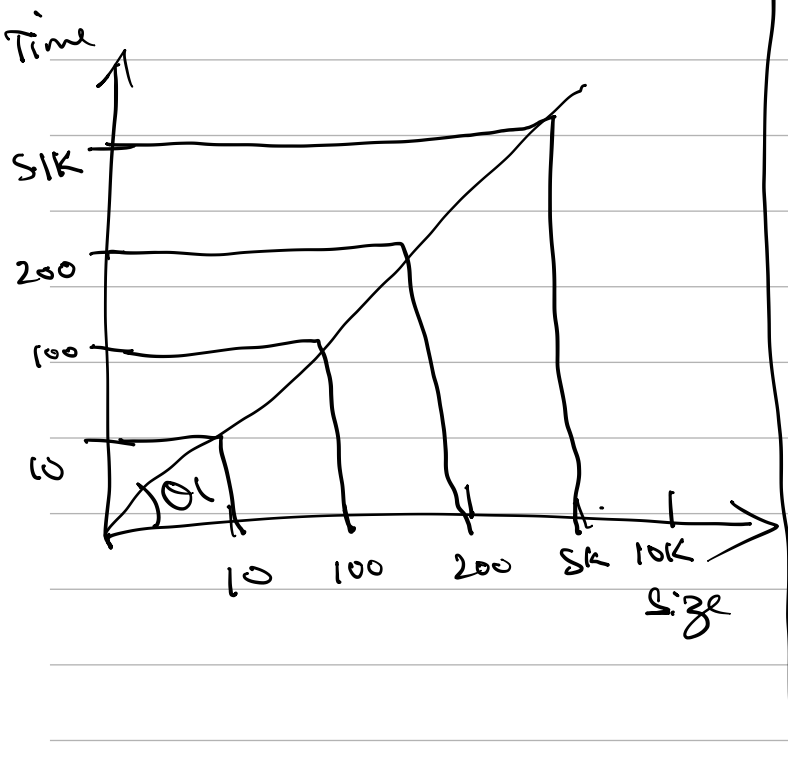
① What is Time Complexity ?

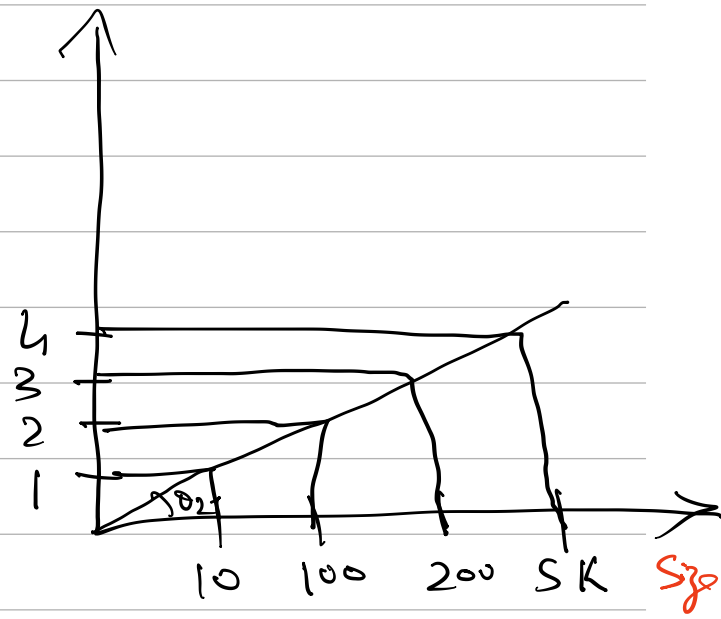
NOTE:- Time complexity does not depend on the speed of the computer,

So, Time complexity != Time taken.

Old Machine



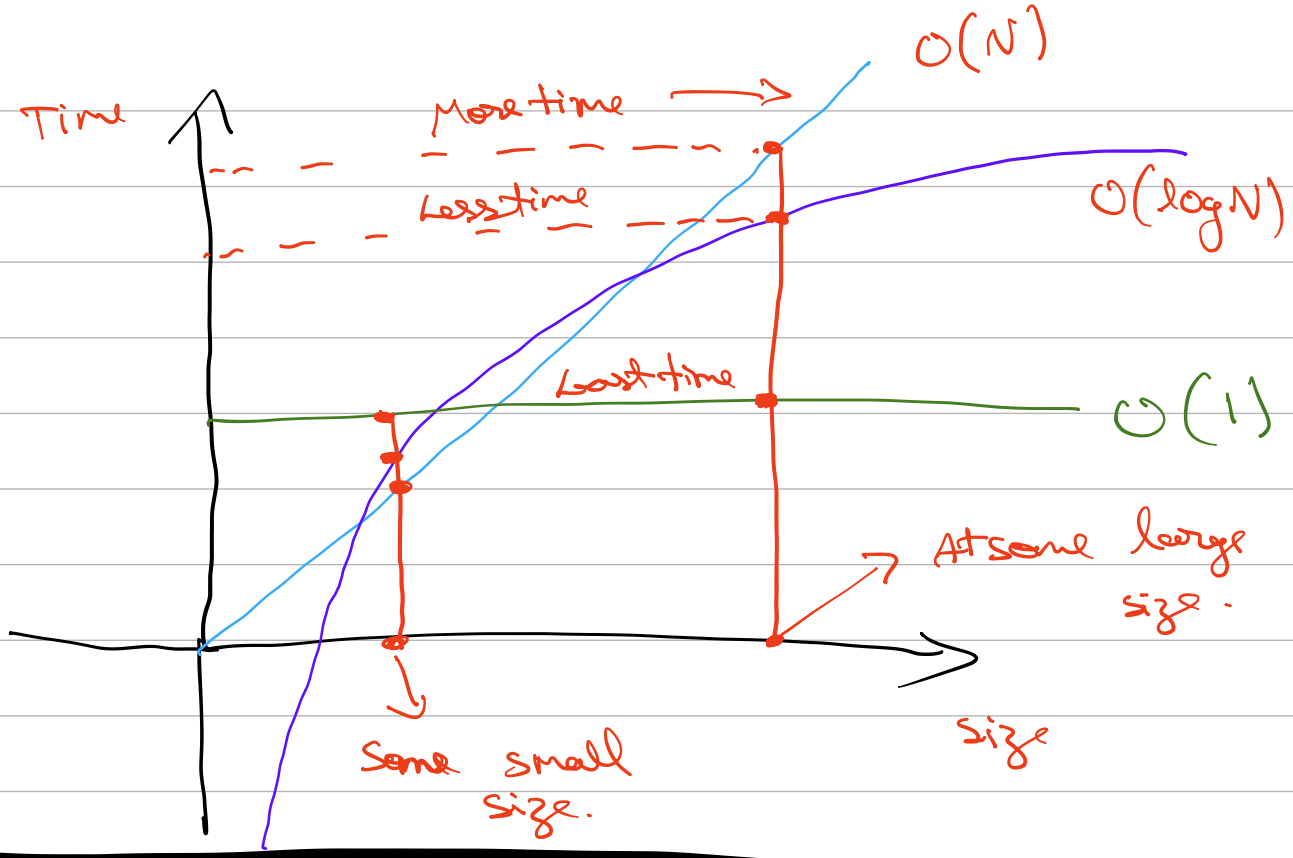
New Machine



*). Time complexity is a function which tells how time grows as the input size grows. i.e Relationship b/w time and input.

Example:- In above diagram even if new machine takes 1sec for 10 input where as old machine takes 10 sec for 10 input, the slope is linear slope in both the cases.

Hence time complexity is same in both the machines.



Therefore at large size.)
Time taken
$O(1) < O(\log N) < O(N)$.

NOTE: We don't consider time complexity for small size of inputs.

As we observe at small size
Time taken $O(1) > O(\log N) > O(N)$.

When to consider Time complexity?

→ Always look for worst case complexity.

→ Always look at complexity for large data.

→ We don't care about time taken, we only think

about relationship between time and size.

→ As we don't care about actual time, we ignore all constants.

Eg:- $3n+5$ → constant is ignored.

→ Always ignore less dominating terms.

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

Ignore constants.

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

Ignore less dominating terms.

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

$= O(N^3)$ Answer Time complexity.

To visualize this suppose $N = 1$ million.

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

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

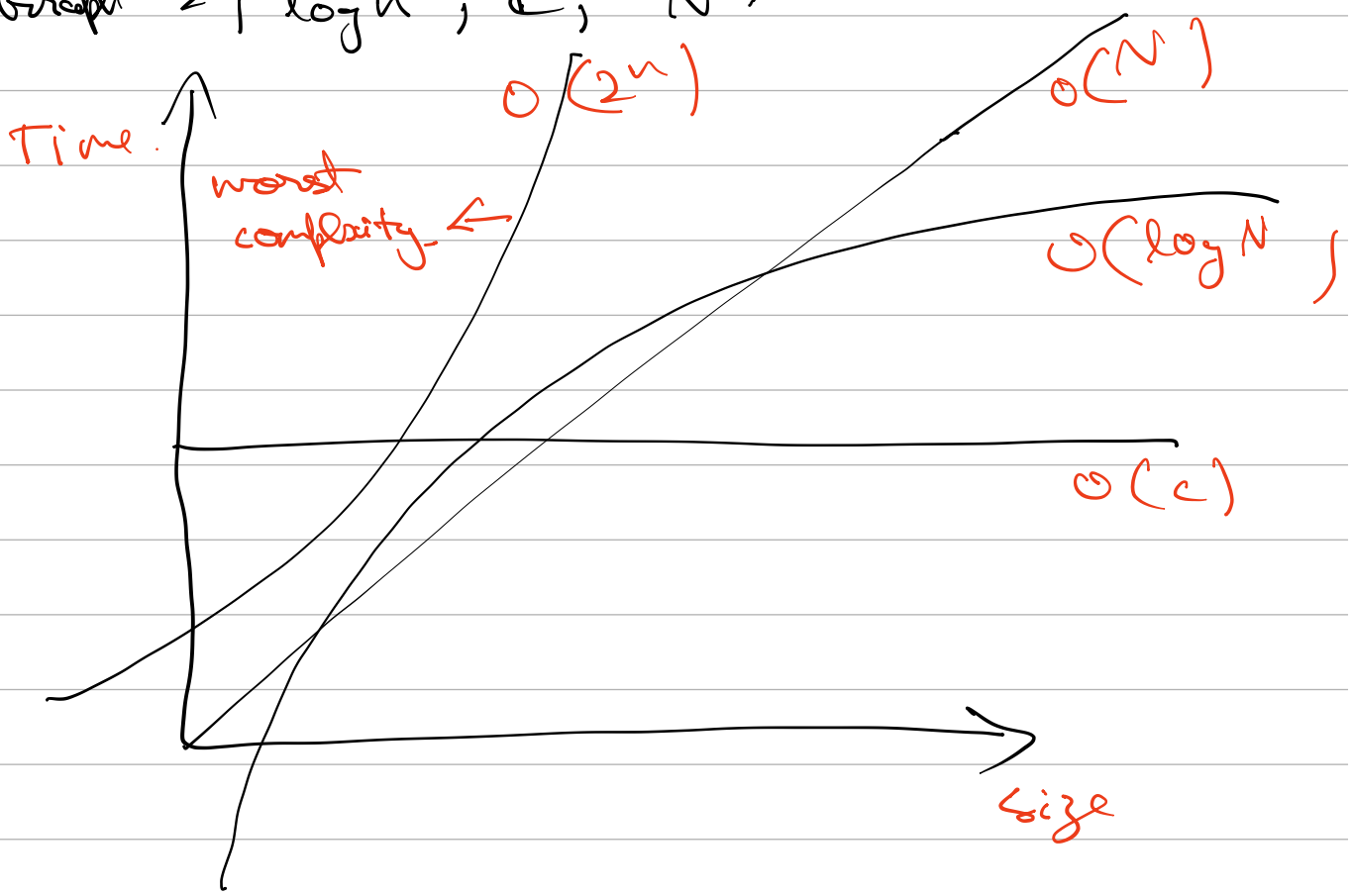
$O((1\text{million})^3 + 6)$

↳ negligible.

So, ignoring less dominating term $(\log N)$

$= O(N^3)$

Graph 2^n, $\log n$, c, N.



TIME COMPLEXITY ORDER.

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

BIG-Oh NOTATION :-

→ Gives upper bound Time complexity.

→ Suppose $f(n)$ → Time complexity.

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

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

$$O(\underbrace{6N^3 + 3N + 5}_{f(n)})$$

Suppose $g(n) = N^3$

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

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

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

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

$$= 6 + 0 + 0$$

$$= 6 < \infty$$

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

So, $g(n) = N^3$ is correct time complexity.

$O(N^3)$

$O(N^3)$ means, the algorithm can be less than or equal to $O(N^3)$ complexity but never more.

BIG OMEGA :-

→ Opposite to Big Oh

→ Lower bound Time complexity.

$\Omega(N^2)$ → means algorithm. greater or equal to $\Omega(N^2)$ time complexity.

Here

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

What if :-

An algorithm has lowerbound and upperbound as (N^2).

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

We use Theta Notation

THETA NOTATION :-

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

LITTLE OH NOTATION :-

→ This also gives upper bound.

→ BUT ! not strictly upperbound.

MEANING

Bigoh	Little oh.
If $f(n) = O(g(n))$	If $f(n) = o(g(n))$
$f(n) \leq g(n)$	$f(n) < g(n)$
	↓ means algorithm is only lesser than $g(n)$.

Here

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

Because $f(n)$ is strictly lower than $g(n)$.

Eg:- $f(n) = n^2$, $g(n) = n^3$

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

$$= 0 \quad //$$

LITTLE OMEGA

→ Strictly lower bound.

If $f(n) = \Omega(g(n))$ ⟶ normal Big omega.

means $f(n) >= g(n)$

BUT!

Little omega

$f(n) > g(n)$.

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

//

---

*]. AUXILIARY SPACE :-

Extra space or temporary space used by an algorithm.

SPACE COMPLEXITY :-

Auxiliary space + Input space.

FINDING TIME COMPLEXITIES :-

Q]. for(i=1; i<=N;) {

for (j=1 ; j<= k ; j=j+1) {

// some operation that takes time t.

}

i = i + k

}

Solu^n Inner loop: O(kt) time

Answer = O(kt x times the outer loop is running)

We observe

i = 1, 1+k, 1+2k, 1+3k, ... (1+k+k)

- - - - - - - - - - - - - -

, 1 + xk.

x -> times the outer loop is running.

$1 + xk \leq N$

$xk \leq N - 1$

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

So, $O(k + x \times 1)$

$\Rightarrow O\left(k + x \frac{(N-1)}{k}\right)$

$\Rightarrow O(N+k)$ Answer.

Complexities of sorting algorithms:-

1). Bubble sort:- Here swapping is optional

Time Best Case = $O(N)$ (when already sorted) No swapping.

Time Worst Case = $O(N^2)$ swapping.

(array is reverse sorted)

STABLE ✓

Auxiliary Space : O(1)

2]. Selection Sort :- Here comparison is done everytime

Time Best, Average, Worst Case

$= O(N^2)$

Space complexity = 1.

NOT STABLE ✓

3]. INSETION SORT :-

WORST TIME COMPLEXITY :- $O(N^2)$

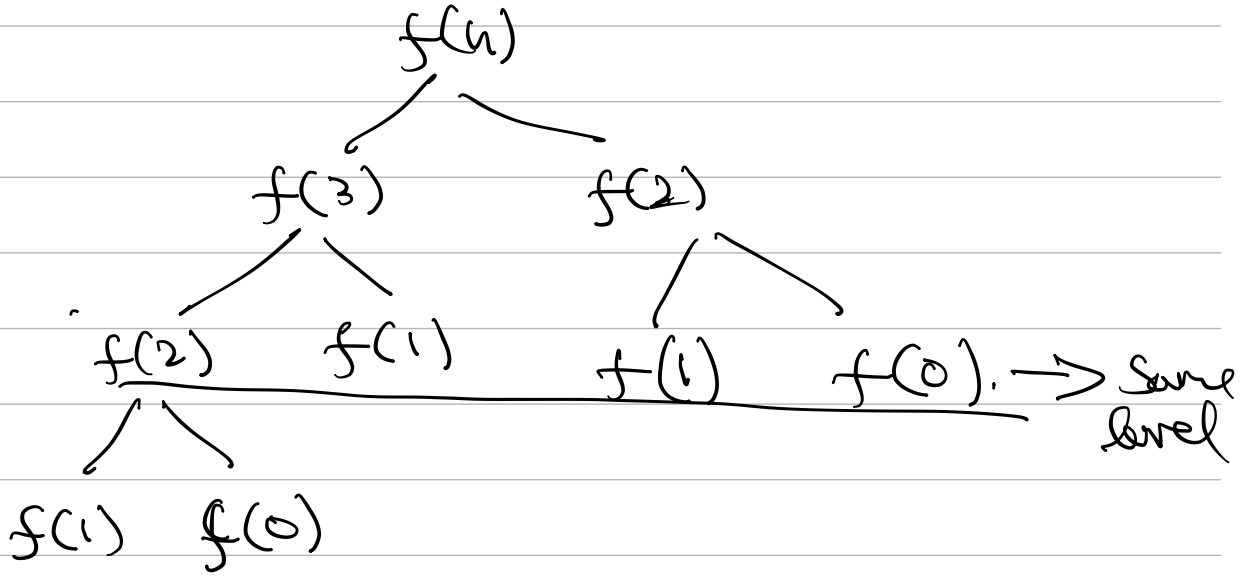
Auxiliary space : O(1)

BEST TIME COMPLEXITY :- O(N).

Here swapping is optional

SPACE COMPLEXITY FOR RECURSIVE ALGORITHMS

Recursive tree for Fibonacci.



NOTE:- No 2 function calls in the same level can be in the stack at same time.

For recursive Algorithms:-

Space complexity = Height of Tree

For example for n^{th} fibonacci number

Space complexity = $O(n)$

2 TYPES OF RECURSIONS:-

① LINEAR	② Divide & Conquer.
Eg:- Fibonacci Number	Eg.- Binary search
$f(n) = f(n-2) + f(n-1)$	$f(n) = f(\frac{n}{2}) + O(1)$.

DIVIDE AND CONQUER RECURRENCES :-

Form:-

$$T(x) = a_1 T(b_1 x + \varepsilon_1(x)) + a_2 T(b_2 x + \varepsilon_2(x))$$

$$+ \ldots\ldots\ldots + a_k T(b_k x + \varepsilon_k(x)) + g(x),$$ Here $x \geq x_0 \rightarrow$ some constant.

Cool down! It's not confusing.

Here $a_1, a_2 \ldots a_k$, $b_1, b_2 \ldots b_k$ are constants.

$\varepsilon_1(x), \varepsilon_2(x) \ldots \varepsilon_k(x)$ are functions.

$g(x)$ is a function at last.

IT IS SIMPLE!

FOR EXAMPLE :-

$$F(N) = F\left(\frac{N}{2}\right) + C.$$ → Binary search recurrence relation.

Also can be written as,

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

Here, $a_1 = 1$

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

$x = N$

$\varepsilon(x) = 0$.

$g(x) = C \rightarrow$ constant.

ANOTHER EXAMPLE :-

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

$9 \rightarrow a_1$, $\frac{N}{3} \rightarrow b_1$, $\frac{4}{3} \rightarrow a_2$, $\frac{5}{6} \rightarrow b_2$, $4N^3 \rightarrow g(N)$

AS EASY AS THIS

HOW TO ACTUALLY SOLVE TO GET COMPLEXITY :-

① **Plug & Check** → expand if further

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

★ In this method $f\left(\frac{N}{2}\right)$ is expanded further taking $N = \frac{N}{2}$.

★). NOT GOOD METHOD SO.,

MOVE ON!

---

② . MASTER'S THEOREM :-

Again Better, but not the best so.....

MOVE ON!

③ Akra Bazzi (1996)

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

HANG ON! DONT BE SCARED OF INTEGRATION..

ONLY SIMPLE FORMULA OF INTEGRATION is used.

$$\int u^p du = \frac{u^{p+1}}{p+1}$$

WHAT IS P?

Remember,

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

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

$a_1 = 2$ $\quad g(n) = n-1$

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

Step 1:-

Finding p.,

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

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

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

Here $\boxed{p = 1}$.

Step-2:- Put p in formula.

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

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

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

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

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

$$= O\left(x + x\left(\log x + \frac{1}{x} - 1\right)\right)$$

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

$$= O\left(x\log x + 1\right)$$

Ignore constant, while finding Time complexity,

$= O(x\log x)$ // Time Complexity.

For array of size N,

$= O(N\log N)$.

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

Soln:- **Step 1:-** Find P.

$a_1 = 2$ $\quad b_1 = \frac{1}{2}$ $\quad g(x) = N^2$

$a_2 = \frac{8}{9}$ $\quad b_2 = \frac{3}{4}$

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

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

Assume $P = 2$.

$$2 \times \frac{1}{4_2} + \frac{8}{9_1} \times \frac{9^1}{16_2} = 1$$

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

$1 = 1$ True, So $P = 2$

Put P in formula,

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

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

$$= \Theta\left(x^2 + x^2 \int_1^x \log u\right)$$

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

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

Ignore x^2,

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

Time Complexity.

SUPPOSE, CANNOT FIND P?

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

Lets try $p = 1$,

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

$$1 + 1 \neq 1$$

$$2 > 1$$

We observe p definitely greater than 1

Lets try $p = 2$,

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

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

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

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

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

We observe p definitely less than 2

$\therefore p < 2$ and $p > 1$

NOTE :- When $p <$ Power of $g(x)$

then answer,
i.e Time complexity $= O(g(x))$

Here,

$g(x) = x^2$, cannot find p.

$p < 2$., So

Time complexity $= O(x^2)$

END OF DIVIDE AND CONQUER RECURRENCES

SOLVING LINEAR RECURRENCES

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

Form:-

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

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

SOLUTION FOR FIBONOCCI NO:-

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

① Put $f(n) = \alpha^n$

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

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

Divide it by α^{n-2}

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

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

$\Rightarrow$ roots of this eqn.

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

$$= \frac{1 \pm \sqrt{1+4}}{2}$$

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

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

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

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

↳ (2)

(3). Fact:- No of roots = no. of answers

Here we have two roots, α_1 & α_2.

Hence we should have 2 answers.

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

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

$\Rightarrow c_1 = -c_2$

↳ (3).

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

from (3).

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

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

Put 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.

As $n \to \infty$, $\left(\frac{1-\sqrt{5}}{2}\right)^n$

will tend to 0.

Hence ignore $\left(\frac{1-\sqrt{5}}{2}\right)^n$

∴ Time Complexity for n^{th} Fibonacci number $= O\left(\frac{1+\sqrt{5}}{2}\right)^n$

$\Rightarrow T(N) = O(1.6180)^n$

So what the use of the formula :-

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

over recursion.

Here is the answer :-

in recursion to find nth fibonacci number.

for larger value of 'n', the time taken is too large. $O(1.6180)^n$ exponential.

So, instead just use.

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

formula.

---

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

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 by α^{n-2}

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

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

$$\Rightarrow \alpha = \frac{2 \pm \sqrt{4-4}}{2}$$

$$\boxed{\alpha = 1}$$

$$\therefore \alpha_1 = 1 \qquad \alpha_2 = 1.$$

★ General Case :- If α is repeated r times then $\alpha^n, n\alpha^n, n^2\alpha^n, \ldots\ldots, n^{r-1}\alpha^n$.

are all the solution to the recurrence.

THE ROOT ABOVE IS 1,
SO TWO SOLUTIONS ARE :-

$1, n\alpha^n$

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

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

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

(c_1)

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

$0 + c_2 = 1$

$c_2 = 1$

Putting $c_1 = 0$ and $c_2 = 1$

$$f(n) = 0 + n\alpha^n$$

$$= n1^n \quad (\alpha = 1)$$

$F(n) = n$

$\therefore$ Time complexity $= O(n)$.

THESE ARE HOMOGENOUS EQNS.

AS THEY DO NOT CONTAIN $G(x)$.

NON-HOMOGENEOUS LINEAR RECURRENCES !-

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

↓

when this extra function is there it is non-homogenous.

HOW TO SOLVE :-

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

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

↳ put 0

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

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

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

Divide by α^{n-1}

$$\alpha - 4 = 0$$

$$\alpha = 4,$$

Homogeous Solution =>

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

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

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

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

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

If $g(n) = 3^n$,

Guess: $f(n) = c3^n$

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

Divide by 3^{n-1},

$$c3 - 4c = 3$$

$$-c = 3$$

$$c = -3$$

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

③. Now we have general solution $c_1 4^n$ and particular solution -3^{n+1}

Add them,

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

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

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

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

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

HOW DO WE FIND PARTICULAR SOLUTION :-

$c3^n$ → guess we made above.

~~4~~ 3. If $g(x)$ is exponential, guess of same type.

Eg:- $g(x) = 2^x + 3^x$

Guess:- $f(x) = a2^x + b3^x$.

*]. If it is polynomial, guess of same degree.

$g(x) = x^2 - 1$

Guess:- $f(x) = ax^2 + bx + c$

or

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

Guess:- $f(x) = a2^x + (bx + c)$

*]. Lets say we guessed, $f(x) = a2^n$ and if it fails, then try $(an+b)2^n$, if that also fails keep increasing degree

$(an^2+bn+c)2^n$, $(an^3+bn^2+cn+d)2^n$

- - - - - - - - So on.

Q]. $f(n) = 2f(n-1) + 2^n$, $f(0) = 1$

Soln:- Homogenous solution.

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

$f(n) = 2\alpha^{n-1} + \underbrace{2^n}_{\hookrightarrow 0}$

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

$\alpha - 2 = 0$

$\boxed{\alpha = 2}$. General soln $= c_1 2^n$

★. Given particular solution,

$g(n) = 2^n$

So, guess:- $f(n) = a2^n$.

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

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

Divide by 2^{n-1}

$$a2 - 2a = 2$$

$$\boxed{a = a+1}$$

↓

wrong, so guess another one.

<u>New Guess:-</u> $f(n) = (an+b)2^n$

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

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

$a = 1$.

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

Ignore b,

$f(n) = n2^n$

(3) General answer

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

Given, $f(0) = 1$

So,

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

$c_1 = 1$.

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

Ignore 2^n,

$$f(n) = n2^n$$

Time complexity.

$$= O(n2^n)$$