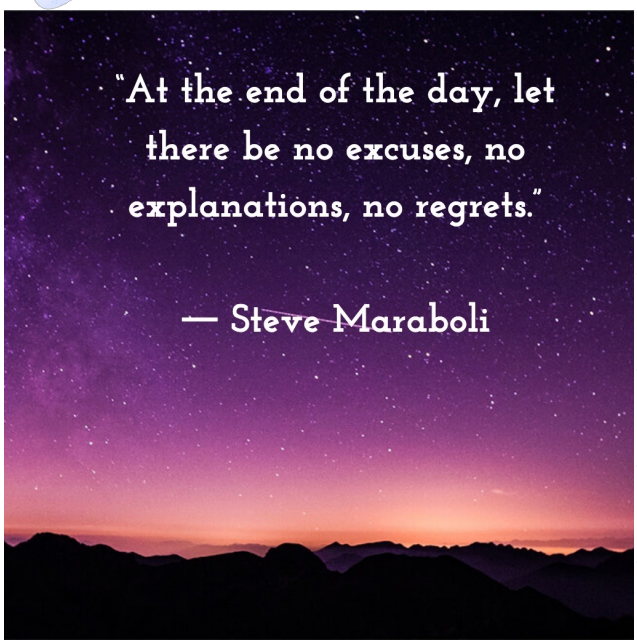
Prime Numbers



{ Doubt session on Saturday → 11 AM }

Today's content

- → Prime number Intro
- → Get all primes from 1 to N
- → Print smallest prime factor for 2 to N
- → Prime factorisation
- → Get the no. of factors/divisors

Prime no. → No. having only 2 factors (1 & itself)

Eg: 2, 3, 5, 7, 11, 13, 17, 19 ..

Q Given a no., check if it is prime or not.

Ans = Count the no. of factors

if (count > 2) → not prime

else if (count == 2) → prime.

```
boolean checkprime (int N)
{
    count = 0
    for( i=1 ; i*i <= N ; i++){
        if ( N % i == 0){
            if ( i == N/i) count ++;
            else count += 2
        }
    }
    if (count == 2) return true;
    else return false
}
```

TC: $O(\sqrt{n})$

SC: $O(1)$

02 Given a no., we need to print all the prime no. from 1 to N.

N = 10 ⟶ 2, 3, 5, 7

N = 20 ⟶ 2, 3, 5, 7, 11, 13, 17, 19

Idea → Iterate from 1 to N & check if it is prime or not.

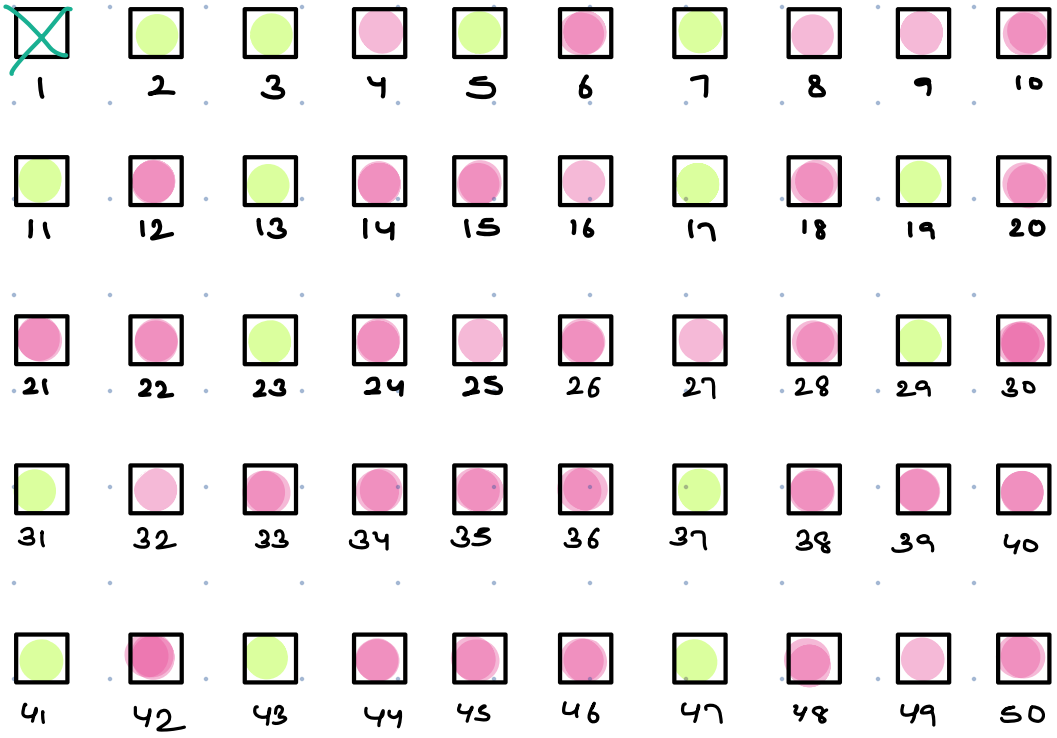
```
void printall(int N)
{
    for(i=1; i<=N; i++){
        if(checkprime(i) == True){
            print(i);
        }
    }
}
```

Tc: $O(N\sqrt{N})$
Sc: $O(1)$

Idea 2 — Sieve of eratosthenes

Given n = 50, we have to find all primes between 1 to 50.

Assumption → Every no. is a prime



If 2 is a prime, what all no. can never be the prime no.

→ Multiples of 2 will never be a prime no.

```
void printallprimes (int n)
    boolean [] prime = new int [n+1];
    // Mark every ele as True;
    Arrays.fill (prime, True);
    prime [0] = false
    prime [1] = false
```

```
for ( i = 2 ; i*i ≤ n ; i++ ) {
    if ( prime [i] == true ) {
        for ( j = i*i ; j ≤ n ; j = j+i ) {
            prime [j] = false ;
        }
    }
}

for ( i = 2 ; i ≤ N ; i++ ) {
    if ( prime [i] == True ) print ( i );
}
```

i	Multiples of i & Mark them as false
2	2*2 2*3 2*4 2*5 2*6 ...
3	3*2 3*3 3*4 3*5 3*6 ...
4	
5	5*2 5*3 5*4 5*5 5*6 ...

Obs → Instead of going on multiples from 2i, we will iterate from i*i

i [2 → n]	j = [i*i → N]
2	= n/2 iterations
3	= n/3 iterations
4	
5	= n/5 iterations
.	
.	
.	
$\sqrt{n}$	j : $[(\sqrt{n})^2 \rightarrow n]$ = 1 iteration
$\sqrt{n}+1$	j : $[(\sqrt{n}+1)^2 \rightarrow n]$ = 0 iteration
$\sqrt{n}+2$	0 iteration
.	.
.	.
.	.
n	0 iterations

$$TC = \frac{N}{2} + \frac{N}{3} + \frac{N}{5} + \frac{N}{7} + \cdots$$

$$= N\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \cdots\right)$$

sum of all reciprocals of prime no.

$= N * (\log(\log N))$

TC: $O(N \log(\log(N))$

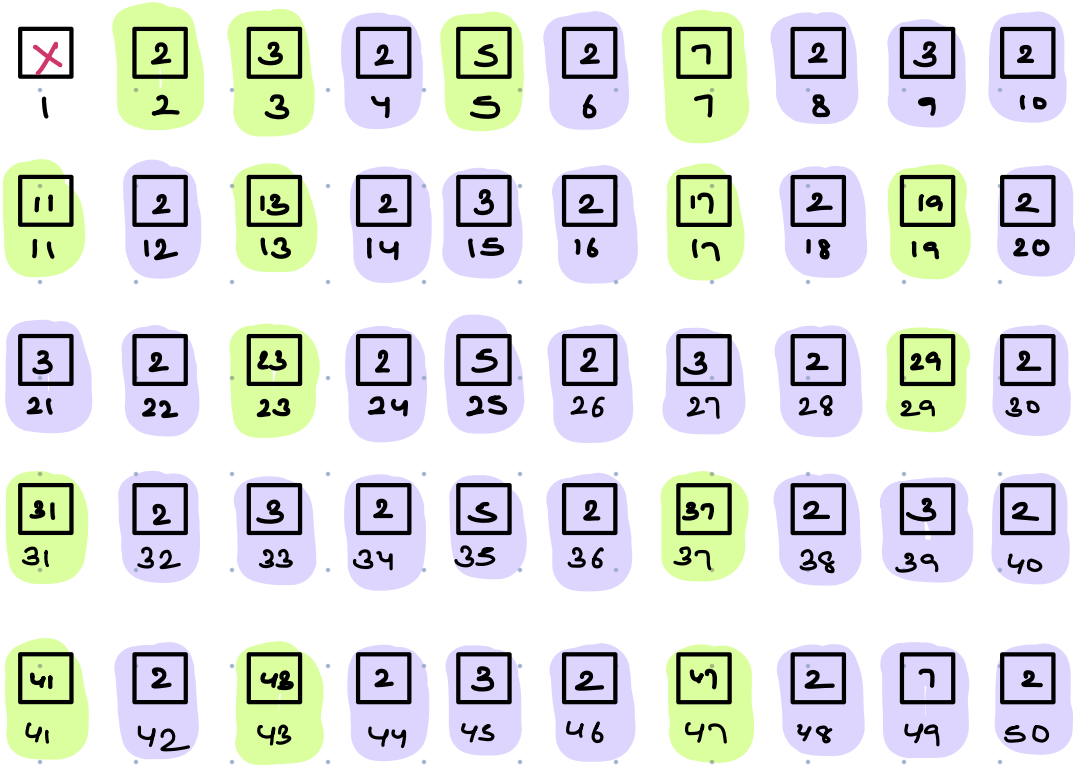
SC: $O(N)$

* Given N, return the smallest prime factor for all no. from 2 to N.

N = 10 — 2 3 4 5 6 7 8 9 10

Ans: 2 3 2 5 2 7 2 3 2

Ass → Every no. is spf of itself



Observations

01. Identify prime → if (A[i] == i) → prime na

02. we should not update ele which have been already updated → if (A[i] != i) → already updated.

```
int [] getspf (int n)
    int [] spf = new int [n+1];
        ∀ all i, spf[i] = i
    for (i = 2; i ≤ n; i++){
        spf[i] = i;
    }

    for (i = 2; i * i ≤ n; i++){
        if (spf[i] == i){   // checking if i is prime
            for (j = i * i; j ≤ n; j = j + i){
                if (spf[j] == j){
                    spf[j] = i;
                }
            }
        }
    }

    return spf;
}
```

TC: O(n log(log n))

SC: O(1)

8:13 AM → 8:23 AM

* **Prime factorisation**

→ process of finding prime no., which are multiplied together to form the given no.

n = 48

2	48
2	24
2	12
2	6
3	3
	1

$n = 48 = 2 * 2 * 2 * 2 * 3$

$= 2^4 * 3^1$

No. of factors = $(4+1) * (1+1) = 10$

Factors of 48 = 1, 2, 3, 4, 6, 8, 12, 16, 24, 48

n = 300

2	300
2	150
3	75
5	25
5	5
	1

$n = 300 \quad 2 * 2 * 3 * 5 * 5$

$= 2^2 * 3^1 * 5^2$

No. of factors = $(2+1) * (1+1) * (2+1)$

$= 3 * 2 * 3$

$= 18$

* Given a no. & its prime factors

$$n = i^{a_1} * j^{a_2} * k^{a_3}$$

No. of factors = $(a_1+1) * (a_2+1) * (a_3+1)$

* Given a no. n, for all the no. from 1 to N, get the no. of factors.

N = 10

1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1
	2	3	2	5	2	7	2	3	2
			4		3		4	9	5
					6		8		10

Ans: 1 2 2 3 2 4 2 4 3 4

N = 360

2	360
2	180
2	90
3	45
3	15

spf[360] = 2
spf[180] = 2
spf[90] = 2
spf[45] = 3

5	5
	1

spf [15] = 3 }
spf [5] = 5 }

Steps

01 get spf [ ]

$2^3 * 3^2 * 5^1$

Ans = (3+1) * (2+1) * (1+1)

= 4 * 3 * 2

= 24

```
int [] spf = getspf [N]

ans = 1
while (N > 1){
    s = spf [N]
    count = 0
    while (N % s == 0){
        N = N/s
        count ++;
    }
    ans = ans * (count + 1)
}
```

No. of factors for one value of N

TC : O (N log (log N)) + O(log n)
SC : O(n)

* HW ⇒ You have to run this process for multiple inputs.

Doubts

Whenever you are give with mod

= making some changes

ans = (ans + k) % mod ✗

ans = (ans % mod + k % mod) % mod

```
public class Solution {
    public int solve(int A, int B, int C) {
        // dp[n][r] stores the value of nCr
        int[][] dp = new int[A + 1][B + 1];
        for(int i = 0; i <= A; i++) {
            for(int j = 0; j <= Math.min(i, B); j++) {
                if(j == i || j == 0) {
                    dp[i][j] = 1;
                } else {
                    // nCr = (n - 1)C(r - 1) + (n - 1)Cr
                    dp[i][j] = (dp[i - 1][j - 1] % C + dp[i - 1][j] %
C) % C;
                }
            }
        }
        return dp[A][B] % C;
    }
}
```