Space & Time Complexity.

1) What is time complexity?

Ans:

Functions that gives us the relationship about how the time will grow as the input grows

'OR'

As the input grows, times grows is known as time complexity

Now as an ~~Expamp~~ Example :-

We have two computer

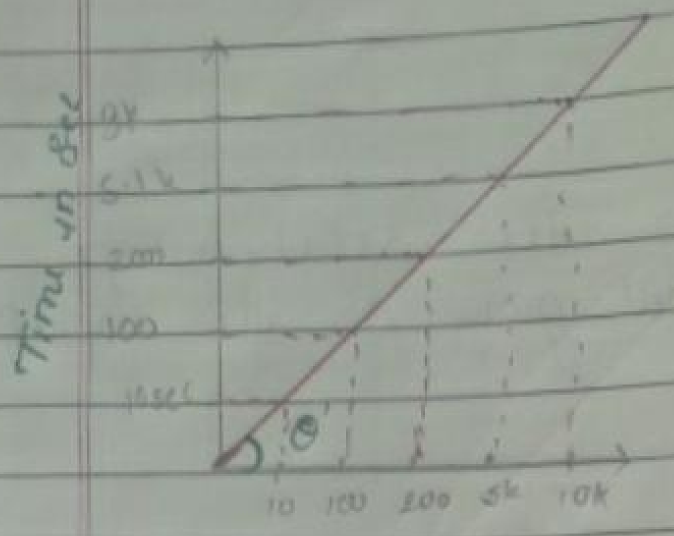
* We run an algo we have :-

	Old Computer	M1 macbook (very fast)
data :-	1,000,000 elements in arr.	10,00,000 elements in arr.
algo :-	linear search for target that doesn't exist in the array	Linear search -11-
Time taken	10 sec	1 sec.

Q. Which machine has a better time complexity in between these?

Ans:- Both of the machine have the same time complexity.

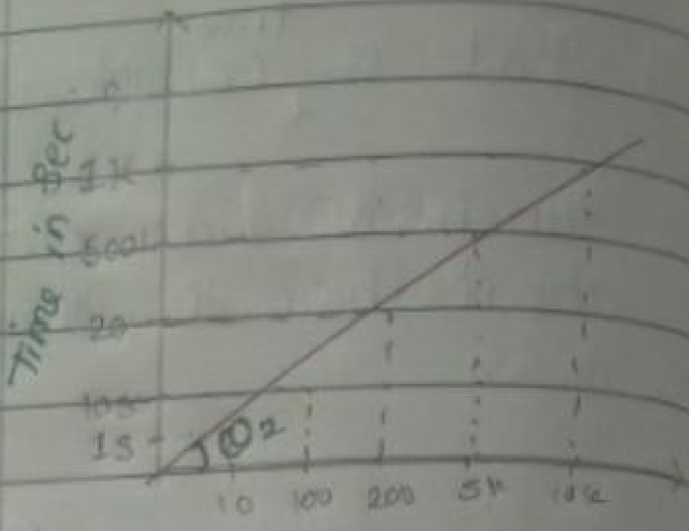
Time Complexity != time taken.



Size of an array.

Eg:- old machine

1) Straight line.



Size of an array.

M1 Mack book.

1) Steeper line with less sleep.

Even though the time taken is different but the relationship between the size and the time is same "Linear".

Over here, time is growing linearly as the size is growing. In both the case though values are different.

Q Why? { Why this relationship is important }.

Ans: Linear search grows linearly. (N)
Binary search grows with logN

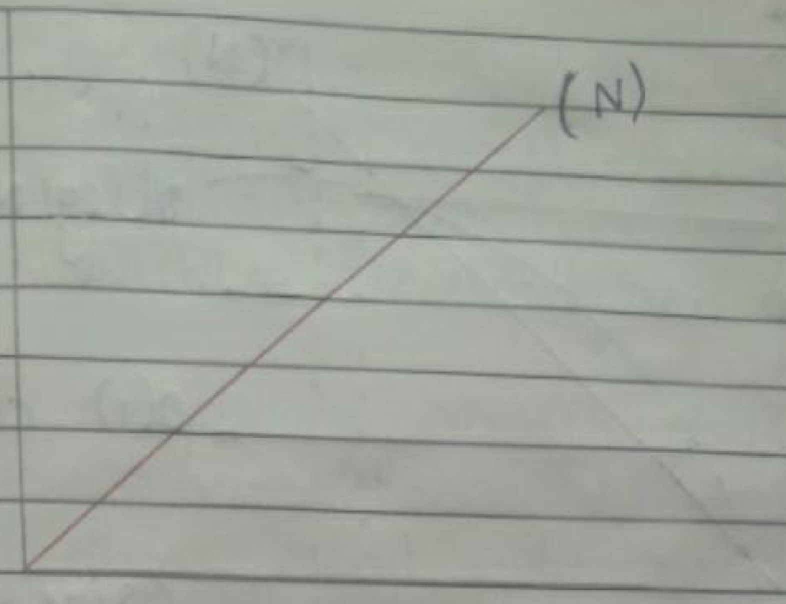


fig: Graph for linear search

Because the time complexity is linear

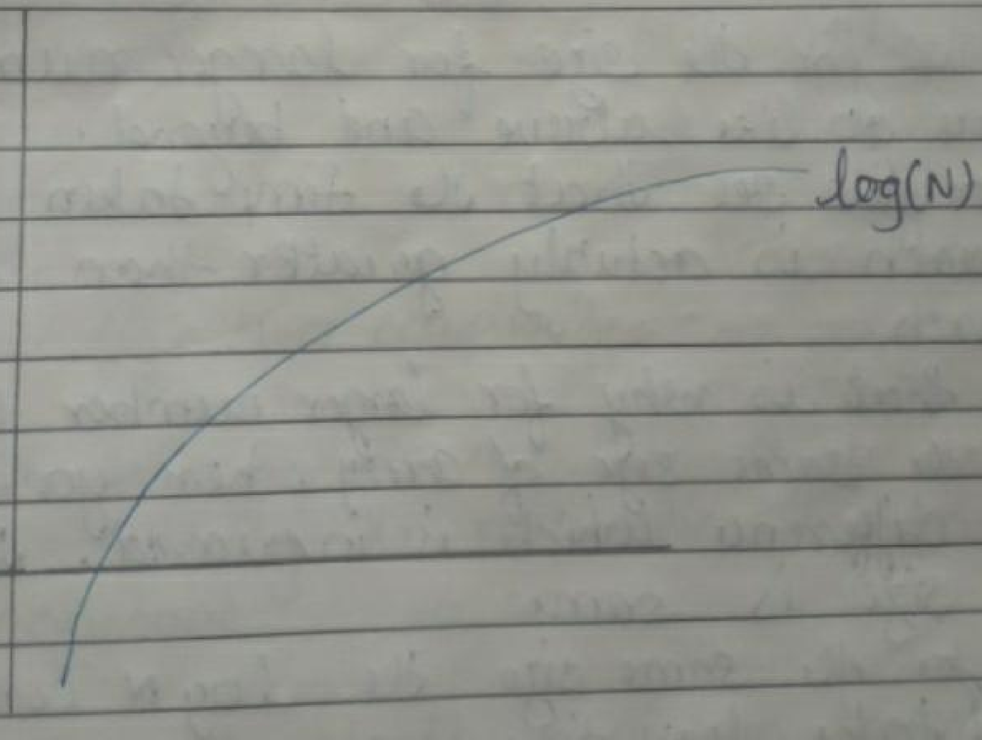
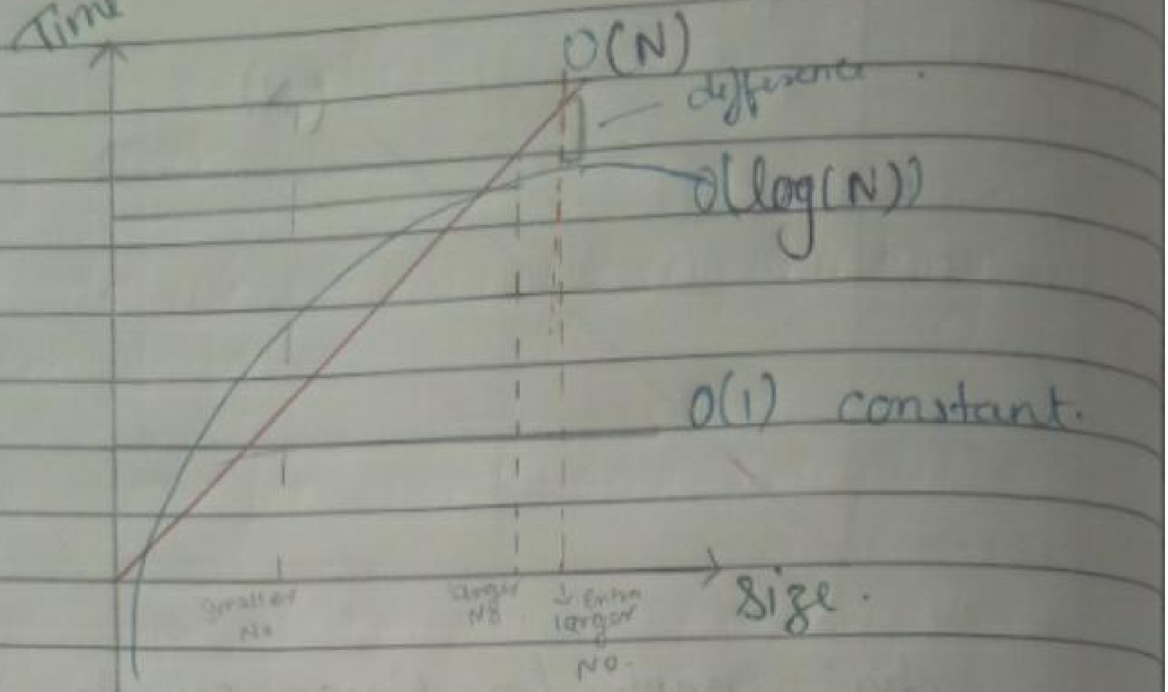


fig: Graph for Binary Search.

This is growing log N times.



Q Now let's Notice. Why does it matter?

Ans:

1) If we fix the size for larger numbers it may go like above and beyond.
2) Can you see that the time taken by linear search is actually greater than the binary search.
3) So, that is why for larger number. (fig: ↑ graph) for the same size of array. here you can see the difference which is increased. though the size is same
4) So, for the same size the log N complexity will take less time
5) And, the linear complexity will take more time.
c) for smaller no, logn will take more time, linear less, O(1) will

Q So, which one is better?

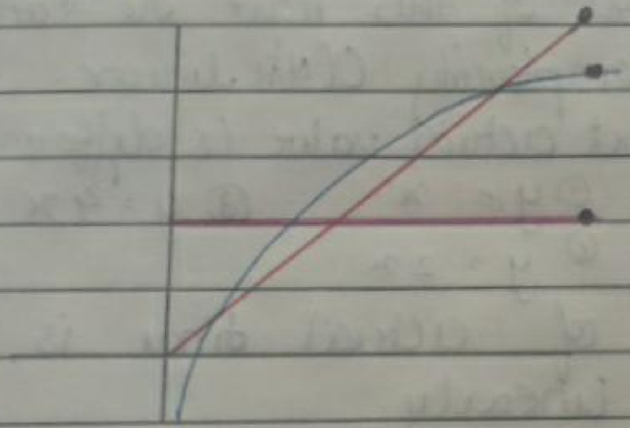
Ans O(logn) is better because it is more efficient So, that's why it matters.

Now let we tak a constant time complexity. so here does not matter what the size is time will always remain constant and for smaller number "we don't care about smaller no."

Note:-

1) In time complexity always look at bigger numbers.
2) Always think about when your data will grow large in size in that case what will happen?

Ans:-



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

~~better~~

fig:- Time complexity.

1) Now in the fig we can see that the (Red) linear is taking the most time, than log(n) then constant.

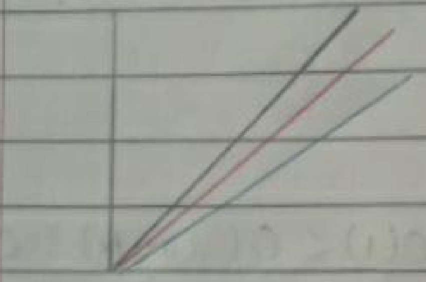
2) ~~Thef~~ Therefore, constantt is always better than O(log(n)) & O(N)

* As you can see when the size was fixed for the same amount of data O(N) was taking the most time. then log(N) & last O(1).

Q Which one is better?
Ans Binary search O(log n).

* Q. What do we consider when thinking about the complexity?

Ans:
1) Always look for worst case complexity
2) Always look at complexity for large/∞ data

3) 

All of this have the same complexity O(N) ie linear.
But actual value is different

Eg:- ① $y = x$ ③ $y = 4x$.
② $y = 2x$

a) Even though value of actual time is differ they're all growing linearly.

b) "We don't actually care about what the time taken is". because that will vary from machine to machine".
We only care about the relationship of how the time will grow, when the input grows.

Q. Do we really need to worry about these consta then?
Ans:- No, we only care about how its growing.

c) This is why, we ignore all constants.

4) Always ignore less dominating terms.

Okay :-

let's say you've complexity of

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

So, from point 2. {Always look at complexity for large / ∞ data}.

$\therefore$ if we take 1million ~~times~~ amt of data.

$\therefore N = 1$ mil.

$\therefore = ((1\ mil)^3 + \log(1\ mil))$

$= (1\ mil)^3$ sec $+ \underbrace{6\ sec}$

It is very small

So does this 6 sec as compared to $(1 mil)^3$ sec has any significant.

Hence, ignore it.

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

- (Ignore the constants)

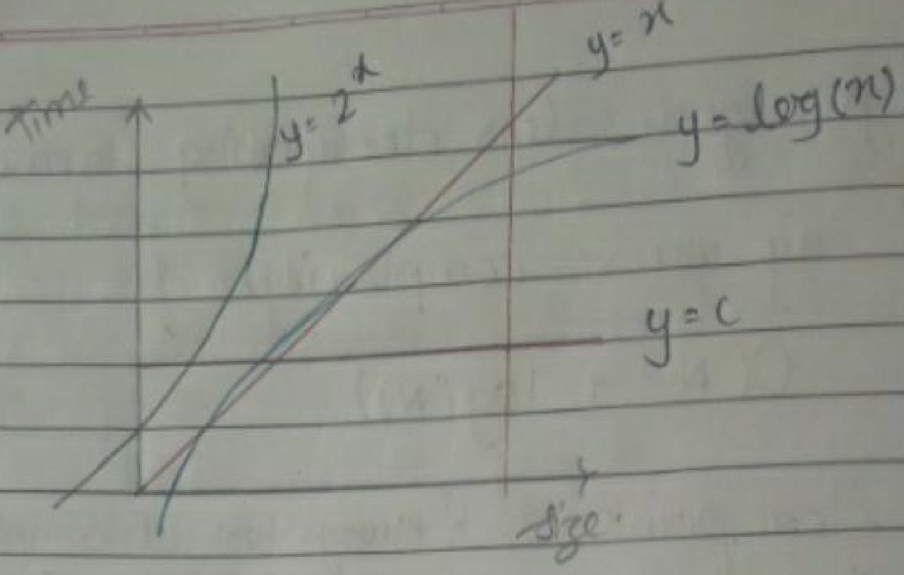
$= N^3 + N^2 + N$

- (Ignore the less dominating term)

$= N^3$

$\therefore O(N^3)$.

So, $O(3N^3 + 4N^2 + 5N + 6) = O(N^3)$.



1) y = c is always be constant so, it will always be less than whatever value I you provide. So, y = c will always be best optimized.

2) log(x) so, if we take some large amount of data, then
for same amount of day y = c will take less amount of time.
After that log(x).
then the 'x' which will be little bit more
& then as you can see 2^x which is very very poor complexity. (which is (exponential complexity).

3) For such a small amount of data, time limit has exceeded alot: (2^x) which is not even visible on the graph.
Eg:- fibonacci like for such a small amount of data time has exceeded alot, that is already not visible on the graph ∴ this is bad.

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

There other complexity also like $O(n \log n)$, $O(N^2 \log^n)$ etc