Today's Content:

→ Comparing 2 Algo's
  a) using execution time ✓
  b) using iterations & graphs ✓

→ Why Big O needed →
  a) Why lower order terms neglected ✓
  b) Why constant coefficient terms neglected ✓
  c) Issues in Big O ✓
  d) Worst Case ✓

→ Space Complexity: → 8:20 AM : 8:28 AM break

→ TLE:
  a) Why TLE occurs & Inf about online editors ✓
  b) How to approach any given problem ✓
  c) Importance of Constraints ✓

→ TC naming conventions

→ 1 adv Problem in calculating iterations.

{ Will be discussed in doubt session / Interview Stay back

→ Mathematically representation of Big O, Ω, Θ

Note: Above content is strictly optional

↳ Context: Given N elements sort them in increasing order

ar[5] = { 3 2 6 8 1 } → { 1 2 3 6 8 }

↳ $N = 10^4$ Input Size

Note: Both Algo will run on same Input sizes

Algo1 {Gaurav}

15 sec

↓

(Windows XP)

↓

(Macbook M2)

↓

7 sec

↓

(C++)

↓

7 sec →

↓

(Top of HOT Volcano)

↓

(Mount Everest)

↓

5 sec

Algo2 {Shala}

10 sec

↓

(Macbook M2)

↓

10 sec

↓

(Python)

↓

(C++)

5 sec

(Mount Everest)

↓

5 sec

Exactly

Execution Time : It depends on so many external factors, hence we generally don't compare, execution time between 2 Algorithms

(Iterations): `i = 1; i <= N; i++) {` → Iterations : N

```
    print(i)
}
```

Conl: To Compare 2 Algorithms, Calulate their Iterations, based on Input Size & Compare them.

Q) →

	Algo1 {Kushal}	Algo2 {Ishaani}
	$100 \log_2 N$	$N/10$
obs:	Iterations	less Execution time
If $N <= 3500$	Kushal > Ishaani	Ishaani : $N <= 3500$ preferd
$N > 3500$	Ishaani > Kushal	Kushal : $N > 3500$ preferd

Ind vs Pak : 10 million

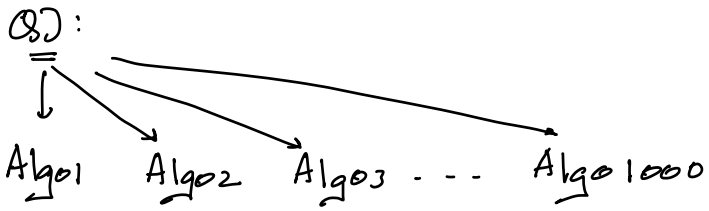
google resu : millions results

Baby Shark : 10.84 Billion

Data → increasing

Pick algorithm which works better for very large inputs.

Col2: Kushal preferred for very large inputs



Assymptotic Analysis of Algorithms

↳ Analysis performance of an algo for very larger Inputs

↳ Big(O) : Algo1 {Khushal} Algo2 {Ishaani}

$100 \log_2 N$ $N/10$

Big(O) → $O(\log_2 N)$ is better than $O(N)$

To Calculate Big(O)

→ Calculate Iterations based on Input size

→ Take higher order term {Neglect lower terms}

→ Ignore constant coefficients

Neglect lower order Terms?

Q) → Algo (Raja's)

Iterations $N^2 + 10N$

Input Size	Total Iterations	% of lower order contribution in total Iterations
$N = 10$	200	$\frac{100}{200} * 100\% \rightarrow 50\%$
$N = 100$	$10^4 + 10^3$	$\frac{10^3}{10^4 + 10^3} * 100\% \approx 9\%$
$N = 10^4$	$10^8 + 10^5$	$\frac{10^5}{10^8 + 10^5} * 100\% \approx 0.1\%$

Obs: → as Input size increases, Contribution of lower order term decreases.

Neglect Constant coefficients

(2) →	Algo1 (Nikhil)	Algo2 (pooja)	For larger Inputs
	$10 \log_2 N$	N	Nikhil
	$100 \log_2 N$	N	Nikhil
	$10^3 \log_2 N$	$N/10$	Nikhil
	$10N$	$N^2/10$	Nikhil
	$N \log N$	$100\ N$	pooja

// Issues in Bpg (O)

Q) → Algo1 (Syed) Algo2 (Rashika) More optimized

$10^3 N$ N^2

Bpg (O) : O(N) $O(N^2)$ Syed is always better?

Claim1: For all input Sizes Syed is Better : {False}

Input

N = 10 → 10^4 10^2 → {Rashika's code optimized}

N = 100 → 10^5 10^4 → {Rashika's code optimized

$N = 10^3$ → 10^6 10^6 → {Both are same}

$N = 10^3 + 1$ → $10^3 * \{10^3 + 1\}$ $(10^3+1)(10^3+1)$ → Syed's code optimized

$N = 10^4$ → 10^7 10^8 → Syed's code optimized

Claim2: For all inputs ≥ 1000 Syed's is Better ✓

Final Claim: When we compare 2 Algo using Bpg O,

Algo1 will be better than Algo2

for all Input Value above a certain point

↳ Threshold point

1) After Threshold Bpg (O) holds
2) Please don't worry above Threshold

Issues in Big O

Q → Algo1 Algo2

Ex1: $2N^2 + 4N$ $3N^2$ → { Algo1 is more optimized }

Big O → $O(N^2)$ $O(N^2)$, { Both are same according to Big O }

Ex2: $5N^3 + 6N^2$ $4N^3 + 10N$ → { Algo2 is more optimized }

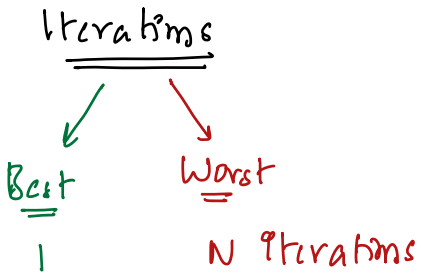
Big O → $O(N^3)$ $O(N^3)$ { Both are same according to Big O }

Obs2: If 2 Algo's have same Big O, we cannot really compare with Big O, we need iterations to compare coefficients of higher order terms in iterations

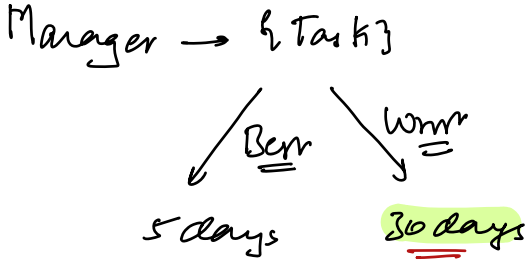
Code ⇒ Iterations ⇒ Big O

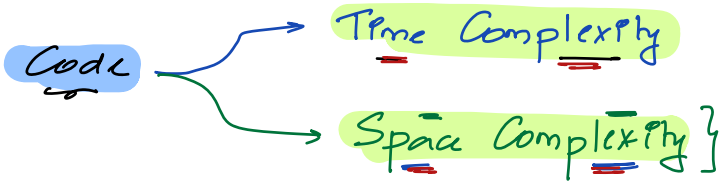
Code: → Searching for an element = k

```
bool search( int[] ar, int k){
    int n = ar.length;
    for(int i=0; i<N; i++){
        if (ar[i] == k) { return True }
    }
    return False
}
```



Note: While writing $B_g(O)$ we consider worst case iterations & write $B_g(O)$





Space Complexity

int - 4B long → 8B

```
____ func ( int N){
|
|  4B. int  x = N
|  4B  int  y = x + x
|  8B: long q = x + y
|
}
```

Total Memory: 4 + 4 + 4 + 8 ⇒ 20B

SC: ⇒ Independent of input

↳ O(1)

```
____ func ( int N){
|
|  4  int  x = N
|  4  int  y = x^2
|  8  long q = x + y
|
|  Declaring int array of size N ⇒ N * 4
|  int[] ar = new int[N]
|
}
```

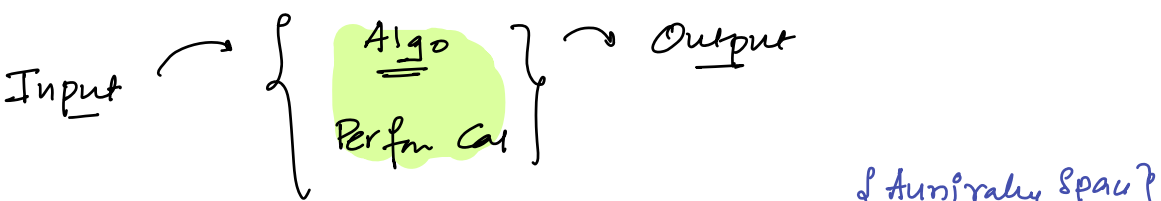
Total Memory:

20B + 4N → O(N)

SC: O(N)

```
____ func ( int N){          Total memory
                              ↳ 20B + 4N + 8N^2
   4B int x = N               SC: O(N^2)
   4B int y = x^2
   8B long q = x + y
   int[] ar = new int[N] : N*4
   Declare long mat[][] of N^2 Elt : N^2 * 8
}  long[][] l = new long[N][N]
```

→ Algo we develop



{Auxiliary Space}

SC of an algorithm: It is amount of space additionly used by algorithm, other than input space to perform necessary computations.

// Given a array write a program to get max of array.

```
int maxarr(int ar[], int N){
    int ans = ar[0]
    for(i=1; i<N; i++){
        ans = max(ans, ar[i])
    }
}
```

{Input}

TC of algorithm : O(N)

SC of algorithm : O(1)

DSA → 6½ months → 600 Problems → TC & SC

// Given an array todo some task

```
int Task(int ar[]){
    int n = ar.length      → 4B
    int pf[N]              → 4N
    pf[0] = ar[0]
    for(i=1; i<n; i++){    → 4B
        pf[i] = pf[i-1] + ar[i]
    }
    // perform few more
       calculations
}
```

Input

SC of algorithm : O(N)

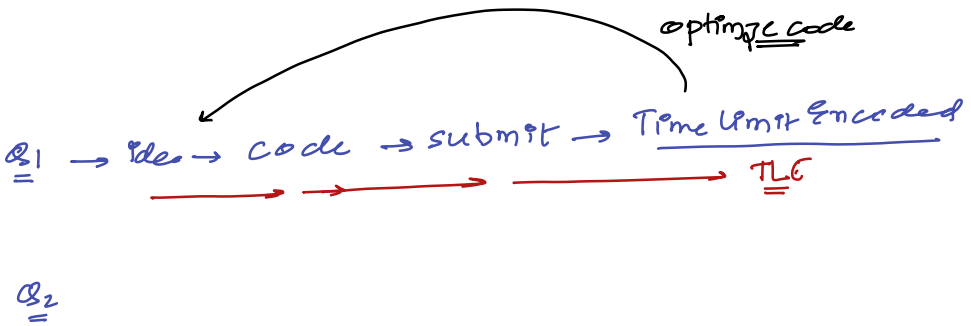
// TC, SC { In most of cases we want to reduce TC first }

→ 0.0001 sec → 2 sec

Microsoft → 0.1 sec → 0.05 sec

TLE : Time Limit Exceeded Error

↳ Mehta → (Amazon) → (Hiring Challenge) → 2Q → duration 1hr



Q2

Remarkable Idea :

↳ { Without even writing a single line code, we can say wether our logic will work }

Online Editors : ⟶ Code Servers ⟶ PS → 1 GHz = 10^9 instructions/sec

↳ Codes should be executed in 1 sec

↳ Mostly

Obs1: At max our code can have 10^9 instructions

↳ variable =
↳ operator =
↳ function calls

Pseudocode

```
bool CountFactors(N){                    → How many iterations = N time
    int c=0                              +1
    for(i=1; i<=N; i++){                 → Per iteration ≈ 6 how many instructions
        if(N%i==0){                      +1  +1  +1  +1
            c=c+1                        +1
        }
    }
    return c                             +1
}
```

Total Instruction ≈ 6N or 7N

App1:

↳ In our code 1 iteration = 10 instructions

→ Our code can atmax contain = 10^9 instruction

$= 10^8 * 10$ instructions

Our code can atmax contain = 10^8 iterations

App2:

↳ In our code 1 iteration = 100 instructions

→ Our code can atmax contain = 10^9 instruction

$= 10^7 * 10^2$ instruction

$= 10^7$ iteration

→ Our code can contain atmax = 10^7 iteration

In general → Code iterations ≈ $[10^7 - 10^8]$ iterations

Structure to Solve a Question

Read Q

Understand Q

Logic

Correctness of logic

Code

Constraints : [More under Information]

Q : Read the Question

Constraints

↳ Input Space

$1 \le N \le 10^5$

Idea → Pseudocode → TC → $O(N^2)$

Idea → TC → $O(N^2)$ →

↓ ↓

$N = 10^5$ → 10^{10} → no need to code

We need to optimize

Q) Read it

Constraints

$1 \le N \le 10^3$

Idea → Pseudocode → $O(N^2)$

↓

$N = 10^3$ → 10^6 → code will work

Q) Read it

Constraints

$1 \le N \le 10^4$

Idea → Pseudo → $O(N^2)$

↓

$N = 10^4$ → 10^8 → {HTT / TLE}

Doubts: → TLE

TODO: