
FACE
Prep

Sieve of Eratosthenes

Sieve of Eratosthenes

Let’s say the given number is 100.

We need to print all prime numbers till 100.

Starting from 2, check for each number till it’s root i.e., 10.

Whenever we get a prime number, delete all of its multiple till 100.

Repeat the process.

In this technique, we are sieving out the prime numbers from 2 to 100.

Sieve of Eratosthenes

This is a simple representation of how the technique works !

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

```
1   import java.util.*;
2   class Main {
3       static void sieveOfEratosthenes(int n)
4       {
5           boolean prime[] = new boolean[n+1];
6           for (int i = 0; i <= n; i++)
7               prime[i] = true;                    //O(n)
8           for (int p = 2; p * p <= n; p++)
9           {
10              if (prime[p] == true)
11              {
12                  for (int i = p * p; i <= n; i += p) //O(n log log n)
13                      prime[i] = false;
14              }
15          }
16          for (int i = 2; i <= n; i++)
17          {
18              if (prime[i] == true)
19                  System.out.print(i + " ");
20          }
21      }
22
```

```
23 public static void main(String args[])
24     {
25         Scanner sc=new Scanner(System.in);
26         int n=sc.nextInt();
27         sieveOfEratosthenes(n);
28     }
29 }
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
```

Segmented Sieve

Segmented Sieve

The idea of a segmented sieve is to divide the range [0..n-1] in different segments and compute primes in all segments one by one.

This algorithm first uses Simple Sieve to find primes smaller than or equal to √(n).

Below are steps used in Segmented Sieve.

- Use Simple Sieve to find all primes up to the square root of ‘n’ and store these primes in an array “prime[]”. Store the found primes in an array ‘prime[]’.
- We need all primes in the range [0..n-1]. We divide this range into different segments such that the size of every segment is at-most √n
- Do following for every segment [low..high]
  - Create an array mark[high-low+1]. Here we need only O(x) space where **x** is a number of elements in a given range.
  - Iterate through all primes found in step 1. For every prime, mark its multiples in the given range [low..high].

```
1  import java.util.*;
2  import static java.lang.Math.sqrt;
3  import static java.lang.Math.floor;
4  class Main
5  {
6      static void simpleSieve(int limit, Vector<Integer> prime){
7          boolean mark[] = new boolean[limit+1];
8          for (int i = 0; i < mark.length; i++)
9              mark[i] = true;
10         for (int p=2; p*p<limit; p++){
11             if (mark[p] == true){
12                 for (int i=p*p; i<limit; i+=p)
13                     mark[i] = false;
14             }
15         }
16         for (int p=2; p<limit; p++){
17             if (mark[p] == true){
18                 prime.add(p);
19                 System.out.print(p + " ");
20             }
21         }
22     }
```

```
23 static void segmentedSieve(int n)
24     {
25         int limit = (int) (floor(sqrt(n))+1);
26         Vector<Integer> prime = new Vector<>();
27         simpleSieve(limit, prime);
28         int low  = limit;
29         int high = 2*limit;
30         while (low < n)
31         {
32             if (high >= n)
33                 high = n;
34             boolean mark[] = new boolean[limit+1];
35             for (int i = 0; i < mark.length; i++)
36                 mark[i] = true;
37             for (int i = 0; i < prime.size(); i++)
38             {
39                 int loLim = (int) (floor(low/prime.get(i)) * prime.get(i));
40                 if (loLim < low)
41                     loLim += prime.get(i);
42                 for (int j=loLim; j<high; j+=prime.get(i))
43                     mark[j-low] = false;
44             }
```

```
45   for (int i = low; i<high; i++)
46                   if (mark[i - low] == true)
47                       System.out.print(i + " ");
48               low  = low + limit;
49               high = high + limit;
50           }
51       }
53       public static void main(String args[])
54       {
55           Scanner sc=new Scanner(System.in);
56           int n = sc.nextInt();
57           segmentedSieve(n);
58       }
59   }
60
61
62
63
64
65
66
```

Incremental Sieve

Incremental Sieve Pseudocode

```
for each number
        for each prime
                loop while current multiple < number
                        get next multiple of the prime
                end
                if number == current multiple
                        go to next number
                end
        end
        add number to list of primes
end
```

THANK YOU

FACE Prep