Sieve of Eratosthenes

Given a number n, sieve of eratosthenes can be used to find all prime numbers less than or equal to given number.

Following is the algorithm to find all the prime numbers less than or equal to a given integer *n* by Eratosthenes' method:

1. Create a list of consecutive integers from 2 to *n*: (2, 3, 4, …, *n*).
2. Initially, let *p* equal 2, the first prime number
3. Starting from *p*, count up in increments of *p* and mark each of these numbers greater than *p* itself in the list. These numbers will be 2*p*, 3*p*, 4*p*, etc.; note that some of them may have already been marked.
4. Find the first number greater than *p* in the list that is not marked. If there was no such number, stop. Otherwise, let *p* now equal this number (which is the next prime), and repeat from step 3.

Code:

```
boolean prime[] = new boolean[n+1];
for(int i=0;i<n;i++)
    prime[i] = true;

for(int p = 2; p*p <=n; p++){
    // If prime[p] is not changed, then it is a prime
    if(prime[p] == true){
        // Update all multiples of p
        for(int i = p*2; i <= n; i += p)
            prime[i] = false;
    }
}
// Print all prime numbers
for(int i = 2; i <= n; i++){
    if(prime[i] == true)
        System.out.print(i + " ");
}
```

Problems with Simple Sieve:

The Sieve of Eratosthenes looks good, but consider the situation when n is large,, the Simple Sieve faces following issues.

- An array of size Θ(n) may not fit in memory
- The simple Sieve is not cache friendly even for slightly bigger n. The algorithm traverses the array without locality of reference

Segmented Sieve

The idea of 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 $\sqrt{}$(n). Below are steps used in Segmented Sieve.

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

   And at the end whichever is unmarked is prime number in that segment.

Problems

https://www.hackerearth.com/practice/math/number-theory/basic-number-theory-2/practice-problems/algorithm/sum-of-primes-7/

https://www.hackerearth.com/practice/math/number-theory/primality-tests/practice-problems/algorithm/roy-and-rangoli-1/description/

http://www.spoj.com/problems/PRIME1/

https://www.codechef.com/APRIL17/problems/CHEFDIV