NOTE IN

Complexity DSA

① What is Time Complexity ?

Lets take an example:
We have 2 computers one is very old and one is very new

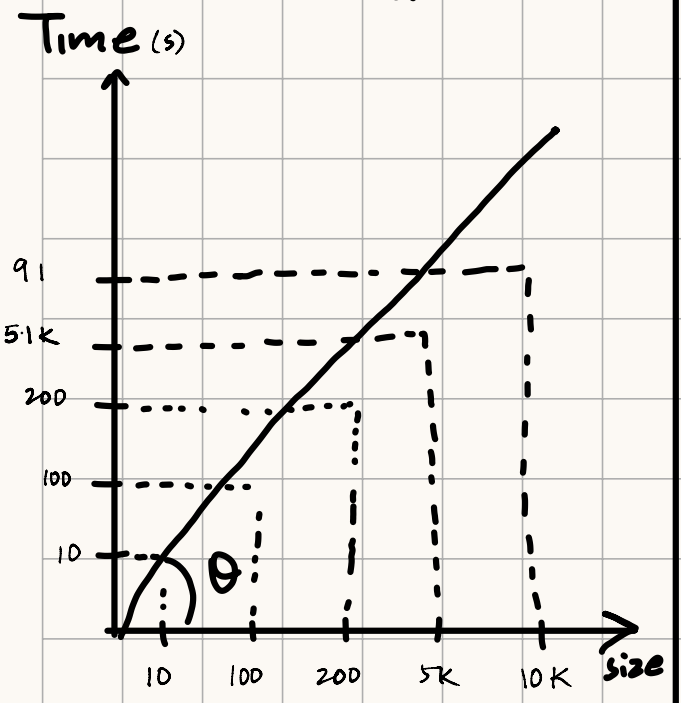
Old computer	M1 macbook (fast)
data 1,000,000 elements in array	data. 1,000,000 elements in array
Algorithm Linear search for target that doesnot exist in array	Algorithm Linear search for target that doesnot exist in array
Time taken: 10 seconds	Time taken: 1 seconds

Which one of the two has better time complexity?

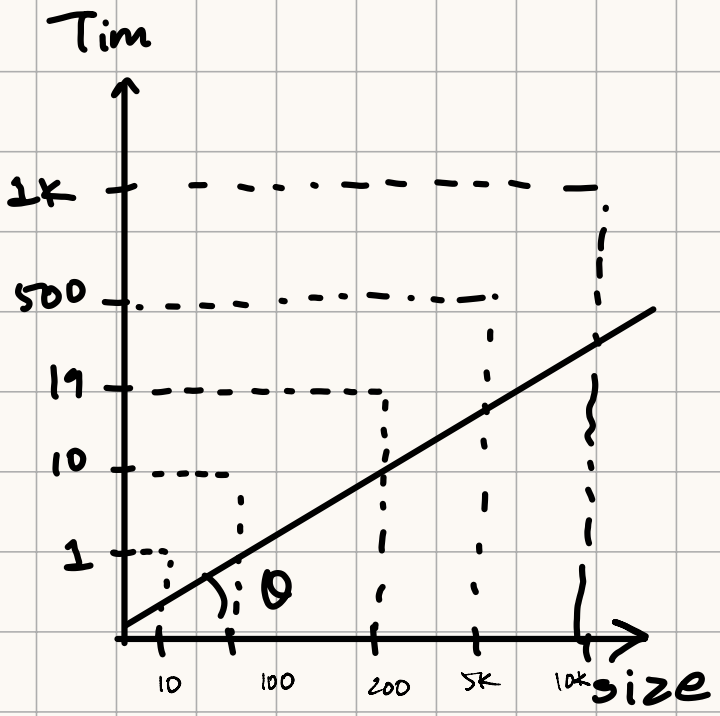
* Both machines have same time complexity

This means that Time complexity != time taken

Old Machine

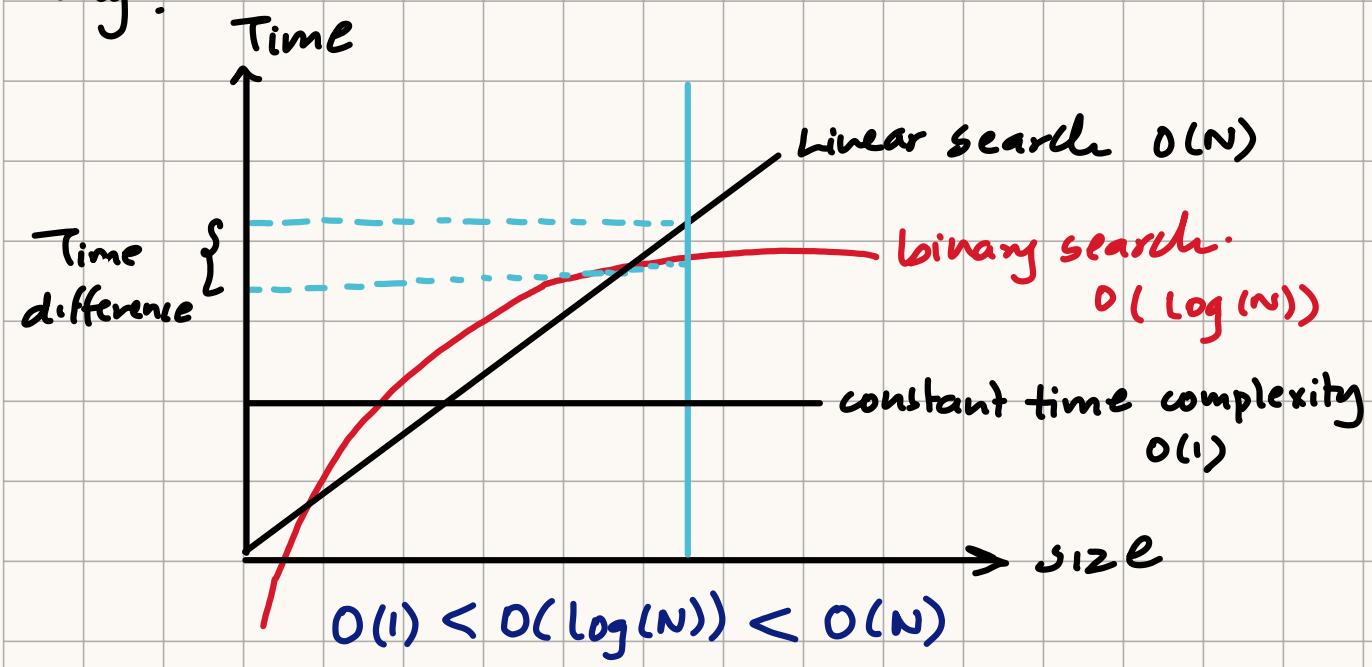


New Machine.



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

Why ?



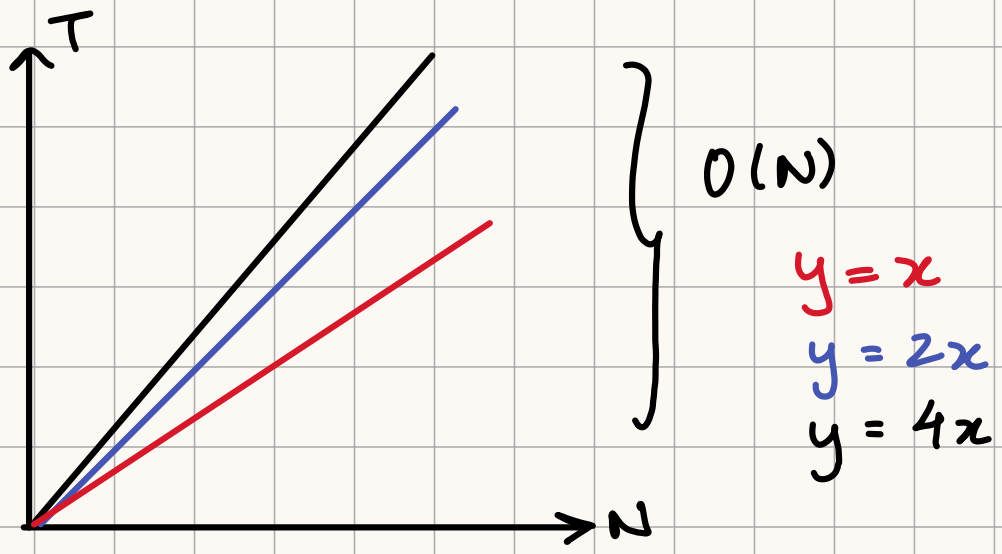
We can see that time taken by linear search is greater than time taken by binary search

what to consider when thinking about complexity?

① Always look for worst case complexity

② Always look for complexity for large / ∞ data

③



i) Even though value of actual time is different they are all growing linearly

ii) We don't care about actual time we only care about the growing input

iii) This is why we ignore all constant

④ $O(N^3 + \log N)$

* From ② → 1 mil size → $((1\text{mil})^3 + \log(1\text{mil}))$

$= [(1\text{ mil})^3 + \underbrace{6\text{ seconds}}_{\text{very small}}]$

Hence Always ignore Less dominating terms

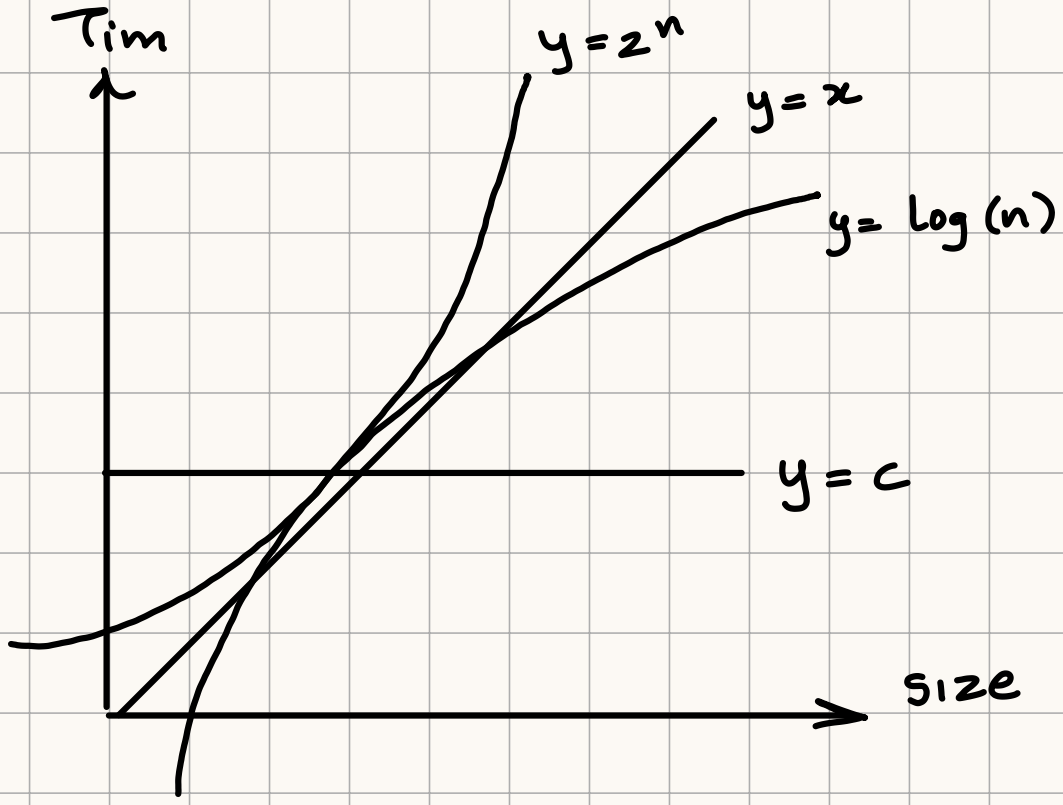
Q) $O(3N^3 + 4N^2 + 5N + 6)$

Ignore constants

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

Ignore Less dominating terms

$\rightarrow O(N^3)$



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

Big-Oh Notation: (O)

Word definition: Lets say an algorithm has a complexity of $O(N^3)$

It means that this is the upper bound

The 'O' here denotes upperbound.

Thus means that complexity of graph cannot exceed N^3

The algorithm may be solved in N^2 $N^2 \log N$ and so on but will not exceed N^3

★ **Math:**

Lets say: $f(n) = O(g(n))$

It means that the limit $n \to \infty$

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

Lets take the above example:

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

$g(n)$ $f(n)$

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

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

$$= 6 + 0 + 0 = 6 < \infty$$

↓ finite value

Big Omega Notation · Opposite of Big-Oh-Notation

Words $\Omega(N^3)$

Big Ω notation is basically the opposite of Big O notation

Here Ω represents the **lower bound**.

This means that complexity of graph always exceeds N^3

Math.

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

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

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

Theta Notation (Θ)

$\Theta(N^2) \Rightarrow$ both upper & lower bound $= N^2$

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

Little-Oh- Notation: (o)

* Like Big-Oh- This is also giving the upperbound, but this is not a strict upperbound

* This is a loose upperbound.

Math.

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

$g(N) = f(N)$

$N^3 = N^2$

$\Omega = \frac{N^2}{N^3} = \frac{1}{N} = 0$

$\lim_{n \to \infty}$

Big-Oh

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

$f = O(g)$

Growth of f is no faster than g

$f \leq g$

growth of f can be slower than or equal to g

Little-o

$f(n) = o(g(n))$

$f = o(g)$

$f < g$

growth of f is strictly slower than g

Little Omega Notation: (ω)

$f = \omega(g)$

Big Ω

$f = \Omega(g)$

$f \geq g$

growth of f can be greater than or equal to g

little ω

$f = \omega(g)$

$f > g$

growth of f is strictly greater than g

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

Example. $\lim_{N \to \infty} \frac{N^3}{N^2} = \lim_{N \to \infty} N = \infty$

* Space Complexity

Space Complexity = Input space + Auxillary space.

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

For binary search:

In binary search space complexity was constant that means auxillary space was constant

When we talk about the 3 variables which we used start, end and mid

Even if the array size is of 10, 100, 1000, 1000000 etc we only take these 3 constant variables hence space is constant

Q)
```
for (i=1, i≤N, ) {
    for (j=1, j≤K, j++){
        // some operation that takes
                                time = t
    }
    i = i+K
}
```

Find time complexity of the program

→ Inner loop is running K-times and taking time of t.

hence complexity of inner loop is

$O(Kt)$ time

Hence ($O(Kt)$ * times outer loop is running)

$i = 1, \ 1+K, \ 1+2K, \ 1+3K, \ 1+4K \ldots \underline{1+xK}$

$1 + xK \leq N$

$xK \leq N-1$

$$x \leq \frac{N-1}{K}$$

→ Number of times the outerloop is running

$O\left(Kt * \left(\frac{N-1}{K}\right)\right)$ → constants gets removed

$O(t * N) \rightarrow \underline{\underline{O(Nt)}}$

Bubble Sort:

Worst case and average case Time complexity : $O(n^2)$

Worst case occurs when array is reverse sorted

Best Case Time Complexity $O(n)$

Best case occurs when array is already sorted

Auxillary Space: $O(1)$ constant.

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

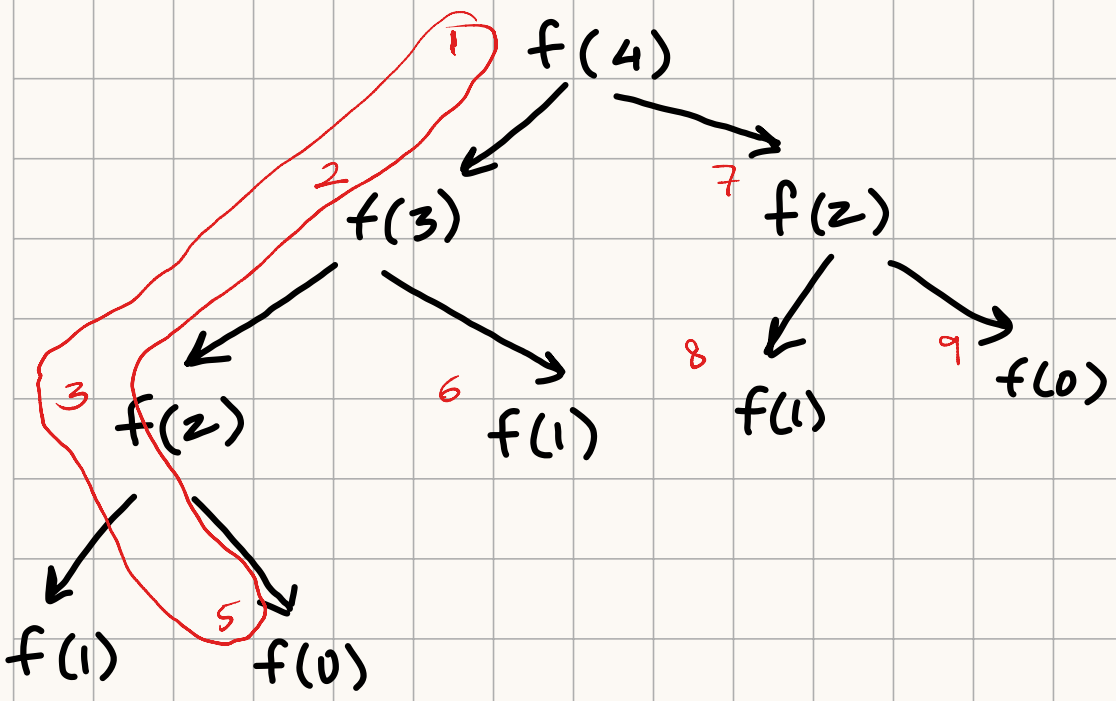
Insertion sort

Time complexity $O(n*2)$

Auxillary space . $O(1)$ → constant

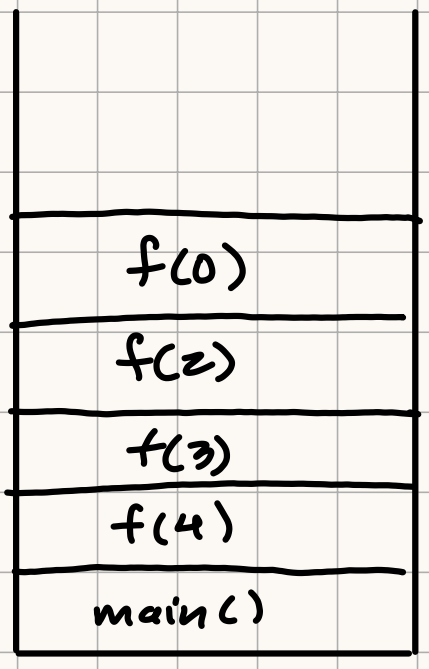
Sorting in place: Yes

Recursive Algorithm



Space complexity of above Recursive function?

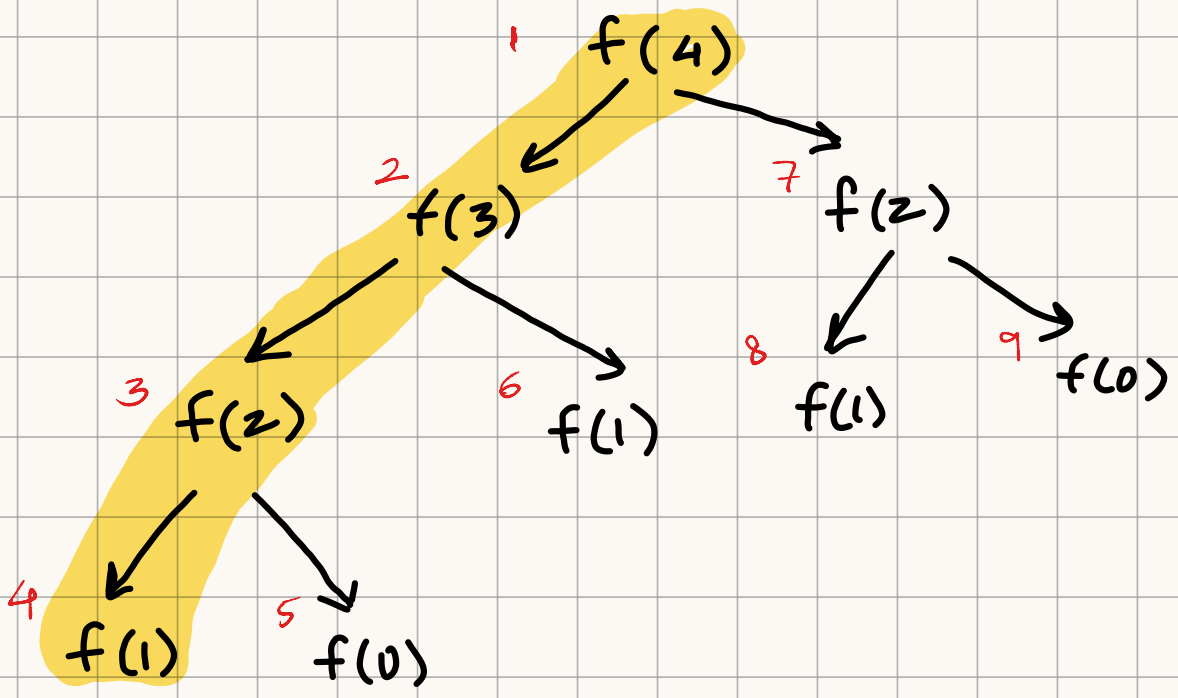
Space complexity of Recursive functions is not constant because function calls are stored in stack. Those function calls takes some memory in stack. Hence recursive function have no constant space complexity.



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

Tip. Only calls that are interlinked with each other will be in the stack at the same time

Space complexity of a recursion tree is the height of the recursive tree (path)



Which is equal to the total number of elements in that recursive program

In this case we have 4 since $N = 4$

Hence Space complexity auxillary space $= O(N)$

2 types of Recursion.

① Linear	② Divide & conquer
$F(N) = F(N-1) + F(N-2)$	$F(N) = F\left(\frac{N}{2}\right) + O(1)$

* Divide & conquer Recurrences

Form.

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

for all $x \geq x_0$ ← some constant

For binary search example.

$T(N) = T\left(\frac{N}{2}\right) + C \rightarrow a_1 = 1$ $\quad b_1 = \frac{1}{2}$ $\quad g(x) = C$ $\quad \varepsilon_1(x) =$

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

a_1 b_1 a_2 b_2

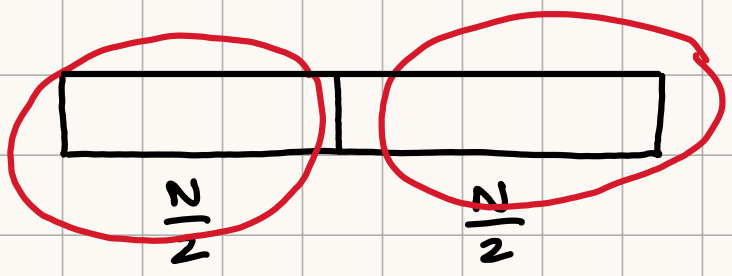
what is $g(x)$?

$g(x)$ basically means that if we take some simple divide and conquer rlu

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

a_1 b_1 $g(x)$

Divide and conquer divides the recurrence in 2 parts when you get the answer from the first part + what you are doing with that answer, takes how much time



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

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

(Recurrence relation of Merge sort)

★ How to solve to get complexity.

① Plug and Chug

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

② Master's Theorem

③ AKra Bazzi Formula. Best

Akra Bazzi Formula:

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

what is p?

$$a_1 b_1 + a_2 b_2 + \cdots = 1$$

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

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

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

a_1 b_1 $g(x)$

$k = 1$ since there is no a_2, b_2

$a_1 = 2$

$b_1 = \left(\frac{N}{2}\right)$

$$T(N) = \sum_{i=1}^{k} a_i b_i^{\,p} = 1$$

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

Via trial and error we can say $p = 1$

putting $P = 1$ in Akra Bazzi formula.

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

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

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

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

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

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

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

$$= \Theta(x\log x) \longrightarrow \text{Time complexity}$$

for array of size N: Merge sort 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 $g(x)$

finding P.

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

$$P = 2$$

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

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

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

→ ignoring less dominating term

$$= \Theta\left(x^2 \log x\right)$$

If you can't find value of P:

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

Let's try $P = 1$

$$T(x) = 3\left(\frac{1}{3}\right) + 4\left(\frac{1}{4}\right) = 1$$

$$= 2 > 1 \rightarrow \text{Increase the denominator}$$

$\therefore P > 1.$

Lets try $P = 2.$

$$T(x) = 3 \times \frac{1}{9} + 4 \times \frac{1}{16}$$

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

P u definitly less than 2

hence $1 < P < 2$

note when P < power of $(g(x))$
then ans = $g(x)$

Here $g(x) = x$

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

ans = $O(g(x))$

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

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

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

$\therefore p < 2$

$x^p \rightarrow$ Less dominating.

hence ans = $\theta(n^2)$

* Solving Linear Recurrences:

Homogeneous eqns

Example: $Fibo(N) = Fibo(N-1) + Fibo(N-2)$

Form

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

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

$n \rightarrow$ order of recurrence.

solution for fibonacci number

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

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

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

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

divide by α^{n-2}

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

(Characteristic equation of recurrence)

Roots of above equation:

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

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

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

(2) $f(n) = c_1 \alpha_1^n + c_2 \alpha_2^n$ is a solution for fibonacci

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

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

(2)

(3) fact: no of roots = no of ans you have already

Here we have 2 roots α_1 and α_2
Hence we should have 2 ans already

$F(0) = 0$ & $F(1) = 1$

$$f(0) = 0 = c_1 + c_2 \Rightarrow c_1 = -c_2 \quad \text{--- (3)}$$

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

from (3)

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

$$\boxed{c_1 = \frac{1}{\sqrt{5}}} \qquad \boxed{c_2 = \frac{+1-\sqrt{5}}{2}}$$

★ Putting this in equation (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 nth fibonacci number

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

$\frac{1}{\sqrt{5}}$ → constant to be ignored

$\left(\frac{1-\sqrt{5}}{2}\right)^n$ → as $n \to \infty$ this will be close to 0

Hence this is less dominating hence ignore.

Hence time complexity → $\boxed{O\left(\frac{1+\sqrt{5}}{2}\right)^N}$

$\frac{1+\sqrt{5}}{2}$ → Golden ratio

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

Q) Equal roots

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

① $f(n) = \alpha^n$

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

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

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

$\hookrightarrow \alpha = 1$ double root

* In general case if α is repeated r times then, α^n, $n\alpha^n$, $n^2\alpha^2$, $\cdots$ $n^{r-1}\alpha^n$ are all solutions to the recurrence

Hence I can take 2 roots as

$1, n\alpha^n$

$$f(n) = c_1(1)^n + c_2 n\alpha^n$$
$$= c_1 + c_2 n$$

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

$f(0) = 0 = c_1$

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

$c_2 = 1$

Ans. $f(n) = n \rightarrow$ Time complexity

$O(N)$

Non homogenous linear recurrences

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

↓

when extra function is there it is non hg-linear

How to solve:

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

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

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

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

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

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

Homogeneous soln → $f(n) = c_1 \alpha^n$.

$f(n) = c_1 4^n$

② Take $g(x)$ on one size and find particular solution

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

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

If $g(n) = n^2$. given a polynomial of degree 2

My guess: $f(n) = C3^n$

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

Particular solution $\Rightarrow f(n) = -3^{n+1}$

③ Add both solutions together.

$f(n) = c_1 4^n + (-3^{n+1})$

$f(1) = 1 \rightarrow c_1 4 - 3^2 = 1$

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

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

How do we guess particular solution?

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

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

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

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

Ex $g(n) = n^2 - 1 \Rightarrow$ guess of some degree $\rightarrow$ 2 degree.

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

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

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

Lets say you guessed $f(n) = az^n$ and if fails, then try $(an+b)z^n$, if this also fails increase the degree.

$$(a^2n + bn + c)z^n$$

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

① 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 solution

$$g(n) = z^n$$

Given $f(n) = az^n$

$$az^n = 2az^{n-1} + z^n$$
$$a = a + 1 \quad \times \text{ answer not coming}$$

Hence guess another one from our rules

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

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

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

$$a = 1$$

$$f(n) = n2^n \quad \text{// particular soln.}$$

③ General answer.

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

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

$$\boxed{c_1 = 1}$$

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

Complexity $= O(n2^n)$