204. Count Primes

问题描述：**Description:**

Count the number of prime numbers less than a non-negative number, ***n***.

（注意：包括0但不包括n）

提示：

Hint1:

Let's start with a *isPrime* function. To determine if a number is prime, we need to check if it is not divisible by any number less than *n*. The runtime complexity of *isPrime* function would be O(*n*) and hence counting the total prime numbers up to *n* would be O(*n*2). Could we do better?

Hint2:

As we know the number must not be divisible by any number > *n* / 2, we can immediately cut the total iterations half by dividing only up to *n* / 2. Could we still do better?

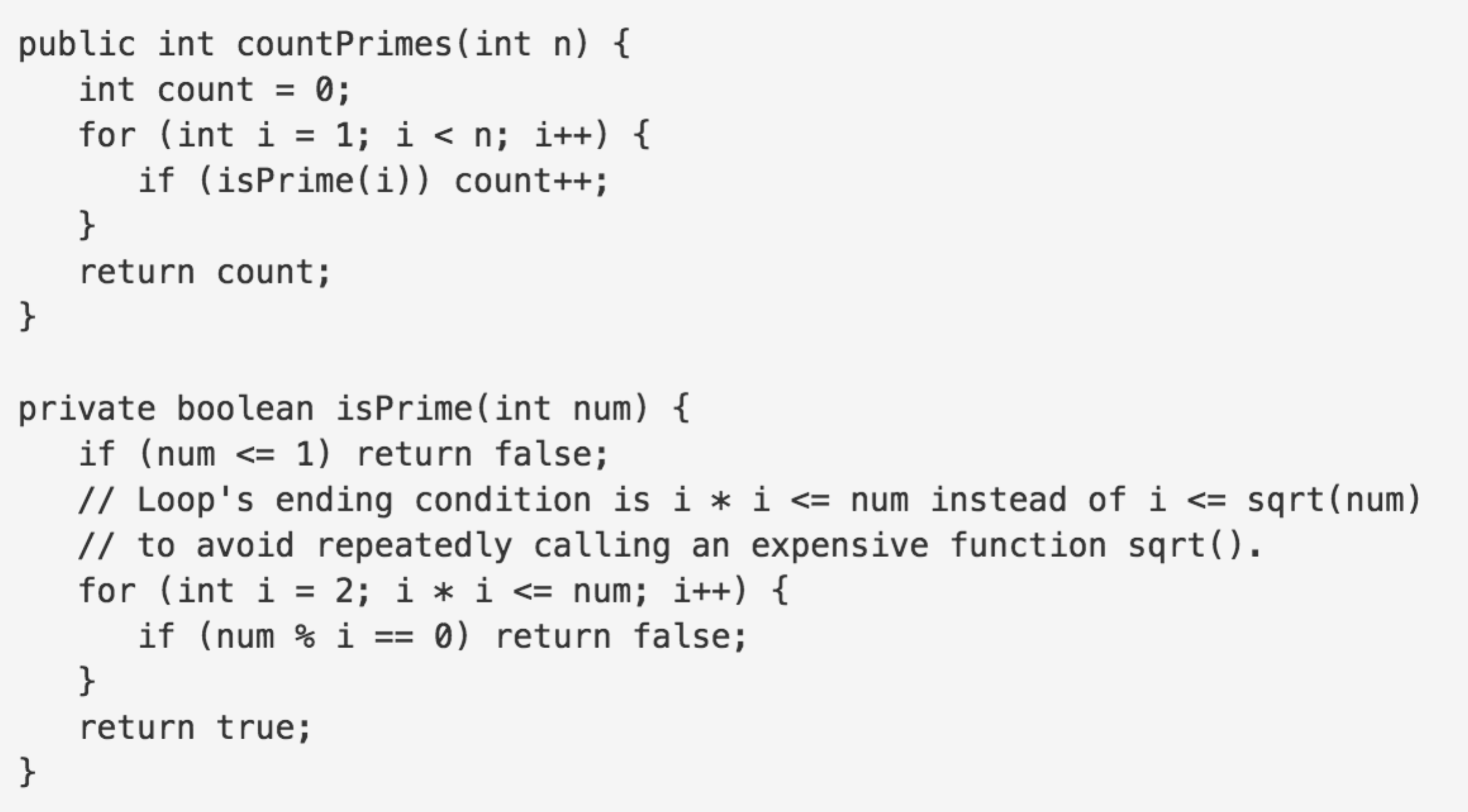
Hint3：

Let's write down all of 12's factors:

2 × 6 = 12 3 × 4 = 12 4 × 3 = 12 6 × 2 = 12

As you can see, calculations of 4 × 3 and 6 × 2 are not necessary. Therefore, we only need to consider factors up to √*n* because, if *n* is divisible by some number *p*, then *n* = *p* × *q* and since *p* ≤ *q*, we could derive that *p* ≤ √*n*.

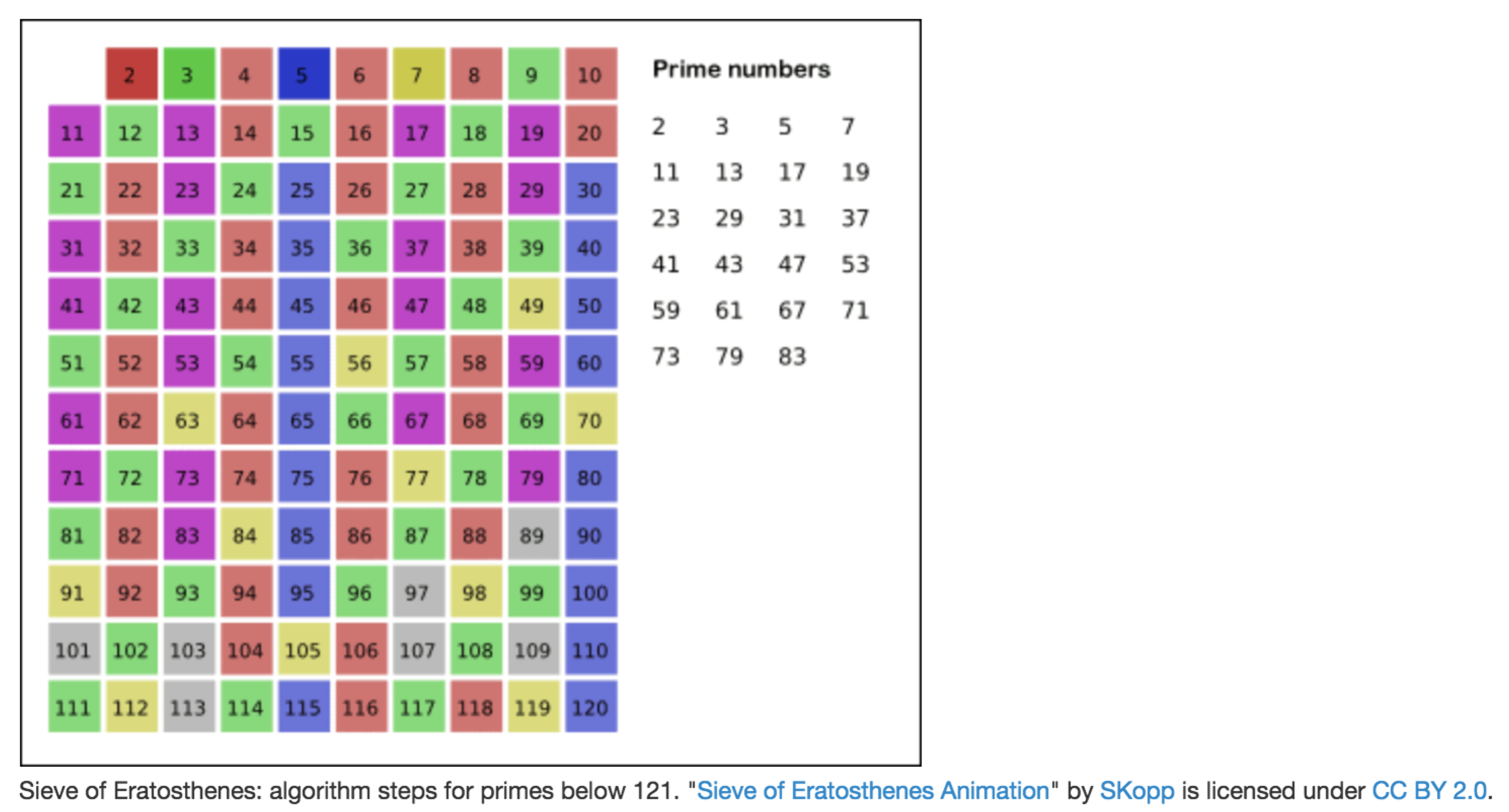
Our total runtime has now improved to O（n1.5）, which is slightly better. Is there a faster approach?



Hint4:

The [Sieve of Eratosthenes](http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes" \t "_blank) is one of the most efficient ways to find all prime numbers up to *n*. But don't let that name scare you, I promise that the concept is surprisingly simple.

We start off with a table of *n* numbers. Let's look at the first number, 2. We know all multiples of 2 must not be primes, so we mark them off as non-primes. Then we look at the next number, 3. Similarly, all multiples of 3 such as 3 × 2 = 6, 3 × 3 = 9, ... must not be primes, so we mark them off as well. Now we look at the next number, 4, which was already marked off. What does this tell you? Should you mark off all multiples of 4 as well?



Hint5:

4 is not a prime because it is divisible by 2, which means all multiples of 4 must also be divisible by 2 and were already marked off. So we can skip 4 immediately and go to the next number, 5. Now, all multiples of 5 such as 5 × 2 = 10, 5 × 3 = 15, 5 × 4 = 20, 5 × 5 = 25, ... can be marked off. There is a slight optimization here, we do not need to start from 5 × 2 = 10. Where should we start marking off?

Hint6:

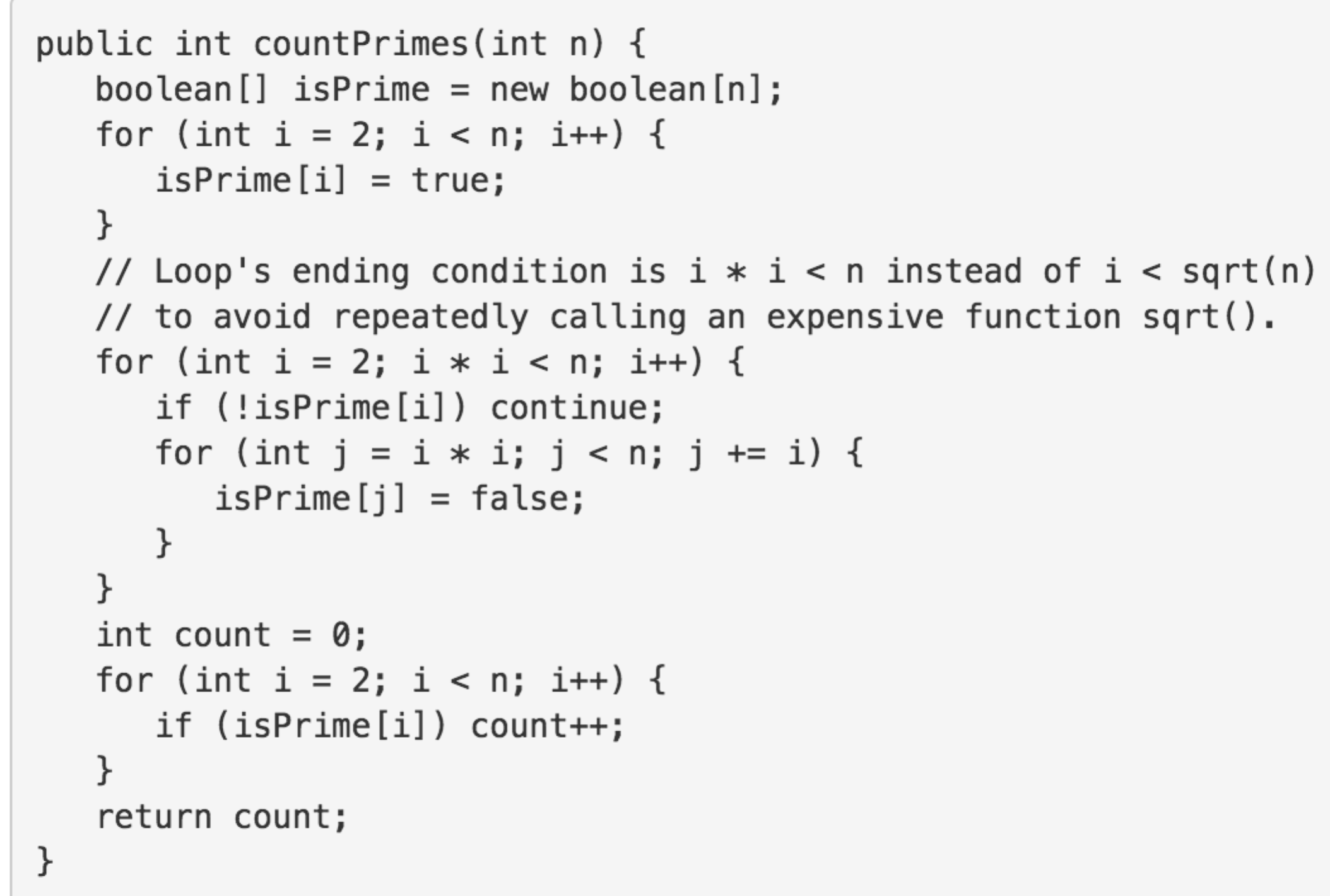
In fact, we can mark off multiples of 5 starting at 5 × 5 = 25, because 5 × 2 = 10 was already marked off by multiple of 2, similarly 5 × 3 = 15 was already marked off by multiple of 3. Therefore, if the current number is *p*, we can always mark off multiples of *p* starting at *p*2, then in increments of *p*: *p*2 + *p*, *p*2 + 2*p*, ... Now what should be the terminating loop condition?

Hint7:

It is easy to say that the terminating loop condition is *p* < *n*, which is certainly correct but not efficient. Do you still remember *Hint #3*?

Yes, the terminating loop condition can be *p* < √*n*, as all non-primes ≥ √*n* must have already been marked off. When the loop terminates, all the numbers in the table that are non-marked are prime.

The Sieve of Eratosthenes uses an extra O(*n*) memory and its runtime complexity is O(*n* log log *n*). For the more mathematically inclined readers, you can read more about its algorithm complexity on [Wikipedia](http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes" \l "Algorithm_complexity" \t "_blank).



Loop's ending condition is i \* i < n instead of i < sqrt(n) to avoid repeatedly calling an expensive function sqrt().

利用i\*i<n取代i<sqrt(n),可以大大提高性能。

代码思路：

首先：定义一个长度为n的boolean数组，默认为false，再利用一个循环，使得2以后的设值为ture；

其次：嵌套循环。外循环：质数p，从2到p\*p<n；

内循环：需要判断p是否质数，不是的话直接退出循环，继续下一个p；

从j=p\*p开始到p\*p+k\*p<n；把j索引下设为false；

最后，统计所有值为true的数目。

该算法的复杂度为O（nloglogn）

Java代码：

public int countPrimes(int n) {

boolean[] isPrime = new boolean[n];//默认为false

for (int i = 2; i < n; i++) {

isPrime[i] = true;

}

for (int i = 2; i \* i < n; i++) {

if (!isPrime[i]) continue;

for (int j = i \* i; j < n; j += i) {

isPrime[j] = false;

}

}

int count = 0;

for (int i = 2; i < n; i++) {

if (isPrime[i]) count++;

}

return count;

}

思路总结：

（1） 判断一个数n是否是质数，利用小于n的数去依次整除n，这样判断一个数是否为质数的复杂度为O（n）；若要判断小于n的质数个数，则复杂度为O（n2）；

（2）经发现，我们只需要利用小于n/2的数去依次整除n，这样复杂度为O（n2/2）；

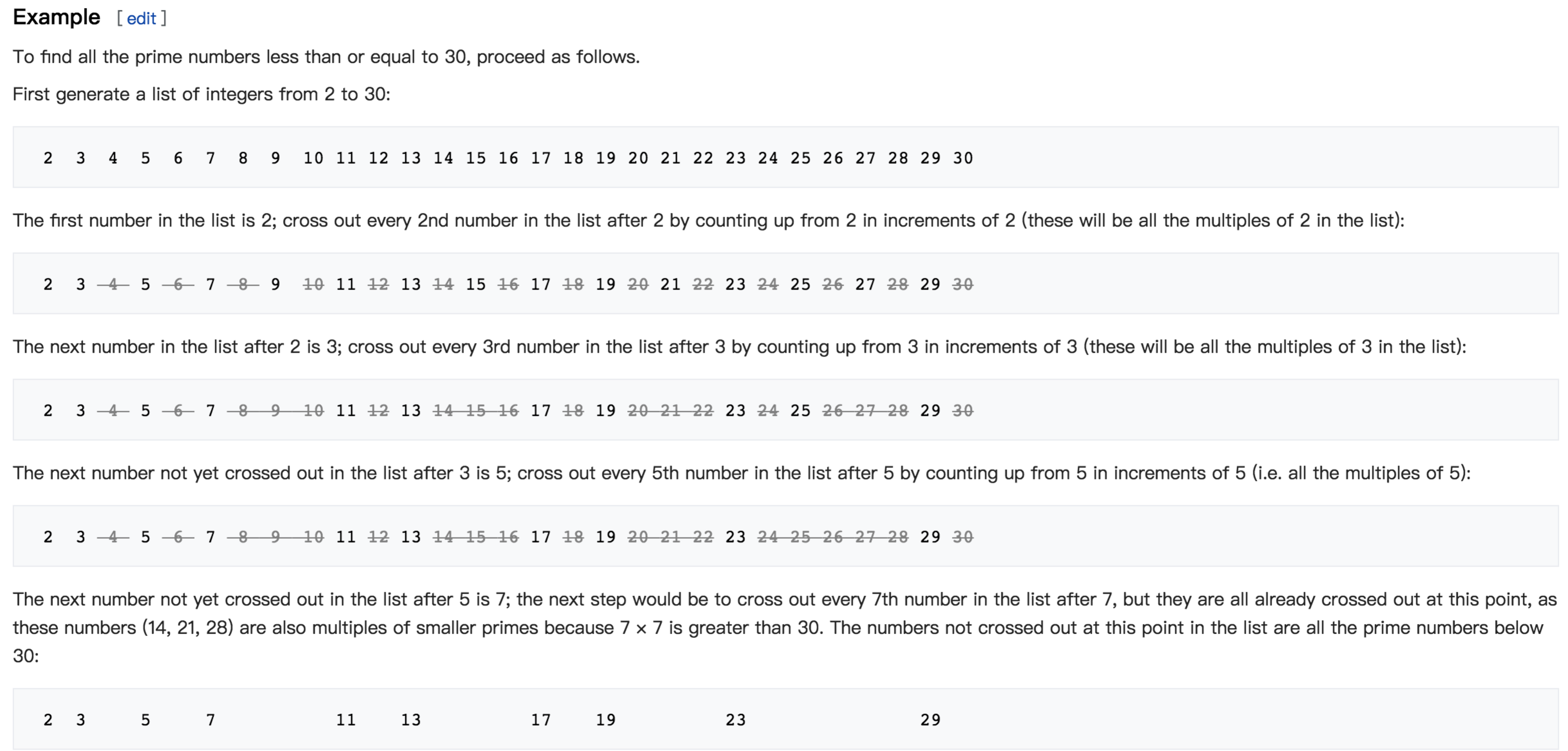
（3）根据Hint3可知，我们只需要利用小于sqrt（n）去依次整除n即可，这样复杂度为O（n1.5）；

（4）最有效的算法：The [Sieve of Eratosthenes](http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes" \t "_blank)  埃拉托斯特尼筛法

英文介绍：比较详细

<https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes>

中文：（不够详细）<https://zh.wikipedia.org/wiki/%E5%9F%83%E6%8B%89%E6%89%98%E6%96%AF%E7%89%B9%E5%B0%BC%E7%AD%9B%E6%B3%95>



该算法基本思想：

筛选法：将所有的合数找出来，然后剩下的数就是质数了；因此这里最重要的就是寻找合数的方法：利用小于sqrt（n）的质数p，它的倍数2p、3p、、、、等都是合数，但是这种筛选会出现很多重叠的，如3X5与5X3，因此为了缩小重复度，从p的平方起步，即p\*p，p\*p+p，p\*p+2p、、、、、，循环终止条件为p\*p + k\*p<n。p是经过判断的质数，循环终止条件为p<sqrt(n)。

还有注意一下：利用i<sqrt(n)性能不如i\*i<n。