Basic Mathematics for DSA

⓪ Prime Numbers →

i/p → n

o/p → Prime or Not.

#Naive approach -

n = 10

1 [- - - - - - - -] 10

i → (2 to 9)

if (n % i == 0) → if any no. completely divides n between 2 to n-1 then that means n is not a prime no.

not prime.

> Basically any number is prime if it has only two factors 1 and that number itself.

prime no. eg → 2, 3, 5, 7

↑ smallest prime no.

> 1 is not a prime no. as it has only one factor.

leetcode 204 → Count Prime

code → Naive approach -

```
int count = 0;
for ( int i = 0 ; i<n; i++){        → O(n)
    if ( isPrime ( i )){            → O(n)
        ++count;
    }
}
return count;
```

```
bool isPrime (int n){
    if(n<=1) return false;
    for (int i = 2; i<n; i++){      — O(n)
        if(n%i == 0)
            return false;
    }
    return true;
}
```

This will give us TLE.

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

#2 → Better isPrime function.

so originally loop runs for → $i=2$ to $i=n-1$

→ Let n is non-prime

means there is atleast 1 ~~fa~~ factor of n between 2 to $n-1$.

$1, \boxed{2, - - - - - - - n-1}, n$

↳ at least 1 factor.

if $a > \sqrt{n}$

and $b > \sqrt{n}$

⇒ $ab > n$ → but this is not possible.

⇒ so atleast one of the factor must be smaller than $\sqrt{n}$.

And if we can't find any factor less than $\sqrt{n}$ then n is a prime no.

so rather than run the loop till $n-1$ we will run the loop till $\sqrt{n}$.

```
bool isPrime (int n){
    if (n<=1) return false;
    for (int i=2; i<= sqrt(n); i++){
        if (n%i ==0)
            return false;
    }
    return true;
}
```

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

sqrt(n) ← C++ function (inbuilt)

$\boxed{\text{Total T.C} \rightarrow O(n\sqrt{n})}$ ($O(n)$ time complexity of countPrime function).

#3 Sieve of Eratosthenes Approach →

$N = 21$

array ↘

	2	3	4	5	6	7	8	9	10	11	12	13	14
	T	T	~~T~~F	T	~~T~~~~F~~F	T	~~T~~F	~~T~~F	~~T~~~~F~~F	T	~~T~~~~F~~F	T	~~T~~F

15	16	17	18	19	20	
~~T~~~~F~~F	~~T~~F	T	~~T~~~~F~~F	T	~~T~~~~F~~F	

Initially everyone marked as True (means prime).

2 is prime but the no. which are completely divided by 2 are not. (mark them non-prime (false)).

3 is prime but it's multiple aren't. (mark them non-prime).

4 → already marked as non-prime.

5 → prime, mark it's multiple as non-prime.

6 → already marked as non-prime.

7 → prime, mark it's multiple as non-prime.

⋮

19 → prime.

Now count the element which are marked as prime (true).

<u>count = 8</u> (2, 3, 5, 7, 11, 13, 17, 19).
<u>ans</u>

Algo →

① 2 → n-1 array represents no.s, mark all of them as prime.

② Start from 2 till end, mark all the no. comes in the table of 2 as non prime.

③ Repeat ② till (n-1). (Only for prime no)

④ Rest elements marked as prime will be ~~con~~ counted.

```
int countPrimes (int n){
    if(n<=01) return 0;
    int ans = 0;
    vector<bool> prime(n, true);
    prime [0] = prime[1] = 0;
    for ( int i=2; i<n; i++){
        if(prime [i]){
            ans++;
            int j = 2* i;
            while (j<n){
                prime[i] = false;
                j+=i;
            }
        }
    }
    return ans;
}
```

#4. Segmented Sieve →

variation of sieve.

In sieve → 0 to n-1

In segmented sieve → l to h (l = starting point, h = end point).

Google this.

T.C of Sieve of Erato

$\boxed{\text{Sieve} \rightarrow \text{the array we made}}$.

Outer loop → $T.C = \underbrace{O(n)}_{\text{outer loop}}$

Inner loop → $\left[\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \frac{n}{7} + \cdots\right]$

↳ H.P of prime numbers

→ Tailor series.

→ $\log(\log n)$

$$\boxed{\text{Total } T.C = O(\underbrace{n}_{\text{outer loop}} * \underbrace{\log(\log n)}_{\text{Inner loop}})}$$

GCD / HCF → Highest Common Factor.

GCD → Greatest common divisor

eg, a, b.

HCF → maximum no. that completely divides both a & b.

a = 24, b = 72

24 = 1 × 2 × 2 × 2 × 3

72 = 1 × 2 × 2 × 2 × 3 × 3

HCF = 1 × 2 × 2 × 2 × 3

= 24

Technique →

$$gcd(a,b) = gcd(a-b, b) \quad \text{if } a > b$$

$$\text{else } gcd(a,b) = gcd(b-a, a) \quad \underline{a < b}$$

OR

$$gcd(a,b) = gcd(a\%b, b), \quad a > b$$

$$gcd(a,b) = gcd(b\%a, a), \quad a < b$$

% is very heavy operator. so subtraction method is preferred.

→ Apply this till one of the parameter becomes 0.

eg → $gcd(72, 24)$
$= gcd(48, 24)$
$= gcd(24, 24)$
$= gcd(\underbrace{0}_{\text{becomes } 0}, 24)$ → so this is my ans.

gfg

- GCD of two numbers →

```
int gcd( int A, int B){
    if (A==0)
        return B;
    if (B== 0)
        return B;
    while (A>0 && B>0){
        if (A>B)
            A= A - B;
        else
            B = B - A;
    }
    return A==0 ? B : A;
}
```

This is Euclid's algo.

- LCM →

$$LCM * HCF = a \times b$$

$$\boxed{lcm(a,b) * hcf(a,b) = a \times b}$$

$$\boxed{lcm(a,b) = \frac{a \times b}{gcd(a,b)} = \frac{a \times b}{hcf(a,b)}}$$

Modulo Arithmetic →

1→ $a \% n$ = ans will lie between $0 ---, n-1$

eg→ $10 \% 3 \Rightarrow [0, 1, 2]$

$5\%4 \Rightarrow [0, 1, 2, 3, 4]$

2→ To avoid overflow while storing integer we do modulo with a large number.

(i) $(a+b)\%M = a\%M + b\%M$
(ii) $a\%M - b\%M = (a-b)\%M$
(iii) $((a\%M)\%M)\%M = a\%M$
(iv) $a\%M * b\%M = (a*b)\%M$

properties

Fast Exponentiation →

1→ Normal solution to find a^b

$$a^b \Rightarrow 2^{10} \Rightarrow \underbrace{2\times2\times2\times ---- \times 2}_{10}$$

$\boxed{T\cdot C \to O(b)}$

```
int ans =1;
for (int i=0; i<b; i++){
ans *=a;
}
cout << ans;
```

multiplying a, b times.

2→ Better solution →

$T.C = O(\log b)$

eg→ a^b

if b is even $\Rightarrow a^b = (a^{b/2})^2$

if b is odd $\Rightarrow (a^{b/2})^2 * a$

eg→ $2^{10} = (2^5)^2 = 2^{10}$

$2^{11} = (2^5)^2 \times 2 = 2^{11}$

now 2^5

$\hookrightarrow (2^4)\cdot 2$

$\hookrightarrow (2^2 \cdot 2^2)\cdot 2$

$\downarrow$

$(2^1\cdot 2^1)\cdot(2^1\cdot 2^1)\cdot 2$

$\boxed{\text{divide and conquer.}}$

```
int fastExponentiation (int a, int b){
    int ans = 1;
    while ( b > 0){
        if (b&1){          ----> if b is odd.
            ans = ans * a;
        }
        a = a* a;
        b >> 1;            ----> right shifting b by 1.
    }
    return ans;
}
```

<u>dry run</u> →

- ans = 1 , a = 5, b = 4

ans $\boxed{\cancel{1}}$ 625 a $\boxed{\cancel{5}}$ ~~25~~ ~~625~~ 625*625 b $\boxed{\cancel{4}\cancel{2}}$ ~~1~~ 0

⇒ b = 4 (even) > 0 → True
a = a * a = 5×5 = 25 , b = 2

⇒ b = 2 (even) > 0 → True
a = a * a = 25 × 25 = 625 , b = 1

⇒ b = 1 (odd) > 0 → True
ans = 1 × 625 = 625
a = 625 × 625 b = 0

⇒ b = 0 > 0 → False return ans. 625

- a = 2 , b = 5 ans $\boxed{\cancel{1}\cancel{2}}$ 32 a $\boxed{\cancel{2}\cancel{4}}$ ~~16~~ 256 b $\boxed{\cancel{5}\cancel{2}\cancel{1}}$ 0

odd
① → b = 5 > 0 → T
⇒ ans = 1 × 2 = 2
a = 2 × 2 = 4 b = 2

even
② → b = 2 > 0 → T
a = 4×4 = 16 b = 1

odd
③ → b = 1 > 0 → T
ans = 2 × 16 = 32
a = 16 × 16 = 256 b = 0

④ → b = 0 > 0 → F
↳ return ans 32

2^5

$(2^2 \cdot 2^2) \cdot 2$

$(2 \cdot 2)(2 \cdot 2) \cdot 2$

```
long long int PowMod(long long int x, long long int n,
                     long long int M){
    long long int ans = 1;
    while (n>0){
        if(n&1){
            ans =(ans * x) % M;
        }
        x = (x * x) % M;
        n>>1;
          ↳ n=n/2;
    }
    return ans % M;
}
```

To avoid overflow.

⊙ Advaced Topics (C.P Scope) →

1. Pigeon Hole
2. Catalan Number
3. Inclusive - Exclusive Principle
4. Chinese ~~Princip~~ Reminder Theorem
5. Lucas' Theorem
6. Fermat's Theorem
7. Probability Concepts.