New Mexico Supercomputing Challenge
Mini-Challenge Problem #2: Squarefree Numbers

Problem Description [1]

A positive integer n is called *squarefree* if no square of a prime number divides n evenly. For example, 1, 2, 3, 5, 6, 7, 10, 11 are squarefree, but not 4, 8, 9, 12 (the square of 2 divides 4, 8, and 12, while the square of 3 divides 9).

How many squarefree numbers are there below 2^{50} ?

Answer

There are 684,465,067,343,069 squarefree numbers below 2^{50} .

Möbius Function

There are a few mathematically-oriented languages and libraries (e.g. Mathematica [2]) that implement the *Möbius function*, $\mu(n)$. This a function on positive integers, defined as follows:

$$\mu(n) = \begin{cases} 1, \text{ if } n = 1 \\ 1, \text{ if } n \text{ is squarefree, with an even number of distinct prime factors} \\ -1, \text{ if } n \text{ is squarefree, with an odd number of distinct prime factors} \\ 0, \text{ if } n \text{ isn't squarefree} \end{cases}$$

For those languages where the Möbius function is available, it makes sense (and requires only a few lines of code) to use it on this problem. In most cases, however, that function isn't available, and it's more efficient to solve this problem using the *inclusion-exclusion principle*, rather than implementing a general-purpose Möbius function. (Even when the Möbius function is available, a well-implemented solution based on inclusion-exclusion generally outperforms Möbius.)

Inclusion-Exclusion Principle

The inclusion-exclusion principle states that when computing the *cardinality* (the number of elements) of the union of multiple finite sets, we can compensate for double-counting by summing the cardinalities of all of the individual sets, then subtracting the sum of the cardinalities of all two-set intersections, then adding the sum of the cardinalities of all three-set intersections, subtracting the sum of cardinalities of all four-set intersections, and so on [3].

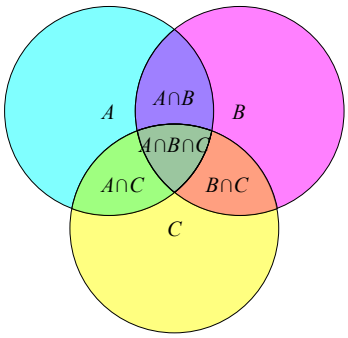

Figure 1: Overlapping sets

In figure 1, for example, it's clear that adding the contents of A , B , and C together will double-count the contents of $A \cap B$, $A \cap C$, and $B \cap C$. On the other hand adjusting for this by subtracting the contents of these three intersections will result in the contents of $A \cap B \cap C$ not being counted at all – so we need to add the contents of that intersection back in.

Symbolically stated, the inclusion-exclusion principle is as follows [3]:

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n}\left|A_i\right| - \sum_{i,\,j:\,1\le i\le j\le n}\left|A_i\cap A_j\right| + \sum_{i,\,j,\,k:\,1\le i\le j\le k\le n}\left|A_i\cap A_j\cap A_k\right| - \cdots + (-1)^{n-1}\left|A_1\cap\cdots\cap A_n\right|$$

Let's look at a concrete example: How many positive integers below 1000 are integral multiples of 2, 3, or 5?

Let's refer to the set of multiples of 2 as S_2, multiples of 3 as S_3, and multiples of 5 as S_5. We can compute the cardinalities of the individual sets by:

$$\begin{aligned}\left|S_2\right| &= \left\lfloor\frac{999}{2}\right\rfloor = 499\\ \left|S_3\right| &= \left\lfloor\frac{999}{3}\right\rfloor = 333\\ \left|S_5\right| &= \left\lfloor\frac{999}{5}\right\rfloor = 199\end{aligned}$$

We can add these values together to get 1031, but that obviously overstates the cardinality of $S_2\cup S_3\cup S_5$. For one thing, the elements in $S_2\cap S_3$ are members of both S_2 and S_3, and have thus been counted twice. To correct for this type of problem, we'll subtract the cardinality of $S_2\cap S_3$, $S_2\cap S_5$, and $S_3\cap S_5$ from the total, so that those elements end up being counted only once. But now we're under-counting: the elements of $S_2\cap S_3\cap S_5$ were originally triple-counted, but they've been subtracted three times, so they aren't counted at all; thus, we need to add them back in.

$$\begin{aligned}\left|S_2\cup S_3\cup S_5\right| &= \left(\left|S_2\right|+\left|S_3\right|+\left|S_5\right|\right) - \left(\left|S_2\cap S_3\right|+\left|S_2\cap S_5\right|+\left|S_3\cap S_5\right|\right) + \left|S_2\cap S_3\cap S_5\right|\\ &= \left(\left\lfloor\frac{999}{2}\right\rfloor+\left\lfloor\frac{999}{3}\right\rfloor+\left\lfloor\frac{999}{5}\right\rfloor\right) - \left(\left\lfloor\frac{999}{6}\right\rfloor+\left\lfloor\frac{999}{10}\right\rfloor+\left\lfloor\frac{999}{15}\right\rfloor\right) + \left\lfloor\frac{999}{30}\right\rfloor\\ &= (499+333+199) - (166+99+66) + 33\\ &= 733\end{aligned}$$

To use this approach with the mini-challenge problem, we'll need a method for finding all the prime numbers whose squares are less than 2^{50}. We'll also need a reasonably simple way of enumerating all of the multiple-set intersections, so that we can compute the cardinality of each. These may seem like daunting tasks, with numbers this large; however, if we tackle the problem in a logical fashion, it won't be that difficult.

Sieve of Eratosthenes

First, we'll use the Sieve of Eratosthenes to build a list of prime numbers [4]. (This is a classic technique that every programmer should learn.)

We begin by listing all of the numbers from 2 to some upper limit (99, in this example) in order:

		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

The algorithm proceeds as follows:

1. L = ordered list of prime candidates
2. u = maximum value in L
3. $p = 2$
4. For all m in $L \cap \left\{ p^2,\ p^2+p,\ p^2+2p,\ \ldots,\ p\left\lfloor \frac{u}{p} \right\rfloor \right\}$, do:
   a. Remove m from L.
5. If a number larger than p and less than or equal to $\lfloor \sqrt{u} \rfloor$ is still in L, then:
   a. p = next value in L larger than p
   b. Go to step #4.
6. Finished; the numbers remaining in L are prime.

In this example, we start with 2, and remove $\{4, 6, 8, \ldots, 98\}$ – i.e. all of the multiples of 2, starting with 2^2 – from the list:

		2	3		5		7		9		11		13		15		17		19
	21		23		25		27		29		31		33		35		37		39
	41		43		45		47		49		51		53		55		57		59
	61		63		65		67		69		71		73		75		77		79
	81		83		85		87		89		91		93		95		97		99

Next, we would advance to 3 (the next higher number still in the list), and remove all of $\{9, 12, 15, \ldots, 99\}$ (i.e. the multiples of 3, starting with 9) that still remain in the list:

	2	**3**		5		7			11		13			17		19
		23		25				29	31				35	37		
	41		43			47		49			53		55			59
	61				65		67			71		73			77	79
			83		85			89		91			95		97	

Now we'll advance to 5, and remove all its multiples, starting with 25:

	2	3		**5**		7			11		13			17		19
		23						29	31					37		
	41		43			47		49			53					59
	61					67				71		73			77	79
			83					89		91					97	

On to 7:

	2	3		5		**7**			11		13			17		19
		23						29	31					37		
	41		43			47					53					59
	61					67				71		73				79
			83					89							97	

At this point, no numbers greater than 7 and less than or equal to $\lfloor \sqrt{99} \rfloor$ remain in the list. So the algorithm stops, and all of the remaining numbers are prime.

In an implementation of the Sieve of Eratosthenes, we use the data structures and syntax appropriate to our chosen language for storing and manipulating the list of numbers. For example, in Python, we'd use a list; in Java or C++, we'd use a bitset. In Python, we might use *list comprehension* (a compact, elegant syntax for expressing operations on the contents of a list), *extended slices* (used for referencing a subset of regularly spaced elements in a list), and *lambdas* and/or *generators* (special types of functions) to express the algorithm; the end result would follow the algorithm described above conceptually, but it might not look much like the pseudocode on the surface. On the other hand, in C, C++, or Java, the implemented algorithm would probably end up looking quite a bit like our pseudocode description.

For the current problem, we'll be looking at positive integers up to 2^{50} (exclusive). Since we need to count the numbers which don't have any squares of prime numbers as factors, we need to start by finding the prime numbers up to $\sqrt{2^{50}} = 2^{25} = 33{,}554{,}432$ (exclusive). That might seem to be a very high number, but none of the languages we've mentioned so far should have any difficulty performing the sieve of Eratosthenes for numbers that large.

Recursive Inclusion-Exclusion Algorithm

Now that we know how to enumerate the prime numbers up to 2^{25}, we'll apply the inclusion-exclusion principle to find out how many numbers under 2^{50} are divisible by the square of at least one of those primes. Then we'll subtract that result from $2^{50}-1$, to find out how many numbers below 2^{50} are squarefree.

As we saw before, we start by counting up all the numbers divisible by 2^2, then the numbers divisible by 3^2, then 5^2, 7^2, etc. Then, because many numbers are divisible by the square of more than one of the primes, and have thus been double-counted, we need to subtract the count of the numbers divisible by $2^2 \cdot 3^2$, then $2^2 \cdot 5^2$, $2^2 \cdot 7^2$, $2^2 \cdot 11^2$, ..., $3^2 \cdot 5^2$, $3^2 \cdot 7^2$, $3^2 \cdot 11^2$, etc. (i.e. all of the numbers divisible by the product of squares of two prime numbers). Now we've under-counted those numbers that are divisible by the product of three squared primes – i.e. those divisible by $2^2 \cdot 3^2 \cdot 5^2$, or $2^2 \cdot 3^2 \cdot 7^2$, or $2^2 \cdot 3^2 \cdot 11^2$, etc. – so we have to add them back in. Of course, we've over-counted again, since some numbers are divisible by the products of four squared primes, so we have to subtract those counts. And so on, and so on.

Things are sounding pretty messy; maybe changing the processing order will help. Let's count all the numbers divisible by 2^2, subtract those divisible by $2^2 \cdot 3^2$, add those divisible by $2^2 \cdot 3^2 \cdot 5^2$, subtract those divisible by $2^2 \cdot 3^2 \cdot 5^2 \cdot 7^2$, etc. When the product gets so large that it exceeds $2^{50}-1$, at $2^2 \cdot 3^2 \cdot 5^2 \cdot 7^2 \cdot 11^2 \cdot 13^2 \cdot 17^2 \cdot 19^2 \cdot 23^2$, we'll drop the last term in the product, and increase the new last term to the square of the next prime, giving $2^2 \cdot 3^2 \cdot 5^2 \cdot 7^2 \cdot 11^2 \cdot 13^2 \cdot 17^2 \cdot 23^2$. We'll do the same thing whenever the product exceeds the limit: take one term off the end of the product, and increase the new last term to the square of the next prime. With each product, we add or subtract the count of numbers divisible by that product, based on the number of distinct primes in the product: if the number is odd, we add; if it's even, we subtract.

Though it might not be apparent, this method enumerates *all* the products of squared primes within the limit. To see this more clearly, let's try it by hand, using an exclusive upper bound of 1000. Since we're dealing with squared primes, we'll start by finding the upper bound for the prime numbers involved, taking the square root of the overall inclusive upper bound. In this example, $\lfloor \sqrt{1000-1} \rfloor = 31$, so we'll be using the prime numbers that are less than or equal to 31: $\{2,3,5,7,11,13,17,19,23,29,31\}$.

Now, let's work through the method described above. Here are the first few steps:

Product	**Limit Reached?**	**# Primes**	**Sign**	**Count =** $\left\lfloor \frac{999}{product} \right\rfloor$
$2^2 = 4$	no	1	+	249
$2^2 \cdot 3^2 = 36$	no	2	-	27
$2^2 \cdot 3^2 \cdot 5^2 = 900$	no	3	+	1
$2^2 \cdot 3^2 \cdot 5^2 \cdot 7^2 = 44100$	**yes**			
$2^2 \cdot 3^2 \cdot 7^2 = 1764$	**yes**			
$2^2 \cdot 5^2 = 100$	no	2	-	9
$2^2 \cdot 5^2 \cdot 7^2 = 4900$	**yes**			
$2^2 \cdot 7^2 = 196$	no	2	-	5

Note that every time the limit is reached, we drop the last squared prime from the product, and increase the new last squared prime to the square of the next prime.

Let's complete the example. However, we can already see that any products of three squared primes greater than $2^2 \cdot 3^2 \cdot 5^2$ will exceed the limit, so we won't bother to include the steps for any more such products – or any other steps where the limit is exceeded; instead, we'll skip straight to dropping the last term in the product when the limit would be reached or exceeded, and increasing the new last term to the square of the next prime.

Product	**# Primes**	**Sign**	**Count =** $\left\lfloor \frac{999}{product} \right\rfloor$
$2^2 = 4$	1	+	249
$2^2 \cdot 3^2 = 36$	2	-	27
$2^2 \cdot 3^2 \cdot 5^2 = 900$	3	+	1
$2^2 \cdot 5^2 = 100$	2	-	9
$2^2 \cdot 7^2 = 196$	2	-	5
$2^2 \cdot 11^2 = 484$	2	-	2
$2^2 \cdot 13^2 = 676$	2	-	1
$3^2 = 9$	1	+	111
$3^2 \cdot 5^2 = 225$	2	-	4
$3^2 \cdot 7^2 = 441$	2	-	2
$5^2 = 25$	1	+	39
$7^2 = 49$	1	+	20
$11^2 = 121$	1	+	8
$13^2 = 169$	1	+	5
$17^2 = 289$	1	+	3
$19^2 = 361$	1	+	2
$23^2 = 529$	1	+	1
$29^2 = 841$	1	+	1
$31^2 = 961$	1	+	1

$$\left|S_2 \cup S_3 \cup S_5 \cup S_7 \cup S_{11} \cup S_{13} \cup S_{17} \cup S_{19} \cup S_{23} \cup S_{29} \cup S_{31}\right| = 391$$

$$\left|\{1,2,\ldots,999\} \setminus \left(S_2 \cup S_3 \cup S_5 \cup S_7 \cup S_{11} \cup S_{13} \cup S_{17} \cup S_{19} \cup S_{23} \cup S_{29} \cup S_{31}\right)\right| = 999 - 391 = 608$$

We can now state that there are 608 squarefree positive integers below 1000.

Note that the first term in the product starts at 2^2 and and moves to the square of the next prime each time it increases, up to a maximum of 31^2. For each initial term in the product, the second term starts at the square of the next prime above the prime squared in the first term, and increases until the product reaches or exceeds the limit. For each combination of first and second terms, the third term starts at the square of the next prime beyond the prime squared in the second term, and increases similarly. This kind of logic can be implemented with nested iteration, or with recursion; the latter is arguably more more flexible in this case, so that's what we'll use.

Squarefree

1. u = upper limit (exclusive)
2. S = set of primes from 2 (inclusive) to $\lfloor \sqrt{u} \rfloor$ (exclusive)
3. e = CountExclusions$(p=1,\ q=1,\ r=-1)$
4. Finished; the count of squarefree numbers in $\{1,\ 2,\ 3,\ \ldots,\ u-1\}$ is $u-1-e$.

CountExclusions (Parameters: *p* [current prime], *q* [current product], *r* [current sign])

1. $x = 0$
2. $r' = -r$
3. $S' = \left\{ s \in S : (s > p) \wedge \left(s < \frac{\sqrt{u}}{q} \right) \right\}$
4. For $p' = \min(S')$ to $\max(S')$, do:
   a. $q' = p'q$
   b. $x = x + r' \left\lfloor \frac{u-1}{q'^2} \right\rfloor$
   c. $x = x + \text{CountExclusions}(p',\ q',\ r')$
5. Return x

Notes:

- In step #3 of the CountExclusions function, ":" denotes a condition on set membership. In this case, the result is that the set S' contains all elements of S that are greater than p and less than $\sqrt{u} / q$.
- In step #4 of CountExclusions, the $\min()$ and $\max()$ functions return the minimum- and maximum-valued elements in a set, respectively.
- We're computing the product of primes, rather than the product of squares of primes, and squaring that product when necessary (in step #4b of CountExclusions).
- The recursive call appears in step #4c of CountExclusions. By recursively calling the function at that point, we're attempting to extend the product by adding a new term, subject to the constraints represented by the current state: the prime currently at the end of the product, the current value of the (non-squared) product, and the current sign.
- The initial call to CountExclusions appears in step #3 of the main algorithm. At that point, there are no primes in the product, so the product has a value of 1, and the current prime is any value less than the lowest prime in S (either 0 or 1 would be a reasonable choice – even though neither is prime, we just need a starting value that's less than the smallest prime). Since the number of primes in the initial product is an even number (0), the initial sign is negative.

Summary

In the end, what might at first have appeared to be a very difficult problem turns out to be manageable with a dozen or two lines of pseudocode (and eventually, a similar number of lines of actual code). More importantly, the two concepts that are essential to the solution – the Sieve of Eratosthenes and the inclusion-exclusion principle – have a wide range of applications in numerical computing.

Implementation Notes

The Java implementations of the algorithms described in "Sieve of Eratosthenes" and "Recursive Inclusion-Exclusion Algorithm" are both contained in the `Main` class, defined in the `Main.java` source file (see appendix, p. i). Since there's only one solution approach implemented, that's the only source file attached.

The executable .jar file `Squarefree.jar` is also attached. This file contains the compiled `Main` class file, and can be executed from the command line, by typing the following (case-sensitive) command:

```
java -jar Squarefree.jar
```

References

[1] "Problem 193," *Project Euler.net*, May 10, 2008. Available: http://projecteuler.net/index.php?section=problems&id=193. [Accessed: Jan. 18, 2010].

[2] E.W. Weisstein, "Möbius Function," *Wolfram MathWorld*. Available: http://mathworld.wolfram.com/MoebiusFunction.html. [Accessed: Jan. 18, 2010].

[3] "Inclusion–exclusion principle," *Wikipedia*, Dec. 15, 2009. Available: http://en.wikipedia.org/wiki/Inclusion%E2%80%93exclusion_principle. [Accessed: Jan. 18, 2010].

[4] P.E. Black, "Sieve of Eratosthenes," *Dictionary of Algorithms and Data Structures*, Dec. 23, 2009. Available: http://www.itl.nist.gov/div897/sqg/dads/HTML/sieve.html. [Accessed: Jan. 18, 2010].

Main.java

```
package org.supercomputingchallenge.minichallenge.squarefree;

import java.util.BitSet;

public class Main {

    private static final int LIMIT_EXPONENT = 50;
    private static final long LIMIT = 1l << LIMIT_EXPONENT;
    private static final long INITIAL_INCLUSIONS = LIMIT - 1;
    private static final int SIEVE_LIMIT =
            1 + (int) Math.sqrt(INITIAL_INCLUSIONS);
    private static final String PRINT_FORMAT =
            "%d squarefree positive integers found below 2^%d.\n(%d ms)\n";

    private static BitSet sieve;

    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        createSieve();
        System.out.printf(PRINT_FORMAT,
                INITIAL_INCLUSIONS - countExclusions(1, 1, -1),
                LIMIT_EXPONENT,
                System.currentTimeMillis() - start);
    }

    private static void createSieve() {
        int root = (int) Math.sqrt(SIEVE_LIMIT - 1);
        sieve = new BitSet(SIEVE_LIMIT);
        sieve.set(2, SIEVE_LIMIT);
        for (int i = 2; i <= root; i = sieve.nextSetBit(i + 1)) {
            for (int j = i * i; j < SIEVE_LIMIT; j += i) {
                sieve.clear(j);
            }
        }
    }

    private static long countExclusions(int prime, long product, int sign) {
        long exclusions = 0;
        int newSign = -1 * sign;
        for (int newPrime = sieve.nextSetBit(prime + 1);
                (newPrime >= 0) && (product * newPrime < SIEVE_LIMIT);
                newPrime = sieve.nextSetBit(newPrime + 1)) {
            long newProduct = product * newPrime;
            exclusions += countExclusions(newPrime, newProduct, newSign)
                    + newSign * INITIAL_INCLUSIONS / (newProduct * newProduct);
        }
        return exclusions;
    }

}
```