WEEK-06 — Lecture - 01

BASIC MATHEMATICS FOR DSA

Prime Number → Naive approach
→ Sqrt approach
→ Sieve of Eratosthenes
→ Segmented Sieve

⇒ Normal i/p = N , o/p = prime or not.

N = 10

(N % i) == 0 ✓ then ~~false~~.

else true.

Prime → exactly/only 2 factors (1 and N)

⇒ 1 is not a prime. (only has one factor)

⇒ 2 is smallest prime.

Q) Count Primes [204]

naive approach :

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

17/66 testcases passed.

~~O(n)~~

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

2 loops ⇒ 0 to n-1 → O(n)

isPrime ⇒ O(n)

∴ O(n²)

M-2 make isPrime() better.

↳ ⊛ `i=2 to i<n` presently.

Let N is non-prime. 1, 2, ... n-1, N

non-prime ⇒ `a x b` = n.

↳ if n is non-prime, atleast 1 factor is there here.

if $a > \sqrt{n}$ and $b > \sqrt{n}$

$ab > n$ ✗ not possible. ($\because ab = n$)

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

Conclusion → if we cant find any factor less than $\sqrt{n}$, then, N is Prime.

∴ `int sqrtN = sqrt(n);`

`for (int i=2; i< sqrtN ; i++)`

∴ Complexity of isPrime ⟹ $\mathcal{O}\sqrt{n}$ (20/66) passed.

M-3 :- Sieve of Eratosthenes.

⇒ when you need to count primes b/w 0 to n.

* • N = 21. ① make an array → 0, 1, 2, 3 ..... 20, 21

② mark all as prime.

③ 1 is non-prime (False). So remove 1 and n (we need b/w)

④ 2 is prime, its multiples will be non-prime.

(4, 6, 8, 10, 12 ---).

⑤ So, remove all even → { mark all even non-prime (False) }

⑥ Same for 3 → (6, 9, 12, 15, 16, 18)

⑦ 4 is itself prime, so next 5 is prime

∴ Mark multiples of Prime ⇒ All true are Prime

TC ⇒ $n \cdot \left[\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \cdots\right]$ → (Harmonic Progress of prime no)

$n^2 = n$ ←

$O[n*(\log \log n)]$

Algorithm :-

∴

① 2 to n-1, array. Mark all as prime (True).

② Start from 2 till end, mark all no comes in the multiples of 2, as Non-Prime.

③ Repeat ②, till (N-1), but only for Prime numbers.

④ Rest elements marked as prime will be counted.

∴ TC = $O[n * \log(\log n)]$

4. Segmented Sieve

(same thing but finding primes b/w low and high)

GCD / HCF (~~Euclid~~)

Greatest Common Divisor / Highest Common Factor.

Eg : a, b

a % hcf = 0

b % hcf = 0

↓ maximum.

24, 72

24 = 1 × 2 × 2 × 2 × 3

72 = 1 × 2 × 2 × 2 × 3 × 3

$1 \times 2^3 \times 3 = \boxed{24}$ //

EUCLID METHOD

* gcd (a,b) = gcd (a-b, b) if a > b → Apply till one of params becomes 0.

gcd (b-a, a) if a < b.

} gcd (a%b, b) a>b

✓

Eg :- gcd (72, 24) ⇒ gcd (48, 24)

⇒ gcd (24, 24)

⇒ gcd (0, 24)

→ ∴ This will be answer (one of them became zero)

```
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

$lcm(a,b) * gcd(a,b) = a * b.$

↳ Euclid algorithm, $\boxed{lcm = \frac{a*b}{gcd}}$

Modulo Arithmetic

* $(a \% n) \Rightarrow [0 \ldots n-1]$ → set of answers.

  $10 \% 3 \Rightarrow [0,1,2]$

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

* Generally to avoid overflow, while storing Integer, we do modulo with a large numbers.

a) $(a+b) \% M = a \% M + b \% M$

b) $a \% M - b \% M = (a-b) \% M$

c) $((a \% M) \% M) \% M = a \% M$

d) $(a \% M * b \% M) \% M = (a*b) \% M$

Eg: int a = 2^{11}
int b = 2^{15}
int c = a * b

(within $M = 10^9$)

$\therefore C = ((a \% M) * (B \% M)) \% M$

Fast Exponentiation

$a^b \rightarrow 2^{10} \Rightarrow \underbrace{2 \times 2 \times \ldots \times 2}_{10}$

* Simple method $\Rightarrow$ $O(b)$ $\quad (x^n)$

ans = 1

→ loop : [0 to b-1]

↳ ans = ans * a; ⟶ will get a^b.

Fast Exponentiation (a^b) $\Rightarrow$ $O(\log b)$

if b = even, $a^b = (a^{b/2})^2$ $\quad 2^{10} = (2^5)^2 = 2^{10}$

if b = odd, $a^b = (a^{b/2})^2 . a$ $\quad 2^{11} = (2^5)^2 . 2 = 2^{11}$

```
int fastExponentiation (int a, int b)
{
    int ans = 1;
    while (b > 0)
    {
        if (b & 1)
        {
            //odd
            ans = ans * a
        }
        a = a * a;  → a^2
        b = b/2;  →or→  b>>=1;  Right shift.
    }
    return ans;
}
```

Dry run: 5^4 $\quad a = 5, b = 4.$

ans = 1

b = 4 → b not odd, a = 5 × 5 = 25

b = 4/2 = 2.

b = 2 → b × odd, a = 25 × 25 = 625

b = 2/2 = 1

b = 1 → b odd ✓ ⟶ ans = 1 × a = 1 × 625

⇒ 625

a = 625 × 625

b = 1/2 < 0 ⟶ out of while loop.

∴ return ans → return 625 //