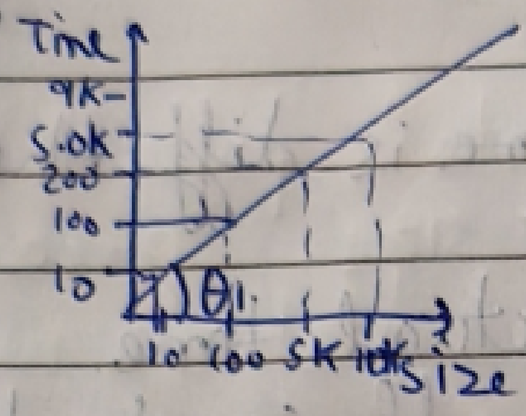
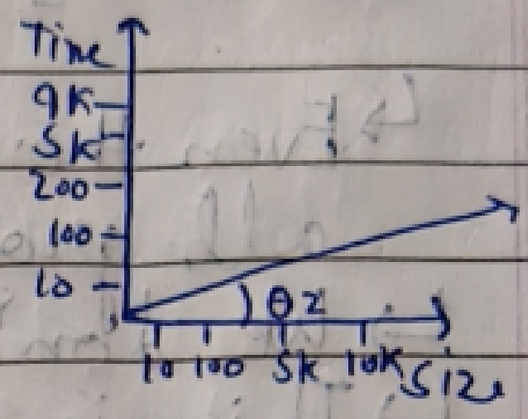
Time and Space Complexity

⇒ Time Complexity ?.

old machine



New machine



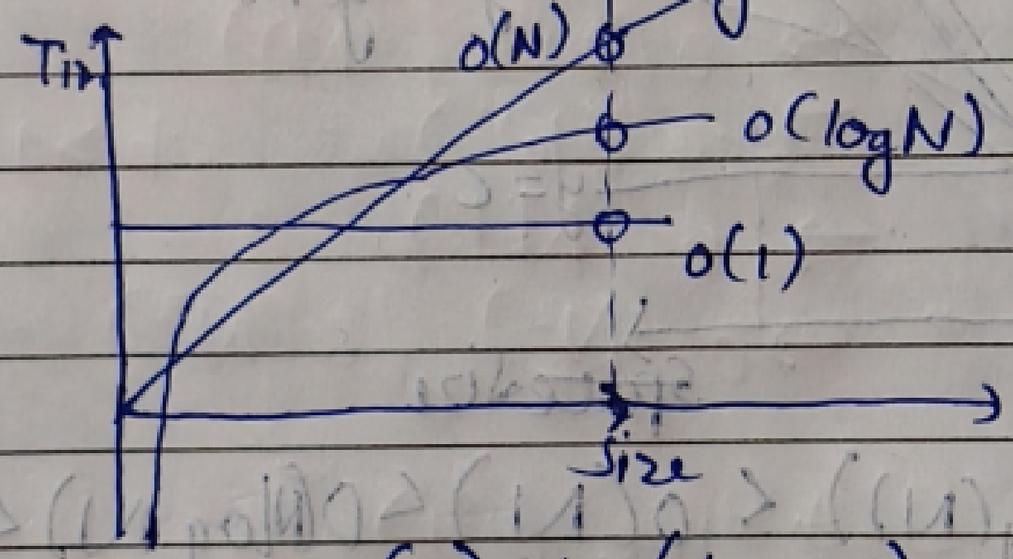
To perform something
It take 10 sec

And this take 1sec

✱ Both machine have same time Complexity

Time Complexity ! = Time Taken

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

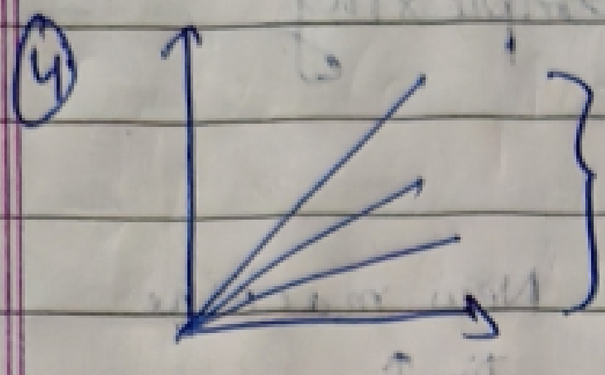


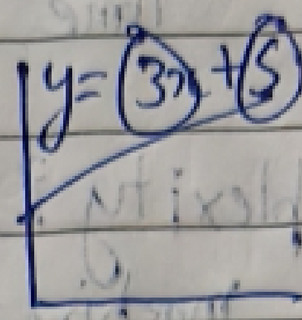
$$O(1) < O(\log N) < O(N)$$

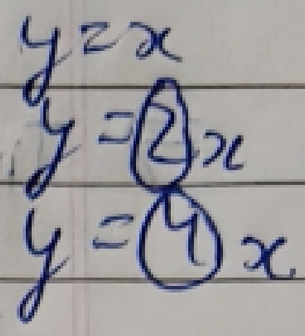
- Things to Consider when thinking about Complexity :
  1. Always look for worst case Complexity
  2. Always look at complexity for large / ∞ data
  3. $O(N^3 + \log N)$

  From point 2, 1 million $\Rightarrow ((1mil)^3 + \log(1mil))$
  $= (1mil^3 + \underbrace{6sec}_{\text{very small}})$
  ∴ Hence Ignore

  Always ignore less Dominating time







↳ Even tho value of actual time is diff they are all growing linearly

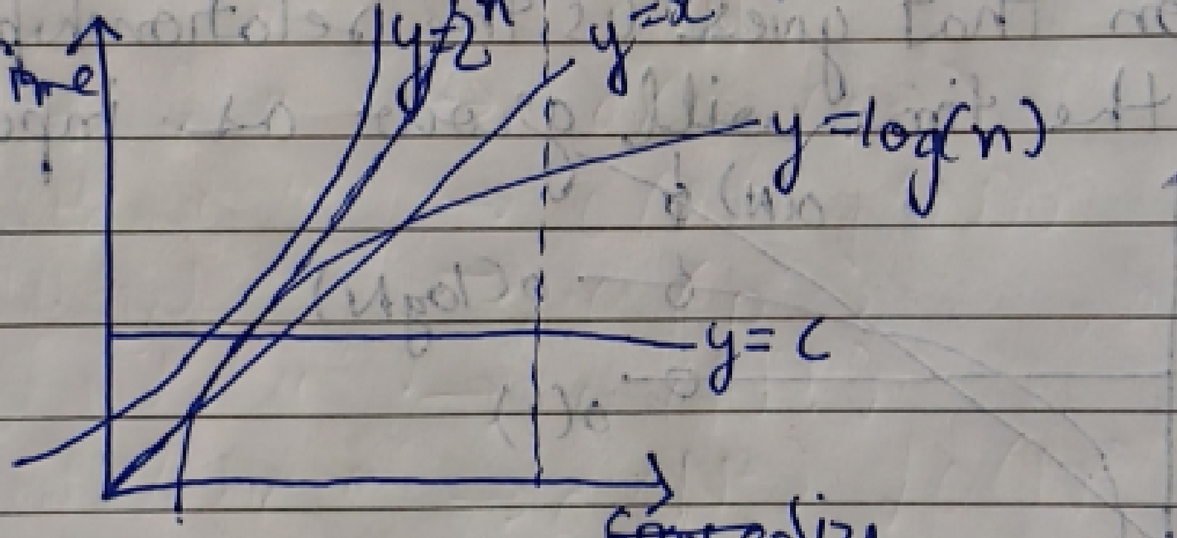
↳ We don't care about actual time

↳ This is why, we ignore all constants.

Ex:- $\Rightarrow O(3N^3 + 4N^2 + 5N + 6)$

$\Rightarrow O(N^3 + N^2 + N)$

$\Rightarrow O(N^3)$



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

(i) Big-Oh Notation

• Word define ⇒ upper bound

• Math ⇒ $\lim_{N \to \infty} \frac{f(N)}{g(N)} \leq \infty$ $\left(f(N) = O(gN)\right)$

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

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

$\Rightarrow 6 + 0 + 0$

Finite value $\Rightarrow (6) \leq \infty$

(ii) Big Omega (opposite of Big-oh)

- word define → $\Omega(N^2)$ (lower bound)
- Math → $\lim_{N\to\infty} \frac{f(N)}{g(N)} \geq 0$

(iii) What if Algo has UB & LB as (N^2)

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

Theta Notation (Combining both)

$\theta(N^2)$:

- words → Both UB & LB
- Maths → $0 \leq \lim_{N\to\infty} \frac{f(N)}{g(N)} \leq \infty$

(IV) little Oh notation

- This is also giving Upper Bound
- words → Lower, upper Bound

Big oh	little o (Stronger Statement)
$f = o(g)$	$f = o(g)$
$f \textcircled{\leq} g$	$f \textcircled{<} g$ (strictly lower)

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

Ex:- $f = N^2$, $g = N^3$

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

(II) little Omega

Big oh	little Ω
$f = \Omega(g)$	$f = \omega(g)$
$f \textcircled{\geq} g$	$f \textcircled{>} g$

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

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

In order $O(1)$ ① $< O(\log n)$ ② $< O(\sqrt{n})$ ③ $< O(n)$ ④ $< O(n \log n)$ ⑤ $< O(n^2)$ ⑥ $< O(n^3)$ $< O(b^n)$ ⑥ $< O(n!)$ ⑦

PAGE NO.
DATE

⇒ Space Complexity / Auxiliary Space?
→ Auxiliary space is the extra space or Temporary space used by Algo.

→ Space complexity of algo is total space taken by algo with respect to input size. Space complexity includes both auxilary space and space used by input.
- Recursive space – Memory used by func^n calls

Big-O Notation — N = The size of input

Constant Time	→	$O(1)$	①
Linear "	→	$O(N)$	③
Logarithmic "	→	$O(\log(N))$	②
Linearithmic "	→	$O(N \log(N))$	④
Exponential "	→	$O(b^n)$, $b > 1$	⑦
Factorial "	→	$O(N!)$	⑧
Quadratic/cubic "	→	$O(N^2)$ / $O(N^3)$	⑤ ⑥

① → ⑧ Sort of small to large

Ex 1 → Constant time $O(1)$

```
a = 1
b = 2
c = a + 5 * b
```

(They don't depend on N)

```
i = 0
while i < 11
    i = i + 1
```

Ex 2 → Linear time $O(N)$

```
i = 0
while i < N
    i = i + 1
```

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

```
i = 0
while i < N
    i = i + 3
```

$f(N) = N/3$
$O(f(N)) = O(N)$

Three Time faster but

Ex3 → Quadratic Time

The first may be obvious since N work done N times is Not $N = O(N^2)$

- for (i=0 ; i<n ; i++) {
  for (j=0 ; j<n ; j++) {

$f(N) = N * N = N^2$, $O(f(N)) = O(N^2)$

- for (i=0 ; i<n ; i++)
  for (j=(i) ; j<n ; j++)

Focus on 2nd loop

∴ i goes from [0, n)

The amt of looping depend by what i is

if i=0, we do n work , if i=1, we do (n-1) wo

So, $\underbrace{(n) + (n-1) + (n-2) \cdots + 3 + 2 + 1}_{\left(\frac{n(n+1)}{2}\right)}$

So, $O\left(\frac{n(n+1)}{2}\right) = O\left(\frac{n^2+n}{2}\right) = O\left(\frac{n^2}{2} + \frac{n}{2}\right) = O(n^2)$

- i=0
  while i<(n)
  j=0
  while j<(3*n)
  j=j+1
  j=0
  while j<(2*n)
  j=j+1
  i=i+1

$f(n) = n * (3n + 2n) \Rightarrow 5n^2$

$\therefore f(n) = O(n^2)$

- i = 0
while i < 3*n.
    j = 10
    while j <= 50
        j = j + 1
    j = 0
    while j < n*n*n
        j = j + 2
    i++ = i + 1

$f(n) = 3n * (40 + n^3/2) \Rightarrow 3n/40 + 3n^4/2$

$\Rightarrow n + n^4$

$\Rightarrow n^4$

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

Ex:4 → Prime no.

```
for (i=2; i*i<=n; i++){
    if (n%i == 0){
        .
        .
    }
}
```

→ constant

$i = 2$ to $i^2 \le n$

$i = 0$ to $i \le \sqrt{n}$ (worst case)

0 to $\sqrt{n}$

$O(\sqrt{n})$

(i) $O(\sqrt{n}) > O(n)$

let $n = 100$

$O(n) = 100$

$O(\sqrt{n}) = 10$

(ii) $O(\sqrt{n}) < O(\log n)$

let $n = 1000000$

$O(\sqrt{n}) = 1000$

$O(\log n) = 20$

Take base 2

Ex → Imp

```
for (i=1 ; i ≤ N ) {
    for (j=1 ; j ≤ K ; j++) {
        // time t
    }
    i = i+K
}
```

→ Inner loop

Inner loop: O(kt) time (constnt

⇒ O(Kt * times outer loop running)

i = 1, 1+K, 1+2K, 1+3K ... x

1 + xK → i final value

1 + xK ≤ N

$\left(x = \frac{N-1}{K}\right)$

⇒ $O\left(Kt * \frac{(N-1)}{K}\right)$ ⇒ O(t * (N-1))

⇒ O(Nt)