Maths for DSA

→ Prime Numbers

n → (2 to √n)

```
for(i=2; i*i <= n; i++){
    if(n%i == 0){
        return "Non-Prime"
    }
}
return "Prime";
```

$i^2 <= n$

$i <= \sqrt{n}$

→ Inner loop

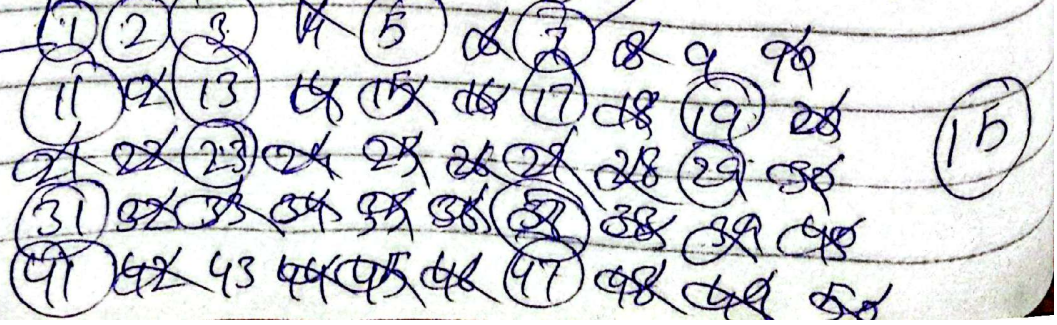
If we want to find the prime numbers in the given range:

```
for(i=2; i<=N; i++){
    IsPrime = TRUE
    Inner loop
        Isprime = false
        break;
}
```

Above approach would give TLE, So, the better approach is:

Sieve of Eratosthenes

1. We will be considering all the Numbers as the Prime Numbers
2. Then the number which we have considered as the Prime Number, multiple of those numbers will be discarded from the given range suppose 1 to N(50)



```
vector<bool> Isprime(n+1, TRUE){
for(i=2; i<n; i++){
    if(Isprime[i]){
        ans++
        for(j = i*2; j<n; j=j+i){
            Isprime[j] = False;
        }}
```

→ Digits in a Number

We can do the following things by this

- Count of Digits
- Sum of Digits
- Print Digits.

n = 358(6) % 10 → 6
35(8) % 10 → 8
3(5) % 10 → 5
(3) % 10 → 3
0

Reverse digits

```
while(n!=0){
    digit=n%10
    digit
    n=n/10
}}
```

Count digits → count=0

```
while(n!=0){
    digit = n%10
    count++
    n=n/10
    count }
```

OR

int(log10(n))+1.

will give the count as well

Sum of Digits → Sum=0

```
while(n!=0){
    digit = n%10
    Sum += digit
    n=n/10
    Sum }
```

→ Amstrong Number

"Number that is equal to the sum of cubes of its digits."

$153 \rightarrow 1^3+5^3+3^3 = 1+125+27 = 153$

$371 \rightarrow 3^3+7^3+1^3 = 27+343+1 = 371$

```
bool IsAmstrong(int n){
    copyN = n
    Sum_of_cubes = 0
    while(n!=0){
        dig = n%10
        Sum_of_cubes += (dig*dig*dig)
        n = n/10
    }
    return Sum_of_cubes == copyN
```

it will be seperating each digit and then adding their cubes as well.

→ GCD/HCF (Greatest Common Divisor / Highest Common Factor)

a = 20, b = 28

2 | 20
2 | 10
5 | 5
1 | 1
| 0

2 | 28
2 | 14
7 | 7
1 | 1
| 0

$20 = 2\times2\times5\times1$ (4)

$28 = 2\times2\times7\times1$ (4)

Brute Force

gcd = 1

```
for(i=1; i<=min(a,b); i++){
    if(a%i==0 && b%i==0){
        gcd = i
    }
}
```

O(min(a,b))

Corner cases

① a == b → b

② b == 0 → a

③ a == b → a or b

Optimized Approach

Euclid Algorithm

gcd(a,b) <
- gcd(a-b,b), a>b
- gcd(a,b-a), b>a

gcd(20,28) → gcd(20,8) → gcd(12,8) →
gcd(4,8) → gcd(4,4) → gcd(0,4)

20-8, 12-8 = 4

20%8, 12%8 = 4

gcd(a,b) <
- gcd(a%b,b), a>b
- gcd(a,b%a), b>a

→ gcd(20,88) → gcd(20,8) → gcd(4,8)
→ gcd(4,0)

```
gcd(a,b){
while(a>0 && b>0){
  if(a>b){
    a=a%b
  else
    b=b%a
  if(a==0)
      return b
  return a;
```

OR Using Recursion

```
gcdRec(a,b)
  if(b==0) return a
  return gcdRec(b,a%b);
```

→ LCM (least common factor)

LCM of two numbers would be like:

$$LCM = \frac{a*b}{gcd(a,b)} : \frac{20*28}{4} = \frac{560}{4}$$

$$= 140$$

```
LCM(a,b){
  gcd = gcdRec(a,b)
  return a*b/gcd;
```

→ Reverse a Number

As we know the logic for extracting digits using the '%' operators in reverse order.

num = 4537

digit	revNum = (revNum * 10) + digit
7	
3	70 + 3 = 73
5	730 + 5 = 735
4	7350 + 4 = 7354

→ Palindrome Number

n = 353

num < 0 → false

① we will calculate the reverse.
② then compare with the original one.

```
IsPalindrome(n){
  if(n<0) → false
  revNum = reverse(n)
  return n == revNum
```

→ Modulo Arithmetic

Modulo Arithmetic are used to store the large or too much big values in the form of exponents. It is used to save us from the overflow condition.

$x \cdot 1 \cdot n \rightarrow [0, n-1]$

$100 \cdot 1 \cdot 3 \rightarrow [0, 1, 2]$

Properties

$(x+y) \cdot 1 \cdot m = x \cdot 1 \cdot m + y \cdot 1 \cdot m$

$(x-y) \cdot 1 \cdot m = x \cdot 1 \cdot m - y \cdot 1 \cdot m$

$(x \cdot y) \cdot 1 \cdot m = x \cdot 1 \cdot m \cdot y \cdot 1 m$

$((x \cdot 1 \cdot m) \cdot 1 \cdot m) \cdot 1 m$