= O(N t)$ Ans

Recursive Algorithms :-



Trick : for space complexity only call that are interlinked will be in the stack at the same time

Space complexity for the recursive program is the height of the tree.

Space complexity = height of the tree (path)

There are 2 types of recursion problems

i) Linear

ii) Divide & conquer.

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

$F(N) = F(N/2) + O(1)$

1) Divide and conquer recurrances:

Form: $T(x) = a_1 T(b_1 x + \Sigma_1(x)) + a_2 T(b_2 x + \Sigma_2(x)) + \ldots + a_k T(b_2 x + \Sigma_k(x)) + g(x)$

for all $x \geq x_0$ ↳ some constant.

$T(N) = T(N/2) + C$

$a_1 = 1, \; b_1 = 1/2, \; g(x) = C, \; \Sigma_1(x) = 0$

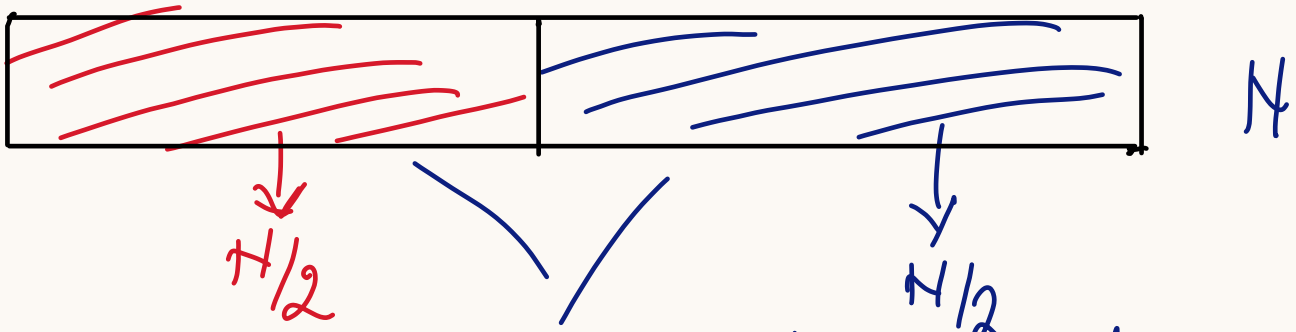
$$T(N) = 9\,T(N/3) + \frac{4}{3}\,T\left(\frac{5}{6}N\right) + \underbrace{4N^3}_{g(N)}$$

a_1 b_1 a_2 b_2

$$T(N) = 2T(N/2) + (N-1)$$

a_1 b_1 $g(N)$

when you get ans from this + what you are doing with the ans takes how much time?



Merge this two sorted arrays (Merge sort)

$(N-1)$ is the time taken to merge $N/2 + N/2$

$$T(N) = T(N/2) + T(N/2) + N-1$$

$$T(N) = 2\,T(N/2) + (N-1)$$

a_1 b_1 $g(N)$

Recurrance relation of Merge Sort

How to actually solve to get complexity:

1) Plug and chug

$$F(N) = F(N/2) + C$$

2) Master's Theorem

3) Akra Bazzi (1996) formula

Akra Bazzi :-

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

What is P ?

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

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

$$T(N) = T(N/2) + C$$

constants

$O(1)$

$O(c)$

$O[(2+1)*1]$

Ex: Merge Sort

$$T(N) = 2T(N/2) + (N-1)$$

$a_1 = 2$

$b_1 = 1/2$

$g(N) = N-1$

$$a_1 b_1^P = 1 \Rightarrow 2 \times \left(\frac{1}{2}\right)^P = 1$$

$$P = 1$$

Put P in formula:-

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

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

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

u^{-2}

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

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

$= \Theta\left(x + x\left[\log x + \frac{1}{x} - 0 - 1\right]\right)$

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

$= \Theta\left(x\log x + 1\right)$ constants gets cancelled

$= \Theta\left(x \log x\right)$

Time complexity.

An array of size N : MergeSort complexity is $O(N \log N)$

Q/

$T(N) = 2T(N/2) + 8/9\ T(3N/4) + N^2$

a_1 b_1 a_2 b_2

$a_1 b_1^p + a_2 b_2^p$

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

If $p=2$

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

$$\Rightarrow 1 = 1$$

L.H.S = R.H.S

$$\boxed{p=2}$$

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

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

$$= \theta\left(x^2 + x^2 \log x\right)$$

Ignoring the less dominating terms

$$\boxed{T(x) \Rightarrow \theta(x^2 \log x)}$$

If you can't find value of P?

$$T(x) = 3T(x/3) + 4T(x/4) + x^2$$

Let's try $P = 1$

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

$$\Rightarrow 1 + 1 \neq 1$$

$$2 > 1 \quad [\text{L·H·S} \neq \text{R·H·S}]$$

increase the denominator, increase the power.

$\therefore P > 1$

Let's Try 2.

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

$$\Rightarrow \frac{1}{3} + \frac{1}{4} = 1$$

$$\Rightarrow \frac{7}{12} < 1$$

L.H.S $\neq$ R.H.S

Hence, P is definitely less than 2

P range 1 to 2.

Note when P is less than power of $g(x)$ than ans is equal to $g(x)$

$g(x) = x^{2}$

i.e, $P < 2$ (i.e. power of $g(x)$)

Hence Ans $= O(g(x))$

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

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

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

$$= \Theta\left(x^p + x^2\right) \quad \because p < 2$$

$x^p \rightarrow$ less dominating term

hence ignore

Ans $O(x^2)$

---

* Solving linear Reccurances $\approx$ Homogenous equations

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

form :-

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

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

for a_i, n is fixed; n is order of reccurance.

Solution: Fibonacci number relation.

$$F(N) = F(N-1) + F(N-2) \quad \text{---(1)}$$

Put $F(N) = \alpha^n$ for some constant α

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

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

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

roots of this going to be quadratic equation.

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

$\frac{\alpha^n}{\alpha^{n-2}}$

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

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

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

α

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

2) $f(n) = c_1\alpha_1^n + c_2\alpha_2^n$

This is also a soln for fibonacci

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

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

3) Fact: Number of roots = number of ans you have already.

Here, we have 2 roots α_1 and α_2

Hence, we should have 2 ans already

$\therefore F(0) = 0, \quad F(1) = 1$

$f(0) = 0 = c_1 + c_2 \Rightarrow c_1 = -c_2$ --- (3)

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

from (3)

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

$$= \frac{C_1 + C_1\sqrt{5}}{2} - \frac{C_1 + C_1\sqrt{5}}{2}$$

$$1 = \frac{2C_1\sqrt{5}}{2} \Rightarrow C_1 = \frac{1}{\sqrt{5}}$$

$$\therefore C_2 = -\frac{1}{\sqrt{5}}$$

putting this in equation number (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} fibonacci number.

ig ore

Time complexity:

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

ignore

as the n tends to Infinity this will be close to 0.

$1-\sqrt{5}$ would be negative

Hence, this is less dominating term

Hence, ignore.

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

Ans

$$\Rightarrow O(1.61803)^n$$

Golden ratio for Mathematics.

Q) Equal roots:

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

1) $f(n) = \alpha^n$

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

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

Divide both L.H.S and R.H.S by α^{n-2}

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

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

If α is repeated n times then

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

are all solution to the recurrance

Hence I can take 2 roots as :

$1, n\alpha^4$

$$f(n) = c_1 (\alpha)^4 + c_2 n\alpha^n$$

$$= c_1 + c_2 n$$

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

$f(0) = 0 = c_1$ | $f(1) = 1 = c_1 + c_2$

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

Non Homogenous Linear Recurrances :-

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

When this extra function is there it is non homogenous linear recurrances

How to solve this

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

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

$3^n \to 0$; $n \to n$

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

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

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

$$\alpha - 4\alpha^0 = 0$$

$$\boxed{\alpha = 4}$$

Homogenous $\alpha^n \Rightarrow f(n) = c_1 \alpha^n$

$$f(n) = c_1 4^n$$

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

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

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

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

Guess: $f(n) = C3^n$

$(C \cdot 3^n) - 4C3^{n-1} = 3^n$ take $n = 1$

$C \cdot 3 - 4C = 3$

$-C = 3$

$\boxed{C = -3}$

Particular solution +

$f(n) = -3 \cdot 3^n$

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

3) Add both solutions together.

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

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

$C_1 4^2 = 10$

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

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

How do we guess particular solution?

* If $g(n)$ is exponential, guess of the same type

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

guess: $f(n) = a 2^n + b \cdot 3^n$

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

$g(n) = n^2 - 1 \rightarrow$ guess of same degree
↳ 2 degree

$a n^2 + bn + c = f(n)$

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

given. $f(x) = a 2^n + (bn + c)$

* 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.

$(an^2+bn+c)2^n$

Ex1

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

1) Homogenous solution!

Put $2^n = 0$

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

$f(n) = \alpha^n$

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

$\alpha = 2$

2) Guess particular solutions.

$g(n) = 2^n$

$f(n) = a2^n$

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

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

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

$a = a + 1$ ✗ wrong

Hence guess another one from our rules.

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

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

$an + b = 2an - 2a + b + 1$

$1 + 1 = 1 - a + 1 + 1$

$\boxed{a = 1}$ discard b

$f(n) = n2^n$ // particular solution

3) General ans:-

$f(n) = c_1 2^n + n \cdot 2^n$

$f(0) = c_1 2^0 + 0 \cdot 2^0$

$f(0) = c_1$ & $f(0) = 1$

$\therefore C_1 = 1$

final Ans :- $\boxed{F(n) = 2^n + n \cdot 2^n}$

Complexity :- $O(n\,2^n)$