CS 355, Fall, 2019, Project 1

The Sieve of Eratosthenes finds the primes less than some limit J. It begins by writing the numbers 1, 2, …, J. Cross out the number 1, which is not prime. After that, let p be the first number not crossed out. Then p is prime, so leave it intact, but cross out every p-th number (including any previously crossed out) starting with $2p$. That is, cross out all multiples of p strictly larger than p. Repeat this process, replacing p by the next number not yet crossed out, so long as $p \leq \sqrt{J}$. Then stop. All numbers crossed out are composite (or 1) and all numbers not crossed out are prime. This algorithm works because every composite number $\leq J$ has a prime factor $p \leq \sqrt{J}$ and so it would be crossed out as a multiple of p.

Example: Let $J = 19$. Then $\sqrt{J} \approx 4.4$. Write the numbers 1 to 19. Cross out 1 and all multiples of 2 starting with 4. We cross out (/) 4, 6, 8, 10, 12, 14, 16, 18. After that, 3 is the next number not crossed out, so cross out all multiples of 3 starting with 6. We cross out (\) 6, 9, 12, 15, 18. The next number not crossed out is 5, which is greater than $\sqrt{19}$, so we stop. The numbers not crossed out are 2, 3, 5, 7, 11, 13, 17, 19, exactly the primes between 1 and 19.

$$\cancel{1}\ \ 2\ \ 3\ \ \cancel{4}\ \ 5\ \ \xcancel{6}\ \ 7\ \ \cancel{8}\ \ \bcancel{9}\ \ \cancel{1}0\ \ 11\ \ 1\xcancel{2}\ \ 13\ \ \cancel{1}4\ \ \bcancel{1}5\ \ \cancel{1}6\ \ 17\ \ 1\xcancel{8}\ \ 19$$

A computer program would use an array (of bits or bytes, say) to represent the numbers 1 to J. (Or remember the prime 2 and let the bytes represent ODD numbers 1, 3, 5, … to save $J/2$ bytes.) Let the value "1" mean that the number is not crossed out and the value "0" mean that the number is crossed out. The algorithm begins by marking the byte that represents 1 as "crossed out" and the bytes that represent the other numbers as "not crossed out." The first inner **while** loop crosses out all multiples of the prime p. The second inner **while** loop finds the next prime p. The outer **while** loop runs through all primes $\leq \sqrt{J}$. Here is the algorithm.

Sieve of Eratosthenes

```
Input: An integer $J > 1$.
$P[1] = 0$
**for** ($i = 2$ to $J$) { $P[i] = 1$ }
$p = 2$
**while** ($p \leq \sqrt{J}$) {
    $i = p + p$
    **while** ($i \leq J$) { $P[i] = 0$; $i = i + p$ }
    $i = p + 1$
    **while** ($i \leq \sqrt{J}$ **and** $P[i] = 0$) { $i = i + 1$ }
    $p = i$
    }
Output: The array $P[\cdot]$ lists the primes $\leq J$.
```

When the algorithm finishes, the value of $P[i]$ is 1 if i is prime and 0 if i is 1 or composite.

The Sieve of Eratosthenes finds all primes $\leq J$ in $O(J \log\log J)$ steps.

(A) Write a program in Java for the Sieve of Eratosthenes. Use it to make a table of all the primes $< 10^5$. (Don't turn in anything for part (A); just do it. You need it for part (B).)

NOTE: Do not copy your program from the Web. I looked at many programs for the Sieve of Eratosthenes on the Web and nine out of ten are wrong.

A variation of the Sieve of Eratosthenes finds all primes in an interval $[I, J]$. First write the integers in the interval. Let $\mathcal{P}$ be the set of all primes less than the square root of the upper endpoint J of the interval, calculated by the first Sieve of Eratosthenes above. Then, for each prime p in the set $\mathcal{P}$, cross out every multiple of p in the interval. The set of numbers not crossed out is the answer. This algorithm assumes that $I > \sqrt{J}$.

Modified Sieve of Eratosthenes

```
Input: Integers $J > I > 1$ and a finite set $\mathcal{P}$ of primes.
**for** ($i = I$ to $J$) { $A[i] = 1$ }
**for each** $p \in \mathcal{P}$ {
    $i =$ the smallest multiple of $p$ that is $\geq I$
    **while** ($i \leq J$) { $A[i] = 0$; $i = i + p$ }
    }
Output: The array $A[\cdot]$ lists the numbers between $I$ and $J$ free of factors in $\mathcal{P}$.
```

Do NOT use a loop to find the smallest multiple of p that is $\geq I$. Recall what I said about integer arithmetic in class to find this number with a simple assignment statement. (Hint: Divide some `int` by p and then multiply it by p.)

When the algorithm finishes, $A[i] = 0$ if some prime $p \in \mathcal{P}$ divides i and $A[i] = 1$ if no prime in $\mathcal{P}$ divides i. In case $I > \sqrt{J}$ and $\mathcal{P}$ is the set of all primes $< \sqrt{J}$, when the algorithm finishes, the value of $A[i]$ is 1 if i is prime and 0 if i is composite.

(B) Write a program in Java for the Modified Sieve of Eratosthenes. Read two even integers I and J from stdin. These integers will satisfy $10^5 < I < J < 2 * 10^9$ and $J - I \leq 10^7$. There will be more than four primes between I and J. Use your program to make a table of all the primes between I and J. Count the primes in your table. Print this count. Print the first 2 primes and the last 2 primes in this table. Your program must run for less than 1 second. If you use some method other than the one described above to find the primes between I and J, your program will probably run for longer than 1 second (and be ingloriously cut off by Vocareum). Your program must have no array larger than 10^7 bytes.

Example: If your program reads these I and J:

```
1000000 2000000
```

then it should print

```
70435
1000003 1000033
1999979 1999993
```

exactly as shown (because Vocareum just compares character strings).

Example: If your program reads these I and J:

```
1000000000 1001000000
```

then it should print

```
48155
1000000007 1000000009
1000999943 1000999949
```

exactly as shown (because Vocareum just compares character strings).

Name your prgram for part (B) `sieve.java`. Submit your program for part (B) to Vocareum by 11:59 PM on the due date. It will be compiled and run ten times with ten different secret input pairs I, J. It must run each time in less than 1 second.