Complexity Analysis

Complexity : A factor involved in complicated process or situation.

time space

If we have written a code / solution to a problem, we want to figure out how good or bad is the solution.

⇒ how good or bad wrt time and space

Time Complexity

→ Can you tell the time complexity of code you have written?

→ Can you improve the complexity?

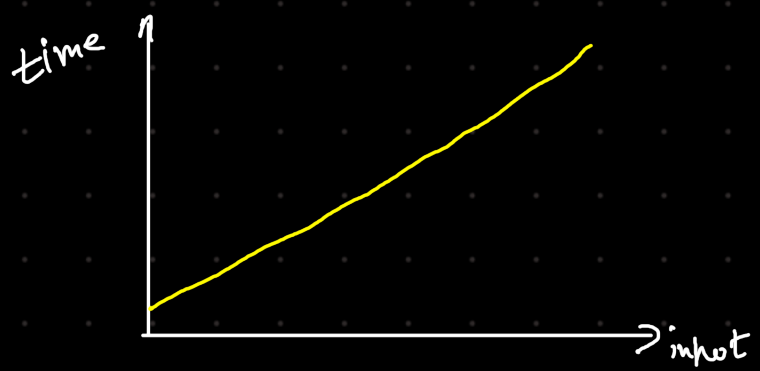
$$A_{algo} \longrightarrow T(n)$$

Experimental analysis

1. different times on diff run
2. few changes, time can be different.
3. time actually varies with input, but we are not getting or establish a relationship.
4. Generate test cases.
5. check for each and every implementation
6. Large inputs are very time consuming.

Sorting	Old	New
10	5	1
100	10	2
1000	15	3
10000	20	4

Time complexity ≠ Time taken



Quantifying the Time Complexity

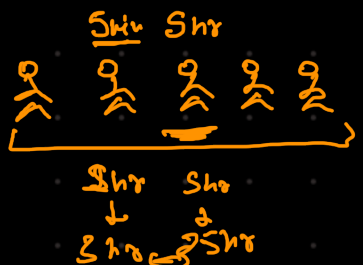
Few points :-

1. we want time complexity when input is very large.

2. worst test case

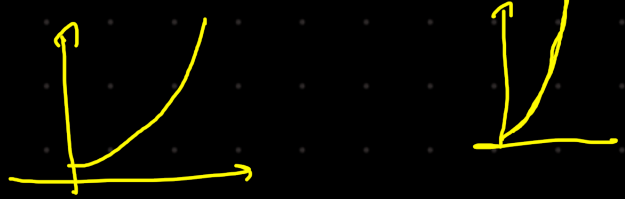
3. We want to look for largest factor in the run time.

   nested loop > single loop ↓



4. We want to express runtime as input size and we don't want to look for precision but 'order of' works

$5n^2 <<< 5n^3$



Bubble sort : $\frac{20n^2}{2} + 5n + 1$

we are fine with most dominant term : $20n^2$

⇒ no. of unit operation $+, -, >, \div, ==$

3 major things :-

1. Talk in terms of no. of operation and not time.
2. Focus only on the highest power.
3. Don't care about coefficient much.

Asymptotic notation

- main idea of our analysis is to do have a measure of the efficiency of algorithm that don't depend on

  → machine specific constants
  → require algorithm to be implemented.

Asymptotic notation are mathematic tools to represent time complexity of our algorithm.

There are mainly 3 types of asymptotic notation:

1. Big-O notation (O notation, worst case)
2. Omega notation (Ω - notation, best case)
3. Theta notation (Θ - notation, avg case)

insertion sort merge

Big O complexity

3 major things :-

1. Talk in terms of no. of operation and not time.
2. Focus only on the highest power.
3. Don't care about coefficient much.

Bubble sort : $20n^2 + 15n + 2$

$f(n)$

Algo $f(n) \longrightarrow O$ complexity

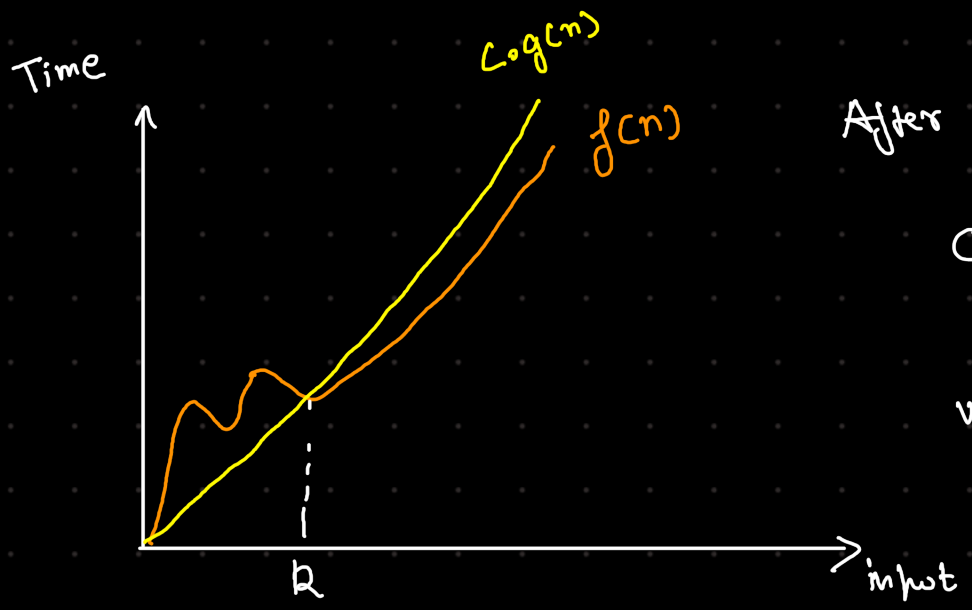
$f(n) \leq c^* g(n)$

$O(g(n))$

$f(n) = 2n^2 + 2 \leq c * g(n)$

$c \rightarrow 3$, $g(n) \rightarrow n^2$

$2n^2 + 2 \leq 3n^2$ $\quad O(n^2)$



After $n \geq R$

$C \cdot g(n) \geq f(n)$

Upper bounds my function.

Big Omega (Ω)

→ Best case scenario

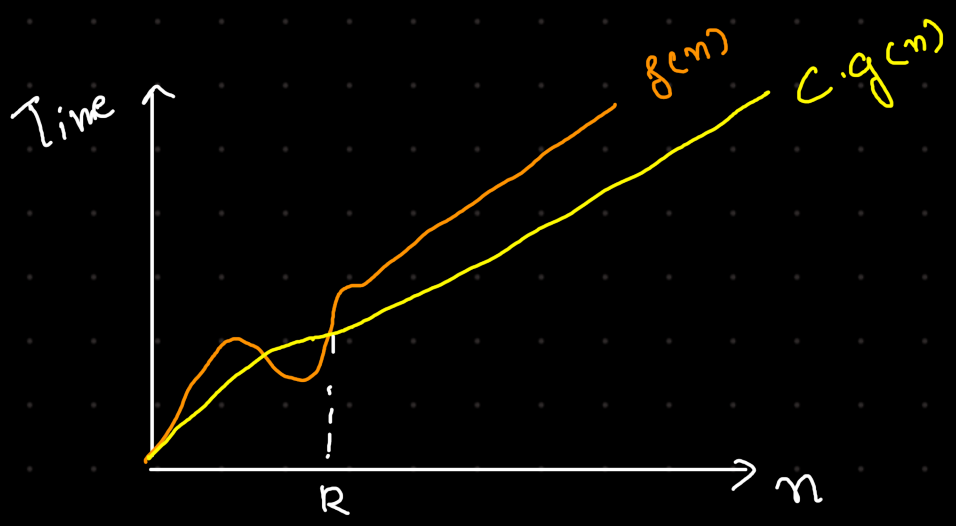
$f(n) = \Omega(g(n))$

when $f(n) \geq c \cdot g(n)$

$$2n^2 + n \geq \frac{c}{2} \cdot n^2$$

$$2n^2 + n \geq 2n^2$$

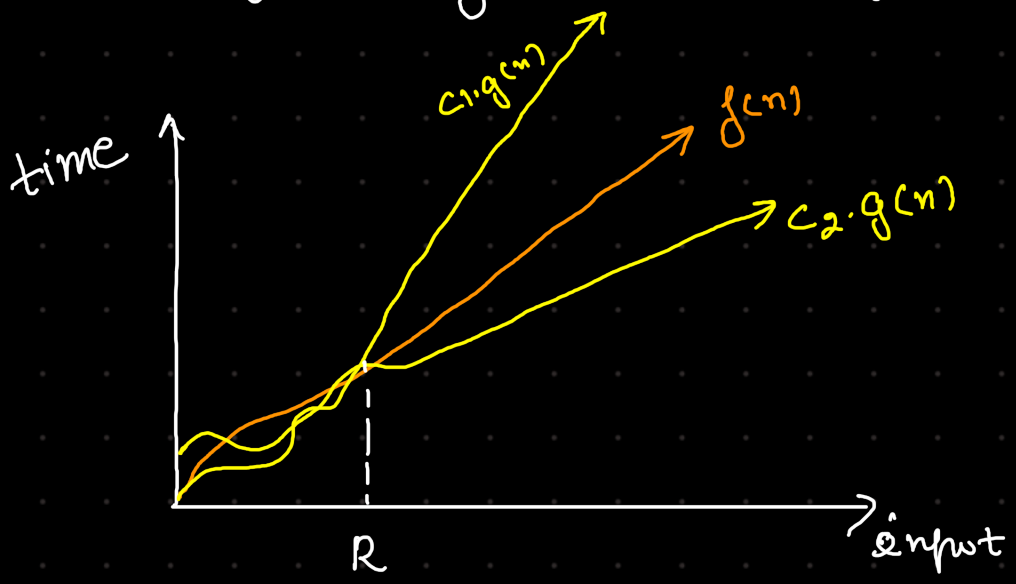
$$n \geq 0$$



Theta (Θ)

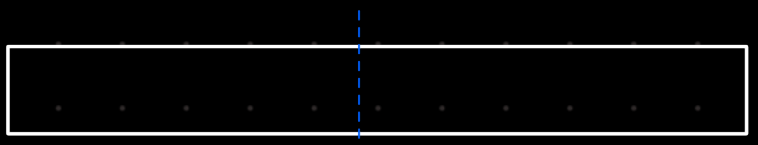
→ Average case complexity

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$



Time complexity ≠ Time taken

Middle of our list.



$n = 10^3$

$n = 10^6$

$n = 10$

mid = n // 2

$$k\, n^0 \leq (k+1)\,\underbrace{(n^0)}_{g(n)}$$

Complexity is O(1)

Largest element in an array

max = a[0]

for i → n
  if a[i] > cmax
    cmax = a[i]
} R operations

1	2	3	. . .	n
R	R	R		R

$\leq = nR \leq c \cdot g(n)$, $(R+1) \quad n \to \infty$

$$O(g(n)) = O(n)$$

Linear Complexity

Bubble Sort

for 1st element $(n-1)R$

for 2nd element $(n-2)R$

for 3rd element $(n-3)R$

2nd last element R

$= R(n-1) + R(n-2) + R(n-3) \cdots \cdots R$

$= R\,[\,[n-1] + [n-2] + [n-3] + \cdots \cdots 1\,]$

$1 + 2 + 3 \cdots n = \frac{n(n+1)}{2}$

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

$\frac{R(n)(n-1)}{2} = R\frac{n^2}{2} - \frac{nR}{2}$ → O

$= \frac{Rn^2}{2} \leq \underset{\left(\frac{R}{2}+1\right)}{C} \cdot g(n)$ $\quad g(n) = n^2$

$g(n) = n^2$

$O(n^2)$

We will have complexity for bubble sort as $O(n^2)$

which is quadratic time complexity.

Insertion sort

for 1st element - 0

for 2nd element - R

for 3rd element - 2R

⋮

n^{th} - $(n-1)R$

$1 + 2 + \cdots + (n-1)$:

$$\frac{(n-1)(n)R}{2} = \frac{Rn^2}{2} - \frac{Rn}{2} \to 0$$

$$= R\frac{n^2}{2} \leq \underset{\downarrow}{C} \cdot \underset{\downarrow}{g(n)}$$

$$\left(\frac{R}{2}+1\right) \quad n^2$$

Complexity is $O(n^2)$ i.e. quadratic

Best case (Ω) time complexity of insertion sort

1 element - 0
2 element - R
3 element - R
4 element - R
⋮
n element - R

$$R + R + R \ldots (n-1)$$

$$R(n-1) = Rn - R \leq (R+1)(n)$$

R is 0

$O(n)$ is the best case complexity for insertion sort

Selection Sort

Time complexity : Recursive algorithm

for recursive algo, we use a diff. approach

- Recurrence relation method

Factorial of a number

```
def fact(n):
    if (n == 0):
        return 1          } R1

    return n * fact(n-1)
           R2
```

$$fact(n) = n * fact(n-1)$$

$$T(n) = \underbrace{R_1 + R_2}_{R} + T(n-1)$$

$$T(n) = k + T(n-1)$$

$$T(n-1) = k + T(n-2)$$

$$T(n-2) = k + T(n-3)$$

$$\vdots$$

$$T(0) = k$$

$$a = b + 5$$

$$b = 5$$

$$a + b = b + 10$$

$$-b \quad -b$$

$$a = 10$$

$$T(n) = k + k + k \quad \ldots \quad (n+1) \text{ times}$$

$$T(n) = k(n+1)$$

$$= kn + k$$

$$= kn \qquad \leq (k+1)\, n$$

Time complexity of factorial is $O(n)$ i.e. linear

Binary Search

$T(n) = R + T(n/2)$ Problem gets half

$T(n/2) = R + T(n/4)$

$T(n/4) = R + T(n/8)$

$\vdots$

$T(1) = R$ ← Some const work (base case)

$T(n) = R + R + R \cdots$? How many times

$n \quad \frac{n}{2} \quad \frac{n}{4} \quad \frac{n}{8} \quad \frac{n}{16} \quad \cdots \quad 1$?

Say x times to reach 1

$\frac{n}{2^x} = 1$

$n = 2^x$

$\log_2 n = x \log_2 2$ ($\log_2 2 = 1$)

$\log_2 n = x$

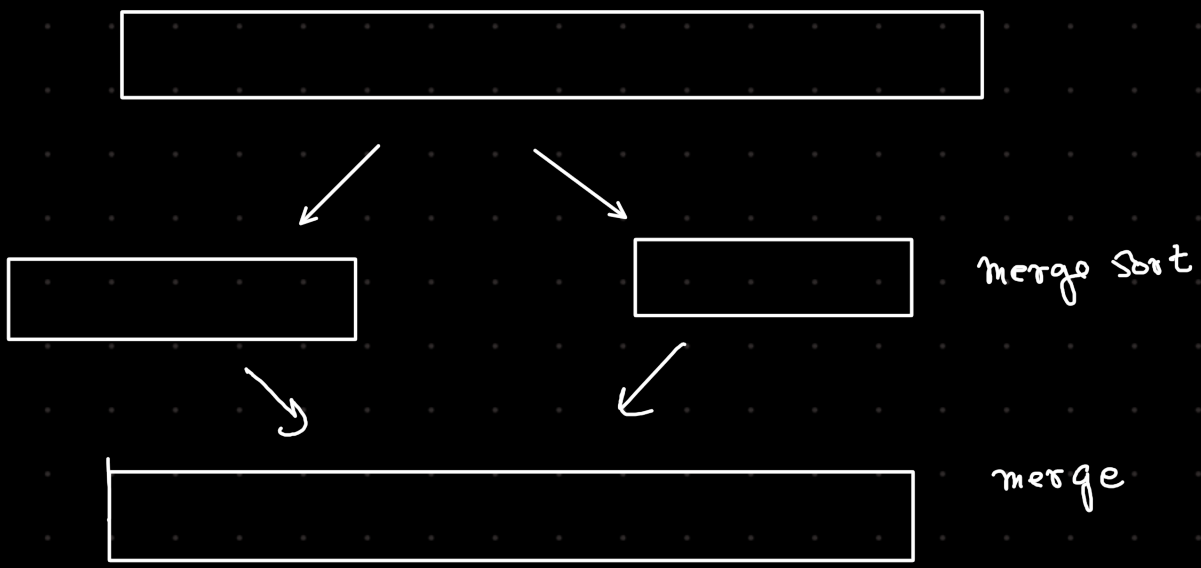
$\Rightarrow \quad k + k \quad \ldots \quad \log_2 n$

$\Rightarrow \quad k * \log_2 n \quad \leq (k+1)(\log_2 n)$

Time complexity of binary search is $O(\log_2 n)$

10^6 elements : 20 comparision

Merge Sort



$$T(n) = R_1 + 2T(n/2) + R_2 n$$

$$= \cancel{R_1}^{0} + 2T(n/2) + R_2 n$$

$$T(n) = Rn + 2T(n/2)$$

$$2T(n/2) = R\frac{n}{2} + 2T(n/4) \quad *2$$

$$4T(n/4) = \frac{Rn}{4} + 2T(n/8) \quad *4$$

$$*8$$

$$\vdots$$

$$T(1) = R$$

$$T(n) = Rn + Rn + Rn \ldots \quad x \text{ times}$$

$T(n) = Rn + Rn + Rn \quad \ldots - \quad x \text{ times}$

$n \quad \frac{n}{2} \quad \frac{n}{4} \quad \ldots \quad 1$

$\frac{n}{2^x} = 1 \qquad x = \log_2 n$

$T(n) = R\, n \log_2 n \quad \leq \quad (R+1)\,(n \log_2 n)$

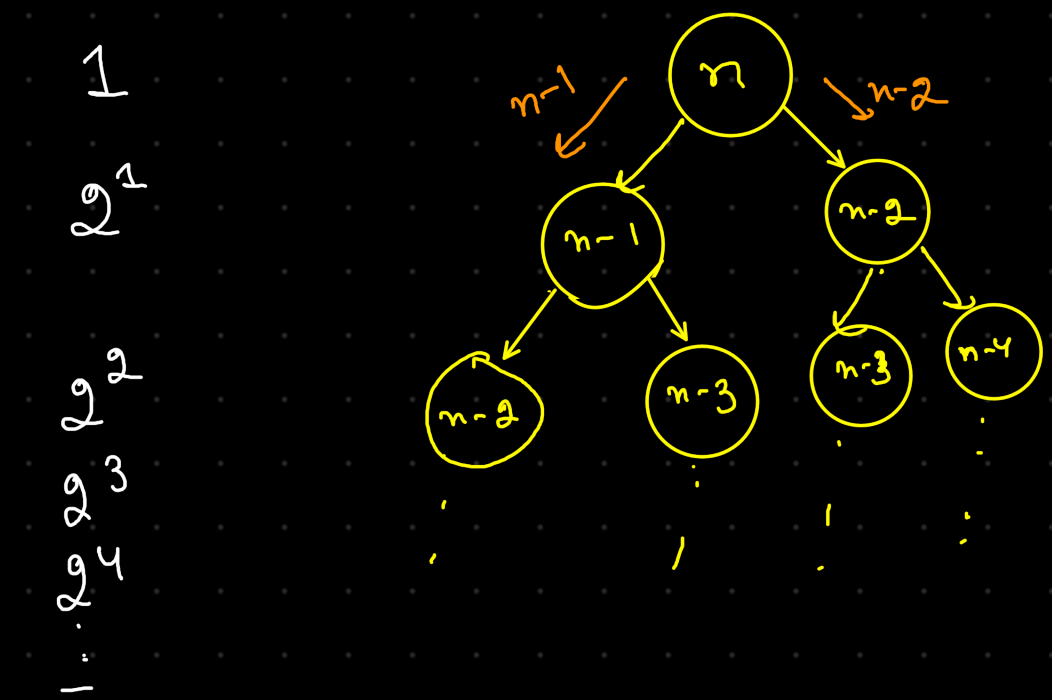
$O(n \log n) \Leftarrow$ worst, avg, best

Fibonicci Number

```
def fib(n):
    if (n <= 1)
        return 1
```
} R

```
return fib(n-1) + fib(n-2)
```

$T(n) = R + T(n-1) + T(n-2)$



R = no. of nodes

* We are assuming that each subtree will in worst case end at n depth / bottom most depth

$1 + 2 + 4 + 8 + 16 \cdots$

$1 + 2^0 + 2^1 \cdots n$

$\frac{2(2^n - 1)}{2 - 1}$ = Sum of G.P.

R * (no. of nodes)

$R * \{ 2 [2^n - 1]]$

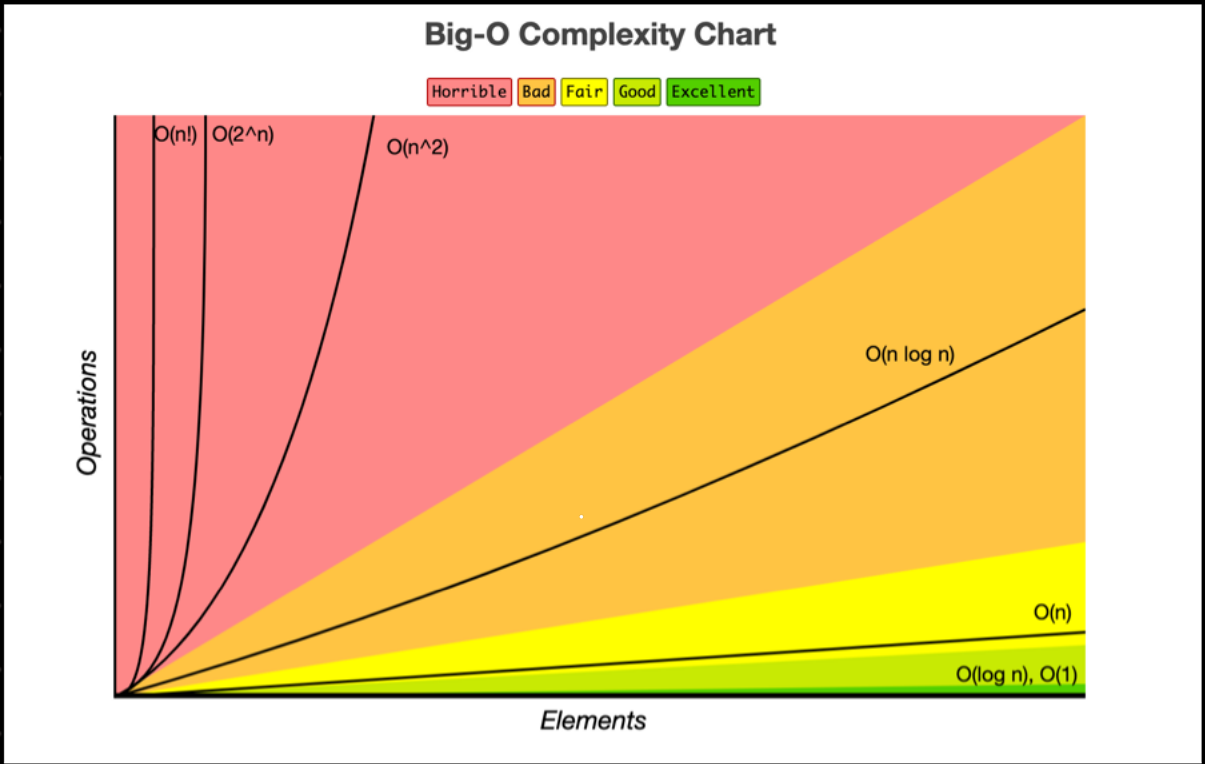
$2^{n+1} R - 2R^0$ $\qquad 2 \cdot 2^n = 2^{n+1}$

$T(n) = R\, 2^{n+1}$

$= 2R\, 2^n \leq 3k(2^n)$

Complexity is $O(2^n)$, it is exponential

Visualizing Time Complexities



n	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$	$O(n!)$
10	3.32	10	33.22	100	1,000	1,024	3,628,80
100	6.64	100	664.39	10,000	1,000,000	Good Luck!	Good Luck!
200	7.64	200	1,528.88	40,000	8,000,000	Good Luck!	Good Luck!
500	8.97	500	4,484.12	250,000	125,000,000	Good Luck!	Good Luck!
1,000	9.97	1,000	9,966.57	1,000,000	1,000,000,000	Good Luck!	Good Luck!
5,000	12.29	5,000	61,434.69	25,000,000	125,000,000,000	Good Luck!	Good Luck!
10,000	13.29	10,000	132,877.12	100,000,000	1,000,000,000,000	Good Luck!	Good Luck!
20,000	14.29	20,000	285,783.99	400,000,000	8,000,000,000,000	Good Luck!	Good Luck!

While doing questions on leetcode, OA, codechef we face time limit exceeded.

Brute force normally not accepted.

⇒ no. of operation allowed per unit second

$\sim 10^8$ ops/s

In problems on internet, the same is also provided in question defn.

How to determine the solution by looking at constraint ?

Constraint	Worst time Complexity	Algo
$n \leq 12$	$O(n!)$	Backtracking recursion
$n \leq 25$	$O(2^n)$	Bit manip, recurs/Bachtr.
$n \leq 100$	$O(n^4)$	DP
$n \leq 500$	$O(n^3)$	DP
$n \leq 10^4$	$O(n^2)$	DP, graph tree
$n \leq 10^6$	$O(n \log n)$	Sorting, Divide & conquer
$n \leq 10^8$	$O(n)$	Mathematical Greedy
$n > 10^8$	$O(1)$ / $O(\log n)$	Mathematical, greedy

One more way to remember,

whatever algo we choose, if we plug in n we should get around $\sim 10^8$

Above is a lot helpful in remembering.