Task → Given 10^6 integers in random order, sort them in ascending order. [3 1 8 2] → [1 2 3 8]

	Siva	Nuthan
	Algo 1	Algo 2
Execution Time →	15 sec (Windos XP)	10 sec (Macbook M2)
	↓ (Macbook M2)	↓
	8 sec (C++) → faster than Python	10 sec (Python)
	↓	↓ (C++)
	8 sec (Very Hot temperature)	5 sec (Very cold temperature)
	↓ (Very cold temperature)	↓
	5 sec	5 sec

∵ Execution time depends on multiple factors
∴ it is not a good parameter to compare algorithms. ✓

```
for i → 1 to N
    print(i)
```

} # iterations = N (do not depend on any factor)

Task → Given N integers in random order, sort them in ascending order.

	Pratik	Shashank
	Algo 1 ✓	Algo 2
# iterations ⇒	100 log (N)	N/10

N	# iterations	
N <= 3500	Algo 1 > Algo 2	→ Algo 2 is better ✓
N > 3500	Algo 1 < Algo 2	→ Algo 1 is better ✓

Real Cases

1) Daily active instagram users → ≈ 500 M
2) Most viewed youtube video ⇒ Baby Shark → ≈ 11 B views
3) Whatsapp messages in one day → ≈ 100 B

∴ system are designed in such a way that can take care of large input.

---

100 Algorithms to compare ↓ Akash

Take random large inputs & compare algorithms to see which is performing better. ↘ Asymptotic Analysis

↓

Big O

iterations → 100 log (N) → O(log (N)) ✓
→ N/10 → O(N) ✓

1) Calculate # iterations based on input size. $2N^2 + 3N - 100$
2) Take higher order terms & ignore it's constant coefficient. → constants do not impact rate of growth.

Neglect lower order terms ?

$N^2 + 10N$

Input Size	Total iterations	% of lower order contribution in total.
$N = 10$	$10^2 + 10*10 = 200$	$\frac{100}{200} * 100 = 50\%$
$N = 100$	$100^2 + 10*100 = 10^4 + 10^3 = 11000$	$\frac{10^3}{11000} * 100 = 9\%$
$N = 10^4$	$10^8 + 10*10^4 = 10^8 + 10^5$	$\frac{10^5}{10^8 + 10^5} * 100 = 0.1\%$

∴ For larger inputs we can ignore lower order terms. ✓

Issues with Big O

1)

	Ashwani	Khushboo
	Algo 1	Algo 2
# iterations →	$1000N$	N^2 ✓
Big O →	$O(N)$	$O(N^2)$

Algo 1 is always better than Algo 2. ✗

Algo 1 is better than Algo 2 for large inputs. ✓

$N > 1000$

2)

	Divya	Sourabh
	Algo 1 ✓	Algo 2
# iterations →	$2N^2 + N$	$5N^2$ ←
Big O →	$O(N^2)$	$O(N^2)$

both algo are equally good. ✗

$2N^2 + N < 5N^2$

$-2N^2 \quad -2N^2$

$N < 3N^2$

10:35 PM

Q → search for an element in array.

```
boolean search(A[], K) {
    N = A.length
    for i → 0 to (N-1)
        if(A[i] == K)
            return true
    return false
}
```

#iterations
- Best = 1
- Worst = N

Big O → TC = O(N)

Space complexity → Rate of growth of space wrt input.

int → 4 Bytes long → 8 Bytes ✓

1)
```
{
    int x = 10;     → 4B
    int y = x*x;    → 4B
    long z = x+y;   → 8B
}
```
Total memory → 4+4+8 = 16 B

SC = O(1)

2)
```
{ //I/P → N
    int x = N;      → 4B
    int y = x*x;    → 4B
    long z = x+y;   → 8B
    int A[] = new int[N]; → 4*N B
}
```
Total memory → 4+4+8+4N = 16+4N

SC = O(N)

3)
```
{ //I/P → N
    int x = N;      → 4B
    long y = x*x;   → 8B
```
Total memory = 4+8+8N+4N² = 12+8N+4N²

long A[] = new long[N]; → 8*N B

int B[][] = new int[N][N]; → 4*N² B

SC = O(N²)

}

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_{2*2}$

Total Space = I/P space + Algo. Space + O/P space

SC is the extra space apart from i/p & o/p.

4N Bytes

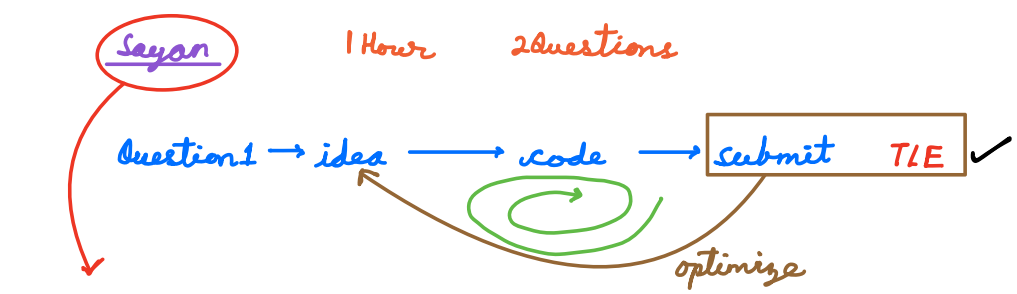
```
int maxArray(int A[]) {
    int N = A.length
    int ans = A[0]
    for int i → 1 to (N-1)
        if(A[i] > ans)
            ans = A[i]
    return ans
}
```

SC = O(1)

int A[3][N]; → SC = O(N)

4* 3*N Bytes

TLE → Time Limited Exceeded



Without writing code → knows about TLE.

TC = O(# iterations)

Online Platforms

code → run on platform server → Processing speed = 1 GHz

Time limit = 1 sec → 10^9 instructions

1 GHz → 10^9 instructions/sec

a = 2 → 1

$a = b + c \rightarrow 2$

Assumption → # instruction in 1 iteration → 10 to 100

For 10^9 instructions → #iterations = 10^8 | #iterations = 10^7

iterations → 10^7 to 10^8 in 1 sec ✓

TC = $O(N^2)$ ✓

Constraints

		#iterations	
$1 <= N <= 10^5$	→	$10^5 * 10^5 = 10^{10}$	TLE
$1 <= N <= 10^3$	→	$10^3 * 10^3 = 10^6$	✓
$1 <= N <= 10^4$	→	$10^4 * 10^4 = 10^8$	may or may not work. ✓

```
for i → 1 to N {
    - -
    - - -
    - - -
    - - -
    - - -
    . . .
}
```

```
for (i = N ; i > 0 ; i /= 2)
    for (j = 0 ; j < i ; j++)
```

i	j	# iterations
N	[0 N-1]	N
N/2	[0 N/2-1]	≈ N/2
N/4	[0 N/4-1]	≈ N/4
⋮		
1	[0 0]	1

$N + \frac{N}{2} + \frac{N}{4} \cdots + 1$