Time & Space 2

→ Quizzes

01. Comparing algos
   (a) Using execution time
   (b) Using iterations & graphs

02. why Big O is needed
   (a) why lower order terms are neglected
   (b) why constant coeff are neglected
   (c) Issues in Big O
   (d) Worst case

03 Space complexity

04 TLE
   (a) why TLE occurs & info about online editors
   (b) Importance of constraints.

Quiz

01. $f(n) = 3n^2 + 6n + 10^3 = O(n^2)$

02 $f(n) = 3n^2 + 6n^3 + 10^3 \rightarrow O(n^3)$

03 $f(n) = 3n + 6n\log n + 10^3 \rightarrow O(n\log n)$

04. $f(n) = 10^3 \rightarrow O(1)$

05.
```
for (int i=1 ; i*i*i ≤ n ; i++) {
    print(—) ;
}
```

Q Given $N = 10^4$ elements, sort them in increasing order.

arr[5] = {3 2 6 8 1} —sort→ {1 2 3 6 8}

→ 2 people have submitted their Algos

Note :- Both are executed on same input

Execution time

Algo 1

15 sec {Windows XP}

↓ Macbook

8 sec {C++}

↓

8 sec {Hot volcano}

↓ Normal temp

6 sec

Algo 2

10 sec {Macbook pro}

↓

10 sec {Python}

↓ C++

6 sec {Normal temp}

↓

6 sec

Issues with execution time

→ Execution time depends upon lot of factors.

this is why not a good idea to compare algorithms

```
void func( int [] ar )
{
    int n = ar.length
    int s = 0
    for ( int i = 0 ; i < n ; i++ ){
        s = s + ar[i]
    }
}
```

→ iterations i : [0 n-1]

= n iteration

Conclusion → To compare 2 algos, we should always consider the no. of iteration

Comparing Iterations

Q Say for a question, two codes are submitted

	Algo 1	Algo 2
Iterations :	$100 \log_2 N$	$N/10$

$N \leq 3550$ Algo 1 has more iterations : Algo 2 ↑

$N > 3550$ Algo 2 has more iterations : Algo 1 is faster

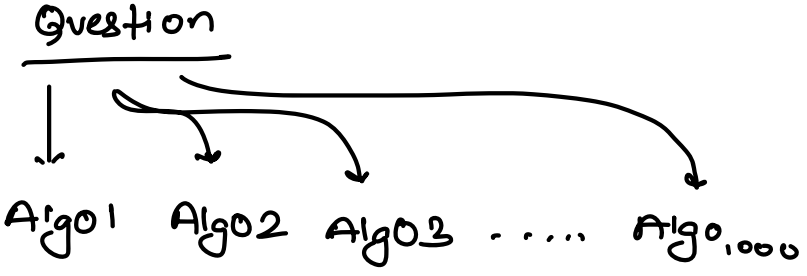
Real world → Data that we works on is in millions or billions, data increasing

Ind vs Pak → 1.8 cr = 2 cr

Google Result → Millions results

Highest viewed → 11 billions
youtube

Obs → Since the data is in millions.
always consider the algo which works better for larger input.



To compare all these algorithm → drawing graph & compare ✗

Asymptotic analysis of algorithm

↳ Analysing the performance of algos with very large input.

Big O

Omega

theta

How to calculate Big O ?

01. Calculate no. of iterations ✓

02. Consider only higher order term ✓

03. Neglect the constant coefficient ✓

— ^ — ^ —

01. Why only higher order term ?

No. of iteration = $N^2 + 10N$ { 10N = lower order term }

Input	Total iterations	Contribution of lower order terms
n = 10	200	$\frac{100}{200} \times 100\% = 50\%$
N = 100	$(100)^2 + 10*100$ $= 11000$	$\frac{1000}{11000} \times 100\% = 9\%$

$N = 10^4$

$(10^4)^2 + 10 * 10^4$

$= 10^8 + 10^5$

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

Obs → Significance of lower order decreases with increase in input size.

02. Why neglect the constant coeff?

	Algo1	Algo 2	Comparison of Algo1 & Algo 2
01.	$10 \log N$	N	Algo 1
02	$100 \log N$	N	Algo 1
03	$1000 \log N$	N	Algo 1
04.	$10 N$	$\frac{N^2}{10}$	Algo 1
05.	$N \log N$	$100 N$	Algo 2

$N = 10^6$

$10 * 10^6$

$= 10^7$

$\frac{10^6 * 10^6 5}{10}$

10^{11}

Issues in Big O

	Algo 1	Algo 2
No. of iterations	$10^3 N$	N^2
Big(O) →	$O(N)$	$O(N^2)$

Claim → Algo 1 is always better Algo 2

N	$10^3 N$	N^2
10	10^4	10^2 : Algo 2 is faster
100	10^5	10^4 : Algo 2 is faster
10^3	10^6	10^6 : Same
10^3+1	$10^3(10^3+1)$	$(10^3+1)^2$: Algo 1 is faster $= (10^3+1)(10^3+1)$ $= 10^3(10^3+1) + 1(10^3+1)$
10^4	10^7	10^8 : Algo 1 is faster

Obs:- $N < 10^3$: Algo 2 is better

$N > 10^3$: Algo 1 is better

Conclusion → One algo is better than the other algo after a certain threshold point.

Issue 2

Algo 1	Algo 2
$2N^2 + 4N$	$3N^2$
Big(O) → $O(N^2)$	$O(N^2)$

$2N^2 + 4N - 2N^2$ → $4N$

$3N^2 - 2N^2$ → N^2

{Algo 1 is more fast}

Conclusion → If same higher order terms are present, Big O can't compare them correctly.

Q → count the odd elements from 1 to N

```
01. for (i=1 ; i≤N ; i++) {
        if ( i%2 != 0) c++;
    }
```

i : [1 N] = N iterations

TC : O(N)

```
02. for (i=1 ; i≤N ; i=i+2) {
        c = c+1
    }
```

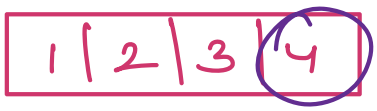
i : [1 N] = N/2 iterations

TC : O(N)

---

data = 1

```
boolean search (int [] ar, data) {
    for ( i=0 ; i<n ; i++) {
        if (ar [i] == data) return true;
    }
    return false;
}
```



Iterations

Best

0th idx

↓

1 iteration

Worst

Not present

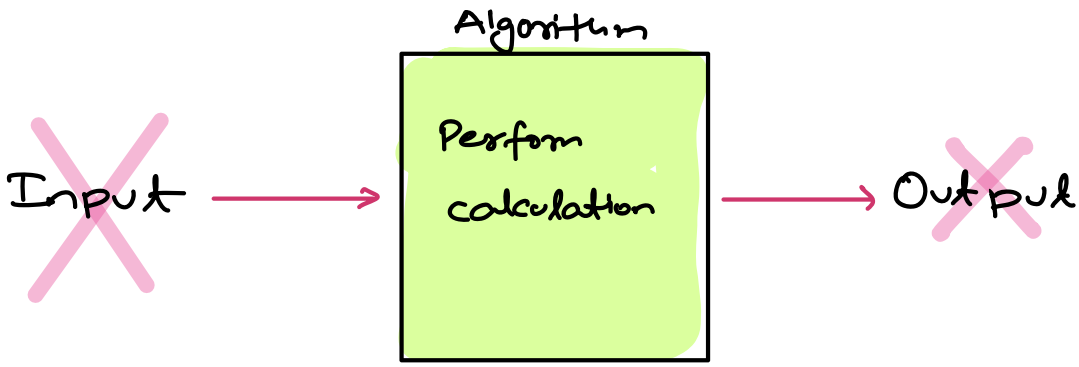
↓

N iterations

Big(O) → Always consider the worst case scenario.

Space complexity

Code
- → Time complexity → Calculate the no. of iterations & get Big O
- → Space complexity → Calculate the space that our algorithm takes & return Big O



Q → Given two values a & b, return an array containing a first & b second

```
int [] create (int a, int b)
    int [] ans = new int [2]
    int [] ar = new int [2];
      ar[0] = a
      ar[1] = b
    return ans;
```

→ ans = a;

ans [0] = a
ans [1] = b

Space complexity → Amount of space additionally used to perform calculations in our algorithm, excluding the input & output space

int → 4B

long → 8B

```
func (int n){
4B |  int x = n;
4B |  int y = x + x;
8B |  long z = x + y;
   }
```

Memory → 4B + 4B + 8B

⇒ 16 B

Big (O) → O(1)

02.

```
func (int n){
4B |  int x = n;
4B |  int y = x + x;
8B |  long z = x + y;
4N |  int []ar = new int[n]; → N integers
   }
```

Total space = 4B + 4B + 8B + 4N

⇒ 4N

Big (O) → O(n)

N * 4

=

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

Total space = $4N + 8N^2$

$\approx 8N^2$

Big(O) = N^2

// → Function to get max in array

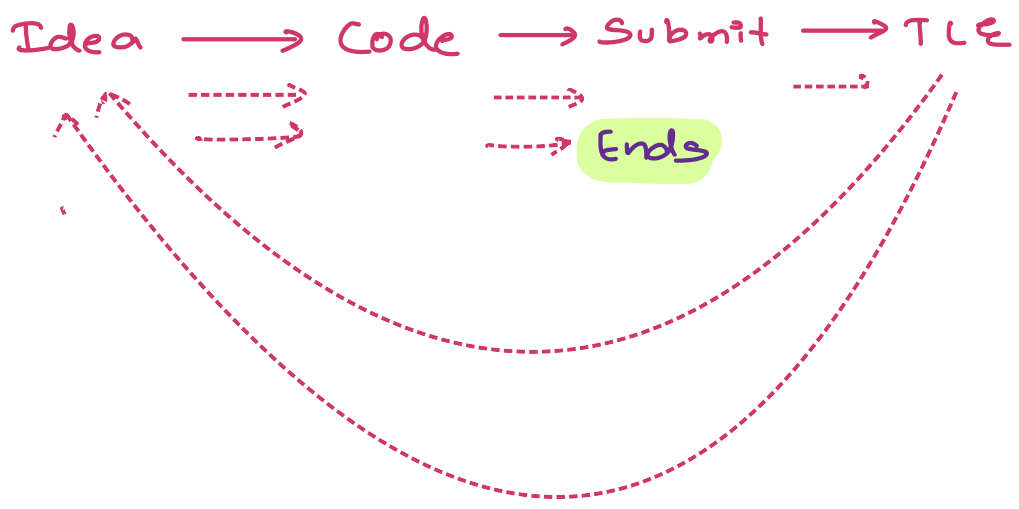
```
int maximum (int []ar)
    int maxi = ar[0]
    for (i=0 ; i<n ; i++){
        if (ar[i] > maxi) maxi = ar[i];
    }
    return maxi;
}
```

TC : O(n)
SC : O(1)

→ consider in our space ✗

TLE → Time Limit Exceeded

Upma (Google) → Hiring challenge (60 mins)



Idea → Without even writing a single line of code, we should be able to judge if our logic is going to work or not

Working on online editor

Online Editors —Run→ Online servers

Processing speed = 1 GHz

$= 10^9$ ____/sec

$\Rightarrow 10^9$ instructions/sec

Code must be executed 1 sec → 10^9 instructions

→ declaring variable

→ operator

→ calling function

→ if clauses

→ assigning values

```
boolean countins (int N)
{
    int c=0          (+1)(+1)
    for (int i=1 ; i<=n ; i++)
    |                (+1)(+1)(+1)(+1)
    |   if (N%i == 0) c=c+1;
    |                (+1)(+1)(+1)(+1)(+1)
    }
    return c;
           (+1)
}
```

Per iteration ≈ 6 or 7 instruction

Total instruction = $7N + 5$

$\approx 7N$

Approximation → 1 iteration = 10 instruction

At max, 10^9 instruction

$= 10^8 * 10$ instruction

$= 10^8$ iterations

Our code can have 10^8 iterations

1 iteration = 100 instruction

At max, 10^9 instructions

$= 10^7 * 100$ instruction

$= 10^7$ iterations

Our code can have 10^7 iterations

Code iterations → $10^7 - 10^8$ iterations

Importance of constraints

Question structure

→ Description

→ Constraints

→ Input & output format

→ Example input & example output

Q → Given arr[N] ... do something

Constraints → $1 \le N \le 10^5$

→ $1 \le arr[i] \le 10^9$

Idea → Three loops → $O(n^3)$ ✗

Two Nested loops → $O(n^2)$ ✗

One loop → $O(n)$

3 loops

Worst case, $n = 10^5$

$O(n^3) = (10^5)^3 = 10^{15}$ iterations

2 loops

Worst case, $n = 10^5$

$O(n^2) \rightarrow (10^5)^2 \rightarrow 10^{10}$ iterations

1 loop

Worst case, $n = 10^5$

$O(n) \rightarrow 10^5$ iterations

Q → Given arr[N] ... do something

Constraints → $1 \le N \le 10^3$

→ $1 \le arr[i] \le 10^9$

2 loops → $O(n^2)$

Worst case = 10^3

TC → $(10^3)^2 = 10^6$ iterations

Ex 3 Given arr[N], calculate & return sum of arr[] ele

Constraints : $1 \le N \le 10^5$

$1 \le arr[i] \le 10^9$

N = size of arr

arr[i] → ele in arr

```
int sum (int [] arr , int n)
{
    int s = 0 , int i;
    for( i = 0; i<n; i++){
        s = s + arr[i];
    }
    return s;
}
```

Min sum

| 1 |

sum = 1

Max sum

arr[10^5] = { 10^9, 10^9, 10^9, 10^9 ... 10^9 }

sum = $10^5 * 10^9 = 10^{14}$

long s ✓

Constraint

↳ Will give the idea for TLE

↳ Data types for the variables

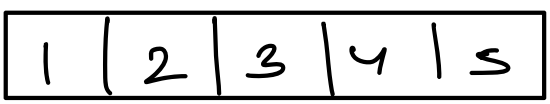
Doubt session

a, b

```
if (a > b) return a
else       return b
```

```
int maxi = Math.max(a, b)
```

or



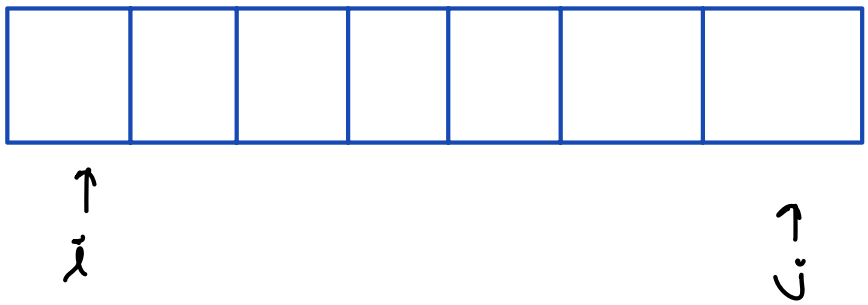
→ Arrays.copy → copying this arr into a new arr

```
int [] ans = Arrays.copy(arr)

    for (int i=0; i<n; i++)
        ans[i] = arr[i];
}
```

TC = O(n)

```
int sum = 0

for (i = 0 , j = arr.length - 1 ; i < j ; i = i*2, j = j*3)
    sum += 
```

2



n = 10

```
(i = 1 ; i ≤ n ; i += 2)
{
    ≡
}
```

1, 3, 5, 7, 9

⇒ no. of iterations = $\frac{n}{2}$

iterate	value of i after iterate
1	3 → 2i+1
2	5 → 2i+1
3	7 → 2i+1
4	9 → 2i+1
⋮	⋮
k	2k+1

$2k+1 = n$

$2k = n-1$

$k = \boxed{\frac{n-1}{2}}$

$\leadsto O(n)$

Q⇒ $3N + 100 \log N$ → No. of iterations

→ ~~$3N$~~ $\leadsto O(N)$

$i \leq \sqrt[3]{n}$

→ `for ( i=1 ; i*i*i ≤ n ; i++) {`

3 ≡

i : ⓐ [1 ⓑ $\sqrt[3]{n}$]

$n=20$ 1, 2, 3

$b-a+1$

$= \sqrt[3]{n} - y + x$

$= \sqrt[3]{n}$

→

```
for (i = 0 ; i<n; i++)
|   for (j = i  ; j<n; j++) {
|   |   break ;
|   3
3
```

i	j	iterate
0	[0 n-1]	1
1	[1 n-1]	1
2	[2 n-1]	1
⋮		⋮
n-1	[n-1 n-1]	1

n