**Fast Calculation of Count of Divisor Sums >= 2**

Nathan McKenzie

September 242009

## Preliminaries

Just to clear up some notation, through out this document, I am going to refer frequently to the following function:

$$D_k{}'(n) = \sum_{j=2} \tau_k{}'(n) \tag{0.1}$$

where

$$\tau'_k(n) = |\{n_1, \dots, n_k \geqslant 2; \;\; n_1 \dots n_k = n\}|.$$

According to Linnik's Identity, this means that

$$\pi^*(n) = \sum_{k=1} \frac{-1^{k+1}}{k} D_k{}'(n) \tag{0.2}$$

where $\pi^*$ is the prime power counting function. Thus, the problem of prime counting turns into the problem of calculating these various D'(n,k) values.

The fundamental relationships for D' are

$$D_1{}'(n) = n-1$$
$$D_0{}'(n) = 1$$

and

$$D_k{}'(n) = \sum_{j=2} D_{k-1}{}'(\lfloor \frac{n}{j} \rfloor) \tag{0.3}$$

## Fast-ish Counting Method

The key to this method of counting is to consider the slightly more general function

$$D_k(n,a) = \sum_{j=a} D_{k-1}(\lfloor \frac{n}{j} \rfloor, a)$$

$$D_1(n,a) = n-a+1$$
$$D_0(n,a) = 1 \tag{1.1}$$

So essentially, this function is a version of (0.3) with iteration starting at a specified integer rather than always at 2. Thus, rewriting (0.2) above, we can say we are looking for

$$\pi^*(n) = \sum_{k=1} \frac{-1^{k+1}}{k} D_k(n,2)$$

(1.2)

Although I won't list here where this relationship comes from, one key feature of the main function in (1.1) is

$$D_k(n,a) = \sum_{j=0}^{k} \binom{k}{j} D_j\left(\frac{n}{a^{k-j}}, a+1\right)$$

(1.3)

So, as an example,

$$D_4(900{,}2) = D_4(900{,}3) + 4\,D_3\left(\frac{900}{2},3\right) + 6\,D_2\left(\frac{900}{4},3\right) + 4\,D_1\left(\frac{900}{8},3\right) + D_0\left(\frac{900}{16},3\right)$$

(1.4)

Pretty obviously, we can reapply identity (1.3) to the leading term on the right side of the equation again, getting

$$D_4(900{,}3) = D_4(900{,}4) + 4\,D_3\left(\frac{900}{3},4\right) + 6\,D_2\left(\frac{900}{9},4\right) + 4\,D_1\left(\frac{900}{27},4\right) + D_0\left(\frac{900}{27},4\right)$$

(1.5)

Now, because D_4(900,k) represents the count of integer solutions to a*b*c*d <= 900 where a,b,c,d >= k, it should be trivially obvious that D_4(900,k) = 0 when k > 900^(1/4). So, more generally, we see that

$$D_k(n,a) = 0 \; when \; a > n^{\frac{1}{k}}$$

(1.6)

So, if we continue the process started in (1.3) and (1.4), once k > 900^(1/4), we will have removed the leading terms, leaving us with

$$D_4(900{,}2) = \sum_{j=2}^{900^{\frac{1}{4}}} 4\,D_3\left(\frac{900}{j}, j+1\right) + 6\,D_2\left(\frac{900}{j^2}, j+1\right) + 4\,D_1\left(\frac{900}{j^3},3\right) + D_0\left(\frac{900}{j^4}, j+1\right)$$

In more general terms, this relationship can be written as

$$D_k(n,a) = \sum_{m=a}^{n^{1/k}} \sum_{j=0}^{k-1} \binom{k}{j} D_j\left(\frac{n}{m^{k-j}}, m+1\right)$$

(1.7)

So, applied recursively, and taking into account the two trivially identities from (1.1), (1.7) can be used to calculate the the prime power counting function from (1.1).

In my C++ implementation of this function, I made aggressive use of a wheel, rejecting all numbers divisible by primes less than 29. This massively speeds calculations up. I have not, in general, been able to find any way to use caching or pre-calculation generally to speed this up any more. Even with a large wheel, the nested loops become slow fairly quickly.

For the sake of redundancy, here is my C# code for (1.7):

```
static int D( int n, int k, int a ){
	if( k == 0 )return 1;
	if( k ==1 ) return n - a + 1;
	int total = 0;
	for( int m = a; m <= (int)Math.Pow( n, 1.0/k ); m++ ){
		for( int j = 0; j < k; j++ ){
			total += D( n/(int)Math.Pow(m, k-j), j, m+1 ) * (int)binom( k, j );
		}
	}
	return total;
}
static double fact( double val ){
	double total = 1.0;
	for( var i = 1; i <= val; i++){
		total*=i;
	}
	return total;
}
static double binom( int val, int div ){
	return fact( val ) / ( fact(div ) * fact( val - div ) );
}
```