→ 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

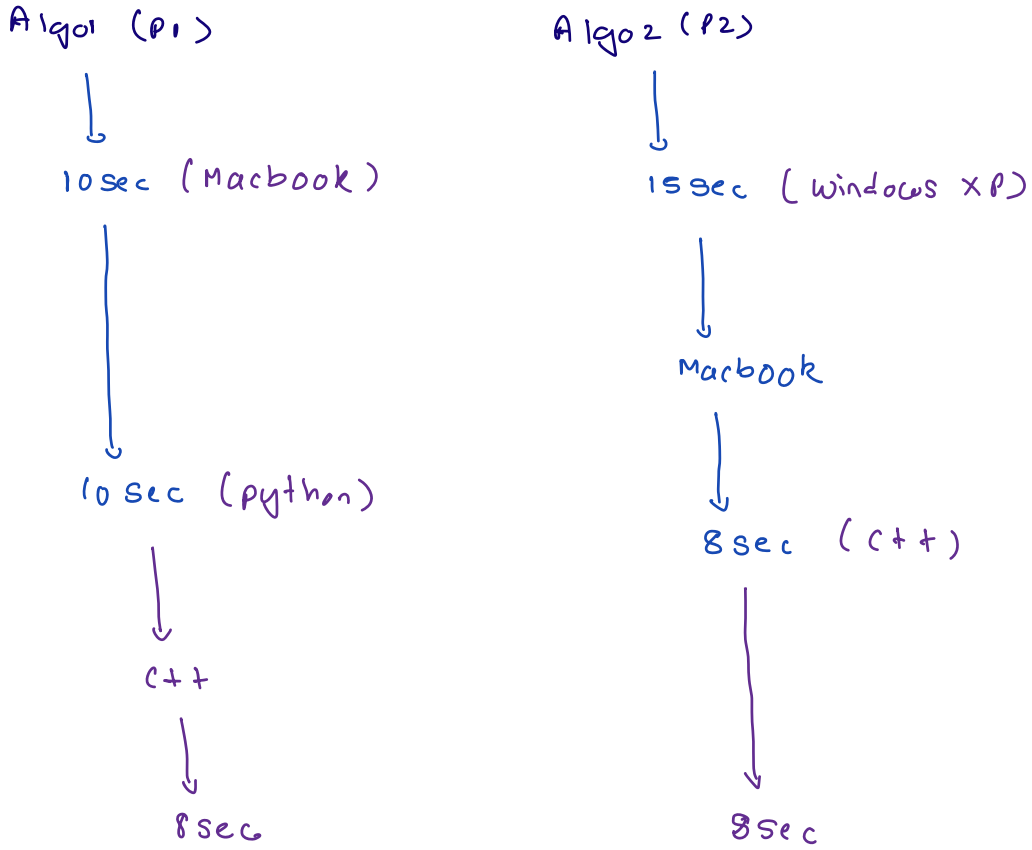
iv) Worst case scenario

→ Space complexity

→ TLE (Time Limit exceeded)

Comparing Algo's using execution time

→ both algo's are running on same input size



conclusion : comparing 2 Algo's based on execution time is not correct because execution time depends on external factors (machine, processing power, language, temp) etc.

```
for (int i=1; i<=N; i++) {
    SOP(i);
}
```

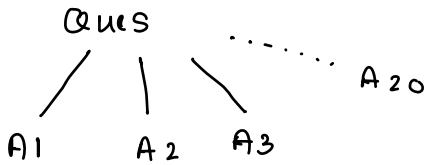
N itr

Comparison based on no. of iterations

	Algo1 (P1)	Algo2 (P2)
itr:	$100\log_2 N$	$\frac{N}{10}$
$N < 3550$	$100\log_2 N > \frac{N}{10}$	Algo2 is better
$N \geq 3550$	$100\log_2 N < \frac{N}{10}$	Algo1 is better

Google search → Millions of results
Hotstar → Millions
} data is huge in real world

Choose the Algo that works better for large input → Algo1



Assymptotic analysis of Algorithms

↳ analyse performance of algo for larger input

↳ Big O

How to find Big O ?

i) calculate total no. of iterations.

ii) neglect all lower order terms (keep the highest term)

iii) neglect constant coefficient.

why to 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 very high value of N, contribution of lower term in total itr is very less and that's why we can neglect lower terms.

why to neglect constant coefficient ?

	Algo 1	Algo 2	Better
itr :	$3N^2$	$5N$	Algo 2
Big O :	$O(N^2)$	$O(N)$	

for large input size, const. coeff. won't play a significant role.

Issues in Big O ?

i)

	Algo I	Algo II
itr →	$10N$	N^2
Big (O) →	$O(N)$	$O(N^2)$

Claim → Algo I is always better ✗

N	itr in Algo I	itr in Algo 2	better
5	50	25	Algo2
8	80	64	Algo2
10	100	100	Same
11	110	121	Algo1

$N \geq 11$ Algo I is always better.

Algo I is better than Algo2 after a specific value (threshold value).

(ii)

	Algo I	Algo II
itr →	$2N^2+7N$	$5N^2$
Big O →	$O(N^2)$	$O(N^2)$

If Big O for two Algo's is coming same then do the comparison based on no. of itr.

$2N^2 + 7N$

$N(2N+7)$

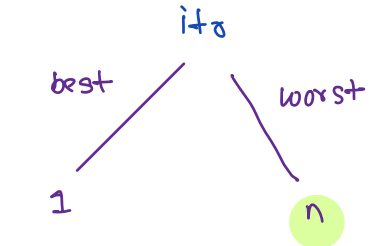
$5N^2$

$N(5N)$

$2N+7 < 5N$

that's why Algo I is better

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



always select the 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 = 25;
    int z = 30;
}
```

Total spaces = 3*4 + 8

= 20 Bytes

```
void fun (int N) {
    int x = 10;
    int [] A = new int [N];
    int [][] B = new int [7] [N];
}
```

Total space

= 8 + 4N + 4*7N

= 8 + 4N + 28N

= 8 + 32N.

How to calculate space complexity of Algo

input → {Algo} → return ans.

Note: In order to calculate space complexity of Algo, please consider the space taken by your code. (Don't consider the input space)

```
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;
}
```

TC	SC
itr: N	space: 8 bytes
TC: O(N)	SC: O(1)
	space of independent N (constant)

```
int freq (int[] A , int K ) {
    int count = 0;
    for(int i=0; i< A.length; i++) {
        if( A[i] == K )
            count++;
    }
    return count;
}
```

TC

itr: N

TC => O(N)

SC

space: 8 bytes

SC => O(1)

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

2 int variables

itr → n

TC : O(n)

space → 4 + 4n + 8

= 12 + 4n

SC : O(n)

Time Limit Exceeded (TLE)

Amazon contest → 2Q, 1 hour

Read → rough code → code → submit → TLE → improve

Online platform / Online IDE → servers → processing speed

10^8 itr / sec

↳ In order to avoid TLE our code should run within 1 sec.

Our code should have at max :

xxx
safe side

10^7 to 10^8 itr

Ques
=

$1 \leq N \leq 10^5$

rough code ⟶ N^2 itr ⟶ if ($N == 10^5$)

(Nested loop)

itr : 10^{10}

TLE

↓

don't write code

↓

try to improvise logic

count factors

Beginner	DSA
$1 \leq N \leq 10^3$	$1 \leq N \leq 10^{12}$
approach : N itr	approach : N itr
if $N = 10^3$	if $N = 10^{12}$
at max itr $\Rightarrow 10^3$	at max itr $\Rightarrow 10^{12}$
	TLE

Doubts

```
for (int i=1, j=1; j<=n; i++) {
    print ( );
    if( i+1 == n-1 ) {
        j++;
    }
}
```

i	j
1	1
2	
3	
4	
⋮	
n	2
n+1	
n+2	
n+3	
⋮	
n+n	3

n

n

itr : n^2

TC : $O(n^2)$

```
int S=0;
for (int i=1; i<=100; i++) {
    S = S+i;
}
return S;
```

i → 1 to 100, i++

itr : 100 (constant)

TC : O(1)

```
for (int i=1; i<=n; i++) {
    for (int j=1; j<=3^i; j++) {
        sop();
    }
}
```

i	j [1 to 3^i]	itr
1	[1 to 3]	3
2	[1 to 9]	+ 9
3	[1 to 27]	+ 27
⋮		+ ⋮
n	[1 to 3^n]	+ 3^n

$3 + 9 + 27 + \cdots + 3^n$

$$S_n = \frac{a(r^n - 1)}{r - 1}$$

$a = 3$

$r = 3$

terms $= n$

$$= \frac{3(3^n - 1)}{2} = \frac{3}{2} \times (3^n - 1)$$

$$= \frac{3^{n+1}}{2} - \frac{3}{2}$$

TC: $O(3^n)$

```
for (int i = 1; i <= n; i += 2) {
    sop();
}
```

$1 \to 3 \to 5 \to 7 \to 9 \to 11 \cdots n$

i → all odd no. in [1 to n]

how many odd no. = $\frac{n}{2}$

itr : $\frac{n}{2}$

TC : O(n)

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

$n * \log_2 n$

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

$1 \to 2 \to 4 \to 8 \to 16 \to 32 \ldots\ldots N$ (approx $\log_2 n$ itr)

$N \to \frac{N}{2} \to \frac{N}{4} \to \frac{N}{8} \ldots\ldots 1$ (approx $\log_2 n$ itr)

```
for(int i=0; i<n; i++) {
    for(int j= i-1 ; j>=0; j++) {
        sop();
    }
}
```

j=0 to 2

i	j	itr
0	-1 to 0	0
1	0	infinite

```
void fun(int n) {
    for(int i=1; i<=50; i++) {
        sop();
    }
}
```

itr: 50

TC: O(1

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

$N \rightarrow \frac{N}{2} \rightarrow \frac{N}{4} \rightarrow \frac{N}{8} \rightarrow \ldots 1$

$a = N$

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

terms $= k$

k^{th} term $\rightarrow ar^{k-1}$

$ar^{k-1} = 1$

$N * \left(\frac{1}{2}\right)^{k-1} = 1$

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

$N = 2^{k-1}$

$\log_2 N = \log_2 2^{k-1}$

$\log_2 N = k-1$

$\boxed{k = \log_2 N + 1}$

TC: $O(\log_2 N)$