✱ Time Complexity ✱

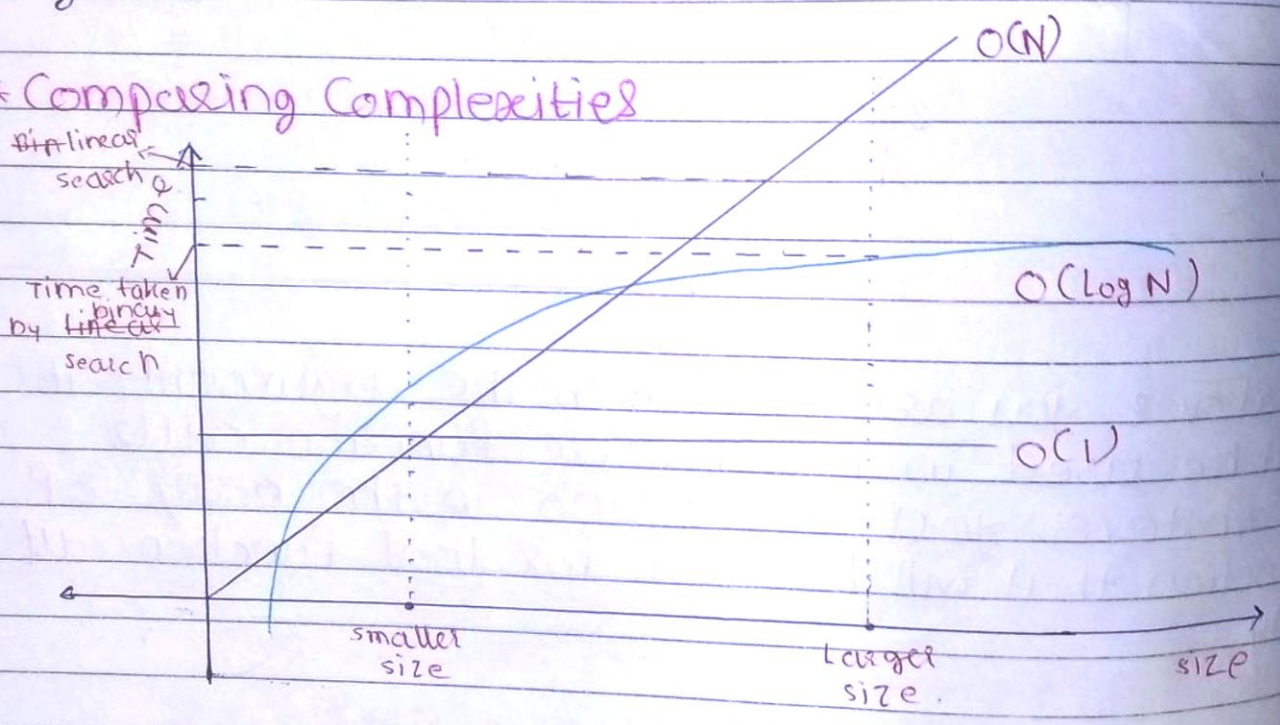
Page No.
Date :

Time Complexity ! = Time Taken.

* What is Time Complexity ?

→ It is a function that gives us the relationship about how the time will grow as the input grows.

* Comparing Complexities



$O(1) < O(\log N) < O(N)$ ← when the input size is larger

In above example, we are comparing linear search time complexity with the time complexity of binary search.

Here are the some key observations from above graph :-

1) For smaller input sizes, the time taken by linear search is less compared to the time taken by binary search.

2) For larger input sizes, the time taken by linear search is more as compared to the time taken by binary search. And it keeps on increasing as compared to the time taken by binary search.

3) We only consider the ~~worst~~ case having large no of input sizes i.e. the worst case because it allows us to know whether the algorithm is efficient in the long term or not. Worst case analysis helps us to determine real world scenario.

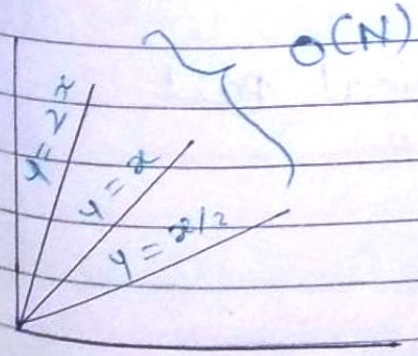
4) Thus we can say that binary search is more efficient as compared to linear search.

Key points to consider when dealing with time complexity :-

1) Always look for worst case scenario

2) Always take large input sizes / infinity into consideration

3)



Here, the actual time is different but in all cases the time is growing linearly as the input grows!

* Don't care about the actual time
* Ignore constants.

Page No.
Date :

4) Ignore less dominating terms

Example :- $O(N^3 + \log(N))$

For $N = 1 \times 10^6 = 1$ Million

$O((10^6)^3 + \log(10^6))$

$= O((10^6)^3 + 6)$ → Here 6 is very small as compared to 1million cube! That's why we ignored it.

$\sim O((10^6)^3)$

$\sim O(N^3)$

* Big O Notation

A mathematical notation which describes the upper limit of the fuction depicting the relationship betⁿ the time & input size i.e. it tells about the maximum complexity any algorithm can have.

Example, $O(N^3)$

→ The time complexity can not exceed $O(N^3)$. It can be $O(N^2)$, $O(N)$, $O(\log N)$, but it cannot be greater than $O(N^3)$

Let's divide into mathematical part

If

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

then,

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

$f(n)$ → $6(N^3) + 3N + 5 = O(N^3)$ ← $O(g(n))$

Page No.
Date :

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

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

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

$$= 6 + 0 + 0$$

$$= 6$$

$< \infty$ And that's why we ignore constants as well as less dominating terms!

*Big Omega - opposite of big O
This is the lower bound, i.e. it will take atleast a certain amount of time.

Ex. $\Omega(N^3)$

→ The complexity can be more than $\Omega(N^3)$.
It will take atleast $\Omega(N^3)$

Mathematical Def^n —

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

* Theta (Θ) Notation :-
It represent both upper & lower bound.

Mathematical Def^n —

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

Page No.
Date :

* Little- Oh Notation - This is loose upper bound

Big O	Little O
$\Rightarrow f = O(g)$	$\Rightarrow f = O(g)$
$\Rightarrow f \leq g$	$\Rightarrow f < g$ (strictly slower)

Mathematical Def^n -

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

* Little - Omega Notation - This is a loose lower bound.

Big Ω	Little Omega (ω)
$\Rightarrow f = \Omega(g)$	$\Rightarrow f = \omega(g)$
$\Rightarrow f \geq g$	$\Rightarrow f > g$

Mathematical Def^n -

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

* Space Complexity :-

- Total space taken by the algorithm with respect to the input size.
- Space complexity include both Auxiliary space (extra space used by an Algo) and space used by input.

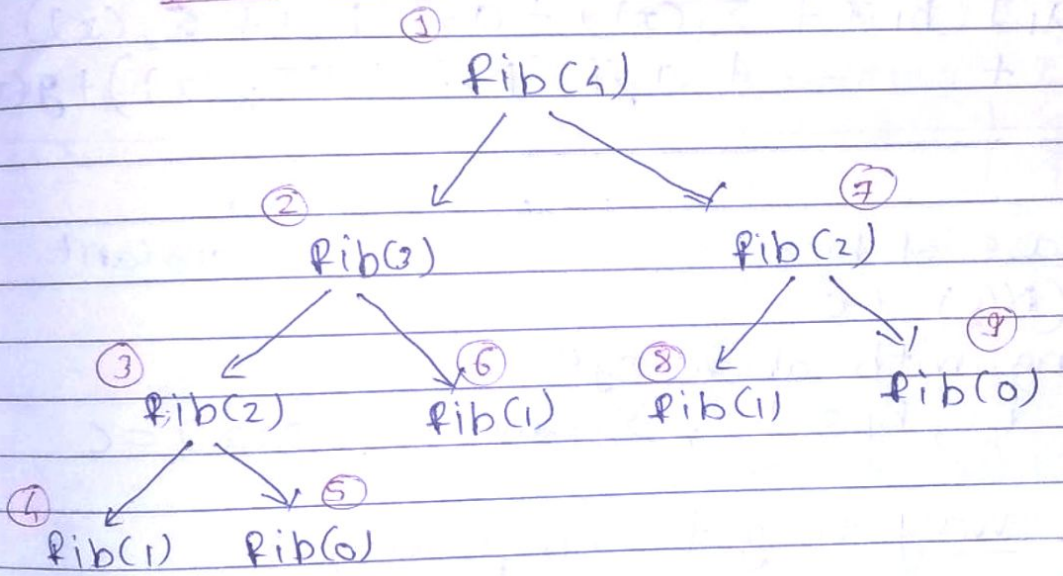
Page No.
Date :

Recursive Algorithms.

In Recursion, we know, ~~that~~ function calls are stored in stack. Hence recursive programs don't have constant space complexity.

* Space complexity of Fibonacci No —

Let's take fib(4)



Note : • At any particular point of time, no two fuction calls at the same level of recursion will be in the stack at the same time.
• only calls that are interlinked with each other will be in the stack at the same time.

⇒ At one particular level of tree, there will be only one call that are in stack at a time. It is not possible that all the function calls (Here, 9) will be in the stack at the same time. maximum space taken = Height of the tree.

Space Complexity = O(N)