Given N Array Elements write a sorting algorithm to sort them

→ ar[5] = { 3 2 9 6 11 } $\xrightarrow[\text{Sorting}]{\text{After}}$ { 2 3 6 9 11 }

→ $N = 10^4$

Naman (one Sort)

Saurabh (HipSort)

10 sec

(Mark book Pro)

15 sec

(Windows 98)

// Both are executing in Mac

10 sec

(python)

7 sec

(C++) → { Game development }

// Both are executing in C++

5 sec

{ Hill Station }

7 sec

{ Close to Volcano }

// Both at Hill Station

5 sec ⟷ 5 sec

Execution time : We cannot use as a metric to compare?

↳ depends on lot of other factors → {SW/ HW/ Environ}

```
//
  i = 0; i < N; i++){ n iterations
  |  // ——— ?
  }
```

↳ It will have n iterations

// Let's Compare Iterations

// Write a Sorting Algorithms to Sort N Numbers?

Bharghav

$100 \log_2 N \rightarrow O(\log_2 N)$

Chetan

$N/10 \rightarrow O(N)$

~ – Bharghav
~ – Chetan

N < = 3550 : Bharghave more iterations : Chetan Algo efficient

N > 3550 : Chetan more iterations : Bharghav Algo efficient

// → $2 * 10^8$ Streamed // Google Search Results : 10^8 – } Very large Bharghav

// → 7 Billion // Data ⟶ Increasing

CSD

Algo1 Algo2 Algo3 . . . . .

Iterations :

// Assymptotic analysis of Algorithms :

observing performance of algorithms for very very large inputs

Big O : Comparing using Big O (Why?)

$100 \log_2 N$ $N/10$

$O(\log_2 N)$ $O(N)$

Better?

a) Calculating Iterations ✓

b) Neglect lower order Terms ✓

c) Neglect Constant coefficients ✓

Neglect lower order Terms?

$N \rightarrow$ Input size

Algo $\longrightarrow$ $N^2 + 10N$

// iterations

// per of lower order term iterations in total

$N: 100 \longrightarrow 10^4 + 10^3$

$$\frac{10^3}{10^4 + 10^3} * 100\% = 9.09\%$$

dec

$N: 10^4 \longrightarrow 10^8 + 10^5$

$$\frac{10^5}{10^8 + 10^5} * 100\% = 0.1\%$$

dec

inc

// Contribution of lower order terms becomes significantly small for very large inputs

Neglect Constant coefficients

Q) →

Aditya	Shashank
$10 \log_2 N$	N
$10^2 \log_2 N$	N
$10^4 \log_2 N$	N
$10^4 \log_2 N$	$10N + 10^6$
$10^4 \log_2 N$	N^2

// Issus in Big (O)

Q) ⟶ Nalya | Abhijeet

$100N \rightarrow O(N)$ | $N^2 \rightarrow O(N^2)$

// $N \rightarrow 50$: $100 * 50$ | $50 * 50$: Abhi is efficient

$N \rightarrow 80$: $100 * 80$ | $80 * 80$: Abhi is efficient

$N \rightarrow 100$: $100 * 100$ | $100 * 100$: Same

$N \rightarrow 110$: $100 * 110$ | $110 * 110$: Nalya

$N - 150$: $100 * 150$ | $150 * 150$: Nalya

$N > 100$: (Nalya Efficient)

// Big (O) Comparism holds after certain Input Size

Threshold value

Issus in Big (O) : 2

Q) ⟶

Sony | Ayush

$\boxed{N^2 + 10N}$ | $\boxed{2N^2 + 5N}$

$O(N^2)$ ⟷ $O(N^2)$

↳ Big (O) cannot say which is Better?

For large Inputs:

Sony is Efficient

Code:

```
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
}
```

Worst Case for Big(O)

Best	Worst
1	N
O(1)	O(N)

Manager

50 days

Task 50 days

1 day

Task 1 day

9:15AM → 9:23AM (Break)

After Break:

Space Complexity

Code → Time Complexity
Code → Space Complexity

Space Complexity

given N input

```
func ( int N ){
    4 : int x = N
    4 : int y = x + x
    8 : long z = x + y
}
```

int : 4B, long : 8B

16 B → O(1) Space
↳ Constant Space

Input Size

```
func ( int N){
    4  int x = N
    4  int y = x^2
    8  long z = x + y
    4N int[] ar = new int[N]
}
```

Total space : 4N + 16
↳ O(N) Space

```
func (int N){
4    int x = N
4    int y = x^2
8    long z = x+y
4N   int[] ar = new int[N]
8N^2 long[][] l = new int[N][N]
}
```

Total space : $16 + 4N + 8N^2$

↳ Space Complexity : $O(N^2)$

// Given N Elements calculate sum of N Elements

{ which is given } → Input

```
int sumarr( int ar[] )
{
4    int sum = 0
4    i = 0; i < N; i = i+1 {
        sum = sum + ar[i]
     }
     return sum
}
```

Total Space :

$4N + 8$

↓

Space Complexity : $O(N)$ ✗ → Wrong

Won't count in SC

→ $(4+4) + 8 \rightarrow O(1)$

Space Complexity : Amount of Extra Space taken by your algorithm other than { Input Space }

given this array

```
// ___ man (int[] ar){
```

you won't consider this in Space

```
    4 : int m = ar[0]
    4 : i=0; i< ar.length; i++){
    8B      if (m < ar[i]) { m = ar[i]}
        }
        return m
    }
```

8B → O(1)

TC/SC

6 months : TC / SC

TLE : Time Limit Exceeded Error

google : → (2 Questions)

1hr

Nagya : → 1Q : → logic → Code $\xrightarrow{\text{Submit}}$ TLE

optimize

// Without even single code → logic → work / TLE

// Online editors $\xrightarrow{\text{code}}$ (Their Servers) $\xrightarrow{\text{(1GHz)}}$ 10^9 instructions / sec

In general Time Limit : 1 Sec

At max code can have 10^9 instructions

{ any operation, function calling, variable declaration }

TODO:

Pseudo Code

```
— func(N){
    int i = 1; i = N; i = i+1){
        if (i % 2 == 0) {
            print(i)
        }
    }
}
```

+1 +1 +1 +1 +1 +1 +1

Total instructions:

$1 + 5N + N$

$1 + 6N$

// Assumption Iteration = Instructions

// At max how many instructions = 10^9 instructions

1 iteration = 10 instructions = $10^8 \times$ 10 instructions

// At max code should have = 10^8 iterations

// Question:

1) Read / Understand
2) Logic
3) Correctness
4) Check TLE occurs
   - If TLE ?
   - If No TLE
5) Write & Execute

Constraints : Given N Array element find Something

Q: Constraints

$1 < = N < = 10^5$

$1 < = arr[i] < = 10^9$

→ logic

→ Pseudo Code

$O(N^2)$: → 10^{10} iterations

$N = 10^5$

TLE (no need code)

optimize logic further

Q): Constraints

$1 \le N \le 5*10^3$ → $25*10^6 \Rightarrow 2.5*10^7$

logic → Pseudocode → $O(N^2)$ → work

not optimized

$1 \le N \le 5*10^2$ → $125*10^6 \Rightarrow 1.25*10^8$ → very rear

logic → Pseudocode → $O(N^3)$

If your code is also having $\approx 10^8$ iterations

{ it can work | it might not }

// Wednesday : (Arrays)

Beginner : 1D arrays & 2 Sessions } → Can you plean review this.

Beginner : 2D arrays & 2 Sessions

(Beginner) / (Inter 1) / (Adv)

→

// Doubts:

Naming Terminologies

- → $O(1)$: Constant
- → $O(N)$: Linear
- → $O(N^2)$: Quadratic
- → $O(N^3)$: Cubic
- → $O(N^k)$: Polynomial
- → $O(2^N)$: Exponential
- → $O(\log N)$: logarthmic
- → $O(N \log N)$: linear logarthemic
- → $O(N!)$: factorial

→ Question: (HW/ Assign)