DSA

16/07/25

Basic Introduction

Q) What is a factor?

'i' is a factor of N → if 'i' divides N completely
i.e remainder = 0

How to check it programmatically check it :

by (%) modulus operator, as it helps us to get remainder, if N%i == 0.

eg: Factors of N = 24

24 → 1, 2, 3, 4, 6, 8, 12, 24 ⇒ (8 factors)

How to count factors? [N=24]

Brute force approach :

Minimum factors of a number? (1)

Max. factor of a number? (24)

∴ so range of factors of N = (1 to N)

Code (Python) :

```
def countfactors (N):
    count = 0
    for i in range (1, N+1):
        if N%i == 0:
            count += 1
    return count
```

If my loop is running from '1 to N+1' so,

No. of iterations ⇒ N

★ Assumption: 10^8 iterations → 1 second

What is the time required for 1 iteration? $1/10^8$ s.

Value of (N)	No. of iterations (N)	Time Required
10^8	10^8	1 second
10^9	10^9	10 secs
10^{18}	10^{18} ↓ $10^8 * 10^{10}$	10^{10} secs ≃ 317 years

★ Optimisation for counting factors:

Let 3 numbers i, j & N

$i * j = N$ → N is divisible by i
→ N is divisible by j
→ i & j both are factors of N

i = 3, j = 4, N = 12 → 3 × 4 = 12
3, 4 are both factors of 12.

Conclusion: first factor → i
second factor → $j = N/i$

Now consider N = 24,

N = 24	first factor (i)	second factor (j) = N/i
Part 1	1	24/1 = 24
	2	24/2 = 12
	3	24/3 = 8
	4	24/4 = 6
Part 2	6	24/6 = 4
	8	24/8 = 3
	12	24/12 = 2
	24	24/24 = 1

Observation: i) first factor is i
ii) second factor is j
iii) Condition to repeat

$$i < \frac{N}{i}$$

We found all factors in part 1 when $i < N/i$ we get all the factors

N = 36	1st factor (i) & 2nd factor (j)		
	1	<	36 → 2
Observation:	2	<	18 → 2
	3	<	12 → 2
i < N → 2 factors	4	<	9 → 2
i	6	<=	6 → 1
i == N/i → 1 fac.	9	>	4 → Repeating factor
i > N/i → repe. factor			X no need to consider it

Date
Page

★ Conclusion →
Range of factors [1 $\sqrt{N}$]

$i <= N$
i
→ $i^2 <= N$
square root on both sides
$\sqrt{i^2} <= \sqrt{N}$
$i <= \sqrt{N}$

★ Code mentioned in .ipynb file

eg→ N = 15 → [1 to $\sqrt{15}$] ⇒ [1 3]

i	N%i == 0	i == N/i	count += 1	count += 2
1	True	1 == 15/1 (X)	-	(2)
2	False			→ skip
3	True	3 == 5	-	(4)
4				→ end of loop.

Total no. of iterations :

i	N/i
1	15
3	5
5 >	3 Repeating

To [1 $\sqrt{N}$] ⇒ $\sqrt{N} - 1 + 1 \Rightarrow \sqrt{N}$ (Range func^n is used)
[a b] ⇒ b - a + 1

★ Prime Numbers →

Approach - 1) Calculate the no. of factors
2) No. of factors > 2 → Not prime
↳ else prime.

```
def isprime(N):
    if countfactors(N) > 2:
        return False
    else:
        return True
```

★ Basic maths properties:

[a, b] = both a & b are included in the range.

[1 3] ⇒ 1, 2, 3

(a, b) = both a & b are excluded

(1 3) ⇒ 2

No. in range [a, b]?

[a b] = end - start + 1
= b - a + 1

- Sum of N natural numbers: $\frac{N*(N+1)}{2}$

$1+2+3+4..... 100 = \frac{100*(100+1)}{2}$

$= 5050$

- **Iterations:** No. of times a loop runs

```
for i in range (1,N+1):
    if (i==N)
        break;
```

Range [1, N]
= N - 1 + 1
= N

```
for i in range (0,101):
    s = s+i+i**2;
```

R = [0 100]
= 100 - 0 + 1
= 101.

```
func()

    for i in range (1,N+1):
        -
        -
        -

    for i in range (1,M+1):
        -
        -
        -
```

Range = [1 N]
= N - 1 + 1
= N

R = [1 M]
= M - 1 + 1
= M

Total iterations ⇒ **N+M**

★ Geometric Progression –

5 10 15 20 40 80 → a ar² ar³ ....

10/5	10/20	40/20	80/40
↓	↓	↓	↓
2	2	2	2 → common ratio

Sum of first N terms of GP: $\frac{a * (r^n + 1)}{(r - 1)}$

a → first terms
r → common ratio
n → no. of terms.

$$\boxed{G.P = \frac{a * (r^n + 1)}{r - 1}}$$

note → if r=1, then $\boxed{GP = a * n}$

a ar ar² ar³ .... ar^(n-1)

r=1 a a a a .... a

n terms

So $\boxed{G.P = a * n}$

Q) How to compare 2 Algorithm's ?

⇒ Execution time depends on no. of factors language, OS etc

But, no. of iterations doesn't depend on any factor.