TIME AND SPACE COMPLEXITY

1. What is Time complexity?

OLD COMPUTER	M1 MacBook (Very Fast)
Data: 1 Million array elements in an array. Algorithm: Linear Search Target: Number that doesn't exist. Time Taken: 10sec	Data: 1 Million array elements in an array Alg: Linear Search. Target: Number that doesn't exist. Time Taken: 1sec.

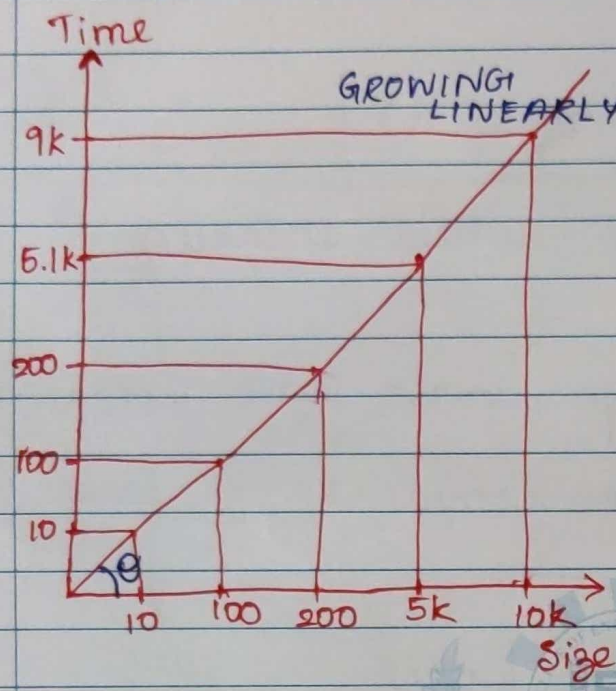
Both have the same time complexity.

Time complexity != Time Taken

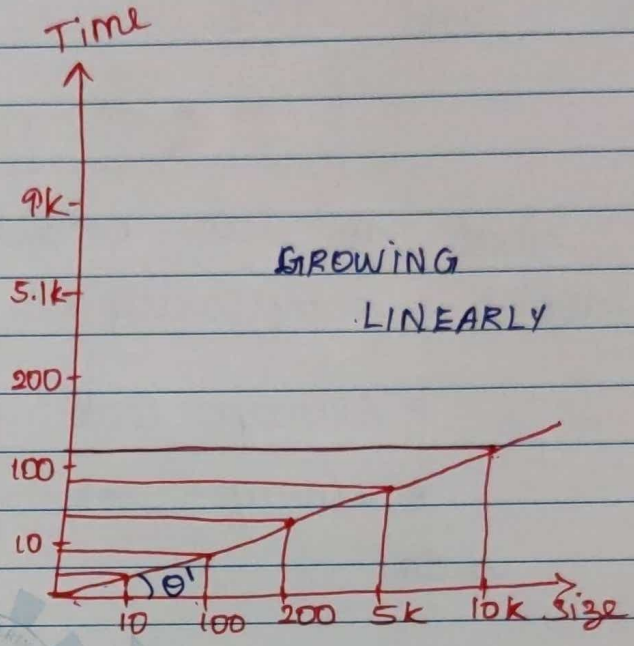
Exp. No.:.....
Date :.....

Page No.:.....

OLD MACHINE

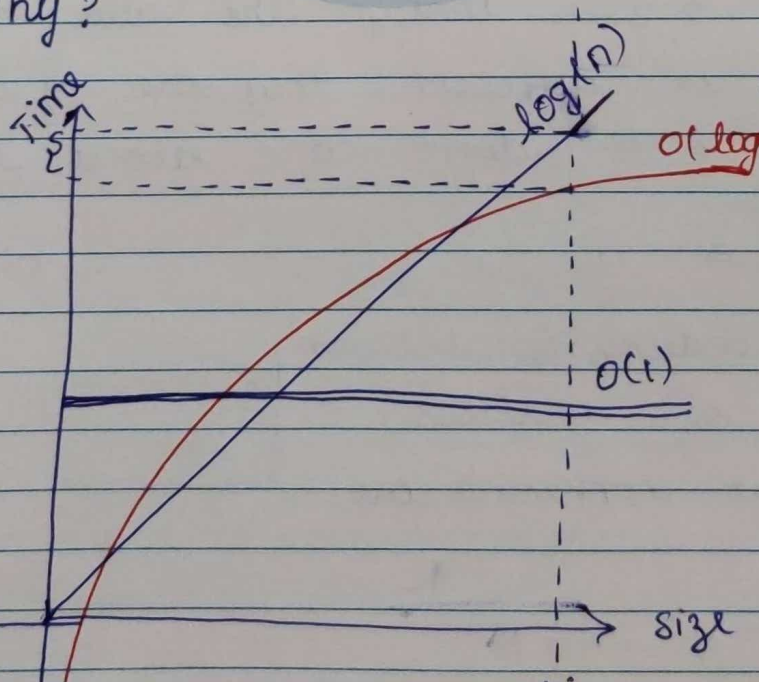


MI MACBOOK



* Time Complexity is the Mathematical function that tells us how the time will grow as the input grows

2. Why?



~~log n~~ linear search

O(logn) - binary search

O(n) - linear search

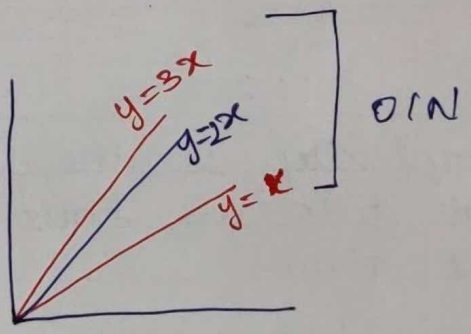
O(1) - Constant time

For larger size, binary search is better than linear search.

NOTE : Always look for larger values / worst case .

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

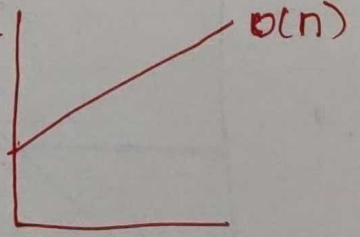
What do we consider when thinking about complexity ?

* Always look for worst case complexity
* Always look for complexity of large / ∞ data.
* 

⇒ Even though the value of actual time is different, they are growing linearly

⇒ We don't care about the actual time.

⇒ $y = 3x + 5$



This relationship between time and size matters, that's what constants are ignored.

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

From point no. 2, always ignore less dominating terms.

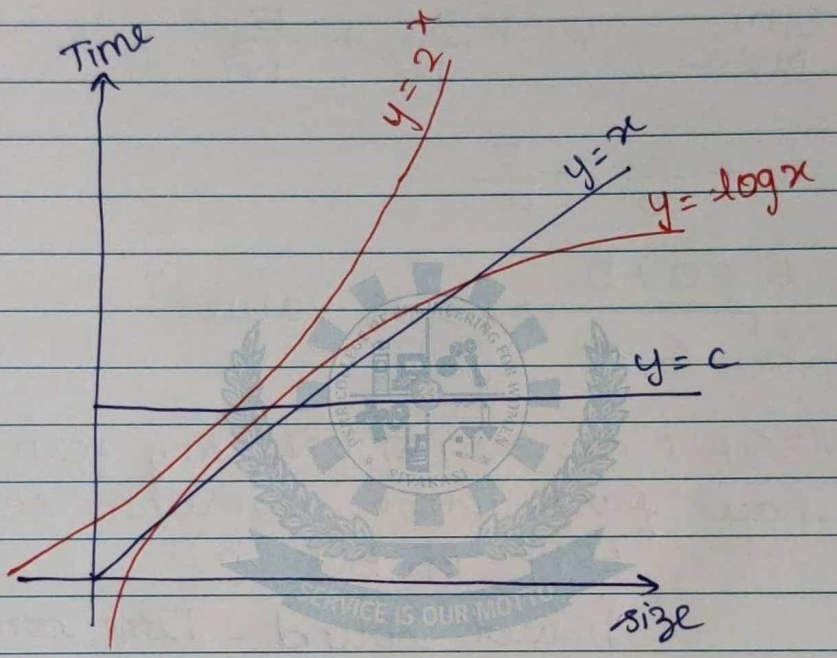
Exp. No.:.....
Date :.....

Page No.:.....

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

* Ignore the constants
* Ignore less dominating terms

$\Rightarrow O(N^3)$



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

Big - Oh Notation (O) : Any formula that will bound how slow your algorithm can be

* Word definition : Upper bound - The complexity cannot exceed $O(N^3)$.

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

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

Eg: $O(N^3) = O(6N^3 + 3N + 5)$
$g(N)$ $\quad f(N)$

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

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

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

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

$$= 6 + 0 + 0$$

$$= 6 \quad (< \infty)$$ → finite value

BIG OMEGA (Ω) NOTATION: Any formula that bound how fast your algorithm can be

Word :. (Opp. of Big-Oh)

Lower bound - Time complexity will never be less than $\Omega(N^3)$

Maths :.

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

NOTE: We always look for big-Oh notation. Since, we ~~go~~ look for worst case.

Exp. No.:.....
Date :.....

Page No.:.....

BIG THETA (Θ) NOTATION :

Word : Both upper & lower bound

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

$= \Theta(N^2)$

Maths :

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

Big-Oh	little-Oh
Strict upper bound.	Loose upper bound.
$f = O(g)$ ie., the growth of f isn't greater than g. which means $f <= g$	$f = O(g)$ ie., $f < g$ strictly slower.

little-Oh Notation :

Maths : $f(N)$ is smaller than $g(N)$

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

Eg :

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

Little Omega Notation (ω)

Loosely lower bound.

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

Big - Omega	Little - Omega
$f = \Omega(g)$ ~~$f = \Theta(g)$~~ i.e, $f >= g$	~~$f = O(g)$~~ $f = \omega(g)$ $\Rightarrow f > g$.

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

SPACE COMPLEXITY:

Both auxiliary space and space used by the input.

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

Inner loop : $O(kt)$ time.

Ans: $O(kt$ * how many times outer loop running)

$O(kt$ x No. of times O.L is running)

Exp. No.:
Date :

Page No.:

i - is running

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

$1 + xk \leq N$

$xk \leq N-1$

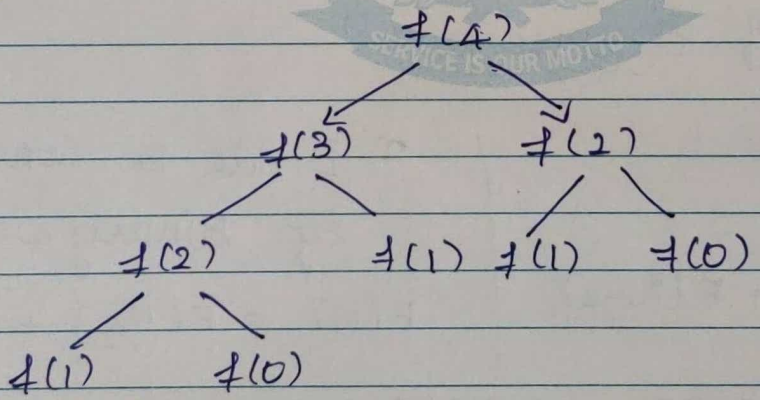
$$x \leq \frac{N-1}{k}$$ ⇒ No. of times the outer loop is running.

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

$$= O(Nt)$$

RECURSIVE ALGORITHMS

Example:



* Recursive calls take some memory in stack. Hence, recursive programs do not have constant space complexity.

*** 1. In any level of recursive tree, is there any possibility for more than one function call in that particular level to be in the stack at any time.

** No two function call at the same level in the recursion will be in the stack. In other words, Only the calls that are interlinked with each other will be on the stack at the same time.

Space Complexity is the longest chain ie, height of the tree.

Two types of Recursions:

1) Linear
Eg: Fibonacci
$F(N) = F(N-1) + F(N-2)$

2) Divide & Conquer
Eg: Binary Search.
$F(N) = F(N/2) + O(1)$

DIVIDE & CONQUER RECRRENCES:

Form:

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

$$\forall\ x \geq x_0$$

where x_0 - some constant

Exp. No.:
Date :

Page No.:

Example

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

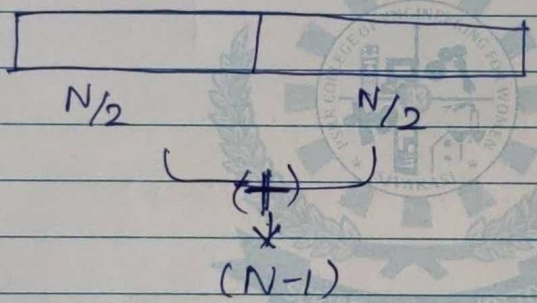
$a_1 = 1 \quad b_1 = 1/2 \quad \leq_1(x) = 0 \quad g(x) = C$

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

$a_1 \quad b_1 \quad a_2 \quad b_2 \quad g(N)$

3) $T(N) = 2\,T\left(\frac{N}{2}\right) + (N-1)$ → It is when we get ans from $2T(N/2)$ + what we are gonna do with the ans takes how much time.

$a_1 \quad b_1 \quad g(N)$



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

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

which is recurrence Relation of Merge Sort.

How to actually solve to get complexity?

* Plug and chug
* Master's Theorem
* 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?

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

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

$\Theta(c)$; $\Theta(2k+c)$;

Θ(Any constant *1)

$\Rightarrow \Theta(1)$

Example: Binary Search.

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

Merge Sort

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

$a_1 = 2$ $\quad g(x) = (N-1)$

$b_1 = 1/2$

$$\Rightarrow 2 * (1/2)^p = 1$$

$$\boxed{p = 1}$$

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

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

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

$\frac{1}{u}\, du = \log u$

$\int u^{-2}\, du$:

$= \frac{u^{-2+1}}{-2+1}$

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

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

Exp. No.:
Date :

Page No.:

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

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

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

$$T(x) = \theta(x \log x)$$

Time Complexity for the array of size N for Merge Sort is $O(N \log N)$

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

$$(2)(1/2)^p + (8/9)(3/4)^p = 1$$

If $p = 1$

$$= 2(1/2) + 8/9\,(3/4)$$

$$= 1 + 2/3 \neq 1$$

If $p = 2$

$$= 2(1/2) + 8/9\,(9/16)$$

$$= 1$$

$$\therefore \boxed{p = 2}$$

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

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

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

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

Q: If we can't find p?

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

If P=1

$= 3(1/3) + 4(1/4)^1$

$= 1 + 1 = 2 \neq 1$

$2 > 1 \Rightarrow$ Increase the denominator.

If P=2

$= 3(1/9) + 4(1/16)$

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

$= \frac{7}{12} < 1 \Rightarrow$ Decrease the denominator

$\Rightarrow$ Decrease the value of P

$\therefore \boxed{P < 2}$

NOTE:

When $p <$ power of $(g(x))$ then ans $= g(x)$

Since, less dominating terms will be neglected.

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

$\therefore$ Ans $= O(g(x))$

$= O(N^2)$

Exp. No.:
Date :

Page No.:

SOLVING LINEAR RECURRENCES (Homogeneous)

Example: $F(N) = F(N-1) + F(N-2)$ → (1)

Form:

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

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

For a_i, n is fixed

$n \Rightarrow$ order of Recurrence

Solution :-

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

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

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

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

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

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

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

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

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

STEP-3: (Fact)

No. of roots = No. of ans we have already

Here, we have 2 roots. α_1 & α_2

Hence, we should have 2 ans already

i.e., $f(0)=0$ & $f(1)=1$

$f(0) = 0$

$c_1 + c_2 = 0$

$c_1 = -c_2$

$f(1) = 1$

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

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

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

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

$$c_1 = \frac{1}{\sqrt{5}} \quad \Big| \quad c_2 = -\frac{1}{\sqrt{5}}$$

(2) $\Rightarrow$

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

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

This is the formula n^{th} Fibonacci No

Exp. No.:
Date:

Page No.:

Time Complexity :-

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

↓

very small, hence neglected.

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

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

or

$= O(1.6180)^n \rightarrow$

⇑

Golden Ratio.

Q: Equal Roots:

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

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

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

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

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

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

$\alpha = 1, 1$

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 solutions to the recurrence

Here $\alpha = 1, 1$. Hence, can take 2 roots as 1 $n\alpha^n$

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

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

$f(0) = 0 \qquad f(1) = 1$

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

$0 = c_1$

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

$1 = 0 + c_2$

$c_2 = 1$

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

$f(n) = n$

$\therefore$ Time Complexity $= O(N)$

Linear Recurrence (Heterogeneous / Non-Homogeneous)

↓

Have $g(x)$

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

↓

This extra function represents 'non-homogeneous.

Exp. No.:
Date :

Page No.:

How to solve

STEP-1: Replace g(x) by 0. & solve usually.

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

① $f(n) = 4f(n-1) + 0$

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

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

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

$\div \alpha^{n-1}$

$\alpha - 4 = 0$

$\boxed{\alpha = 4}$

Homogeneous Solution $f(n) = C_1 \alpha^n$

$f(n) = C_1 (4)^n$

STEP-2 : Take $g(n)$ on one side and find particular solution.

We need to find something similar to $g(n)$ → particular solution

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

* If $g(n)$ is n^2, then guess a polynomial of degree 2.

My guess: $f(n) = c3^n$

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

$$\div 3^{n-1}$$

$$c(3) - 4c = 3$$

$$-c = 3$$

$$c = -3$$

particular solution $= -3(3)^n$

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

STEP - 3: Add both solution:

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

$$f(1) = 1$$

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

$$4c_1 = 1 + 9$$

$$c_1 = \frac{10}{4}$$

$$\boxed{c_1 = 5/2}$$

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

Exp. No.:
Date :

Page No.:

How do we guess the particular solution:

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

Eg: if $g(n) = 2^n + 3^n$

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

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

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

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

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

Guess: $a2^n + bn + c$

* Lets say, if the guess $f(n) = a2^n$ fails, then try $(an+b)2^n$, if this too fails then try, then increase the degree like $(a^2n + bn + c)2^n$ and so on.

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

① $2^n = 0$

$f(n) - 2f(n-1) = 0$

② $f(n) = \alpha^n$

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

$\div \alpha^{n-1}$

$\alpha - 2 = 0$

$\alpha = 2$

③ Guess particular solution.

$g(n) = 2^n$

then Guess = $f(n) = a2^n$

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

$a = a + 1$ ✗ wrong

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

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

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

$+a = 1$

$\boxed{a = +1}$

discard b.

$f(n) = n2^n \Rightarrow$ particular solution

③ General solution

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

Exp. No.:.....
Date :.....

Page No.:.....

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

$\boxed{c_1 = 1}$

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

Complexity $= O(n2^n)$

x