Prime Sieve Notes

The prime sieve allows us to efficiently generate a list of primes upto some given integer, n. In works as follows:

1) Write down all the numbers from 2 to n.
2) Go through each number, in order.
3) For each of these, if it's not crossed off, circle it.
4) Then, cross off each multiple of that number. Thus, when we circle 2 at the beginning of the algorithm, then cross off 4, 6, 8, 10, and so forth, until we get to the last even number less than or equal to n.

The numbers not crossed off at the end of these (the circled ones) are all the primes in the range.

One note:

If a number, n, is NOT prime, then we know there exist a and b such that $1 < a \leq b$ and

$n = a \times b$

In the equality case, we have:

$n = a \times a = a^2$, so $a = \sqrt{n}$

This corresponds to the largest value that a could be because if it were any bigger, it would no longer be smaller than b!

Thus, if a number is composite (not prime), then it has at least one factor less than or equal to its square root. So, we can technically stop our outer loop when we get to the square root of the number we are checking.

Here is a code segment that implements a basic prime sieve for all primes upto MAX-1:

```
boolean[] isPrime = new boolean[MAX];
Arrays.fill(isPrime, true);
isPrime[0] = false;
isPrime[1] = false;

for (int i=2; i<MAX; i++)
      for (int j=2*i; j<MAX; j+=i)
            isPrime[j] = false;
```

If a number is already crossed off, there is no need to cross off its multiples. For example, when we get to 6, all of its multiples were crossed off when we circled 2, so there's no need to cross them off again. Can you think about what to edit in the code above to skip these unnecessary loop

iterations? Also, what could you edit to stop looking for factors after you reach the square root of MAX?

After implementing the prime sieve, we are left with a boolean array such that sieve[i] is set to true if and only if i is prime. For some questions, this formation of the data is good enough to solve problems. For other problems, it's necessary to have a list of integers (or an array) with the prime numbers in successive order.

Here is a code segment that places all of the prime numbers identified by the sieve in an ArrayList, in numerical order:

```
ArrayList<Integer> primes = new ArrayList<Integer>();
for (int i=2; i<MAX; i++)
    if (isPrime[i])
        primes.add(i);
```

One of these two storage methods should suffice for most problems that require the generation of all primes up to some bound.