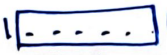
Basic Maths :-

1. **Prime number :**

1 is not a prime number
2 is smallest prime number
input — N

. Tell N is prime or not ?

N = 10



↳ i

(N % i) == 0

↳ X

else

prime

count primes :- Q204 Leet code

[0 ⟶ N-1]

↳ ? prime no.s count

① Naive

⇒

```
for (i=2; i<n ; i++){
    if (isPrime (i))
        count++;
}
return count
```

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

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

→ n: normal method Prime Yes / No
O(n)

2 loop → countprint ( )
↳ O(n)

is Prime → O(n) → T.C. ⇒ $O(n^2)$

M2: Better kro [redacted] is Prime()

→ finding if N is prime or not ?

originally → i=2 ⟶ i<n

Let N is non-prime.

1 [2 . . . . n-1] N

n = a x b

if [a > √n, b > √n] ⇒ ab > n x

at least one of factor must be greater than √n.

if we can't find any factor less than √n, then N is prime.

Code:

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

TC:- O(n√n)

M3: **Sieve of Eratosthenes**

Given → N : total count of primes < N.

N = 21

~~1~~ 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 ~~21~~

as 2 is prime, so we can remove its multiples.

2 3 5 7 9 11 13 15 17 19

now, 3 is prime, so we can remove its multiples

2 3 5 7 11 13 17 19

now, 5 is prime, 7, 11, 13, 17, 19

∵ We have '8' prime no.s under 21.

Algorithm:

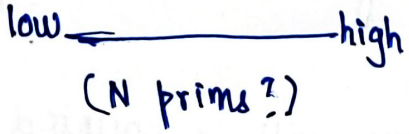
1. 2 → n-1, array no.s
   Mark all no.s as prime
2. start from 2 till end, mark all multiples of 2 as non prime.
3. Repeat (2) till (N-1), only for no.s marked prime no.
4. Rest elements marked as prime will be counted.

Code:

```
int countPrimes(int n) {
    if(n==0)
        return 0;

    vector<bool> prime(n, true);
    prime[0] = prime[1] = false;

    int ans=0;

    for(int i=2; i<n; i++){
        if(prime[i]){
            ans++;

            int j=2*i
            while(j<n){
                prime[j] = false;
                j+=i;
            }
        }
    }
    return ans;
}
```

T.C.:- $n\left[\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \frac{n}{7} + \cdots \frac{n}{n}\right] = O(n(\log n))$

M4: Segmented sieve (Google करलो)



2. GCD/HCF :
   - Highest Common factor
   - Greatest common divisor

GCD(a, b) = GCD (a-b, b) ; a > b
GCD (b-a, a) , a < b

GCD (a, b) = GCD (a % b, b) a > b (best practise)

Apply till one of parameter is zero.

eg: gcd (72, 24)
gcd (48, 24)
gcd (24, 24)
gcd (0, 24)

Code:

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

LCM →

LCM * HCF = a * b

⇒ lcm(a,b) * gcd(a,b) = a * b

↓

Euclid algo

$$lcm = \frac{a * b}{gcd}$$

//

3. Modulo Arithmetic:

(a %. n) ⇒ [0 ... n-1]

10 %. 3 ⇒ [0, 1, 2]

5 %. 4 ⇒ [0, 1, 2, 3]

Properties:-

1. (a+b) %. M = a %. M + b %. M
2. a %. M − b %. M = (a−b) %. M
3. ((a %. M) %. M) %. M = a %. M
4. a %. M * b %. M = (a*b) %. M

Ratio

4. **Fast Exponentiation** – a^b

$\Rightarrow 2^{10} = \underbrace{2 \times 2 \times \cdots 2}_{10}$

$x^n \rightarrow 3^{10}$

5^{20}

1. **Normal soln:**

To find $a \hat{} b = a^b$

T.C. = O(b)

Loop → [0 – b-1] ans = 1

↳ ans = ans * a;

a^b O(b)

Code:

```
int slowExponentiation (int a, int b) {
        int ans = 1;
        for (int i=0; i<b; ++i){
                ans *= a;
        }
        return ans;
} // O(b)

int main(){
    cout << slowExponentiation(5,4) << endl;
    return 0;
}
```

2. **Better Solution:-**

a^b

$O(\log b)$

if b is even,

$a^b = (a^{b/2})^2$

if b is odd

$a^b = (a^{b/2})^2 \cdot (a)$

eg: $2^{10} = (2^5)^2$

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

2^5

$\hookrightarrow (2^4)\ 2$

$\downarrow$

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

$\downarrow$

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

Code:-

```
int fastExponentiation(int a, int b){
    int ans = 1;
    while(b > 0){
        if(b & 1)
            // odd
            ans = ans * a;
        }
        a = a * a;
        b >>= 1;
    }
    return ans;
} // O(logb)
```

Advanced Topics (C.P Scope)

1. Pigeon Hole
2. catalan number (BST)
3. Inclusion - Exclusion Principle
4. Chinese Reminder Theorem
5. Lucas' Theorem
6. Fermat's Theorem
7. Probability concepts.