## Problem 12 - Highly Divisible Triangular Number

## Gautam Manohar

10 June 2018

This document originally appeared as a blog post on my website. Find it at gautammanohar.com/euler/12.

## 1 Problem Statement

The sequence of triangle numbers is generated by adding the natural numbers. So the 7th triangle number would be 1+2+3+4+5+6+7=28. The first ten terms would be:

$$1, 3, 6, 10, 15, 21, 28, 36, 45, 55, \dots$$

What is the value of the first triangle number to have over N divisors?

## 2 My Algorithm

If we prime factor a number n as

$$\prod_{p|n} p_i^{e_i},\tag{1}$$

where  $p_i$  are the prime factors of n and  $e_i$  is the largest exponent such that  $p_i^{e_i}$  divides n, then we can write the number of factors of n as

$$\prod_{i,p_i|n} (e_i + 1). \tag{2}$$

We will also use the fact that the n-th triangular number is  $\frac{n(n+1)}{2}$ . We could find the number of factors of this number in  $O(\sqrt{\frac{n(n+1)}{2}}\log\log\frac{n(n+1)}{2}) = O(\sqrt{n^2}\log\log n) = O(n\log\log n)$  time. But we can also use the fact that n and n+1 are coprime. Because they share no factors, the number of factors of

n(n+1) is the product of the number of factors of n and n+1, which we can find in  $O(\sqrt{n}\log\log n)$  time.

We also need to account for the factors lost by dividing by 2. Suppose that the greatest power of 2 that divides n(n+1) is  $2^k$ . Then  $2^k$  divides exactly one of n and n+1, so we need not factor n(n+1). This factor of  $2^k$  contributes a k+1 term to the product in (2). Dividing by 2 turns this into a k-1+1=k term. And so the number of factors decreases by  $\frac{k}{k+1}$ . We then iterate over n until a number with more than N divisors is found, then return the corresponding triangular number.

This solution has time complexity  $O(\frac{M^2\log\log M}{\log M})$ , where M is the index of the smallest triangular number with over N divisors, We can improve this solution by precomputing a sufficiently large list of primes. Then the solution has time complexity  $O(\sqrt{M}\log\log M + \frac{M\sqrt{M}}{\log M}) = O(\frac{M\sqrt{M}}{\log M})$ . We can further optimize the constant factor in our algorithm by caching the number of prime factors of the last n+1 that we tested.