Time Complexity

Content

- Basics
- Count of factors
- Big O
- TLE
- Bitwise Operators

Rules

1> Private chat to answer.
2> Question tab to ask questions
3> Quizzes don't answer in chat.
4> Be patient.

Sum of first n natural numbers ?

$$S = 1 + 2 + \ldots\ldots N-1 + N$$
$$S = N + N-1 + \ldots\ldots + 2 + 1$$
$$2S = \underbrace{(N+1)+(N+1)\ldots\ldots+(N+1)+(N+1)}_{N \text{ times}}$$

$$2S = N*(N+1)$$
$$S = \frac{N*(N+1)}{2}$$

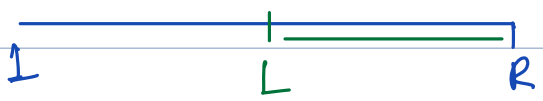
Numbers in range [3, 8]

3 and 8 are inclusive

3 4 5 6 7 8

6

[L, R]



R

L - 1

R - (L-1) = R - L + 1

[L, R] = # numbers R - L + 1

Factors

—— If i is a factor of N

$\frac{N}{i}$ there is no remainder.

How to programatically check

```
if N % i == 0
```

Count factors of N = 24

ans = 8

1 2 3 4 ~~5~~ 6 ~~7~~ 8 ~~9~~ 12 24

Count factors of N = 10

1 2 5 10 ans = 4

Given N

min factor = 1

max factor = N

Pseudocode to count factors of given N

```
count = 0
for i ——> 1 to N {
|     if (N % i == 0)  count++
}
return count
```

Lets say you submitted the above code.

1 sec to process 10^8 iterations {loop count}

$N = 10^8$	# iteration 10^8	time = 1 sec
$N = 10^9$	# iterations 10^9	time = 10 sec
$N = 10^{18}$	# iterations 10^{18}	time = 10^{10}

$10^{10} * 10^8$ iterations

10^{10} 1 sec

10^{10} sec.

$\approx$ 317 yrs ☹

10^8 iterations = 1 sec

10^9 iterations = ?

$\frac{10^9}{10^8}$ seconds = 10 sec

$\Longrightarrow$ 10^9 iterations $\longrightarrow$ $10 * (10^8$ iterations$)$

1 sec

Observation

$a * b = N \quad a <= b$

If a is known what is the value of $b = \frac{N}{a}$

N = 24

a		b
1	*	24
2	*	12
3	*	8
4	*	6

N = 10

1	*	10
2	*	5

$a_{min} = 1$

$a_{max} = \sqrt{N}$

$a = \frac{N}{a}$

$a * a = N$

$a = \sqrt{N}$

Optimised

```
count = 0

for i ⟶ 1 to √N   // for(a = 1 ; a*a ≤ N ; a++)
    if (N % i == 0) {
        count += 2
    }
}

print (count)
```

N = 9

1	9	+2
3	3	+2

After you write the code think of edge cases.

The above code doesn't work for perfect square.

Optimised

```
count = 0

for i ⟶ 1 to √N    // for(a = 1 ; a*a ≤ N ; a++)
    if (N % i == 0) {
        a = i
        b = N / i    // integer division
        if (a == b) { count += 1 }
        else { count += 2 }
    }
print (count)
```

Total no. of iterations for above code = $\sqrt{N}$ iterations

$N = 10^8$ # iteration $= 10^4$ time $= 10^{-4}$ sec

$N = 10^{18}$ # iterations $= \sqrt{10^{18}} = 10^9$ time $= \frac{10^9}{10^8}$ secs $= 10$ sec

317 yrs to 10 sec

:)

10^8 iteration $= 1$ sec

1 iteration $= \frac{1}{10^8}$ secs

10^4 iteration $= \frac{10^4}{10^8} = 10^{-4}$ secs

Follow up — Check if a given N is prime or not.

Exactly 2 factors. HW

```
i = N        // N > 0
while (i > 1) {
    i /= 2
}
```

1 iteration ⟶ $\frac{N}{2}$

2 iterations ⟶ $\frac{\left(\frac{N}{2}\right)}{2} = \frac{N}{4} = \frac{N}{2^2}$

3 iterations ⟶ $\frac{N}{2^3}$

⋮

K iterations ⟶ $1 = \frac{N}{2^K}$

$2^K = N$

$\log(2^K) = \log N$

$K \log(2) = \log N$

$K = \log N$

$\log_2 2^K = K$

$\log_b c = a$ if $b^a = c$

```
for i ⟶ 1 to N {
    for j ⟶ 1 to N {
        print (i+j)
    }
}
```

i	j	# iteration
1	[1 N]	N
2	[1 N]	N
3	[1 N]	N
⋮		
N	[1 N]	N

N times

N * N

```
for i ⟶ 0 to N-1 {
    for j ⟶ 0 to i {
        print (i+j)
    }
}
```

i	j	# iteration
0	[0 0]	1
1	[0 1]	2
2	[0 2]	3
⋮		⋮
N-1	[0 N-1]	N

$$\frac{N*(N+1)}{2} = \frac{N^2}{2} + \frac{N}{2}$$

Big O

Rate of growth of function wrt input.

3 steps

→ Calculate no. of iterations

→ Drop the lower order terms

→ Drop the coefficient.

$f(N) = 4N^3 + 3N^2 + 2N + 10$

Drop lower order terms

$4N^3 + \cancel{3N^2} + \cancel{2N} + \cancel{10}$

Drop the coefficient.

$\cancel{4}N^3$

$O(N^3)$

```
bool search (A[], K) {                                   TC: O(N)
    for i ⟶ 0 to N-1 {
        if (A[i] == K) return true
    }
    return false
}
```

# iteration if A[0] == K	1
# iterations if array doesn't contain K	N

NOTE: we always take worst case # iterations

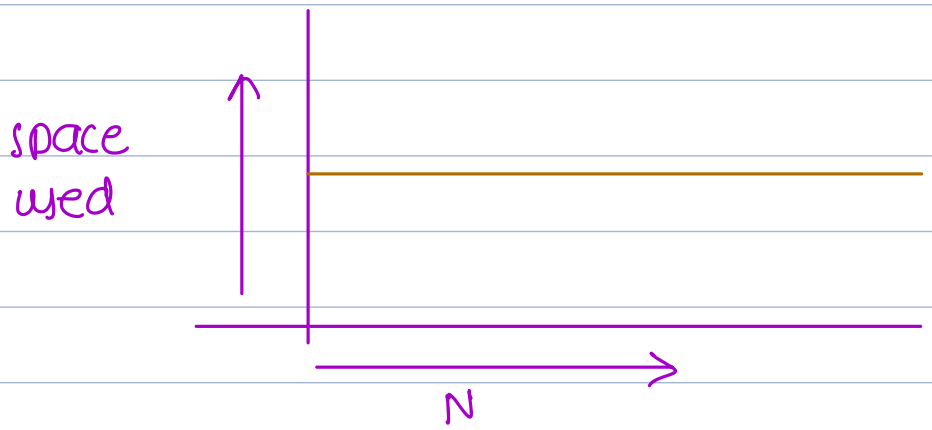
Space complexity

Rate of growth of space used wrt input

input = N

```
int a = 10      // 4 B
int b = 20      // 4 B
long c = a * b  // 8 B
```

Total space = 16 B



SC : O(1)

input N

```
int A[] = new int[N]
```

SC : O(N)

input → algorithm → output

Space complexity is calculate by the max space wed at any instant by your algo

Input & Output space doesn't contribute to SC

```
int f (int[] A) {
    total = 0
    for i ⟶ 0 to N-1 {
        total += A[i]
    }
    return total
}
```

SC : O(1)

Break : 22 : 43