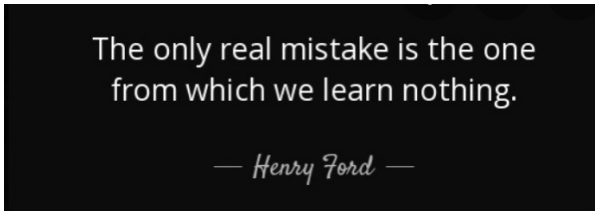
Today's Quote



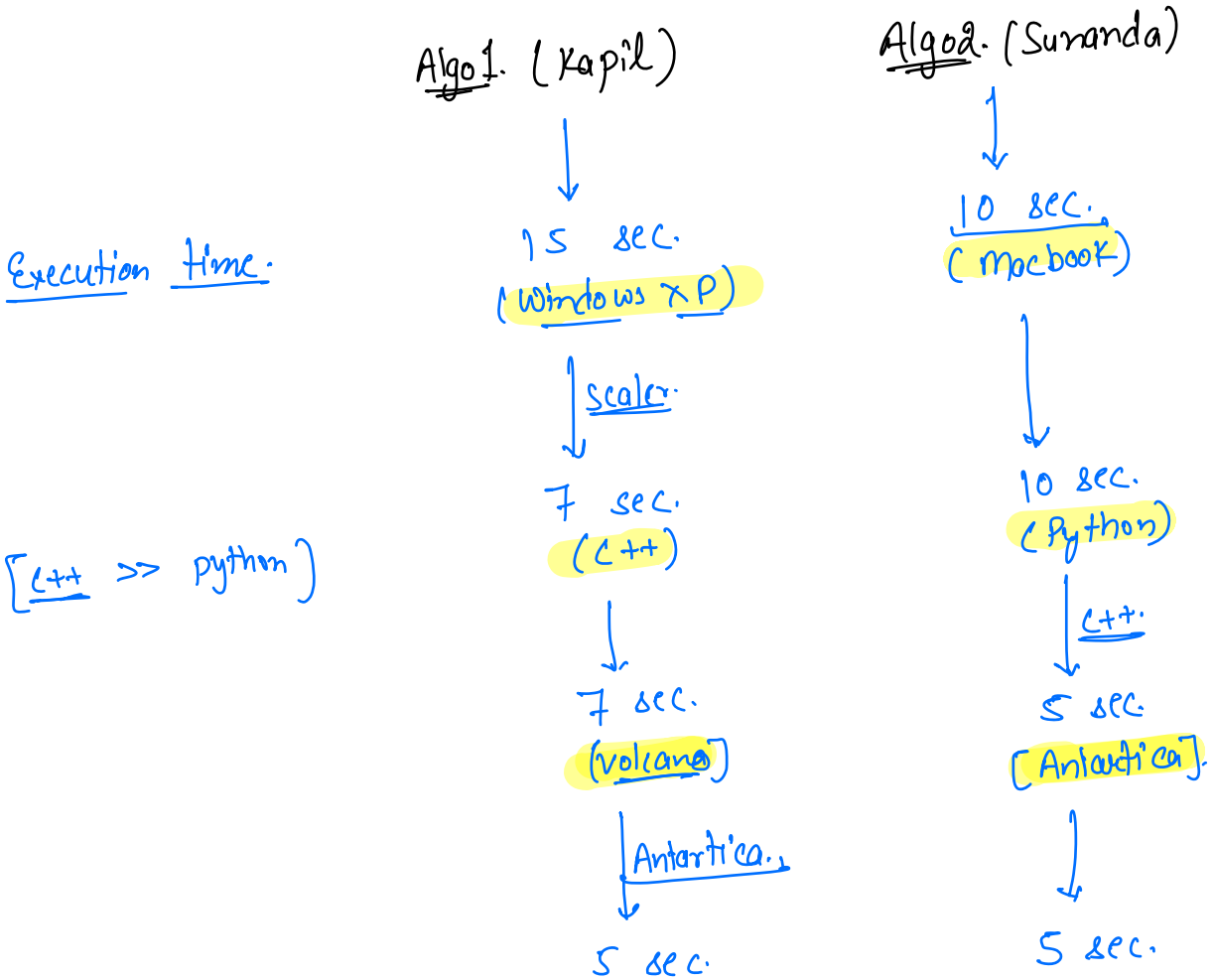
Do mistakes → learn from them → grow.

Today's content:

- Comparing two Algo's.
  - a) using execution time
  - b) using iterations & graphs.
- Why Big-O needed.
  - a) Why lower order terms neglected
  - b) Why constant coefficients neglected
  - c) Issues in Big-O
  - d) Worst case.
- Space Complexity
- T.L.E
  - a) Why TLE occurs & info about online editors.
  - b) How to approach any given problem.
  - c) Importance of constraints.
- TC Naming conventions.

Task: Given N elements. Sort them in increasing order.

- arr → {3 2 5 7 1 11} → {1 2 3 5 7 11}
- input size [N] → 10^9.



Execution time. → It depends on so many external factors, hence we don't generally compare 2 algorithms on the basis execution time.

iterations

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

[iterations = N-1].

Conclusion 1.: To compare 2 algorithms, calculate their iterations, based on input size and then compare.

Algo1. (Gaurav) $100 \log_2 N$

Algo2. (Mayank) $N/10$

till $N <= 3500$ — Mayank's Algo is better.

$N > 3500$ — Gaurav's Algo is better.

Ind vs Pak : 1.3 cr

Google results : millions results.

Baby shark : 11 Billion.

∴ Pick an algo which performs better for very large inputs.

In real world, data is always increasing

To make the comparison simpler

Assymptotic Analysis Of Algorithms ⇒

↳ Analysing the performance of algo's for very large inputs.

→ Big-O

Algo 1.
$100 \log_2 N$

Algo 2.
$N/10$

Big-O. $O(\log_2 N)$ is better than $O(N)$

Biggest advantage of using Big-O.

We can directly tell which algorithm is better by using Big-O notation.

[No need to draw graphs]

Steps to calculate Big-O

→ Calculate iterations. ✓
→ Neglect lower order terms ?
→ Neglect constant co-efficients. ?

Neglect lower order Terms ?

Q) iterations → $N^2 + 10N$

i/p. size.	total iterations	% of lower order terms in total iterations
$N = 10$	200	$\frac{100}{200} \times 100^{50} = 50\%$
$N = 100$	$10^4 + 10^3$	$\frac{10^3}{10^4+10^3} \times 100 \approx 9\%$
$N = 1000$	$10^6 + 10^4$	$\frac{10^4}{10^6+10^4} \times 100 \approx 1\%$
$N = 10^4$	$10^8 + 10^5$	$\frac{10^5}{10^8+10^5} \times 100 \approx 0.1\%$

observation : As Input size ↑es, contribution of lower order terms ↓es.

Neglect constant co-efficients

Q.1	Algo1 (Meenakshi)	Algo2 (Karthik)	for larger input.
	$10 \log_2 N$	N	Meenakshi
	$100 \log_2 N$	N	Meenakshi
	$10^3 \log_2 N$	$N/10$	Meenakshi
	$10 N$	$N^2/10$	Meenakshi
	$N \log N$	$100 N$	Karthik

{Growth rate of $N^2 >> N$}

* Growth rate.

Issues in Big-O

Q.1

	Algo 1	Algo 2
	$10^3 N$	N^2
Big-O.	$O(N)$	$O(N^2)$

Claim 1 : For all input sizes, algo-1 will perform better. : false

$N = 10$	10^4	10^2	{Algo 2 is better}
$N = 100$	10^5	10^4	{Algo 2 is better}
$N = 1000$	10^6	10^6	{same}.
$N = 10^3 + 1$	$10^3 (10^3+1)$	$(10^3+1)(10^3+1)$	{Algo 1 is better}

Claim 2 : For all input $N > 1000$, algo1 will perform better.

Final claim : when we compare 2 algorithms using Big-O, Algo-1 will be better than Algo2.

for all input values → greater than a certain value.

⇓

Threshold value

→ After threshold value, Big-O holds.

→ Don't worry about threshold value.

Issues in Big-O

Q.) Algo1 $2N^2+4N$ Algo2 $3N^2$ {Actually, Algo1 will perform better}

Big O: $O(N^2)$ $O(N^2)$ {Both are same according to Big-O}

$2N^2+4N$ $2N^2+N^2$ growth rate of $N^2 > 4N$

Q.) $5N^3+6N^2$ $4N^3+10N$ {Actually, algo2 is better}

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

observation: If two algorithms have the same Big-O notation, we can't really compare them using Big-O.

{code → iterations → Big-O}

Break till 10:28 PM → 10:37

Code : Searching for element = K

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

iterations.

best. → 1

worst. → N

Manager. → task.

best. → 5-day

worst → 30-days.

Code → Time Complexity

Code → Space Complexity.

Space Complexity.

①
```
void fun ( int N ) {        // 4B
    int x = N ;             // 4B
    int y = x * x ;         // 4B
    long z = x + y ;        // 8B
}
```

total memory →
= (4 + 4 + 4 + 8) Bytes
= 20 Bytes.

S.C → O(1)
constant.

②
```
void fun ( int N ) {        // 4B
    int x = N ;             // 4B
    int y = x * x ;         // 4B
    long z = x + y ;        // 8B
    int[] arr = new int [N] ;
}
```
4 * N Bytes.

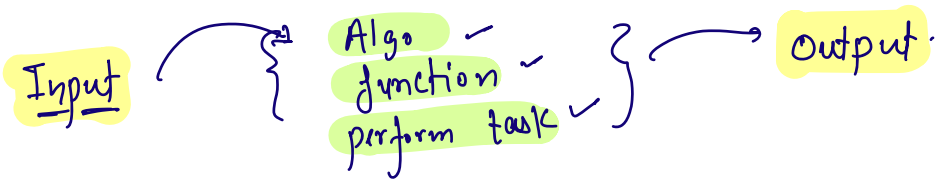
total memory →
(20 + 4N) B

S.C → O(N)

③
```
void fun ( int N ) {        // 4B
    int x = N ;             // 4B
    int y = x * x ;         // 4B
    long z = x + y ;        // 8B
    int[] arr = new int[N];
    long[][] l = new long[N][N];
}
```

total memory →
20B + 4N + 8N²

S.C → O(N²)



Space Complexity of an algorithm : It is amount of extra/additional space needed by algorithm, other than input and output space.

Q.) Max of an array :

```
int max(int[] arr, int n){
    int ans = arr[0];
    for(int i=1; i<n; i++){
        ans = Math.max(ans, arr[i]);
    }
    return ans;
}
```

T.C → O(N)
S.C → O(1)

D.S.A → 600 problems → T.C & S.C

Q.)

```
int task ( int[] arr, int n){
    int[] pf = new int [n];
    pf [0] = arr[0];
    for( int i = 1 ; i < n ; i++){
        pf[i] = pf[i-1] + arr[i];
    }
    // few more calculations
}
```

S.C → O(N)

→ In most of the cases, first try to reduce T.C.

Microsoft → 0.01 sec

↓

0.05 sec

TLE Time Limit Exceeded Error.

Remarkable idea.:

Without even writing a single piece of code, we can say whether our logic will work or not.

Online Editor

code are executed on their code servers.

⇓

Processing speed = 1 GHz.

10^9 instructions/sec.

→ Codes should be executed in 1 sec.

observation: At max our code can have 10^9 instructions.

- → variable declaration
- → operator.
- → function calling
- → comparing.

pseudo-code.

```
int countfactors (int N){      : iterations → N
    int c = 0;
    for(int i=1; i<=N; i++){   instructions ≈ 5N or 6N
        if (N%i == 0) c++;
    }
    return c;
}
```

Approx 1 :

In our code 1 iteration = 10 instructions.

→ Our code at max contain = 10^9 instructions

= $10^8 * 10$ instructions.

[Max. iterations possible = 10^8 iterations.]

Approx 2. [our code will contain 50-60] ≈ 100 instructions.

In our code 1 iteration = 100 instructions.

Our code at max contain = 10^9 instruction

= $10^7 * 10^2$ instruction

[Max iteration possible = 10^7 iterations]

In general, code iterations ≈ $[10^7 - 10^8]$

Structure to solve a question.

Read Q
↓
understand Q
↕
logic
↓
check correctness of logic
↓
code.

constraints : [Most useless info]

$1 \leq N \leq 10^5$

idea → pseudo-code in your mind

↙ 1 loop ↓ nested loop ↘ sorting

idea → T.C → $O(N^2)$

⇒ 10^{10} iterations.

{no need to code this approach}

⇓

Start thinking of optimisation.

Eg : $1 <= N \leq 10^3$

idea → pseudo-code → T.C → $O(N^2)$

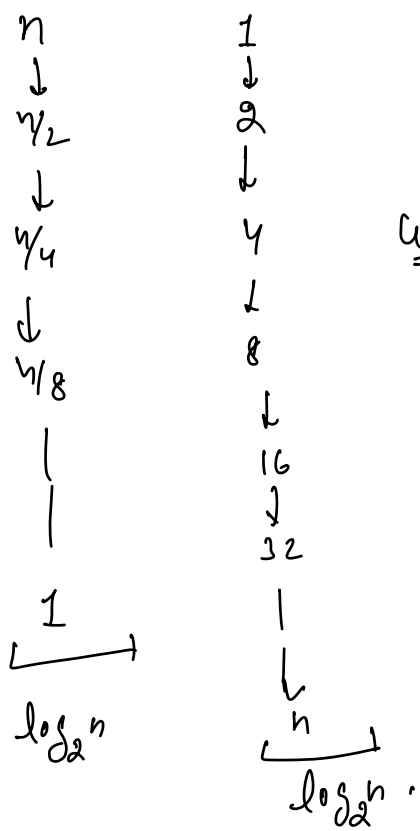
$N = 10^3$ # no. of iterations = 10^6

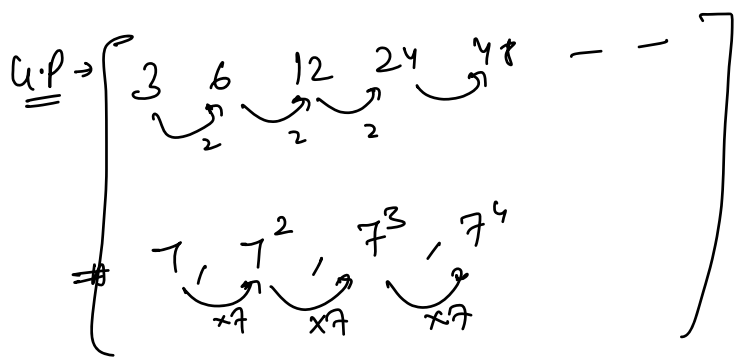
Q.) $1 \leq N \leq 10^4$

idea → pseudo code → T.C → $O(N^2)$

$N = 10^4 \Rightarrow 10^8$ iterations {hit / miss}

→ think of the optimisation.





$2^x = 16$

$x \log_2 2 = \log_2 16$

$x = 4$