DSAJ - 22 Maths For DSA

① Prime number code-

```
For (i = 2;  i < N ; i++) {
    if (N % i) {
            not prime;
    }
    ~~else~~ }
        prime ✓;
```

Nowe we know for a num == 36, its division is

1 × 36
2 × 18
3 × 12
4 × 9
6 × 6
9 × 4
12 × 3
18 × 2
36 × 1

$\left.\begin{matrix} 9 \times 4 \\ 12 \times 3 \\ 18 \times 2 \\ 36 \times 1 \end{matrix}\right\} \rightarrow$ repeated

we can avoid these by writing efficient code till $\sqrt{n}$

code →

```
if (n <= 1) {
        return false;
}   int c = 2;
while ( c × c <= n) {
    if (n % c == 0)
        return False;
    }
    c++;
}   return true;
```

basically means
$c \leq \sqrt{n}$
squaring both sides
$c^2 \leq n$

② sieve of eratosthenes →

as we have to include n in array

```
int n = 40;
boolean[] primes = new boolean [n+1];
sieve (n, primes);

static void sieve (int n, boolean[] primes) {
    for (int i=2 ; i*i <= n ; i++) {
        if (primes [i] == false) {
            for (int j = i+i ; j <= n ; j = j+i) {
                primes [j] = true;
            }
        }
    }
    for (int i = 2; i <= n ; i++) {
        if primes [i] == false;
            print ( i );
    }
}
```

$i \leq \sqrt{n}$

in array if arr[i] = false → prime

arr[i] = true , not prime.

Time complexity →

outer loop running till n

inner loop → $n\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} \cdots\right)$ → sum of primes (Taylors) HM

↓

log (logn)

∴ Time complexity = $O(n \log(\log n))$

we use binary search

③ Finding square root of a num.

```
int n = 40;
int p = 3;
System.out.printf("%.3f", sqrt(n,p));
```

pretty printing.

```
static double sqrt(int n, int p){
    int s = 0;
    int e = n;
    double root = 0.0;

    while ( s<=e) {
        int m = s + (e-s)/2;

        if (m x m == n)
            return m;
        if (m x m > n)
            e = m-1;
        else
            s = m+1;

    root = e;
```

returns the last value, possible root!

for precision we use

```
double incr = 0.1;        precision digits, = 3 in this case
for (int i=0 ; i< p ; i++) {
        while (root * root <= n) {
            root += incr;
        }
        root = root - incr;        → here the root value had exceeded √n ∴ we need to subtract 0.1 to get our previous step value
        incr /= 10;        gets to the next decimal place
    }
    return root;
  }
}
```

Time complexity == $O(\log(n))$

④ Newton Raphson method for root

Formula → $$\sqrt{N} = \frac{\left(X + \frac{N}{X}\right)}{2}$$

X → guessed root

This formula works because try to put $\sqrt{N}$ in place of X, we will have LHS = RHS

error = |root − x| → we will keep minimising error.

Math.abs (a-b) $\rightarrow$ gives the exact value (like mod | |).

Process / Approach

1. Assign X to N
2. we will find our ans when error <1
3. we will update x to $X = \sqrt{\frac{X+N}{X}}$

$$X(\text{root}) = \frac{X + \frac{N}{X}}{2}$$

$\rightarrow$ initial $X = N$

Code $\rightarrow$

```
double x = n;
double root = 0;

while (true) {
    root = 0.5 x (x + n/x);

    if (Math.abs (root - x) < 0.5) {
        break;
    }

    x = root;

}
return root;
}
```

⑤ Factors of a number:-

eg if we take num → 20

20 has factors

1 2 4 5 10 20

we can check all nums below 20 like

i % n == 0 ✓ is a factor.

OR to reduce ~~space~~ time complexity.

↳ we can run it till like $\sqrt{n}$

because

1 × 20
2 × 10
4 × 5

~~too~~ factors are repeating

∴ no need to print / do twice.

also we store that later half, 20, 10, 5 in a array list and print reverse of it.

```
ArrayList<Integer> list = new ArrayList<> ();
for (int i= 1; i<= Math.sqrt(n); i++) {
    if (n % i == 0) {
        if (n / i == i)
            return (i)
        else
            return(i)
            list.add (n/i);
}
```

prevents printing duplicates ←

adds the other factor to list ←

$\sqrt{n}$ Time

now print reverse arraylist.

⑥ Properties of Modulo

- $(a+b)\%m = [(a\%m)+(b\%m)]\%m$
- $(a-b)\%m = [(a\%m)-(b\%m)+m]\%m$
- $(a*b)\%m = [(a\%m)*(b\%m)]\%m$
- $\left(\frac{a}{b}\right)\%m = [(a\%m)*(b^{-1}\%m)]\%m$

$b^{-1}\%m \Rightarrow$ multiplicative modulo inverse

↓

This means that b and m are coprimes

↳ only 1 as common factor

- $(a\%m)\%m = a\%m$ only
- $m\%m = 0$ ✓ $\forall\, n \in$ +ve integer

7 Die hard Example

What is HCF / GCD ?

For an equation x & y as in integers, what is the min. +ve value you can get of the equation.

OR

It is the highest common factor of 2 no-s
eg

HCF (4, 18) = 2

1, 2, 4   1, 2, 3, 6, 9, 18

$\min(3x + 9y) = 3$ → 3L measurement to be done.

3L bucket   9L bucket

$\cancel{3}(x + 3y) = \cancel{3}$

put $x = -2$, $y = 3$

you get min value.

eg- $ax + by = L$

$2x + 4y = 5$

HCF of 2 and 4 is 2

$2(x+2y) = 5$

$x+2y = 2.5 \rightarrow$ as we are getting decimal value

Die hard not possible

eg- $3x + 5y = 17$

$1(3x+5y) = 17$

1 divides 17 $\therefore$ Die hard is possible

$3x + 6y = 9$

$\cancel{3}(x+2y) = \cancel{9}\ 3$

as HCF divides, die hard possible.

(8) Euclids Algorimm for <u>GCD</u>

$$gcd(a, b) = gcd(rem(b, a), a)$$

eg- $gcd(105, 224) = gcd(rem(224, 105), 105)$

$$\downarrow$$

$$= gcd(14, 105)$$

Why this is working?

Our eq - $105x + 224y$

$\downarrow$ gets converted to

$14x + 105y$

because gcd of $(105, 224)$ also divides a linear combination of 105 and 224.

$\therefore$ as soon as it is a linear combination, it is okay ✓

x and y can be anything.

as $224 - 2 \times 105 = 14$

$224 = 14(1) + (2) \times 105$

$14x + 105y = 224 = L$

Code → use Recursion-

Function-

```
static int gcd (int a , int b) {
    if ( a == 0) {
        return b;
    }
    return gcd ( b%a , a)
}
```

(9) LCM

$lcm(a,b)$ = min number divisble by both a and b

we know that

$$a \times b = lcm(a,b) \times GCD(a,b)$$

FORMULA :

$$\boxed{LCM(a,b) = \frac{a \times b}{GCD(a,b)}}$$