→ Comparing 2 Algos

i) using execution time

ii) using iterations and graphs

→ Why Big O needed

i) Why lower order terms are neglected

ii) Why constant coefficients are neglected

iii) Issues with Big O

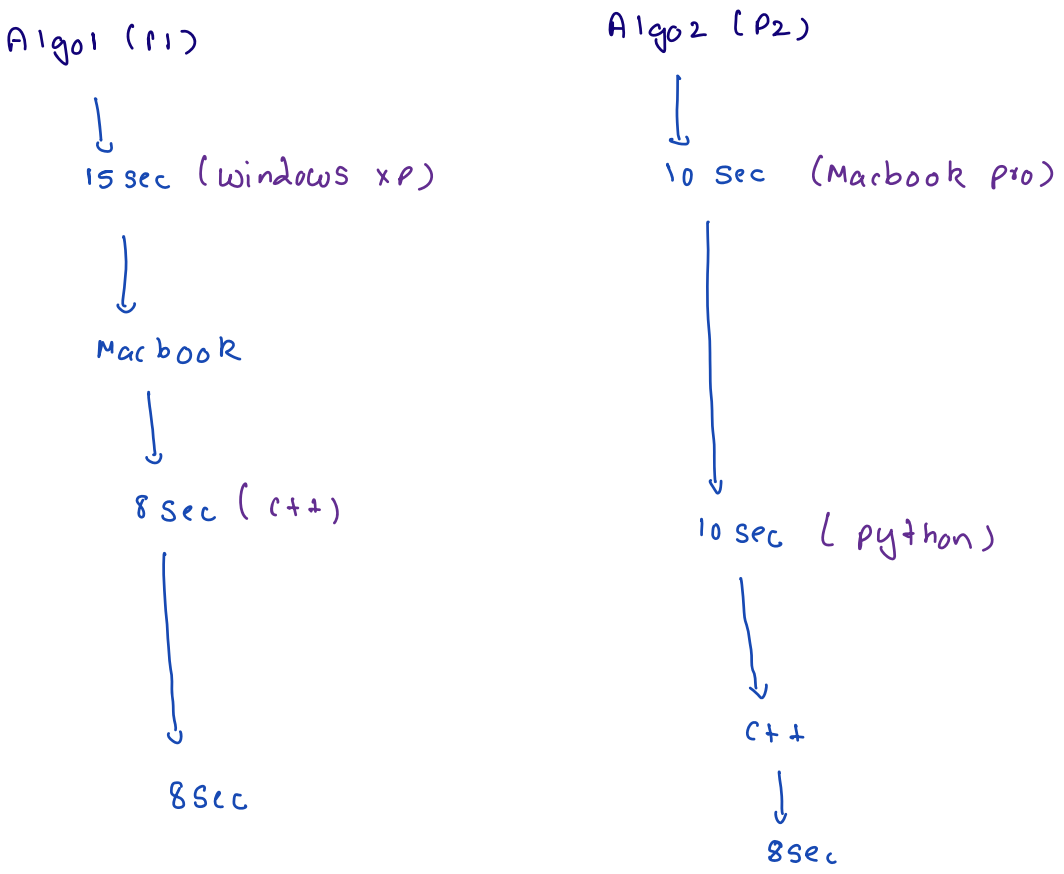
iv) Worst case scenario

→ Space complexity

→ TLE (Time Limit exceeded)

Comparison using execution time

→ both algo's are running on same input size



Conclusion : comparing two algo's on the basis of execution time is not correct, because execution time depends on a lot of external factors (processor, language, temp etc.)

Comparing using iterations

```
for (int i=1; i<=n; i++) {
    sop(i);
}
```

n itr

	Algo1 (P1)	Algo2 (P2)
itr	$100\log_2 N$	$\frac{N}{10}$

$N < 3550$ $\quad 100\log_2 N > \frac{N}{10}$ $\quad$ Algo 2 is better

$N >= 3550$ $\quad 100\log_2 N < \frac{N}{10}$ $\quad$ Algo 1 is better

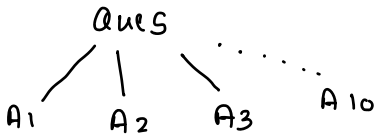
Google Search → Millions of results

Hotstar → Millions

} data is always inc.

→ Pick Algo that is better performer for larger value of N

⇒ Algo1



Assymptotic analysis of algo's

→ analyse performance of algorithms for larger input.

→ Big O.

How to find Big O of a code ?

a) calculate total no. of iterations.

b) neglect all lower order terms (keep highest order term)

c) remove constant coefficient.

Why neglect lower order terms :

itr : $N^2 + 10N$

N	total itr	% of lower order term contribution in total itr
10	$100 + 100$	$\frac{100}{200} \times 100 = 50\%$
100	$10^4 + 10^3$	$\frac{10^3}{10^4+10^3} \times 100 \approx 10\%$
1000	$10^6 + 10^4$	$\frac{10^4}{10^6+10^4} \times 100 \approx 1\%$

Conclusion: for larger value of N, contribution of lower order terms in total itr is very less.
That's why we can ignore lower order terms while calculating Big O.

Why to remove constant coefficient:

		Algo 1	Algo 2	(better)
i)	itr →	$10 \log_2 N$	$\frac{N}{100}$	Algo 1
ii)	itr →	$3N^2$	$15N$	Algo 2

for larger values of N, const. coeff. don't play a very big role in itr.

Issues with Big O

1)

	Algo1	Algo 2
itr :	$10N$	N^2
	$O(N)$	$O(N^2)$

Claim: Algo1 is always better than Algo2 ✗

N	itr in Algo1	itr in Algo2	better
5	50	25	Algo2
8	80	64	Algo2
10	100	100	same

$N > 10$ Algo1 is always better

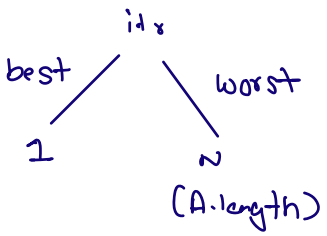
Algo1 is not always better, it is better than Algo2 after a specific value (threshold value).

2)

	Algo 1	Algo 2
itr:	$2N^2 + 7N$	$5N^2$
Bigo:	$O(N^2)$	$O(N^2)$

When Algo's have same Big O, then we should do the comparison based on itr.

```
boolean search (int [] A, int K) {
    for (int i=0; i < A.length; i++) {
        if (A[i] == K) {
            return true;
        }
    }
    return false;
}
```



always consider itr. in worst case scenario

itr: N

TC : O(N)

Space complexity

int → 4 bytes

long → 8 bytes

```
void fun (int N) {
    int x = 10;
    long y = 30L;
    int z = 94;
}
```

total space = 3*4 + 8

= 20 bytes

```
void fun (int N) {
    int x = 10;
    int [] A = new int [N];
    int [][] B = new int [5] [N];
}
```

total space

= 8 + 4N + 4*5N

= 8 + 24N

How to calculate space complexity

input ⟶ { Algo } ⟶ return ans

Note: When we calculate space complexity of Algo we don't consider input space, consider space taken by Algo.

```
int max(int[] A) {
    int max = A[0];
    for (int i=0; i< A.length; i++) {
        if (A[i] > max) {
            max = A[i];
        }
    }
    return max;
}
```

Time	Space
itr: N	Space: 8 Bytes (max, i)
TC → O(N)	SC → O(1)

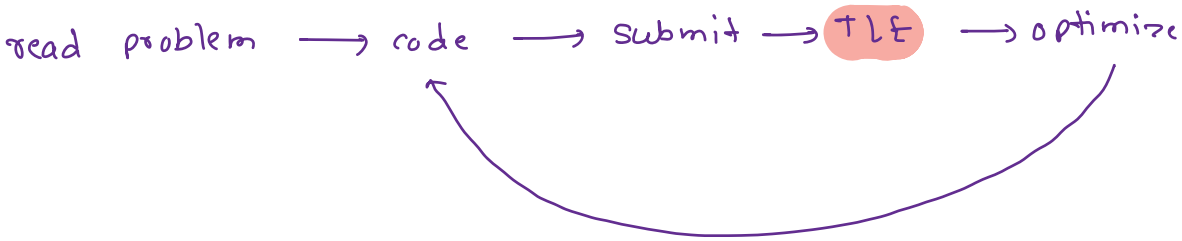
```
int solve(int[] A, int K) {
    int n = A.length;
    int[] B = new int[n];
    for(....) {
        ==
        ==
        ==
    }
    return B[K];
}
```

TC: O(N)

SC: O(N)

Time limit exceeded

Amazon contest → 2Q, 1 hour



how to check that approach will give TLE without writing code.

online contest / online IDE → Server → processing power ≈ 10^8 itr / sec

↳ to avoid TLE our code should within a second.

Your code should have at max :

** Safe side

10^7 to 10^8 itr

Ques
=

constraints $1 \leq N \leq 10^5$

rough code
(nested loop) $\longrightarrow$ itr $= N^2$, itr $= 10^{10}$
if $(N = 10^5)$

↓

don't write code

|

Improvised idea

Count factors

Beginner

$1 \leq N \leq 10^6$

N itr

if $(N == 10^6)$
$\rightarrow$ itr : 10^6

DSA module

$1 \leq N \leq 10^{12}$

N itr

if $(N == 10^{12})$
$\rightarrow$ itr : 10^{12}

TLE

```
int a = 0, i = N;
while (i > 0) {
    a += i;
    i /= 2;
}
```

itr	i value after
1	$\frac{N}{2}$
2	$\frac{N}{4}$
3	$\frac{N}{8}$

loop breaks at $i = 0$

// assume the loop k times

$$i = \frac{N}{2^k}$$

no. of itr till $i = 1$

$$\frac{N}{2^k} = 1$$

$$k = \log_2 N$$

total itr $= \log_2 N + 1$

Big O $\rightarrow O(\log_2 N)$

```
for(int i=1; i<=n; i=i*2) {
    for(int j=1; j<=n; j++) {
        Sop();
    }
}
```

i	j	itr
1	[1 n]	n
2	[1 n]	+ n
4	[1 n]	+ n
8	[1 n]	+ n
⋮		+ ⋮
n	[1 n]	+ n

$O \rightarrow O(n * \log_2 n)$

$1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow \dots \rightarrow n$

approx. $\log_2 n$ times

```
for(int i=1; i<=100; i=i*2) {
    for(int j=1; j<=n; j++) {
        Sop();
    }
}
```

i	j	itr
1	[1 n]	n
2	[1 n]	n
4	[1 n]	n
8	[1 n]	n
16	[1 n]	n
32	[1 n]	n
64	[1 n]	n
128		7n

Big O → O(n)