leetCode / Codechef

unit operations they allowed. ← (1s) = 10^8

Time Complexity → measures the total no. of basic (unit) operations a program performs as a function of input size n.

How to measure T.C ?

Program → Basic operation → collection of unit operation

- A single loop running n times = n unit ops.
- Also count the no. of ops or function calls, especially in recursion.

$\boxed{\text{T.C represented as : } O(1)}$

O ↳ Worst case representation

(1) ↳ constant K overall operations, when a program is executed. ($K <= 10^8$)

For exa : N → `for (int i=1, i<=N; i++) print (i);` $\boxed{T.C: O(n)}$

N ↑ user input

$T.C : \underset{\text{for } i=1}{1} + \underset{\text{assignment operator}}{n+1} + \underset{\text{inc. i (i++)}}{n} + \underset{\text{print (i)}}{n}$

(~~print~~)

$\therefore T.C = O(3n+2)$ [Ignore constant]

constant ↓ bcoz n is the changing factor. It can inc or dec.

$\therefore \boxed{T.C = O(n)}$

For $n \to 10^6$, $n \to 10^5$ } → Use $O(n)$ T.C.

For : $O(10n)$

↳ $n \approx 10^7$ (bcoz $10 \times n \le 10^8$)

↓ $n \approx 10^7$

Exa :

```
for (int i=1; i<=n; i++){
    for (int j=1; j<=n; j++){
        print (j);
    }
}
```

$\boxed{T.C : O(n^2)}$

↳ Max. value of n : (10^4)

bcoz $n^2 = (10^4)^2 = 10^8$

Exa:

```
for (i....)
  for (j....)
    for (k....)
```

→ $\boxed{\text{Overall T.C: } O(n \times n \times n) = O(n^3)}$

Exa:

```
for (i=1; i<=n; i++){
   for (j=i; j<=i; j++)
       print (j);
}
```

$\boxed{\text{T.C: } O(n^2)}$

Explanation:

i	j	
i = 1 :	1	→ 1 iterations
i = 2 :	1 2	→ 2 iterations
i = 3 :	1 2 3	→ 3 iterations
i = 4 :	1 2 3 4	
i = n :	1 ..... n	→ n iterations

No. of iterations $= \frac{1+2+\dots n}{(n+1)}$

$= O\left(\frac{n^2+n}{\cancel{2}}\right)$

$= O(n^2)$

* n^2 will cross overall limit of 10^8 faster than n.

Exa:

```
for (i=1; i<=N; i*=2){
      print (i);
}
```

$i = 1$
$i = 2$ (×2)
$i = 4$ (×2)
$i = 8$ (×2)
⋮
$i = 2^x <= N$

$\hookrightarrow 2^x = n$

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

$x \cdot \underbrace{\boxed{\log_2(2)}}_{1} = \log_2 n$

$x = \log_2 n$

$\therefore \text{T.C} = O(\log_2(n) + \cancel{1})$

$\boxed{\text{T.C: } O(\log_2(n))}$

$2^0 \text{ ------- } 2^x = (x+1)$ operations performed.

Exa:-

```
while (n>0){
    n/=2;
}
```

$n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \text{ ------ } 1$

$\frac{n}{2^0}, \frac{n}{2^1} \rightarrow \frac{n}{2^2}, \frac{n}{2^3} \text{ ---- } \frac{n}{2^x} = 1$

2^0
⋮
2^x

$\Rightarrow (x+1)$ operation performed.

$$\frac{n}{2^x} = 1 \Rightarrow 2^x = n \Rightarrow x = \log_2(n)$$

$\therefore$ $T.C : O(\log_2(n) + \cancel{1}) \Rightarrow \boxed{T.C : O(\log_2(n))}$

* $i \to i * k \quad (i <= n)$
  $n \to n/k \quad (n > 0)$

  $\therefore$ $\boxed{T.C : O(\log_k(n))}$

When we use $O(\log(n))$?

If $n > 10^8 \to$ We can't use $O(n)$.

$\therefore$ Whenever $n > 10^8 \to$ use $\log(n)$ or $O(1)$ complexity based algo.

Exa : Binary Search Algo. ($\log n$)

If $\log(n)$ inc.

↳ with $i = 2, 4, 8 ----$ $\Rightarrow$ then $O(\log_2(n))$

with $i = 3, 9, ----$ $\Rightarrow$ then $O(\log_3(n))$.