Introduction to Time and Space Complexity

* Time complexity

Question – Find the sum of all numbers from 1 to n ($1 \le n \le 10^8$)

Bruteforce Approach –

```
static int sum(int n) {
    int sum = 0;
    for (int i=0; i<n; i++) {
        sum = sum + i;
    }
    return sum;
}
```

n = 1, 2 .. 10^8 more time

this

using Formula – This is time optimized solution than

```
static int sum(int n) {
    int Ans = n*(n+1)/2
    return sum;
}
```

* Algorithms – A1 A2

overall running time – 1 sec 2 sec

we have to choose which algorithm is better we have not given anything like which language / Machine is used, Generally we choose A1 is better because overall running time is less. but we have to not judge any Algorithm by seeing their overall running time

→ Intution of Time complexity is probably overall running / execution time of an Algorithm (which is wrong)

→ In Any Algorithm / program overall running time depends on multiple factors i.e

1. logic / Algorithm
2. Language → c, c++, java, python, JS
3. Machine dependent
   eg- i3 (slow) Macbook (fast)

python, JS are scripted languages which executes in more time where as c, c++, java are compile / Interpreted languages which are faster than scripted languages - which takes less time to execute.

* Analysis of overall running time of a algorithm also called as <u>experimental Analysis</u>

* We have Machine which executes the same java program multiple times we cant say all the outputs will give us same running time that depend on cpu etc.

* eg we have an Algorithm A and we are saying A is best Algorithm, write A in <u>c++ / Java / python / I3 / M1 pro etc</u>

A should be best in every configuration related / compared to other Algorithm. To Analyze this Time Complexity helps. Time Complexity is Independent of this

* Dont measure Time Complexity Analysis in Overall running time / experimental Analysis

* To analyze time complexity we consider

Def: → No of operations performed in an Algorithm / program. (no. of operations are based on Input size) (No of operations as a function of Input size)

eg -

Algorithm 1 → 10^8 operations / Instructions

Algorithm 2 → 10 operations / Instructions

both the Algorithms are solving the same problem. now we can say that Algorithm 2 is better Algorithm than Algorithm 1. Instead considering with overall running time we cant say which one is better.

* To analyze / calculate time complexity we use <u>Asymptotic Analysis</u> (i.e no. of operations as a function of Input which is performed in a Algorithm)

eg - How many operations / Instructions that are performed in an program as a function of n

Q. Find the Sum of all Numbers from 1 to n

```
int sum(int n){
    int ans = 0;          ①
    for(int i=1; i<=n; i++)   ②
        ans = ans + i;        ③
    return ans;           ①
}
```

In one Iteration 3 operations for n Iterations there will be 3n operations

≃ 3n + 3 operations

→ 3n + 3 operations

→ let $n = 10^8$

$3(10^8) + 3$

→ +3 doesnt give any significance because the machine performing $3(10^8)$ operations is so big so extra 3 operations we can ignore

→ ∴ 3n + 3 = 3n ⇒ $3(10^8)$

$3(10^8) \simeq 10^8$ but $10^8 \simeq 10^{16}$ ×

→ Hence we can say that

3n + 3 ⇒ n operations

② int sum(int n){

int Ans = n*(n+1)/2
① ① ③ ② ④ ①

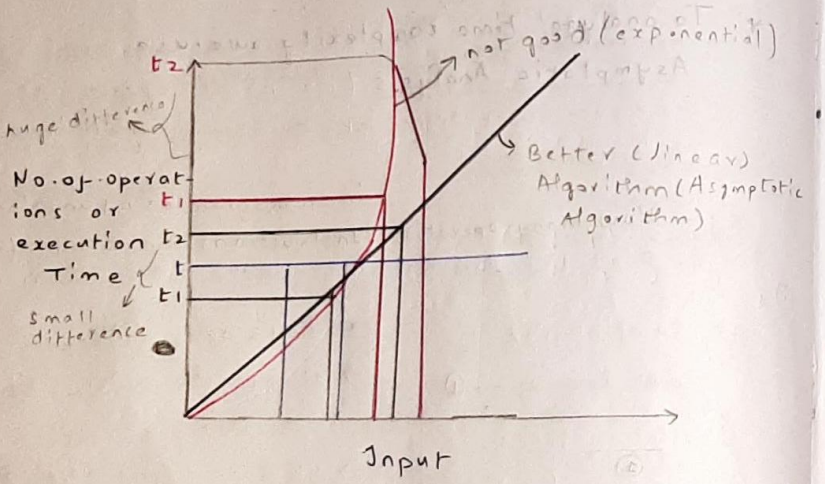
return ans;
⑤ ①

}

≅ 5 operations

for big Input i.e n=10^8 int can't store that amount so use long.

→ For any Input i.e n=5, n=10, n=100, n=10^8, n=10^16 the no of operations are 5 only (for this type of Algorithm we can say that it is running in constant time because for any Input the no of operations which are performed are only 5. (they doesn't depend on input the running time is constant) so it is called as constant time Algorithm.



* If Input changes in blackline (linear) the running time obviously increasing (only some)
* but for red line if Input changes the running time is increasing more drastically
* (The Algorithm for which the growth in Input is less compared to the Algorithm for which the Input increases, running time increases drastically) the 1st Algorithm is called better Algorithm.
* for any Algorithm the running time is constant called as constant time Algorithm. — blue line in graph

* Types of Time complexity Analysis & their Notations -

→ Worstcase Time Complexity → O(n) (3n+3) from below question
→ Bestcase Time Complexity → O(1)
→ Average Case Time Complexity → O(n) (n/2, n/3 ..)

Question - Find the index of number x in a given array Arr.

arr	1	5	6	8	3

x = 1

```
static int findIndex (int arr[], int x){
    int n = arr.length;
    for(int i=0; i<n; i++){
        if (arr[i] == x){
            return i;
        }
    }
}
```

→ If x=1 only one Iteration/operation (It is best case)
If x = 2/3 more operations (worst case)
If x = 5/6/3 Avg operations (Avg case)

* Representations

→ Worst case Time complexity → Big O i.e

O(n) (from previous que Tc)

→ Best case Time Complexity → Big Omega i.e

Ω(1) (from prev que Tc)

→ Average Case Time Complexity → Big theta i.e

Θ(n) (from prev que Tc)

* From question ① sum of all Numbers from 1 to n Time Complexity is O(n)

from/by using formula the Time Complexity is O(1) constant time Algorithm

* On different practicing platforms (leetcode etc) they will given as Timelimit → 1 sec

or

Timelimit → 2 sec

etc.

* In 1 sec the machines / platforms till today date will perform $10^8 \sim 10^9$ operations

* So for timelimit 1 sec if it is given in any question Assume

1 sec → 1×10^8 maximum operations

2 sec → 2×10^8 operations

* one common error we will face i.e TLE (Time limit exceeded) which means in 1 sec it is performing more than 10^8 operations.

Question -

* Time Complexity for traversing an Array of length n.

```
Void f1(int[] arr){
    int n = arr.length;          ①
    for(int i=0; i<n; i++){      ① ① ①
        System.out.println(arr[i]);
                                 ①
    }
}
```

In one iteration 3 operations

③

for 1 itteration 3 operations

for n itterations there will be 3n operations

i.e 3n + 2

Hence we ignore small terms ∴ answer is n operations

Time Complexity → O(n) (worst case)

* Time complexity when traversing 2 individual Arrays of length M and N respectively.

```
Void f2(int[] arr1, int arr2[]){
    int n = arr1.length;         1
    int m = arr2.length;         1
    for(int i=0; i<n; i++){      1
        cout(arr1[i]);
    }                            3n ~ n
    for(int i=0; i<m; i++){      3m+3 ~ n
        cout(arr2[i]);
    }                            3m ~ m
}
```

Time Complexity:

O(n+m)

* Time Complexity for Nested Loops

```
Void f3(int n){
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            Sout("Hello");
        }
    }
```

for every iteration there are n iterations

eg - n = 3

i = 0
- j = 0 Hello
- j = 1 Hello
- j = 2 Hello

so on

1 i iteration → 3 operations (3 = n)

n i iterations → n x n operations

∴ n x n iterations

Time Complexity → $O(n^2)$

* Time complexity of Nested loop type 2

```
Void fy(int n){
    for(int i=0;i<n;i++){
        for(int j=0;j<i;j++){
            Sout("Hello");
        }
    }
```

n = 3

for every iteration there is upto i iteration

i = 0

i = 1
- j = 0 Hello

i = 2
- j = 0 Hello
- j = 1 Hello

i = 3 , j = 0, 1, 2

i = 4 , j = 0, 1, 2, 3

i = n , j = 0, 1, 2, ... n-1

count the no. of operation - 1+2+3+...(n-1)

Total operation - $\frac{n(n-1)}{2}$

i.e

$$\frac{(n-1)(n-(+1))}{2} = \frac{n^2}{2} - \frac{n}{2}$$ constants

Ignore low degree values i.e

$n^2 - n \Rightarrow n^2$ (because from n^2 comparision n do not have any value)

i.e $n = 10^5$

$(10^5)^2 - 10^5 \simeq (10^5)^2$

Kimath

Time Complexity → $O(n^2)$

* Time Complexity for Nested Loop type 3

```
Void f(int n){
    for(int i=0;i<n;i++){
        for(int j=0;j<Math.sqrt(n);j++){
            Sout("Hello");
        }
    }
```

for every iteration there are $\sqrt{n}$ iterations

	j
i = 0	$\sqrt{n}$
i = 1	$\sqrt{n}$
i = 2	$\sqrt{n}$
i = 3	$\sqrt{n}$
:	
i = n-1	$\sqrt{n}$

i.e $n \times \sqrt{n}$ (for n iteration → $\sqrt{n}$ operations)

$O(n \times \sqrt{n})$

* Time complexity for traversing the Array and Multiply the Increment point by 2

```
int count = 0
for(int i=1;i<n;i*=2){
    count++;
}
```

$O(k) \sim O(\log n)$

Iteration No (K)	i	
1	1	$2^0 \to 2^{1-1}$
2	2	$2^1 \to 2^{2-1}$
3	4	$2^2 \to 2^{3-1}$
4	8	$2^3 \to 2^{4-1}$
5	16	$2^4 \to 2^{5-1}$
6	32	$2^5 \to 2^{6-1}$
:	:	:
K^{th}		2^{K-1}
$(K^{th}+1)$		$2^{(K+1)-1} \simeq 2^K$

Total No of operations $(K+1)$

Time complexity $\to O(K+1) \sim O(K)$ (O(No of Iterations))

→ From code - we have to represent K in terms of N because we dont have any K value in code

$i < N$ (from question condition)

$\therefore 2^K < N$

$\log_2(2^K) < \log_2(N)$

$K \log_2 2 < \log_2 N$

$K \cdot 1 < \log_2 N$

$\log_a x^n = n \log_a x$

$\log_b b = 1$

$\log_a b = \frac{\log_c b}{\log_c a}$

$\therefore$ Answer $\to O(\log_2 n) \sim O(\log N)$

Time complexity.

* calculate the time complexity for below Nested loops:

```
int val = 0;
for (int i = 1; i <= N; i *= K)
    val++;
}
```

Step1- find Total no. of Iterations, In code the values are N, K the Answer should be In terms of O(N, K)

Iteration No	i	(i = i * K (from question)
1	$1 = K^0$	K^{1-1}
2	K	K^{2-1}
3	K^2	K^{3-1}
4	K^3	K^{4-1}
5	K^4	K^{5-1}
6	K^5	K^{6-1}
:	:	
P^{th}	K^{P-1}	K^{P-1}
$(P+1)^{th}$	K^P	K^{P+1-1}

Total no. of Iterations

$O(P+1) \simeq O(P)$

From code we have to represent P in terms of (N, K)

$i \leq N$ (from question condition)

$K^P \leq N$

$\log_K K^P \leq \log_K N$ (we are taking base K because so that the Ans should be 1 i.e $\log_K K = 1$)

$P \log_K K \leq \log_K N$

$P \cdot 1 \leq \log_K N$

$P \leq \log_K N$

Time complexity $\to O(\log_K N) \simeq O(\log N)$

* Some common Time complexities are -

→ O(1) constant

→ O(log n)

→ O(n) linear

→ O(n log n)

→ O(n²) ⟶ (if n=4 operations = 16
n=5 operations = 25
n=20 operations: 400
n=30 operations: 900
if Input Increases time taken Increases rapidly)

→ O(n³)

→ O(n^n)

Space Complexity

* If an Algorithm running time is less and it is having minimum no of operations said to best Algorithm Inefficient Algorithm using Asymptotic Analysis we can determine

* Algorithm 1 - using 5mb extra space
Algorithm 2 - using 500mb extra space } both are referring to same problem but in one logic 5mb space and another logic 500mb so space wise Algorithm 1 is better than Algorithm 2

Def:

* The extra memory space used by an Algorithm ∝ proportional to Input size

* Input & o/p is not considered in space complexity Irrespective of the logic we make which is having extra space. (So we are using the term extra in definition)

* It is also calculated using Asymptotic Analysis & It Also have Average, worst, Best case Space complexity which is represented as θ, O, Ω

Question: Reverse an Array

Approach 1 -

```
static int[] reverseArray(int[] a, int n){
    int[] Ans = new int[n];
    int j = 0;
    for(int i = n-1; i >= 0; i--){
        Ans[j] = a[i];
        j++;
    }
}
```

* In this logic we are making an Ans array of size n so Space complexity : O(n)

Approach 2 -

```
static void reverseArray(int[] a, int n){
    int i = 0, j = n-1;
    while(i < j){
        swap(a[i], a[j]);
        i++;
        j--;
    }
}
```

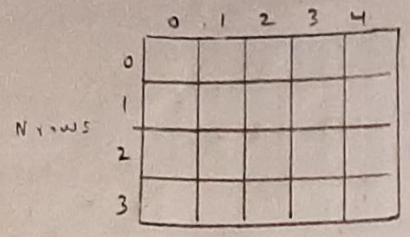
* In this logic we are changing In-place only we are not using extra space so Space complexity : O(1) (constant space)

* The extra space that is used called as Space Complexity.

* Space Complexity Analysis for an Array of length N. (Suppose extra Array of length N)

Space complexity - O(N) (reverse Array Approach :)

* Space Complexity Analysis for a 2-d Matrix of N rows and M columns. (Suppose extra extra of size N x M)



N = 4
M = 5
4 x 5
N x M = 20

Space complexity. O(N x M) ← No of Iterations
↳ with respect to Input size

* By default if anyone Asks the Time/space complexity of the given code we have to answer worst case only.

Problems on Time & Space Complexity

Problem - Calculate the time complexity for the following code snippet.

```
int val = 0
for(int i=1; i<=N; i*=2){
    val++;
}
```

Iteration No	i	i=i*2
1	1	2^0
2	2	2^1
3	4	2^2
4	8	2^3
5	16	2^4
6	32	2^5
7	64	2^6
⋮	⋮	
K	2^{K-1}	
K+1	$2^{K+1-1} = 2^K$	

Total Iterations = (K+1)

Time complexity - O(No. of Iterations/operations)

T.C → O(K+1) ≃ O(K)

Input is in terms of N so, we have to write K in terms of N

→ Acc. to question

$i \le N$

$2^K \le N$

Apply $\log_2$ on B.S base 2 because $\log_2 2 = 1$

i.e $\log_2 2^K \le \log_2 N$

$K \log_2 2 \le \log_2 N$

$K \cdot 1 \le \log_2 N$

$K \le \log_2 N$

i.e

Time complexity → $O(\log_2 N)$

base is inter, so ≃ convertable $O(\log N)$

Problem - Calculate the time complexity for the following code snippet

```
int val = 0
for(int i=1; i<=N; i=i*2){
    val++;
}
```

Iteration No	i
1	2^0
2	2^1
3	2^2
4	2^3
5	2^4
6	32
7	64
⋮	
K	2^{K-1}
(K+1)	2^{K+1-1}

Total Iterations = (K+1) ≃ K

T.C → O(K)

we have to represent K in terms of N

$i \le N$

$2^K \le N$

$\log_2 2^K \le \log_2 N$

$K \le \log_2 N$

Time complexity → O(log N)

(i*=2 & i=i) both are Same

Problem - Calculate the time complexity for the following code snippet.

```
int val = 0
for(int i=1; i<=N; i*=2) {
    for(int j=1; j<=i; j++) {
        val++;
    }
}
```

(range) How many times j loop will run

i-Itteration No	i-val	j[ ]	j - Itterations Total
1	1	[1, 1]	1
2	2	[1, 2]	2
3	4	[1, 4]	4
4	8	[1 to 8]	8
5	16	[1 - 16]	16
:	:		
K	2^{K-1}	$[1 - 2^{K-1}]$	2^{K-1}
K+1	2^K	$[1 - 2^K]$	2^K

→ Calculate the Total No. of Itterations

* We know that for every i value how many times j will run
* So we sum all the j total values we will get the total no of Itterations
* So we have to focus for, value i how many times j will run (focus on inside loop)
* So add j values to find total Itterations

we know that K value = $\log_2 N$

from outer loop i.e. $O(K+1) \simeq O(K)$

$i \le N$

$2^K \le N \Rightarrow \log_2 2^K \le \log_2 N$

$K \le \log_2 N$

So $K = \log_2 N$

$\therefore 2^K = 2^{\log N}$

i.e.

Add all the j values for Total Itterations

→ $1 + 2 + 4 + 8 + 16 + \dots 2^K$

Total K+1 terms i.e. $\simeq K$

$2^0 + 2^1 + \dots 2^{K+1}$

→ The series is of Geometric progression

G.P sum formula is

$$\frac{a(r^n - 1)}{r - 1}$$

where, a → first term

r → common ratio ($\frac{4}{2} = 2$, $\frac{8}{4} = 2$, $\frac{16}{8} = 2$)

n → total terms

$$\frac{1(2^K - 1)}{2 - 1} = \frac{1(2^{\log_2 N} - 1)}{1}$$ i.e. $K = \log_2 N$

$= 2^{\log_2 N} - 1$

$= \log_2 N \; 2$

$= N$

Time complexity → O(N)

[ ) [→ Inclusive,) → exclusive

eg N = 5, i = 1, j = 5, 4, 3, 2

Problem - what is the time complexity for the following code

```
int val = 0
for(int i=1; i<=N; i*=2) {
    for(int j=N; j>i; j--) {
        Val++;
    }
}
```

A) O(logN)

B) O(N)

✓ C) O(N logN)

d) None

Itteration no	i value	j[ ) range	Total j Itterations
1	1 2^0	[N, 1)	(N-1)
2	2 2^1	[N, 2)	(N-2)
3	4 2^2	[N, 4)	(N-4)
4	8 2^3	[N, 8)	(N-8)
.			
.			
(K+1)	2^K	$[N, 2^K)$	$(N - 2^K)$

→ for Total no.of iterations i.e $K+1$

$O(K+1) \cong O(K)$

$i \leq N$

$2^K \leq N$

$\boxed{K \leq \log_2 N}$

i.e outer loop runs $\log N$ times

→ we have to add all the inner value(j) iterations to get the total no.of iterations

$(N-1)+(N-2)+(N-4)+\cdots(N-2^K)$

Total No of iterations $= K+1 \cong K$

So if total No of iterations are K the value N is also written K times i.e.

$KN - \underbrace{(1+2+4+8+\cdots 2^K)}_{GP}$

$KN - \frac{a(r^n-1)}{r-1}$

$KN - \frac{1(2^K-1)}{1}$

$KN - 2^{\log_2 N} - 1$

$N(\log_2 N) - N - 1$

we can ignore 1 & N with comparision of $N\log N$

i.e $(N\log_2 N)$

Time Complexity → $O(N\log N)$

Problem - what is the time complexity of the following code

```
int val=0
for(int i=N; i>0; i/=2) {
    for(int j=0; j<i; j++) {
        val++;
    }
}
```

A) $O(\log N)$ B) $O(N)$ c) $O(N\log N)$ d) None

range → it can also be written as $[0, N)$

Iteration No	i value	j[]	Total Iterations
1	N	$[0, N-1]$	N
2	$N/2$	$[0, \frac{N}{2}-1]$	$N/2$
3	$N/4$	$[0, \frac{N}{4}-1]$	$N/4$
4	$N/8$	$[0, \frac{N}{8}-1]$	$N/8$
⋮	⋮	⋮	
K+1	$N/2^K$	$[0, \frac{N}{2^K}-1]$	$\frac{N}{2^K}$

* $i>0$ means $i \geq 1$

$\frac{N}{2^K} \geq 1$

$N \geq 2^K$

$\log_2 N \geq \log_2 2^K$

$\log_2 N \geq K$

$\boxed{K \geq \log_2 N}$

* Sum of all Inner loop values to get the total no of iterations

$N + \frac{N}{2} + N/4 + N/8 + \cdots \frac{N}{2^K}$

$N\underbrace{\left(1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots\frac{1}{2^K}\right)}_{GP}$ n = Total no of terms $K+1 \cong K$

$N\left(\frac{a(r^n-1)}{r-1}\right) \Rightarrow N\left(\frac{1\left(\frac{1}{2}^K-1\right)}{\frac{1}{2}-1}\right)$

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

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

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

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

$$N\left(\frac{1}{N} - 1\right)$$

$$\frac{N}{N} - N = 1 - N \simeq N$$

$2^{\log_2 N} = N$

i.e

$\log_2 N$ 2

$\therefore N = N$

Time Complexity - $O(N)$

Problem - Calculate the time complexity for the following code snippet

```
int val = 0
for (int i = 2; i <= N; i = i * i) {
    val++;
}
```

Iteration No.	i value	
1	2	2^1 2^{2^0}
2	4	2^2 2^{2^1}
3	16	2^4 2^{2^2}
4	256	2^8 2^{2^3}
	:	
	.	
(t+1)	2^K	$2^{2^{(t+1-1)}} \Rightarrow 2^{2^t}$

Total Iterations (t+1)

Time complexity $O(t+1) \simeq O(t)$

we have to represent t in terms of N

$i \leq N$

$2^K \leq N$

$\log_2 2^K \leq \log_2 N$

$\boxed{K \leq \log_2 N}$

$K = 2^t$ ← ② ^{K} = ② $^{2^t}$

$\log K = \log 2^t$

$\log_2 K = \log_2 2\ (t)$

$\boxed{\log K = t}$

we know that $K = \log N$ & $t = \log K$

i.e

$t = \log(\log N)$

Time complexity → $O(\log(\log N))$