Time and Space Complexity:

Time Complexity: we can start from a question.

Q:
You're given a number x and y where ($1 \le x \le 10^5$, $x \le y \le 10^8$). calculate sum of all numbers in the range $[x, y]$.

Approaches: 1st: $x=5$, $y=8$ => $5+6+7+8=26$
2nd: iterate from x to y.

```
int tempP=0;
for (int i=x; i<y; i++) { temp=tem+i; } cout<<temp;
```

3rd: $x=2$ (first term), $y=6$ (last term) $[2,3,4,5,6]$ => A.P
we have formula: $S_n = \frac{n}{2}(2a + (n-1)d)$

No. of terms $= (y-x+1) = 6-2+1 = 5$

Experimental Analysis:

- Actual time taken by the code to get executed.
- we not take this analysis in consideration because time of execution varies.

No. of operations:

for $a_1 \Rightarrow$ $t_1 \rightarrow (x \times 10^5 ms)$ take ms to Perform 1 instruction.

$\boxed{m_1}$

$t_2 \rightarrow (x \times 10 ms)$

so, $t_2 < t_1$ (t_2 is good)

$a_1 \rightarrow 10^5$ operations / instructions

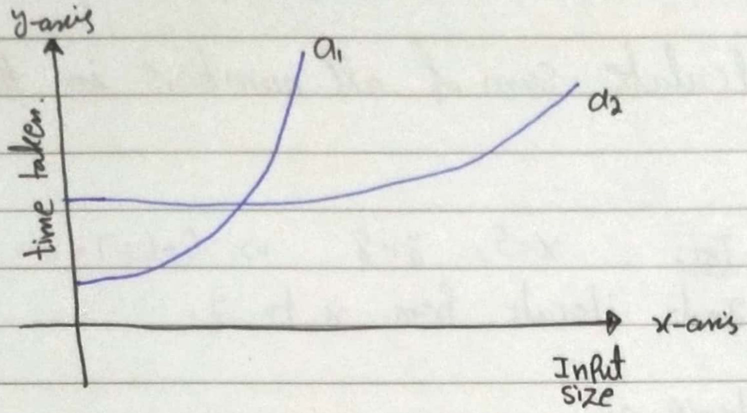
$a_2 \rightarrow 10$ operations / instructions

Date 12/06 2024
M T W T F S S

↳ Before that we check algorithm on the basis of # of operations.

Asymptotic Analysis:-

Basic concept:
No. of operations w.r.t change in input.



$a_1 \to$ first algorithm

$a_2 \to$ 2nd algorithm

* As a_1 → At small input → run fast (time taken less)
but
→ At large input → (time consumption exponentially increase)

But
As a_2 → At small input → run slow
→ At large input → (time taken is still considerable)

~~Bas~~ foundation concept:
↳ By changing the input size, how the time taken by algorithm is changes.

↳ For time taken, we consider the # of operations.

For Normal computers:
1 sec → $10^8 \simeq 10^9$ instructions execute.

Example: ① Not optimized

In the worst case, execute n-instructions.

```
int sum_range(int x, int y)        →  n = y - x + 1
{                      ①
   int result = 0;  ① - opt.            for loop ⇒ n-iterations.
   for (int ② i = x; i <= y; i++) ③ → increment;    ⇒ 3n instructions execute.
   {                 ② → add "i"                      So,
      result = result + i;                             total instructions ⇒ 3n + 3.
   }
   return result ③
}
```

So, In given Problem on Page-5

$x \to$ minimum 1, $y \to$ maximum $\to 10^8$

⇒ So, $n \cong 10^8$ ⇒ from $3n+3$ ⇒ $\boxed{3 \times 10^8 + 3}$ instructions.

So,

$3 \times 10^8 + 3 \cong 3n + 3 \cong n$

this also be neglected — lower term neglected

∴ By changing input, no. of iterations increases and with no. of instructions, not remain constant.

Fact:

↳ In Asymptotic analysis, lower degree terms which not put a large effect on instructions of algorithms, So, they can be neglected.

any value (does not impact)

Example: 2 → optimized. → $x=1$, $y=10^8$

```
int optimized_sum(int x, int y)
{   int n = y - x + 1;
    int a = x;
  int result = (n * (2*a + (n-1)*1)) / 2;
    return result;
}
```

≅ 10 instructions

∴ By changing input of x and y, But no. of instructions remains (10) constant.

Compare algorithms of example ① and ②, So, example ② is best.

Date ______ 20__
M T W T F S S

Summary:
w.r.t input, we see how much change is taken place in # of instructions.

we always care about Big input values. (In that small constants does not matter)

$\therefore$ Growth $\Rightarrow \dfrac{\Delta \text{ \# of instructions}}{\Delta \text{ input size}}$

Algorithm having slow growth, will Perform better.

So,

# of instructions	
const	$\rightarrow$ fast
$\log n$	$\rightarrow$ little fast
$\sqrt{n}$	$\rightarrow$ slow
n	$\rightarrow$ more slow
$n \log n$	$\rightarrow$ more slow
n^2	$\rightarrow$
n^3	$\rightarrow$
2^n	$\rightarrow$

In example-2 our algorithm is constant, So, it is extremely fast.

Example-1, algorithm is in terms of n, So, this will be Slow.

Types of time Complexity and their Notations:

1) Worst Case $\rightarrow$ Big O $\rightarrow$ O e.g. $O(n)$ — w.r.t change in input $\rightarrow$ # of instructions change in $n, n^2, n\log n$, etc.

2) Best Case $\rightarrow$ Big omega $\rightarrow$ Ω e.g $\Omega(1)$ — for constant Instructions.

3) Average Case $\rightarrow$ Big theta $\rightarrow$ Θ e.g $\Theta(n)$

* It is the extra memory space requirement of algorithm.

Three steps to calculate time complexity.

1) T.C , worst case scenerio
2) avoid constants.
3) aviod lower values.

Space Complexity:

use any extra variable or space ⇐ Auxilary Space ⟹ space that you take to solve the Problem.

\+

Input Space ⟹ the space take you take to store the input

e.g.

a b

c = a+b