Lecture :- Maths: Prime numbers

Agenda

- Introduction to prime numbers.
- Get all primes from 1 to n.
- Print smallest prime factor for 2 to n.
- Prime factorisation
- Get the number of factors / divisors.

Prime numbers

- Numbers having only 2 factors
- factors must be 1 and number itself

Example:

4 ⟶ no

7 ⟶ yes

11 ⟶ yes

19 ⟶ yes

18 ⟶ no

2 ⟶ yes [smallest prime no]

1 ⟶ no [factor = 1]

Qu: Check whether a number is prime or not?

└ No of factors = 2
1 and no itself.

```
boolean isprime(int n) {
    cnt = 0;
    for(i=1; i*i<=n; i++) {        // i <= √n
        if(n % i == 0) {
            if(i == n/i) {
                cnt += 1;
            } else {
                cnt += 2;
            }
        }
    }
    if(cnt == 2) {
        return true;
    }
    return false;
}
```

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

SC: $O(1)$

Qu Print all prime numbers from 1 to n.

n = 10 2 3 5 7

n = 20 2 3 5 7 11 13 17 19

Brute force:

```
void printAllPrimes(int n) {
O(n) —— for(i=2; i<=n; i++) {
    O(√n) —— if(isPrime(i)) {
                print(i);
            }
    }
}
```

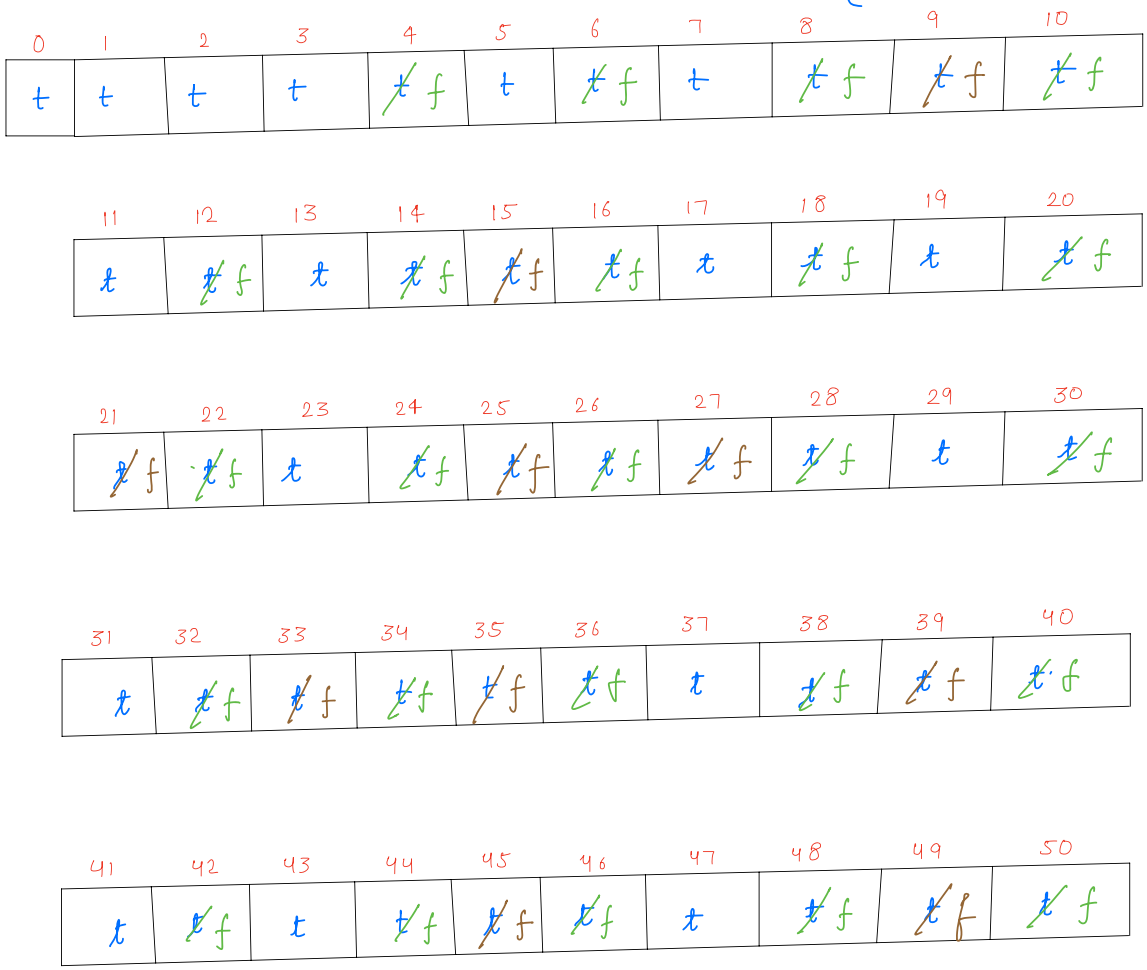
TC: $O(n\sqrt{n})$

SC: $O(1)$

Sieve of eratosthenes [Prime optimisation]

Qu Print all prime numbers from 1 to 50.

[boolean[n+1] {true}




```
i ← 2 to √n {
    if( i is prime) {
        mark all multiples of i as
        non-prime.
    }
}
```

Algorithm

```
boolean[] sieve(int n) {
    boolean[] sieve = new boolean[n+1];
    for(i = 2; i<=n; i++){
        sieve[i] = true;
    }
    for(i=2; i*i<=n; i++){
        if( sieve[i] ) {
            for(j = 2*i; j<=n; j+=i) {
                sieve[j] = false;
            }
        }
    }
    return sieve;
}

void printAllPrime(int n) {
    boolean[] s = sieve(n);
    for(i=2; i<s.length; i++) {
        if(s[i] == true) {
            print(i);
        }
    }
}
```

i=2; 4, 6, 8, 10, 12, 14, 16 --
(+2, +2)

i=3. 6, 9, 12, 15, 18, 21 --
(+3, +3)

i=5 10, 15, 20, 25, 30 ...
(+5, +5)

.....

i — 2*i , += i

Time complexity of sieve [Proof not needed]

i		j [2*i, n, j=j+=i]
2	≃	$\frac{n}{2}$ itr
3	≃	$\frac{n}{3}$ itr
4	≃	0 itr
5	≃	$\frac{n}{5}$ itr
⋮		⋮
$\sqrt{n}$	≃	$\frac{n}{\sqrt{n}}$ itr $= \sqrt{n}$ itr

```
boolean[] sieve(int n) {
    boolean[] sieve = new boolean[n+1];
    for(i=2; i<=n; i++){
        sieve[i] = true;
    }
    for(i=2; i*i<=n; i++){
        if(sieve[i]) {
            for(j=2*i; j<=n; j+=i) {
                sieve[j] = false;
            }
    }
    return sieve;
}
```

$\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \frac{n}{7} \cdots\cdots \frac{n}{\sqrt{n}}$ [approximation]

$n\left[\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} \cdots \frac{1}{\sqrt{n}}\right]$

$n * \log_2(\log_2(n))$

TC: $O(n * \log_2(\log n))$

SC: $O(n)$

Small optimization in sieve of eratosthenes

i	Multiple j	
2	4, 6, 8, 10, 12, 14, 16, 18 - ...	2*2
3	6, 9, 12, 15, 18, 21, 24 - ...	3*3
5	10, 15, 20, 25, 30, 35 - --	5*5
7	14, 21, 28, 35, 42, 49, 58 - ~	7*7
i		i*i

Updated code

```
boolean[] sieve(int n) {
    boolean[] sieve = new boolean[n+1];
    for(i = 2; i<=n; i++){
        sieve[i] = true;
    }
    for(i=2; i*i<=n; i++){
        if(sieve[i]) {
            for(j = i*i; j<=n; j+=i) {
                sieve[j] = false;
            }
        }
    }
    return sieve;
}

void printAllPrime(int n) {
    boolean[] s = sieve(n);
    for(i=2; i<s.length; i++) {
        if(s[i] == true) {
            print(i);
        }
    }
}
```

Smallest prime factor

Given n, return smallest prime factor for all numbers from 2 to n.

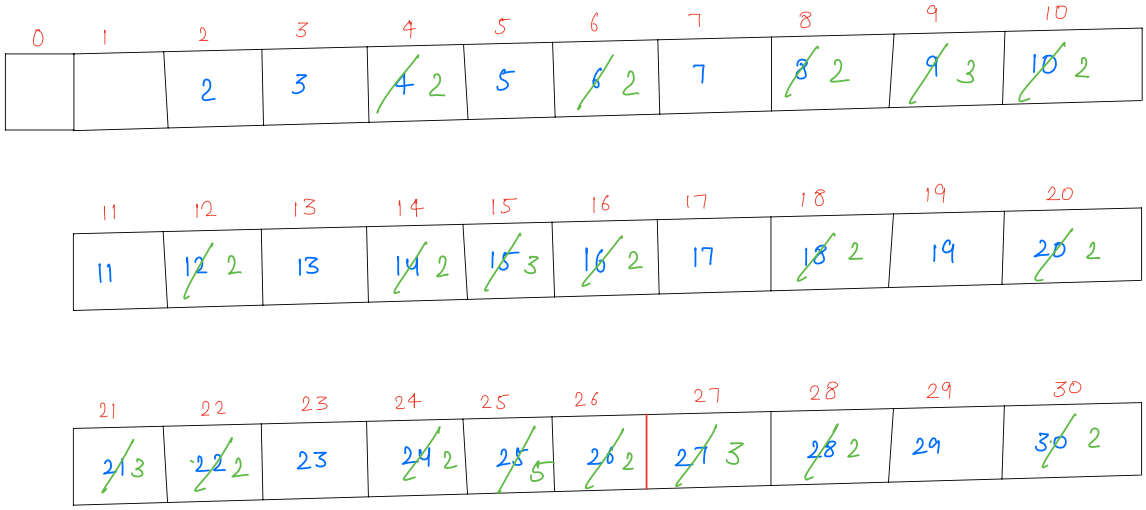
n = 10	2	3	4	5	6	7	8	9	10
	2	3	2	5	2	7	2	3	2

Observation : if (no == its smallest factor) {
prime number;
}
Exception: n == 1.

Number	Smallest prime factor
2	2
3	3
29	29
31	31
37	37
41	41
5	5

Approach

if (i == arr[i]) — prime no



int[8] ⟶ smallest prime factor for every no indicated by i

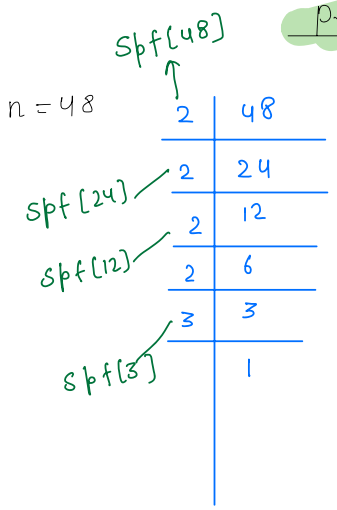
Code

```
int[] smallestPrimeFactor(int n) {
    int[] sieve = new int[n+1];
    for(i=2; i<=n; i++) {
        sieve[i]=i;
    }
    for(i=2; i*i<=n; i++) {
        if(sieve[i]==i) {
            for(j=i*i; j<=n; j+=i) {
                if(sieve[j]==j) {
                    sieve[j]=i;
                }
            }
        }
    }
    return sieve;
}
```

TC: $O(n \log_2 \log_2 n)$
SC: $O(n)$

Break: 8:28 - 8:39 AM

Prime factorisation



$\Rightarrow 2^4 * 3^1$

Trick: factors = $(4+1) * (1+1)$

$= 5 * 2 = 10$

1 2 3 4 6 8 16 12 24 48

n = 45

spf[45]	3	45
spf[15]	3	15
spf[5]	5	5
		1

$3^2 * 5^1$

factors = $(2+1) * (1+1)$

$= 6$ Ans

1, 3, 5, 15, 9, 45

Generalisation

$$n \Rightarrow p_1^{a_1} * p_2^{a_2} * p_3^{a_3} \cdots p_x^{a_x}$$

$$\text{factors} \Rightarrow (a_1+1) * (a_2+1) * (a_3+1) \cdots (a_x+1)$$

Qu Given an integer n, for all numbers from 1 to n. count no. of factors of all numbers.

Eg: number = 48. → 48 = 24

2 spf[48]

$2^4 * 3^1 \Rightarrow$

factors = (4+1) * (1+1) = 10.

24/2 = 12

12/2 = 6

6/2 = 3

3/3 = 1.

⟶ info of smallest prime factor

⟶ Map<Int, Int>

↑ smallest prime factor

↑ frequency

Code

```
int countfactors(int n) {
    int[] spf = smallestprimefactor(n);
    Map<Integer, Integer> map;
    int temp = n;
    while (temp > 1) {
        int spf = spf[temp];
        if(map.containskey(spf)) {
            int freq = map.get(spf);
            map.put(spf, freq+1);
        } else{
            map.put(spf, 1);
        }
        temp = temp / spf;
    }
    int factors = 1;
    for(int key : map.keyset()) {
        int freq = map.get(key);
        factors *= (freq+1);
    }
    return factors;
}
```

n = 48

map:

int	int
2	4
3	1

TC :-
SC :-

Thankyou

Doubts

```
int spf = 2;
while ( temp > 1 ) {
    if ( temp % spf == 0 ) {
        ≡≡≡  temp = temp / spf;
    } else {
        spf++;
    }
    temp = temp / spf;
}
```

temp = 10^9+7

(2)
3
4
5
6
⋮
10^9+7

1 — n.

10^9+7 ⟶ 10^9+7 iters

$+6$ — $+6$ iters

5 — $+5$ iters

$(10^9+7) * ($ ⋮ $)$ ✓