① What is Time Complexity?

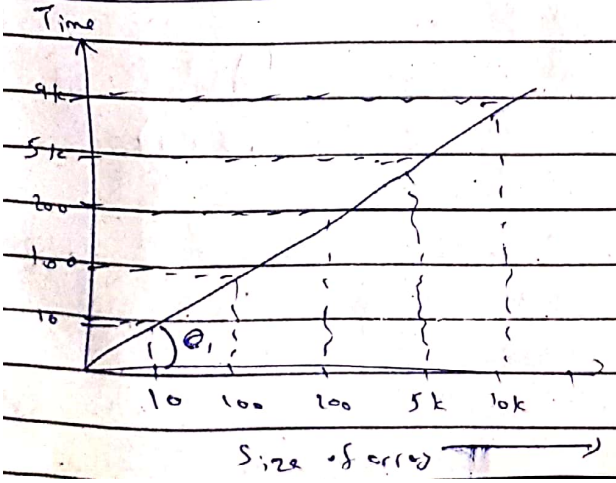
Old Computer	M1 Macbook
data: 1000000 elements in an array	//
Algorithm: Linear search → for target that is not present in an array	//
Time taken: 10 seconds	Time taken = 1 sec

② Which machine has better time complexity ???

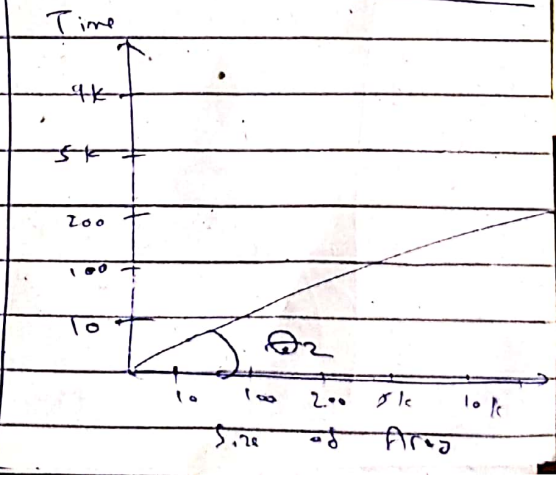
→ Both machines have same time complexity.

> Time Complexity ! = Time taken to run a program

Old machine



M1 Macbook



So the conclusion is that one lac size of array in old machine would take one lac seconds while one lac size of array in new machine would take 10 thousand seconds.

→ Time complexity is basically this graph,

Time complexity

How time grows as the size grows

→ Answer linearly

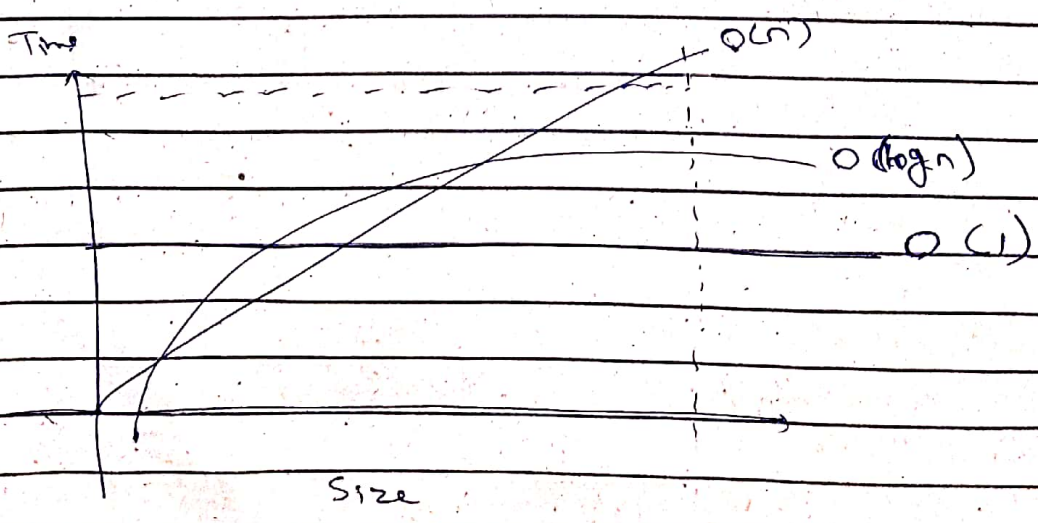
→ In both cases

→ So its a function that tells us how the time will grow as the input grows

* Why does it Matter?

→ Linear Search time complexity: O(n)

→ Binary Search time complexity: O(log n)



For a larger size of array, time complexity for binary search is less than linear search

⇒ So Log n is taking less time so it is efficient

⇒ Constant time complexity (O(1)) is lesser than Log (n) for large size of input. And we care only about larger size of input and not smaller.

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

What do we consider when thinking about complexity ?

1) Always look at worst case complexity.

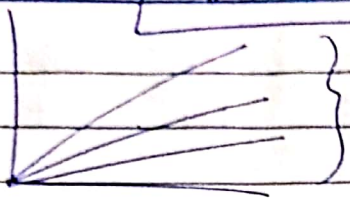
eg 2 million people using your website is more worth worrying than 10 people doing the same.

→ $O(1) < O(\log n) < O(n)$ is not true when the size is small But after some time, after a point when the input size is greater than the condition will hold true indefinately. That is what we should think about.

2) Always look for complexity for large input data.

Ignore Constants

3)



slopes

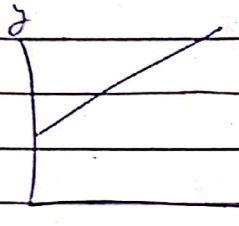
$y = x$

$y = 2x$

$y = 4x$

→ Even though value of actual time is different, they are all growing Linearly.

→ We dont care about actual time taken we care about how time grows as input grows.



$y = 3x + 5$ (Exact time taken)

→ We dont care about the exact time taken, we only care about relation

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

→ From part (2), we consider Large input data.

Hypothetically,

→ For 1 million size it is going take,

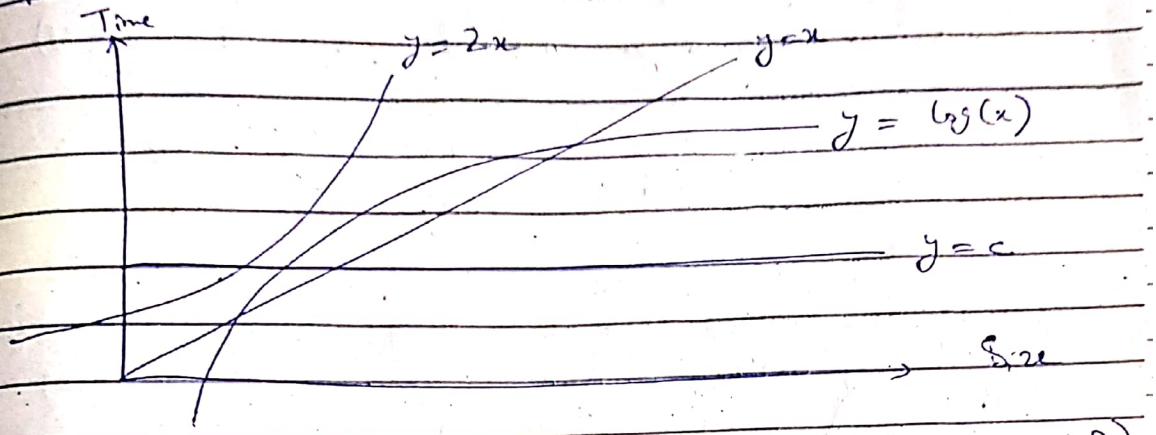
- $O((1\text{million})^3 \text{ sec} + \log(1\text{million}) \text{ sec})$
- $((1\text{million})^3 \text{ sec} + 6 \text{ sec})$

→ Does this 6 seconds have any significance when compared to 1 million

So,

Always ignore less dominant terms

Example :-



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

→ $O(2^n)$ represents exponentially growing complexity eg Fibonacci number using recursion i-e even for small amount of data it is taking a lot of time

* Big O Notation :-

→ Word Definition

→ $O(n^3)$

→ We know it means that size of array grows as input grows in an n^3 fashion

→ Big O is saying that this graph relationship is upper bound

→ It cannot exceed the relationship or graph or value n^3.

→ Maths : if

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

then

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

→ From point 2, Always look for larger values, here we are considering the limit from N to infinity.

→ Example, we know that

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

putting in formula

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

$$\Rightarrow \lim_{N \to \infty} 6 + \frac{3}{N^2} + \frac{5}{N^3}$$

$$\Rightarrow 6 + \frac{3}{\infty} + \frac{5}{\infty}$$

$$\Rightarrow 6 + 0 + 0$$

$$\Rightarrow 6 \quad (\text{which is less than infinity})$$

So for a limit in which N tends to go till near infinity $\frac{f(N)}{g(N)}$ will always be less than infinity.

→ So $O(N^3)$ means it will take at most N^3 time complexity.

Big Omega Notation :-

→ Word Definition

- Lower Bound
- Opposite of Big O notation
- $\Omega(n^3)$ here means, it will take at least $\Omega(n^3)$ time complexity.

→ Maths :-

if

$$f(n) = \Omega(g(n))$$

then

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

we need the worst case complexity so we consider Big O (upper bound)

Question :- What if an algorithm has Lower bound and upper bound as (N^2)

So, mathematicians introduced Theta Notation

Theta Notation :-

if

$$f(n) = \Theta g(n)$$

→ It means that function $f(n)$ grows at the same rate as the function $g(n)$ for Large values of n.

→ Word Defination

$\Theta(n^2)$ → Both upper bound and lower bound is (n^2)

→ Math

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

(*) Little O Notation :-

→ Word Defination

→ Loose Upper Bound

Little O Notation	Big O Notation
→ When we say $f(n) = o(g(n))$ it means	→ When we say $f(n) = O(g(n))$ it means
→ Function $f(n)$ grows strictly slower than function $g(n)$ for sufficiently large values of n	→ $f(n)$ grows as fast as $g(n)$ function for sufficiently large values of n
→ $f(n) < g(n)$	→ $f(n) \leq g(n)$

→ Math

→ if

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

then

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

Example, if $f = N^2$, $g = N^3$

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

$$\Rightarrow 0$$

(4) Little Omega Notation

→ Word Definition

↳ Loose Lower Bound

Little Omega Notation

→ When we say $f(n) = \omega(g(n))$ it means

↳ function $f(n)$ grows strictly faster than function $g(n)$ for sufficiently larger values of n

→ $f(n) = \omega(g(n))$

$f(n) > g(n)$

Big Omega Notation

→ When we say $f(n) = \Omega(g(n))$ it means

↳ function $f(n)$ grows at 'least as fast as $g(n)$ for sufficiently larger values of n

→ $f(n) = \Omega(g(n))$

$f(n) \geq g(n)$

→ Math.

↳ if $f(n) = \omega(g(n))$

then

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

Example:- $\lim_{n \to \infty} \frac{N^3}{N^2} = \lim_{N \to \infty} N = \infty$

① Space Complexity :-

Space Complexity = Input Space + Auxiliary Space

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

→ If you have 3 variables in your code then space complexity is constant

↳ Because if array is of size 100 or 2000 or 10000 every time it gonna take space of 3 only. Hence constant.

② Question :- Find the Time complexity?

```
for (i = 1 ; i <= N) {
    for (j = 1 ; j <= k ; j++) {
        // Some operation taking time T
    }
    i = i + k;
}
```

Inner loop = O (k t) time

Ans = O((k t) * (Times outer loop is running))

We know that outer loop is running as

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

And at the end, by loop condition I can say

$$1 + xk \leq N$$

$$xk \leq N-1$$

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

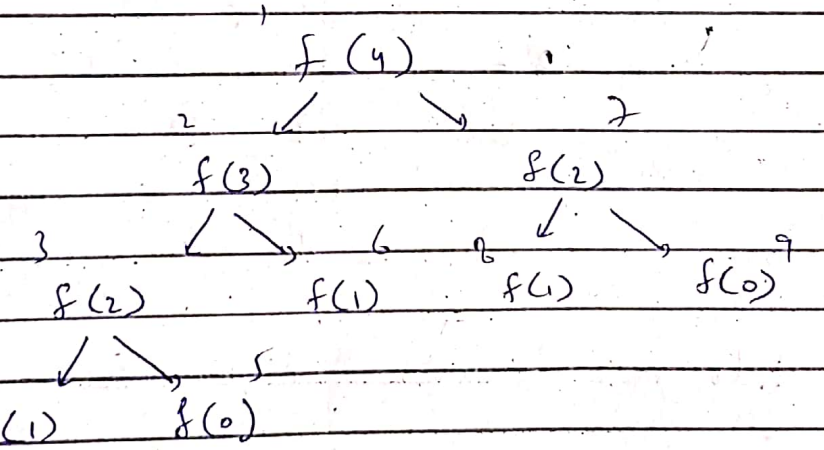
So,

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

$$= O(Nt)$$

⑧ Recursive Algorithms :-

1- Fibonacci

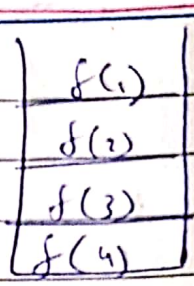


→ Here, at any particular point of time, no two function calls at the same level of recursion will be in the stack at the same time.

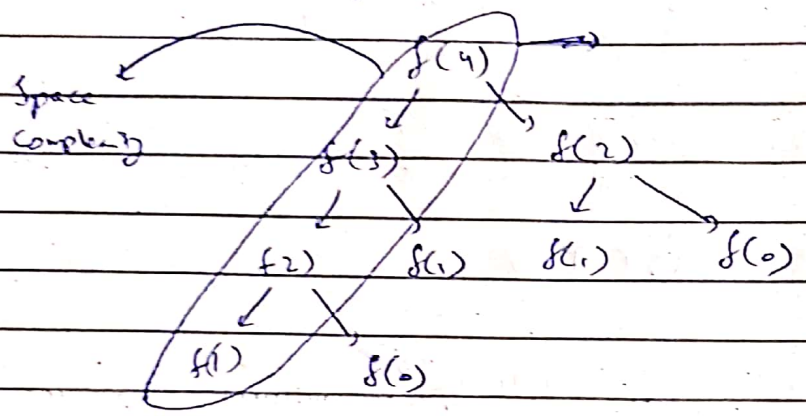
↳ ~~f(0)~~ Call 3 and 9 left say

→ Only calls that are interlinked will be in the stack at the same time.

→ Example Stack



Space Complexity is equal to the height of Tree of recursion



For n = 4, space complexity will be O(n)

↳ Because these all are in the stack at the same time i.e Maximum time required at the same time.

⊛ Solving Recursion Problems:

2 Types of Recursion

① Linear → Homogeneous / Heterogeneous

eg Fibonacci

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

② Divide & Conquer

eg Binary Search

$f(N) = f\left(\frac{N}{2}\right) + O(1)$

(4) Divide and Conquer Recurrence :-

- To identify if it is of divide and conquer or Linear. ⟶ Look at form

⟶ 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 $\forall\ x \geq x_0$ ↓ some constant

Here
- $g(x)$ is some other function
- $\varepsilon(x)$ is also a function

⟶ Example : Binary Search

$f(N) = f\left(\frac{N}{2}\right) + O(1)$

Can be written as

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

Here,

$\therefore a_1 = 1$, $\varepsilon(x) = 0$

$b_1 = \frac{1}{2}$, $g(x) = c$

⟶ Another example

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

a_1 b_1 a_2 b_2 $g(n)$

→ Another Example : merge Sort

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

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

a_1 b_1 $g(n)$

Here the intution is,

When you get answer from this + what are you doing with ans (takes how much time)

★) How to Solve to get the complexity ?

1. Plug and chug
2. Master's Theorem
3. Akra Bazzi (1996) (Recommended)

★) Akra Bazzi

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

What is p ?

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

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

Example :- Merge Sort

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

Here, $a_1 = 2$, $b_1 = \frac{1}{2}$, $g(x) = (N-1)$

Finding P

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

$$1^P = 1$$

$$\boxed{P = 1}$$

Putting in formula of Akra Bazzi

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

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

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

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

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

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

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

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

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

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

$$= \Theta\left(n\log n\right)$$ ∴ constants ignored

For Array of size n

↳ Time complexity of merge sort is $\Theta(n\log n)$

Question No 2:

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

Finding P:

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

By trial and error if $p = 2$ then

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

$$2 * \frac{1}{4} + \left(\frac{8}{9}\right) * \left(\frac{9}{16}\right) = 1$$

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

So $\boxed{p = 2}$

Putting in Akra Bazzi :-

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

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

$$= \Theta\left(x^2 + x^2 \left[\log u\right]_1^x\right)$$

$$= \Theta\left(x^2 + x^2 \left[\log x - \log(1)\right]\right)$$

$$= \Theta\left(x^2 + x^2 \left[\log x - 0\right]\right)$$

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

$$= \Theta\left(x^2 \log x\right)$$ ∴ ignoring less dominating terms.

*) Case :- If you cant find the value of P !!

Example. $T(n) = 3T\left(\frac{n}{3}\right) + 4T\left(\frac{n}{4}\right) + x$

Lets try P=1

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

$$1 + 1 = 1$$

$$2 > 1$$

∴ So $\boxed{P > 1}$,

bcz I need to increase the denominator value

Let's try $p=2$

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

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

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

So I need to increase the value of numerator or decrease the value of denominator

↳ So $\boxed{p < 2}$

> Note: if $p <$ power of $g(n)$ then ans $= g(n)$

↳ Here, $g(n) = n^2$ and $p < 2$

So, $\boxed{\text{ans} = g(n)}$

i.e Time complexity is $O(g(n))$

Proof: Let's say we have

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

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

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

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

Since we know $p < 2$ so n^p will be less dominating term, hence ignored

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

which is $g(x)$.

⊛ Linear Recurrence Relations :-

Example: Fibonacci Number

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

Form

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

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

$n :=$ order of recurrence

Solution :=

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

Step 1 :- Put α^n for some constant α in $f(n)$

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

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

Dividing by α^{n-2}

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

Step 2 :- Roots of this Quadratic Equation

$$\Rightarrow \alpha = \frac{1 \pm \sqrt{5}}{2} \qquad \therefore \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

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

Note:- → If in any equation, we get the number of roots like 2 in this case,

→ Take that many number of constants and multiply that many number of roots with the power of n.

→ Add All

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

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

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

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

→ So,

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

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

$\Rightarrow$ From (3)

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

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

Putting in (2)

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

↳ Formula for n^{th} Fibonacci number

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

$\therefore \left(\frac{1-\sqrt{5}}{2}\right)$ result in 0.6 which is < 1 so it is less dominating term

$\frac{1}{\sqrt{5}}$ is constat so,

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

↙ Golden ratio

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

*) What if you get equal number of roots?

Example

↳ $f(n) = 2f(n-1) + f(n-2)$ ... (1)

Step 1:- Put $f(n) = \alpha^n$

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

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

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

Dividing on both sides

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

$\Rightarrow \quad \alpha = 1 \longrightarrow$ double roots

$\longrightarrow$ (2)

> Note:- If α is repeated r times then $\alpha^n, n\alpha^n, n^2\alpha^n, n^3\alpha^n, \ldots$ $\ldots n^{r-1}\alpha^n$ are all solutions to the recurrence

Hence I can take two roots as one is 1 itself and other can be lets say $n\alpha^n$

Putting in formula

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

From (2)

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

If answers given are,

$f(0) = 0$

$\Rightarrow c_1 + c_2(0) = 0$

$\boxed{c_1 = 0} \longrightarrow (3)$

$f(1) = 1$

$\Rightarrow c_1 + c_2(1) = 1$

$c_1 + c_2 = 1$

From (3)

$\boxed{c_2 = 1}$

Time complexity = $O(n)$

Note :- These were all homogeneous equations as they did not have $g(n)$

★ Non-Homogeneous recurrence relations :

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

Note: This is non-homogeneous equation as it has $g(n)$

Replace $g(n)$ with 0 and solve usually

~~Step 1.~~ ~~Put $f(n) = \alpha^n$~~

Example :-

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

Step 1 - Replacing $g(n)$ with 0

$$f(n) = 4f(n-1) + 0$$
$$f(n) = 4f(n-1)$$

Step 2. Putting $f(n) = \alpha^n$

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

Homogenous Solution is:

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

$$\boxed{f(n) = c_1 4^n} \quad \text{---(1)}$$

Step 2:-

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

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

Imp

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

↳ Particular Solution

↳ Example if $g(n) = n^2$, guess something that is a polynomial of degree 2

For this question if $g(n) = 3^n$,

guess $= c3^n$

putting in equation

$$c3^n - 4c3^{n-1} = 3^n$$

$$\Rightarrow c3^n - 4c3^{n-1} - 3^n = 0$$

$$c = -3$$

Particular solution would be putting c in guess

$$f(n) = (-3)(+3)^n$$

$$\boxed{f(n) = -3^{n+1}} \quad \text{---(2)}$$

Step 3: Add both solutions together

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

Given that $(f(1) = (1)$

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

$$1 = c_1 4 + 9$$

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

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

Ans

* How to guess Particular Solution ?

① If $g(n)$ is exponential, guess of same type

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

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

② If $g(n)$ is polynomial, guess of same degree

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

guess: ⟶ will be of degree 2

⟶ $f(n) = an^2 + bn + 1$

③ If there is combination of exponential and polynomial.

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

Guess: $f(n) = a2^n + (an + b)$

Note ⟶ Lets say you guessed $f(n) = a2^n$ and it fails, the increase the degree $f(n) = (an+b)2^n$, if it also fails then guess $f(n) = (an^2+bn+c)2^n$ and so on.

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

Put $2^n = 0$

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

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

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

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

Guess particular solution

Since $g(n) = 2^n$

Guess $\Rightarrow f(n) = a2^n$

Put in main eq taking $g(n)$ on one side

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

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

$a = a+1$ ⟶ This is wrong

Hence make another guess

Guess $\Rightarrow f(n) = (an+b)2^n$ ——— (1)

Putting again

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

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

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

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

$\boxed{a = 1}$

Putting in (1)

$f(n) = (1(n)+b)2^n$

Discarding b

$f(n) = n2^n$

Step 3 :- Adding two solutions :-

$f(n) = c_1 2^n + n2^n$ ——— (2)

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

$\Rightarrow \boxed{c_1 = 1}$

Now, Putting in (2)

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

Time Complexity $= O(n2^n)$