Time Complexity-2

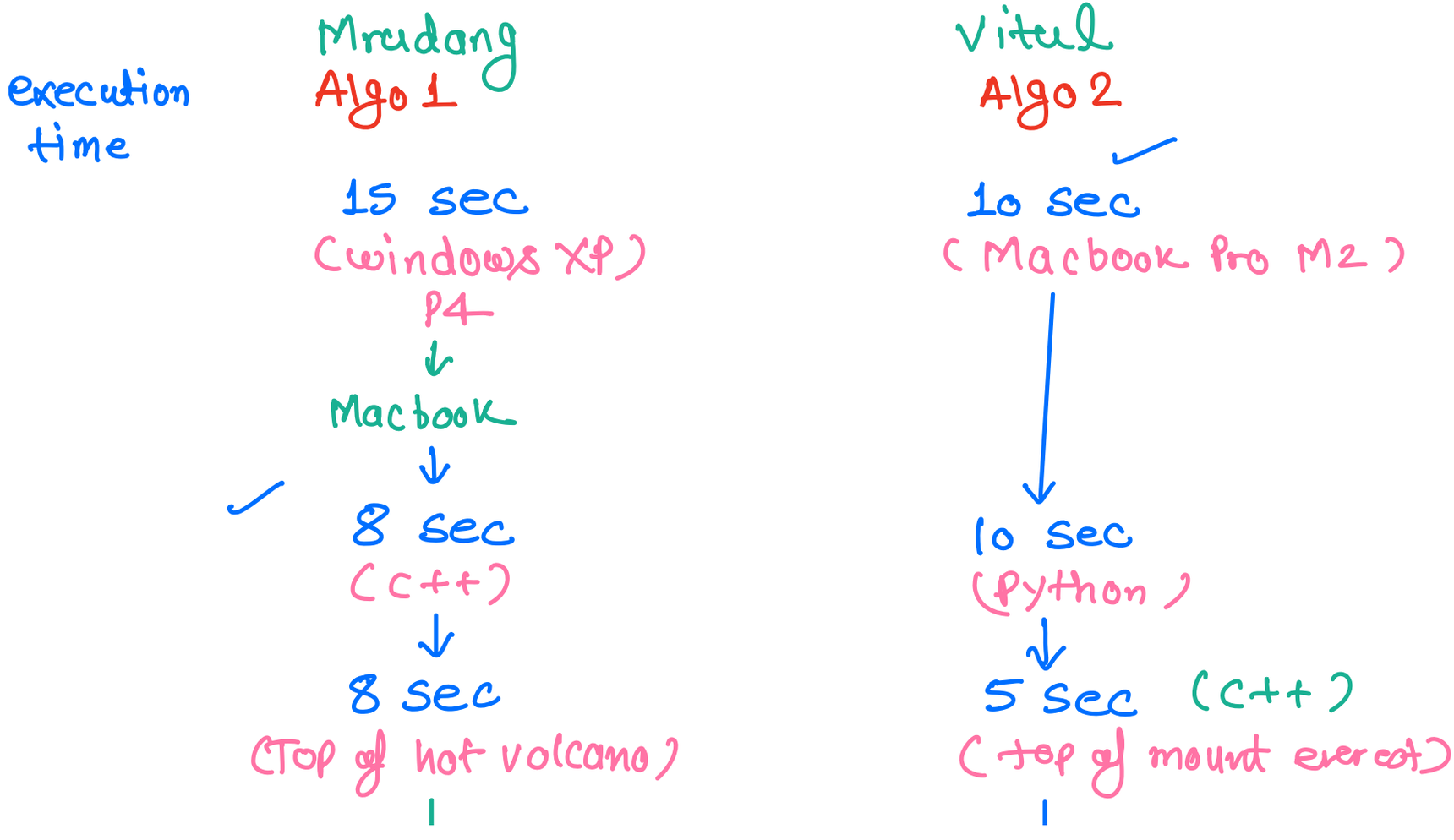
Vinay Neekhra

Senior Instructor & Mentor

Reachable in Scaler Lounge

"Time moves in one direction. Memory in another"

1. Comparing execution time
2. Comparing iterations
3. Asymptotic analysis - Big-O
4. Space complexity
5. TLE



↓ M1 execute

4 sec

↓

5 sec.

Execution time: it depends on so many factors, hence it's usually not a good idea to compare 2 algos based on this.

$N = 1024$

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

(1024 iterations) → N

$N = 1024$

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

(10 iterations) → $\log N$

Obsⁿ: To compare 2 algos. calculate their iterations.

⇒ iterations depends on input

time ∝ iterations [iterations depends upon input]

Q Find the sum of first N natural numbers.

Doubt

Algo 1

```
for (i=1; i<=N)
    sum+=i
```

Algo 2

$sum = \frac{N*(N+1)}{2}$

↙ 1 instruc

N = 100
N = 10

Q

Ankit
Algo 1

$100 \log_2 N$

↑ iterations

Pooja
Algo 2

$N/10$

↑ execution time

Obsⁿ:

Till

$N <= 3500$ Algo 1 < Algo 2

$N > 3500$ Algo 1 > Algo 2

Algo 1 > Algo 2

⇒ Ind vs Pak : 20 million +

⇒ google result: million results (0.001 sec).

⇒ amazon ⇒ # of products ↑

⇒ Pick the algorithm which works better for very large inputs

§.

Algo 1
Algo 2
⋮
Algo 100

Asymptotic analysis of algorithm.

- analysing the performance of algos for very large inputs
- Big-(O)

Ankit
Algo 1

Pooja
Algo 2

$100 \log_2 N$ $\qquad$ $N/10$

$\Rightarrow O(\log N)$ $\qquad$ $O(N)$

$\Rightarrow O(\log N)$ is better than $O(N)$.

Q. To calculate Big-O.

1. Calculate iterations based on input size
2. Remove the lower order terms.
3. Ignore the constant coefficients

$\Rightarrow$ computation time gets affected wrt input.

→ Algo 1 $\Rightarrow O(N)$
$\Rightarrow$ computation time $\propto N$

→ Algo 2 $\Rightarrow O(N^2)$
$\Rightarrow$ computation time $\propto N^2$

5. Neglect lower order terms?

Harshita

iterations $N^2 + 10N$

input size	total iterations	% of lower order terms contribution in iteration.
$N = 10$	200	$\frac{100}{200} \times 100\% \Rightarrow 50\%$
$N = 100$	$10^4 + 10^3$	$\Rightarrow \frac{10^3}{(10^4+10^3)} * 100\% \Rightarrow \sim 9\%$
$N = 10^4$	$10^8 + 10^5$	$\frac{10^5}{(10^8+10^5)} * 100\% \approx 0.1\%$

Obsn: as input size increases, contribution of lower order term decreases.

6. ignoring the constant coefficients

Sumit Algo 1	Jayapratha Algo 2	for larger inputs
$10 \log_2 N$	N	Sumit
$100 \log_2 N$	N	Sumit
$10^3 \log_2 N$	$N/10$	Sumit
$10 N$	$N^2/10$	Sumit
$N \log N$	$100 N$	Jayapratha

Doubt

	Algo 1 $N \log N$	Algo 2 $100 N$
$N = 1024$	I	II
$\Rightarrow$	$1024 * 10$	102400

iteration in Algo 1 are 10 times lesser than Algo 2

$N = 10000$

I: $10^4 * 13$

II: $10^4 * 100$

Obs^n: we don't need to find the inflection point

$\Rightarrow \log N > 100$

II I

inflection pt $[2.5 \times 10^{30})$

5. Issues in Big-O

	Divya (Algo1)	Shilpa (Algo2)	
	$10^3 N$	N^2	Divya's Algo is faster
Big-O	$O(N)$	$O(N^2)$	

claim: for all inputs Divya's code is better. [False]

input			
$N = 10$	10^4	10^2	Shilpa's code.

$N = 100$	10^5	10^4	is more optimised Shilpa's code is faster
$N = 10^3$	10^6	10^6	<>
$N = (10^3+1)$	$10^3 * (10^3+1)$	$(10^3+1) * (10^3+1)$	Divya's code
$N = 10^4$	10^7	10^8	Divya's code

Claim: for all inputs > 1000, Divya's code is faster.

Final claim: when we compare 2 algos using Big-O notation Algo1 will be better than algo 2 for all the input value above a certain point.

inflection point
threshold point

(1) after the threshold Big(O) holds true.

(2) Don't worry about when

Inflection point is coming

8. Issues in Big-(O)

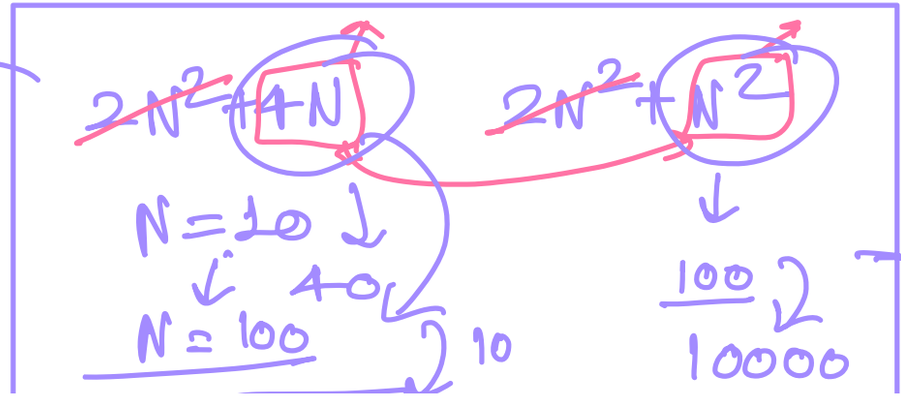
# iterations	Algo 1 Brian	Algo 2 Deepthi
ex.1	$2N^2+4N$	$3N^2$
	$O(N^2)$	$O(N^2)$
$N=10 \Rightarrow$	240	300

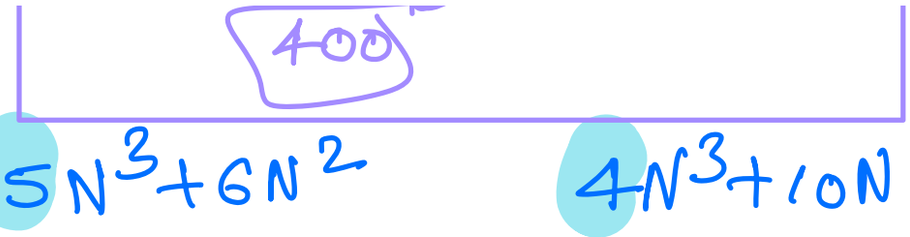
{Brian's code is more optimised}

(Big-O notation is same for both the algos)

Google → 0.01 → 2sec

Microsoft → 0.1sec → 0.5sec





{Deepthi's code is better}

ex2

Big(O) $O(N^3)$ $O(N^3)$ {Big-O is same}

Obs^n: If 2 algo(s) have same Big-(O) notation, we can not compare with Big-(O). we need iterations to compare contributions of higher power terms

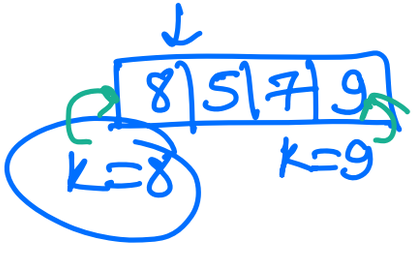
$\Rightarrow$ Sol^n $\Rightarrow$ iterations $\Rightarrow$ Big (O)

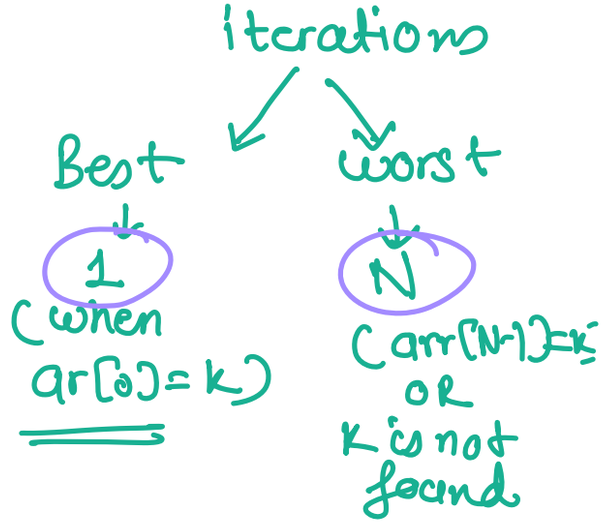
10:35 $\Rightarrow$ 10:45

Code: searching for a number = K.

```
bool search (int[] arr, int k) {
    int N = arr.size();

    for (int i=0; i<N; i++){
        if (arr[i] == k) return true
    }
    return false
}
```





Big (O) ⇒ O(N)

Note: while writing Big (O), we consider worst case iterations.

Code: → time complexity
→ space complexity

int: 4 Bytes
long : 8 Bytes

Quiz

```
func (int N) {          ← 4 Bytes
  4B int x = N
  4B int y = N * N
  8B long z = x * y
}
```

total memory
$= 4+4+4+8$
$= 20$ Bytes

$N = 10 \Rightarrow 20$ Bytes

$N = 10^6 \Rightarrow 20$ Bytes

Space complexity $\Rightarrow$ independant of input

$= O(1)$

[$\propto N \quad O(N)$
$\propto N^0 \quad O(N^0)$

Quiz

```
func (int N) {          ← 4 Bytes
  4B int x = N
  4B int y = N * N
  8B long z = x * y
```

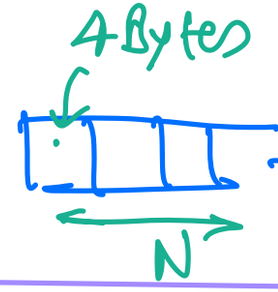
total memory
$= 4+4+4+8+4N$
$\Rightarrow 20+4N$ Bytes.

SC ⇒ O(N)

// Declaring int array of size N

4*N int[] arr = new int[N]

}



Quiz

func (int N) { — 4 bytes

4B int x = N

4B int y = N*N

8B long z = x*y

total memory

= 4+4+4+8+4N
+8N²

SC ⇒ O(N²)

// Declaring int array of size N

4*N int[] arr = new int[N]

long[][] l = new long[N][N]

8 bytes

}

Space Complexity : It is amount of space needed by the algorithm to compute the output.
(other than input)

Q. Given an array, find max of the array

```
int    getMax ( int[] arr ) {
          int N = arr.size();      4B
          int ans;                 4B
          for (i=0; i<N; i++) {    4B
                ans = max (ans, arr[i])
          }
          return ans;
}
```

T.C. = O(N), SC = O(1)

DSA → ≈6 ½ months 600 problems, TC, SC

§. TC, SC { in most cases we prefer optimising time }

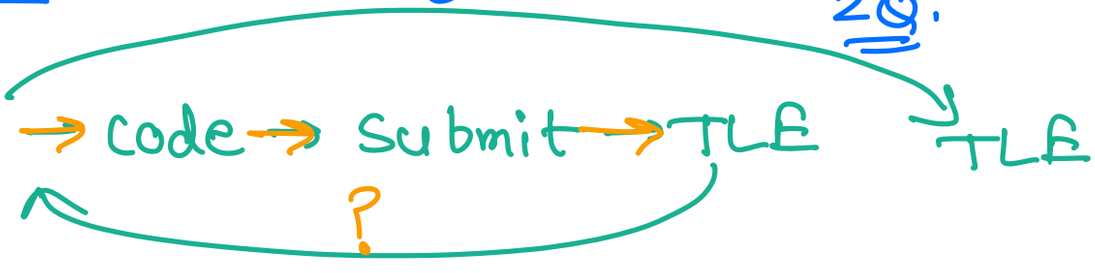
time — ✗ reusable
space — ✓ reusable

0.1 ↓ million $
0.5 ↓

---

§. TLE : Time limit exceeded

Rakesh : Amazon ⇒ (Coding challenge) → 1 hr.



Q2 :

idea : without even writing a single code of

Cine, how we can predict TLE.

⇒ First find the solution, then only code it

Computers ⇒ dumb + fast [0, 1 AND, NOR]

Outsourcing soln to a machine, because they are faster.

100 units → ~30 understand the problem
— 10 Brute force soln
→ 30 unit optimised soln
→ 10 unit edge cases /
→ 15 coding
→ 5 dry running the code

Online editors: Code servers ⇒ 1 GHz = 10^9 instructions/sec.

⇒ Code should run in < 1 sec.

Obs^n : Our code can have $\sim 10^9$ instruction,
command

→ variable declaration
→ a = b + c
→ function call, return

```
bool countfactor(int N) {
    int c = 0                        +1
    for (i = 1; i <= N; i++) {       +1  +1  +1
        if (N % i == 0) {            +1
            c += 1                   +1
        }
    }
    return c;                        +1
}
```

per iteration we have ≈ 5 instruction

~~total~~ instruction ≈ 5N

Situation 1

⇒ in 1 iteration = 10 instructions.

our code can at max have = 10^9 instruction

$= 10^8 * 10$ instruction

$= 10^8$ iterations

Situation 2

in 1 iteration = 100 instructions

iterations = 10^7 iteration

in general ⇒ code iterations ⇒ $(10^7 - 10^8)$ iterations

Q. constraints (kinda not so useful, it seems)

count of factors

$1 <= N <= 10^5$ (I)

$1 <= N <= 10^{10}$ (II)

Bruteforce ✓ ✗

5×10^{10} inst $<= 10^9$

→ First show up, then do it, then doing it right.

⇒ then do it better

① Read the constraints:

$1 <= N <= 10^3$ / $N \in [1, 10^3]$

idea ⇒ pseudocode ⇒ $O(N^2)$

no TLE

$1 <= N <= 10^4$

$O(N^2)$

$N = 10^4 \approx 10^8$ iteration

TLE (Hit / miss)

$1 <= N <= 10^5$

idea $O(N^2)$

⇒ TLE

Doubt session

(I) more practice : interviewBit.com

→ Time complexity
Chapter ⇒ Practice Theme

(II) active & passive learning:
learnings

(I) → lectures
(II) → assignments
(III) → Projects
(IV) → ~~fun~~ (contemplation)

⇒ $O(N^3)$ computation time $\propto N^3$

⇒ $O(N^2)$ ⇒ computation time $\propto N^2$

Constraint
(I) $N : [1 \; 10^3)$ → $N^2 \rightarrow [1 \; 10^6]$ $\propto 10^6$

$< 10^8$ iteration $\quad O(N^2)$

(II) $N = [1 \quad 10^5] \quad N^2 \; [1 \quad 10^{10}] \propto 10^{10}$

$< 10^8$ Iterations $\quad$ TLE

$\Rightarrow$

$\Rightarrow N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \ldots$

$\Rightarrow N \left[1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots\right]$

$\Rightarrow N \left[\frac{a(1-r^n)}{(1-r}\right] \Rightarrow N * \left[\frac{1 * (1-(1/2)^n)}{(1-\frac{1}{2})}\right.$

$\Rightarrow 2N * \frac{(1-(\frac{1}{2})^n)}{1/2} \to 0$

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

$\Rightarrow 2N$

$O(N)$