Southern Adventist University
KnowledgeExchange@Southern

Senior Research Projects | Southern Scholars

2004

Calculating the unrestricted partition function Towards an investigation of its arithmetic properties

Robert Jacobson

Follow this and additional works at: https://knowledge.e.southern.edu/senior_research

 Part of the Applied Mathematics Commons

Recommended Citation
Jacobson, Robert, "Calculating the unrestricted partition function Towards an investigation of its arithmetic properties" (2004). *Senior Research Projects*. 43.
https://knowledge.e.southern.edu/senior_research/43

This Article is brought to you for free and open access by the Southern Scholars at KnowledgeExchange@Southern. It has been accepted for inclusion in Senior Research Projects by an authorized administrator of KnowledgeExchange@Southern. For more information, please contact jspears@southern.edu.

Calculating the unrestricted partition function

Towards an investigation of its arithmetic properties

Robert Jacobson
Southern Adventist University
December 7, 2004

■ Contents

■ Introduction

A partition of a positive integer n is a set of positive integers, called parts, that sum to n. For example, 4 has the partitions:

4
3+1
2+2
2+1+1
1+1+1+1.

The unrestricted partition function $p(n)$ counts the number of partitions of n. Thus $p(4) = 5$, since there are 5 partitions of the number 4. We can restrict the partitions in various ways. For example, we may wish to count the number of partitions of n into *distinct* parts, that is, in a given partition no part occurs more than once. The number of such partitions of 4 is 2, namely 4 and 3+1, since each other partion repeats parts. For the most part, however, we will be concerned with the unrestricted partition function.

Many interesting questions can be asked about the arithmetic of this special function. For example, when is it even, or when is it odd? When is it evenly divisible by 29? What is its congruence distribution modulus certain prime numbers? In 1938 G.H.Hardy remarked, "In spite of the simplicity of $p(n)$, very little is known about its arithmetic properties" [1]. We might begin by writing down $p(n)$ for the first few values of n and looking for patterns, as Ramanujan and Hardy did about 80 years ago to develop some striking results. After the first, say, eight terms counting the number of partitions of n as we did above becomes too cumbersome. We have already learned two things: first, $p(n)$ grows approximately exponentially, and second, we need an efficient method of calculating $p(n)$ without enumerating each partition of n. If we wish to investigate the arithmetic properties of $p(n)$, we need to be able to compute $p(n)$ efficiently, perhaps for several hundred, a few thousand, or millions of values of n.

In this paper several methods for calculating $p(n)$ will be explored with the motivation of determining $p(n)$'s arithmetic properties. Also, an exposition of the Hardy-Ramanujan-Rademacher method will be given, and indirect methods of determining some arithmetic properties will be discussed.

■ Preliminaries

First, some elementary definitions and theorems must be understood. This section developes the mathematical foundations assumed in subsequent sections.

■ Generating Functions

Definition 1: The generating function $f(q)$ for the sequence $a_0,\ a_1,\ a_2,\ a_3,\ \ldots$ is the power series $f(q) = \sum_{n\geq 0} a_n\, q^n$.

As an example, $(q+1)^m$ is a generating function for the sequence $\{a_i\}_{i=0}^{\infty} = \left\{\binom{m}{i}\right\}_{i=0}^{\infty}$ since

$$(q+1)^m = \binom{m}{0}q^0 + \binom{m}{1}q^1 + \binom{m}{2}q^2 + \cdots + \binom{m}{m-1}q^{m-1} + \binom{m}{m}q^m .$$

Remarks: Note that this sum may still be regarded as an infinite series as in the definition since $a_n = \binom{m}{n} = 0$ for $n > m$. We say that $f(q)$ *generates* $\{a_i\}_{i=0}^{\infty}$.

■ Elementary Series-Product Identities

Theorem 2: The infinite product $\prod_{n\geq 1}(1-q^n)^{-1}$ is a generating function for the sequence $a_n = p(n)$ $(n \geq 1)$.

Proof. Recalling the identity $\frac{1}{1-q} = 1 + q + q^2 + q^3 + q^4 + \ldots.$, the infinite product becomes an infinite product of infinite sums,

$$\begin{aligned}\prod_{n\geq 1}(1-q^n)^{-1} = (1 + q + q^2 + q^3 + \ldots)\\ \times(1 + q^2 + q^4 + q^6 + \ldots)\\ \times(1 + q^3 + q^6 + q^9 + \ldots)\\ \ldots.\end{aligned}$$

The first "row", that is, the first infinite sum can be thought of as contributing "parts" (the numbers in the exponent) that occur once, the second row parts that occur twice, etc. It can now be seen that when this product is expanded the coefficient of q^n will be the number of ways to add nonnegative integers (the exponents of q) to sum to n. ∎

Theorem 3: The infinite product $\prod_{n\geq 1}(1+q^n)$ is a generating function for the number of partitions of n into *distinct* parts.

Referring to the list of unrestricted partitions of 4 in the introduction, observe that there are 2 partitions in which each part is distinct, that is, only used once, namely, 4 and 3+1.

Definition 4: Let $p(m,\ n)$ be the number of ways to partition n such that in a given partition no part occurs more than m times.

Again, referring to the list of unrestricted partitions of 4 in the introduction, observe that there are 3 partitions that have 2 or fewer parts. Thus, $p(2,\ 4) = 3$.

Theorem 5: The infinite product $\prod_{n\geq 1}(1 + q^n + q^{2n} + q^{3n} + \cdots + q^{mn}) = \prod_{n\geq 1}(1-q^{(m+1)n})(1-q^n)^{-1}$ is a generating function for $p(m,\ n)$.

Notice that theorem 3 is just a special case of theorem 5, namely, theorem 3 is a generating function for $p(1, n)$.
The proofs of theorems 3 and 4 procede as that of 2 and are left as an exercise to the reader.

Note: We have not been careful with the question of convergence. Suffice it to say that all of our sums and products converge for $|q| < 1$. Questions of convergence will generally be proscribed since our considerations usually deal only with functions' forms.

■ Inefficient Methods of Computing $p(n)$

If $p(n)$ is to be investigated empirically, the first problem that must be solved is, how can $p(n)$ be computed? This section surveys several methods one might use and discusses associated issues.

■ Using the Generating Function Directly

An Algorithm

The most obvious way to compute $p(n)$ is to use theorem 5, expanding enough terms to determine the coefficient of q^n. Clearly $p(m,\ n) = p(n)$ when $m \geq n$. Thus, the coefficient of q^n in

$$\prod_{m=1}^{n}(1 + q^m + q^{2m} + q^{3m} + \cdots + q^{nm}) = \prod_{m=1}^{n}\left(\sum_{i=0}^{n}(q^{im})\right)$$

is $p(n)$. In fact, the coefficient of q^i for *every* $i \leq n$ is $p(i)$. The following pseudocode algorithm performs the expansion of the above product.

```
p(input n)
Begin p:
1)      If n < 0  then
2)              output 0
3)              halt
4)      If n = 0 then
5)              output 1
6)              halt

7)      Allocate Coefficients Array with Length n^2(n+1)/2 + 1

8)      Allocate Coefficients2 Array with Length n^2(n+1)/2 + 1

9)      Initialize Coefficients to 0
10)     Initialize Coefficients2 to 0
11)     Set Coefficients[0] to 1

12)     For m = 1 to n
        Do:
13)             For i = 0 to n
                Do:
14)                     For j = 0 to n(m-1)m/2
                        Do:
15)                             Set Coefficients2[j + i m] to Coefficients2[j + i m] + Coefficients[j]
                        Loop
                Loop

16)             Copy Coefficients2 to Coefficients
```

```
17)          Reset Coefficients2 to 0
     Loop
End p
```

The array holding the coefficients needs to have as many spaces as the largest exponent plus one. The innermost loop need only go to $n(\frac{m(m+1)}{2})$, the largest exponent so far.

Complexity

Theorem 6: The above algorithm is $O(n^5)$.

Proof. To determine the running time, we must determine how many times line 15 executes. Now, line 15 executes a total of

$$\begin{aligned}
\sum_{m=1}^{n}\sum_{i=0}^{n}\sum_{j=0}^{\frac{1}{2}n(m-1)m} 1 &= \sum_{m=1}^{n}\sum_{i=0}^{n}\left(\frac{n(m-1)m}{2}+1\right) \\
&= \sum_{m=1}^{n}\left(\frac{n(m-1)m}{2}+1\right)(n+1) \\
&= \sum_{m=1}^{n}\left[n+1-m\left(\frac{n(n+1)}{2}\right)+m^2\left(\frac{n(n+1)}{2}\right)\right] \\
&= n(n+1)-\left(\frac{n(n+1)}{2}\right)^2+\left(\frac{n(n+1)(2n+1)}{6}\right)\left(\frac{n(n+1)}{2}\right) \\
&= \frac{1}{6}n^5+\frac{1}{6}n^4-\frac{1}{6}n^3+\frac{5}{6}n^2+n
\end{aligned}$$

times. Thus this algorithm is $O(n^5)$. ∎

A Refined Algorithm

This can be refined significantly by realizing that any q^c with $c > n$ does not contribute to the coefficient of q^n. Thus the inner sum need only be over $0 \le im \le n$, or, more precisely, until $i = \lfloor n/m \rfloor$. We may thus use

$$\prod_{m=1}^{n}\sum_{i=0}^{\lfloor \frac{n}{m} \rfloor} q^{im}.$$

In addition, we need not multiply two terms $a\,q^{\alpha} \cdot b\,q^{\beta}$ if $\alpha + \beta > n$; thus the inner loop need only go from 0 to $n - im$.

```
p(input n)
Begin p:
1)     If n < 0 then
2)          output 0
3)          halt
4)     If n = 0 then
5)          output 1
6)          halt
```

```
7)      Allocate Coefficients Array with Length n+1
8)      Allocate Coefficients2 Array with Length n+1

9)      Initialize Coefficients to 0
10)     Initialize Coefficients2 to 0
11)     Set Coefficients[0] to 1

12)     For m = 1 to n
        Do:
13)             For i = 0 to ⌊n/m⌋
                Do:
14)                     For j = 0 to n − i m
                        Do:
15)                             Set Coefficients2[j + i m] to Coefficients2[j + i m] + Coefficients[j]
                        Loop
                Loop

16)             Copy Coefficients2 to Coefficients
17)             Reset Coefficients2 to 0
        Loop
End p
```

Since we no longer are concerned with coefficients of q^i ($i > n$), the array holding the coefficients (lines 7 and 8) need only be $n + 1$ in length.

Complexity

Theorem 7: The above algorithm is $O(n^2 \operatorname{Log}(n))$.

Proof. We proceed as before:

$$\begin{aligned}
\sum_{m=1}^{n}\sum_{i=0}^{\lfloor i;\rfloor}\sum_{j=0}^{n-im} 1 &= \sum_{m=1}^{n}\sum_{i=0}^{\lfloor \frac{n}{m} \rfloor}(n+1-im) \\
&= \sum_{m=1}^{n}\left((n+1)\left(\left\lfloor\frac{n}{m}\right\rfloor+1\right)-m\sum_{i=0}^{\lfloor \frac{n}{m} \rfloor} i\right) \\
&= \sum_{m=1}^{n}\left((n+1)\left(\left\lfloor\frac{n}{m}\right\rfloor+1\right)-m\sum_{i=1}^{\lfloor \frac{n}{m} \rfloor} i\right) \\
&= \sum_{m=1}^{n}\left((n+1)\left(\left\lfloor\frac{n}{m}\right\rfloor+1\right)-m\,\frac{1}{2}\left(\left\lfloor\frac{n}{m}\right\rfloor\right)\left(\left\lfloor\frac{n}{m}\right\rfloor+1\right)\right) \\
&= \sum_{m=1}^{n}\left(n+n\left\lfloor\frac{n}{m}\right\rfloor-m\,\frac{1}{2}\left\lfloor\frac{n}{m}\right\rfloor^2-m\,\frac{1}{2}\left\lfloor\frac{n}{m}\right\rfloor+\left\lfloor\frac{n}{m}\right\rfloor+1\right). \qquad (1)
\end{aligned}$$

(We expand (1) for the sake of simplicity.) We now obtain an upper bound:

$$
\begin{aligned}
(1) \quad &\le \sum_{m=1}^{n} \left(n + n\left(\frac{n}{m}\right) - m\,\frac{1}{2}\left(\frac{n}{m} - 1\right)^2 - m\,\frac{1}{2}\left(\frac{n}{m} - 1\right) + \frac{n}{m} + 1 \right) \\
&= \sum_{m=1}^{n} \left(\left(\frac{3}{2}\,n + 1\right) + \left(n + \frac{1}{2}\,n^2\right)\frac{1}{m} \right) \\
&\le \frac{1}{2}\,n^2 \operatorname{Log}(n) + n \operatorname{Log}(n) + \frac{3}{2}\,n^2 + n,
\end{aligned}
$$

where the last line follows from the inequality

$$\sum_{m=1}^{n} \frac{1}{m} \le \operatorname{Log}(n) + 1.$$

Again using (1), we obtain a lower bound:

$$
\begin{aligned}
(1) \quad &\ge \sum_{m=1}^{n} \left(n + n\left(\frac{n}{m} - 1\right) - m\,\frac{1}{2}\left(\frac{n}{m}\right)^2 - m\,\frac{1}{2}\left(\frac{n}{m}\right) + \frac{n}{m} \right) \\
&= \sum_{m=1}^{n} \left(-\frac{1}{2}\,n + \left(n + \frac{1}{2}\,n^2\right)\frac{1}{m} \right) \\
&\ge \frac{1}{2}\,n^2 \operatorname{Log}(n+1) + n \operatorname{Log}(n+1) - \frac{1}{2}\,n^2,
\end{aligned}
$$

where the last line follows from the inequality

$$\sum_{m=1}^{n} \frac{1}{m} \ge \operatorname{Log}(n+1).$$

Hence,

$$\frac{1}{2}\,n^2 \operatorname{Log}(n+1) + n \operatorname{Log}(n+1) - \frac{1}{2}\,n^2 \le \sum_{m=1}^{n} \sum_{i=0}^{\lfloor \frac{n}{m} \rfloor} \sum_{j=0}^{n-im} 1 \le \frac{1}{2}\,n^2 \operatorname{Log}(n) + n \operatorname{Log}(n) + \frac{3}{2}\,n^2 + n.$$

Thus this algorithm is $O(n^2 \operatorname{Log}(n))$.■

Note: A "tighter" bound is possible by algebraic manipulation of terms involving $\lfloor n/m \rfloor$ before applying an inequality, though the resulting bound will of course still be $O(n^2 \operatorname{Log}(n))$.

The running time of this algorithm is a large improvement over $O(n^5)$.

■ A Method Involving A Recurrence

Mathematical Development

A fundamentally different technique for computing $p(n)$ uses a recurrence relationship we now develop. (This development follows work done in [11].)

Theorem 8: $p(n) = \frac{1}{n}\sum_{k=1}^{n}\sigma(k)\,p(n-k)$, where $\sigma(k)$ is the sum of factors of k.

Proof. Let $P(q) = \prod_{i\geq 1} 1/(1-q^i)$ be the generating function for $p(n)$ as in theorem 2. Taking the log of both sides gives

$$\mathrm{Log}(P(q)) = \sum_{i=1}^{\infty}\mathrm{Log}\left(\frac{1}{(1-q^i)}\right).$$

Differentiating both sides with respect to q and moving $P(q)$ to the right hand side yields

$$P'(q) = P(q)\sum_{i=1}^{\infty}\frac{i\,q^{i-1}}{(1-q^i)}. \tag{2}$$

We now simplify $\sum_{i=1}^{\infty}(i\,q^{i-1})/(1-q^i)$:

$$\begin{aligned}\sum_{i=1}^{\infty}\frac{i\,q^{i-1}}{(1-q^i)} &= \sum_{i=1}^{\infty}(i\,q^{i-1})(1+q^i+q^{2i}+q^{3i}+\cdots)\\ &= \sum_{i=1}^{\infty} i\,q^{i-1}\sum_{j=0}^{\infty}q^{ji}\\ &= \sum_{i=1}^{\infty} i\,q^{i-1}\sum_{j=1}^{\infty}q^{(j-1)i}\\ &= \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} i\,q^{ij-1}. \end{aligned}\tag{3}$$

In (3), any term q^{ij-1} with $ij - 1 = k$ contributes i to the coefficient of q^k. That is, any i that is a factor of $k+1$ contributes i to the coefficient of q^k. Thus, the coefficient of q^k is $\sigma(k+1)$. We may rewrite (3) as

$$\sum_{k=0}^{\infty}\sigma(k+1)\,q^k.$$

Rewriting (2) now gives

$$\begin{aligned} P'(q) &= p(1) + 2\,q\,p(2) + 3\,q^2\,p(3) + \cdots = \sum_{n=0}^{\infty} (n+1)\,p(n+1)\,q^n \\ &= P(q) \sum_{k=0}^{\infty} \sigma(k+1)\,q^k \\ &= \left(\sum_{r=0}^{\infty} p(r)\,q^r\right)\left(\sum_{k=0}^{\infty} \sigma(k+1)\,q^k\right) \\ &= \sum_{n=0}^{\infty} \left(\sum_{k=0}^{n} \sigma(k+1)\,p(n-k)\right) q^n. \end{aligned}$$

Equating the coefficients of q^n in the extremes of the above yields

$$n\,p(n) = \sum_{k=0}^{n-1} \sigma(k+1)\,p(n-k-1).$$

Hence

$$p(n) = \frac{1}{n} \sum_{k=1}^{n} \sigma(k)\,p(n-k). \blacksquare$$

Algorithm

To construct an algorithm to exploit this recurrence we note that to compute $p(n)$ we must first compute $p(i)$ for $0 \le i < n$ and $\sigma(i)$ for $1 \le i \le n$.

```
p(input n)
Begin p:
1)      If n < 0  then
2)              output 0
3)              halt
4)      If n = 0 then
5)              output 1
6)              halt

7)      Allocate Sigma Array with Length n
8)      Allocate PV Array with Length n + 1

9)      Initialize Sigma to 0
10)     Initialize PV to 0
11)     Set PV[0] to 1

12)     For i = 1 to n
        Do:
13)             For k = 1 to i
                Do:
14)                     If k ≡ 0 (mod i) then
15)                             Set Sigma[i – 1] = Sigma[i – 1] + k
                Loop
```

```
        Loop

16)     Initialize variable partialsum to 0

17)     For i = 1 to n
        Do:
18)             For k = 1 to i
                Do:
19)                     Set partialsum = partialsum + Sigma[k – 1]*PV[i – k]
                Loop

20)             Set PV[i] = partialsum / i
21)             Reset partialsum to 0
        Loop
End p
```

Sigma[k] holds $\sigma(k+1)$ and PV[k] holds $p(k)$ at the end of this algorithm.

Complexity

Theorem 9: This algorithm is $O(n^2)$.

Proof. Clearly lines 14-15 and line 19 execute the same number of times, namely

$$\sum_{i=1}^{n}\sum_{k=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n.$$

Thus this algorithm is $O(n^2)$. ∎

■ Considerations

All of these algorithms have the property that they compute all $p(i)$ for $1 \le i \le n$ which is useful for investigating $p(n)$'s arithmetic properties, since often a survey of properties of $p(n)$ for many different n is desired. The last algorithm has the added benefit that, as can easily be shown, if its arrays are retained after computing $p(n)$, $p(n+1)$ can be computed in $O(n)$ time. Still, they are far too inefficient to be used for large n.

■ Euler's Pentagonal Recurrence

Euler's pentagonal recurrence formula is the standard method for computing $p(n)$ for small n. For example, the mathematical software packages *Mathematica* and *Maple* both use Euler's recurrence for small n. (In the case of *Mathematica*, "small n" means $n < 5000$.) Since this method sits on the cutting edge of several efforts to empirically investigate $p(n)$, we shall look at it in detail.

■ Euler's Pentagonal Number Theorem

In order to prove Euler's Pentagonal Number Theorem an additional theorem is required. The interested reader, however, must be satisfied to consult, for example, page 10 of [7] for the proof of this additional theorem, as it is omitted here.

Theorem 10: Let $p_e(n)$ and $p_o(n)$ be the number of partitions of n into an even number of distinct parts and an odd number of distinct parts respectively. Then

$$p_e(n) - p_o(n) = \begin{cases} (-1) & \text{if } n = \frac{1}{2}\, m(3\,m \pm 1), \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 11 (Euler's Pentagonal Number Theorem):

$$\prod_{n=1}^{\infty}(1-q^n) = 1 + \sum_{m=1}^{\infty}(-1)^m\, q^{\frac{1}{2}\, m(3\,m-1)}(1+q^m) = \sum_{m=-\infty}^{\infty}(-1)^m\, q^{\frac{1}{2}\, m(3\,m-1)}.$$

Proof. (This proof is found in [7].) Clearly

$$\begin{aligned} \sum_{m=-\infty}^{\infty}(-1)^m\, q^{\frac{1}{2}\, m(3\,m-1)} &= 1 + \sum_{m=1}^{\infty}(-1)^m\, q^{\frac{1}{2}\, m(3\,m-1)} + \sum_{m=-1}^{-\infty}(-1)^m\, q^{\frac{1}{2}\, m(3\,m-1)} \\ &= 1 + \sum_{m=1}^{\infty}(-1)^m\, q^{\frac{1}{2}\, m(3\,m-1)} + \sum_{m=1}^{\infty}(-1)^m\, q^{\frac{1}{2}\, m(3\,m+1)} \\ &= 1 + \sum_{m=1}^{\infty}(-1)^m\, q^{\frac{1}{2}\, m(3\,m-1)} + \sum_{m=1}^{\infty}(-1)^m\, q^{\frac{1}{2}\, m(3\,m-1)+m} \\ &= 1 + \sum_{m=1}^{\infty}(-1)^m\, q^{\frac{1}{2}\, m(3\,m-1)}(1+q^m) \\ &= 1 + \sum_{n=1}^{\infty}(p_e(n) - p_o(n))\, q^n \end{aligned}$$

by theorem 10. We now must show that

$$1 + \sum_{n=1}^{\infty}(p_e(n) - p_o(n))\, q^n = \prod_{n=1}^{\infty}(1-q^n).$$

Now

$$\prod_{n=1}^{\infty}(1-q^n) = \sum_{a_1=0}^{1}\sum_{a_2=0}^{1}\sum_{a_3=0}^{1}\cdots(-1)^{a_1+a_2+a_3+\cdots}\, q^{1\cdot a_1+2\cdot a_2+3\cdot a_3+\cdots}.$$

Now each partition with a distinct number of parts (see theorem 3) is counted with a weight $(-1)^{a_1+a_2+a_3+\cdots}$ which is $+1$ if the number of parts is even and -1 if the number of parts is odd. Thus

$$\prod_{n=1}^{\infty}(1-q^n) = \sum_{a_1=0}^{1}\sum_{a_2=0}^{1}\sum_{a_3=0}^{1}\cdots(-1)^{a_1+a_2+a_3+\cdots}\, q^{1\cdot a_1+2\cdot a_2+3\cdot a_3+\cdots}$$

$$= 1+\sum_{n=1}^{\infty}(p_e(n)-p_o(n))\,q^n.\ \blacksquare$$

Theorem 12 (Euler's Recurrence): If $n > 0$, then

$$p(n)-p(n-1)-p(n-2)+p(n-5)+p(n-7)+$$
$$\ldots+(-1)^m\,p\left(n-\frac{1}{2}m(3m-1)\right)+(-1)^m\,p\left(n-\frac{1}{2}m(3m+1)\right)+\cdots=0.$$

Note: $p(0) = 1$ and $p(N) = 0$ for $N < 0$.

Proof: Let a_n be the left-hand side of the above equation. Then clearly

$$\sum_{n=0}^{\infty}a_n q^n = \left(\sum_{n=0}^{\infty}p(n)\,q^n\right)\cdot\left(1+\sum_{m=1}^{\infty}(-1)^m\,q^{\frac{1}{2}m(3m-1)}(1+q^m)\right)$$

$$= \left(\prod_{n=1}^{\infty}(1-q^n)^{-1}\right)\cdot\left(\prod_{n=1}^{\infty}(1-q^n)\right) \qquad \text{(by theorems 2 and 11)}$$

$$= 1.\ \blacksquare$$

▪ Recurrence Algorithm

If we move everything but $p(n)$ to the right hand side of theorem 12 we obtain

$$p(n) = p(n-1)+p(n-2)-p(n-5)-p(n-7)+\ldots.$$

To compute $p(n)$ we need to know $p(n-1)$, the first term in our recurrence. But to compute $p(n-1)$, we need to know $p(n-2)$, the first term in the recurrence for $p(n-1)$. Thus, our algorithm is eventually going to compute every partition from $p(n)$ down to $p(0)$. We stop there because $p(N) = 0$ when $N < 0$. Now, $p(0)$ is the base case, since $p(0)$ is defined to be 1. It would be efficient, then, to start by computing $p(0)$, then compute $p(1)$, then $p(2)$, and so on until we compute $p(n)$.

We compute $p(n)$ by subtracting generalized pentagonal numbers from n, so it might be useful to precompute them. But how many pentagonal numbers are we going to need? We need as many as will make $p(n-r)$ zero, where r is the pentagonal number. That is, when $n = r = \max(\frac{1}{2}(3k^2+k),\ \frac{1}{2}(3k^2-k)) = \frac{1}{2}(3k^2+k)$. Now, $n = \frac{1}{2}(3k^2+k)$ when $k = \frac{1}{6}\left(-1+\sqrt{1+24n}\right)$. So we will need to generate the first $\left\lfloor\frac{1}{6}\left(-1+\sqrt{1+24n}\right)\right\rfloor$ pentagonal numbers. Pseudocode for the algorithm follows.

```
p(input n)
Begin p:
1)      If n < 0  then
2)              output 0
3)              halt
4)      If n = 0 then
5)              output 1
6)              halt
```

```
7)    Set MaxK = ⌊(1/6)(−1 + √(1 + 24 n))⌋ + 1
8)    Allocate Pent Array with Length 2 × MaxK
9)    Allocate Partitions Array with Length n + 1

10)   Initialize Partitions to 0

      /*This loop initializes the Pent array with the pentagonal numbers*/
11)   For m = 1 to MaxK
      Do:
12)         Set Pent[2 m − 2] to (1/2) m(3 m − 1)
13)         Set Pent[2 m − 2 + 1] to (1/2) m(3 m + 1)
      Loop

14)   For I = 1 to n
      Do:
15)         Set PartialSum to 0
16)         Set j to 1

17)         While Pent[j − 1] ≤ i
            Do:
18)               If j ≡ 1 (mod 4) or j ≡ 2 (mod 4) then
19)                     Set PartialSum to PartialSum + Partitions[i − Pent[j − 1]]
                  Else
20)                     Set PartialSum to PartialSum − Partitions[i − Pent[j − 1]]
21)               Increment j by 1
            Loop

22)         Set Partitions[i] to PartialSum
      Loop
End p
```

When the algorithm exits, Partitions[i] contains $p(i)$. Note that one extra pentagonal number is computed as an "escape" case.

• Complexity Analysis

Theorem 13: The above algorithm is $O(n^{3/2})$.

Proof. Clearly lines 12 and 13 execute $\left\lfloor \frac{1}{6}\left(-1+\sqrt{1+24n}\right)\right\rfloor + 1$ times. Now, for each value of i, $\left\lfloor \frac{1}{6}\left(-1+\sqrt{1+24i}\right)\right\rfloor$ numbers are added together, since, as previously mentioned, there are $\left\lfloor \frac{1}{6}\left(-1+\sqrt{1+24i}\right)\right\rfloor$ positive pentagonal numbers less than or equal to i. Hence, lines 19 and 20 are executed

$$\sum_{i=1}^{n}\left\lfloor \frac{1}{6}\left(-1+\sqrt{1+24i}\right)\right\rfloor$$

times. It is obvious that

$$\frac{2}{3}n^{3/2} = \int_1^{n+1} \sqrt{x-1}\,dx < \sum_{i=1}^{n} \sqrt{i}\,. \tag{4}$$

Now,

$$\begin{aligned}\sum_{i=1}^{n} \left\lfloor \frac{1}{6}\left(-1+\sqrt{1+24\,i}\right)\right\rfloor &> \sum_{i=1}^{n}\left(\frac{1}{6}\sqrt{1+24\,i}-\frac{7}{6}\right)\\ &> \frac{\sqrt{24}}{6}\sum_{i=1}^{n}\sqrt{i}-\frac{7}{6}n\\ &> \frac{2\sqrt{6}}{9}n^{3/2}-\frac{7}{6}n,\end{aligned}$$

where the last line follows from (4). Similarly,

$$\sum_{i=1}^{n} \sqrt{i} < \int_1^{n+1} \sqrt{x}\,dx = \frac{2}{3}(1+n)^{3/2} - \frac{2}{3}. \tag{5}$$

Thus,

$$\begin{aligned}\sum_{i=1}^{n} \left\lfloor \frac{1}{6}\left(-1+\sqrt{1+24\,i}\right)\right\rfloor &< \frac{5}{6}\sum_{i=1}^{n}\sqrt{i}\\ &= \frac{5}{9}(1+n)^{3/2}-\frac{5}{9},\end{aligned}$$

where the last line follows from (5). Hence

$$\frac{2\sqrt{6}}{9}n^{3/2}-\frac{7}{6}n < \sum_{i=1}^{n}\left\lfloor \frac{1}{6}\left(-1+\sqrt{1+24\,i}\right)\right\rfloor < \frac{5}{9}(1+n)^{3/2}-\frac{2}{3}.$$

Therefore this algorithm is $O(n^{3/2})$.∎

■ Considerations

This algorithm shares the advantages of the previous algorithms in that it calculates $p(i)$ for $0 \le i \le n$. In addition, it can easily be shown that if $p(i)$ for $0 \le i \le n$ have already been computed, then $p(n+1)$ can be computed in $O(\sqrt{n})$ time, much faster than the last algorithm in the previous section. Algorithms based on this one have been developed for parallel computers. One such algorithm running on 128 processors is able to calculate $p(n)$ modulo all primes < 100 for $0 \le n \le 10^9$ in about a day [10].

■ The Hardy-Ramanujan-Rademacher Formula

Donald Knuth, the preeminent computer scientist of our time, writes "The Hardy-Ramanujan-Rademacher formula for $p(n)$ is surely one of the most astonishing identities ever discovered" [16]. Indeed, otherwise stolid authors of books on analysis and number theory rarely fail to offer a gushing remark when they treat the Hardy-Ramanujan-Rademacher formula (HRR). George Andrews provides a restrained example when he writes that the HRR is "one of the crowning achievements in the theory of partitions," yet cannot resist affirming that Hardy and Ramanujan's approach is "truly *remarkable*" (emphasis his) [7]. Other authors simply revert to superlatives like "beautiful" [13] and "spectacular" [14]. In a curious struggle between letting the mathematics speak for itself (as a mathematician presumably should) and a desire to express his awe, J. E. Littlewood writes, "The reader does not need to be told that this is a very astonishing theorem..." [17]. Therefore, this author will make no comments regarding the aesthetics of HRR, prefering rather that the reader be the judge.

The form of HRR as we will use it is:

$$p(n) = \frac{2\sqrt{3}}{24n-1} \sum_{k=1}^{\infty} \frac{A_k(n)}{\sqrt{k}} \left\{ \left(1 - \frac{k}{\nu(n)}\right) \exp\left[\frac{\nu(n)}{k}\right] + \left(1 + \frac{k}{\nu(n)}\right) \exp\left[-\frac{\nu(n)}{k}\right] \right\},$$

where

$$A_k(n) = \sum_{\substack{1 \le h < k, \\ \gcd(h,k)=1}} \omega_{h,k} \exp\left[\frac{-2\pi i n h}{k}\right],$$

$$\nu(n) = \sqrt{2/3}\, \pi \sqrt{n - 1/24},$$

and $\omega_{h,k}$ is a certain 24 kth root of unity given by

$$\omega_{h,k} = e^{\pi i s(h,k)}$$

where $s(h, k)$ is the Dedekind sum

$$\begin{aligned} s(h, k) &= \sum_{m=1}^{k-1} \left(\frac{m}{k} - \left\lfloor \frac{m}{k} \right\rfloor - \frac{1}{2}\right)\left(\frac{hm}{k} - \left\lfloor \frac{hm}{k} \right\rfloor - \frac{1}{2}\right) \\ &= \sum_{m=1}^{k-1} \left(\frac{m}{k} - \frac{1}{2}\right)\left(\frac{hm}{k} - \left\lfloor \frac{hm}{k} \right\rfloor - \frac{1}{2}\right), \end{aligned}$$

the last line following from the fact that $\lfloor x \rfloor = 0$ when $0 \le x < 1$. (This form along with a short survey of other equivalent representations is found in [14].) Hardy and Ramanujan's collaboration on this problem, and Rademacher's subsequent completion of their work has led to a powerful technique, the Circle Method, for solving certain types of additive problems. Hardy and Ramanujan developed an asymptotic expansion for $p(n)$ which was later proved by D. H. Lehmer to diverge [14]. Hans Rademacher later managed to identify an additional term that caused the infinite series to converge [18].

■ A Brief Mathematical Exploration

An exposition complete enough to justify the HRR would be far too cumbersome to include here. Indeed, even a brief sketch is a daunting task. ([13] provides a sketch of the circle method, the technique used to prove HRR, though not necessarily in the context of the partition function. [7], from which much of our discussion is derived, provides a detailed discussion of the Hardy-Ramanujan-Rademacher formula and its mathematical justification. See also [16] for a slightly different approach.) Thus we will hint at some of the most important highlights, the major facts which contribute to the result.

Let $P(q) = \prod_{n\geq 1} (1-q^n)^{-1}$ as in theorem 2. It is then obvious that $P^{(n)}(0) = n!\, p(n)$. Computing this directly is clearly more difficult than expanding the generating function, but recall Couchy's Integral Formula:

$$f^{(n)}(z) = \frac{n!}{2\pi i} \int_C \frac{f(s)}{(s-z)^{n+1}}\, ds$$

where C is any simple closed contour around the origin, we see that

$$p(n) = \frac{1}{2\pi i} \int_C \frac{P(q)}{q^{n+1}}\, dq, \tag{6}$$

where C is a contour within the unit circle. But how can (6) be evaluated? The problem with evaluating (6) is that $P(q)$ has an infinite number of factors. Notice also that $P(q)$ has a singularity at every integral root of unity. By exploiting a fact about these singularities, specifically the behavior of $P(q)$ "near" these singularities, we can work toward a formula for (6).

The key is to see that $P(q)$ is a modular form. Specifically, it can be shown that

$$P\left(\exp\left[\frac{2\pi i(h+iz)}{k}\right]\right) = \omega_{h,k} \exp\left[\frac{\pi(z^{-1}-z)}{12k}\right] P\left(\exp\left[\frac{2\pi i(h'+iz^{-1})}{k}\right]\right), \tag{7}$$

where Re(z) > 0, the principle branch of $z^{1/2}$ is selected, h' is a solution of the congruence $h\,h' \equiv -1 \pmod{k}$, and $\omega_{h,k}$ is a 24 kth root of unity defined in the previous section. Now when z is small, the argument of P on the LHS is near an integral root of unity, whereas the argument of P on the RHS is very close to zero (and thus P is close to 1). In other words, as $z \to 0$ with Re(z) > 0, it is clear that $\exp[2\pi i\,(h' + i\,z^{-1})/k] = \exp[2\pi(i\,h' - \frac{1}{z})/k] \to 0$ very quickly. Therefore, the contour C of integration should in some way be "centered" near these integral roots of unity in such a way as to leverage (7) so that the contribution of $P(z)$ is negligible.

Farey Fractions

Before the contour C is dissected, it will be quite useful to discuss our approach to these integral roots of unity. Let us call $\exp[2\pi i\, h/k]$ a *rational point* if $\frac{h}{k}$ is rational. (Note that the rational points are simply the integral roots of unity.) Clearly the rational points are dense on the unit circle. But recall the product expansion for P. The partial product $\prod_{n=1}^{N}(1-q^n)^{-1}$ $(N\epsilon\mathbb{Z}^+)$ has a pole of order N at $q=1$, a pole of order $[N/2]$ at $q=-1$, poles of order $[N/3]$ at $q=\exp[2\pi i/3]$ and $q=\exp[4\pi i/3]$, etc. Just as in our generating function algorithm above, we notice that factors appearing "early" in our product contribute more to the result. Indeed, were we to fix n, we might calculate $p(n)$ by simply selecting N large enough, and hence our set of relevant rational points would not be the countably infinite set of roots of unity, but $\{\exp[2\pi i\,\frac{h}{k}]: 0\le h\le k,\ 1\le k\le N\}$. Hence our strategy will be to divide the unit circle in such a way as to in some way "center" our curve of integration near all rational points $\exp[2\pi i\, h/k]$ with $0\le k\le N$, where N is fixed. Our guiding principle is that rational points with least denominator are most important to our calculations.

We now develop some useful theorems which will be useful in dissecting the contour of integration, which will be helpful in building an algorithm, and which are fascinating in their own right. (The proofs of theorems 15 and 16 can be found in [1].)

Definition 14: The ascending series of all irreducible fractions $\frac{h}{k}$, $0\le h\le k$ and $1\le k\le N$, is called the Farey series of order N, denoted F_N. The fractions $\sim$ are called Farey fractions.

Informally, we often take F_N to be a set. Thus,

$$F_6=\left\{0,\frac{1}{6},\frac{1}{5},\frac{1}{4},\frac{1}{3},\frac{2}{5},\frac{1}{2},\frac{3}{5},\frac{2}{3},\frac{3}{4},\frac{4}{5},\frac{5}{6},1\right\}.$$

Theorem 15: If $\frac{h}{k}$ and $\frac{h'}{k'}$ are two successive terms of F_n, then $k\,h'-h\,k'=1$.

Proof: Since $\gcd(h,k)=1$, the equation

$$k\,x-h\,y=1 \tag{8}$$

has integer solutions. If $(x_0,\ y_0)$ is a solution, then $(x_0+r\,h,\ y_0+r\,k)$ is also a solution for any positive or negative integer r. We can choose r so that

$$n-k<y_0+r\,k\le n.$$

There is therefore a solution $(x,\ y)$ of (3) such that

$$\gcd(x,\ y)=1,\quad 0\le n-k<y\le n. \tag{9}$$

Since $\frac{x}{y}$ is in lowest terms, and $y\le n$, $\frac{x}{y}$ is a fraction of F_n. Also, rearranging (8) yields

$$\frac{x}{y}=\frac{h}{k}+\frac{1}{k\,y}>\frac{h}{k},$$

so that $\frac{x}{y}$ comes later in F_n than $\frac{h}{k}$. If $\frac{x}{y}\ne\frac{h'}{k'}$, then $\frac{x}{y}$ comes later than $\frac{h'}{k'}$, and

$$\frac{x}{y} - \frac{h'}{k'} = \frac{k'x - h'y}{k'y} \geq \frac{1}{k'y};$$

while

$$\frac{h'}{k'} - \frac{h}{k} = \frac{kh' - hk'}{kk'} \geq \frac{1}{kk'}.$$

Hence

$$\frac{1}{ky} = \frac{kx - hy}{ky} = \frac{x}{y} - \frac{h}{k} \geq \frac{1}{k'y} + \frac{1}{kk'} = \frac{k+y}{kk'y} > \frac{n}{kk'y} \geq \frac{1}{ky}$$

by (9). But this is a contradiction. Thus it must be that $\frac{x}{y} = \frac{h'}{k'}$, and $kh' - hk' = 1$. ∎

Theorem 16: If $\frac{h}{k}$, $\frac{h''}{k''}$, and $\frac{h'}{k'}$ are three successive terms F_n, then $\frac{h''}{k''} = \frac{h+h'}{k+k'}$, the *mediant* of $\frac{h}{k}$ and $\frac{h'}{k'}$.

Proof: From theorem 15, we have $kh'' - hk'' = 1$, and $k''h' - h''k' = 1$. Solving these two equations for h'' and k'', we obtain

$$h''(kh' - hk') = h + h', \text{ and}$$
$$k''(kh' - hk') = k + k'.$$

Dividing equations, we obtain

$$\frac{h''(kh' - hk')}{k''(kh' - hk')} = \frac{h''}{k''} = \frac{h+h'}{k+k'}.$$ ∎

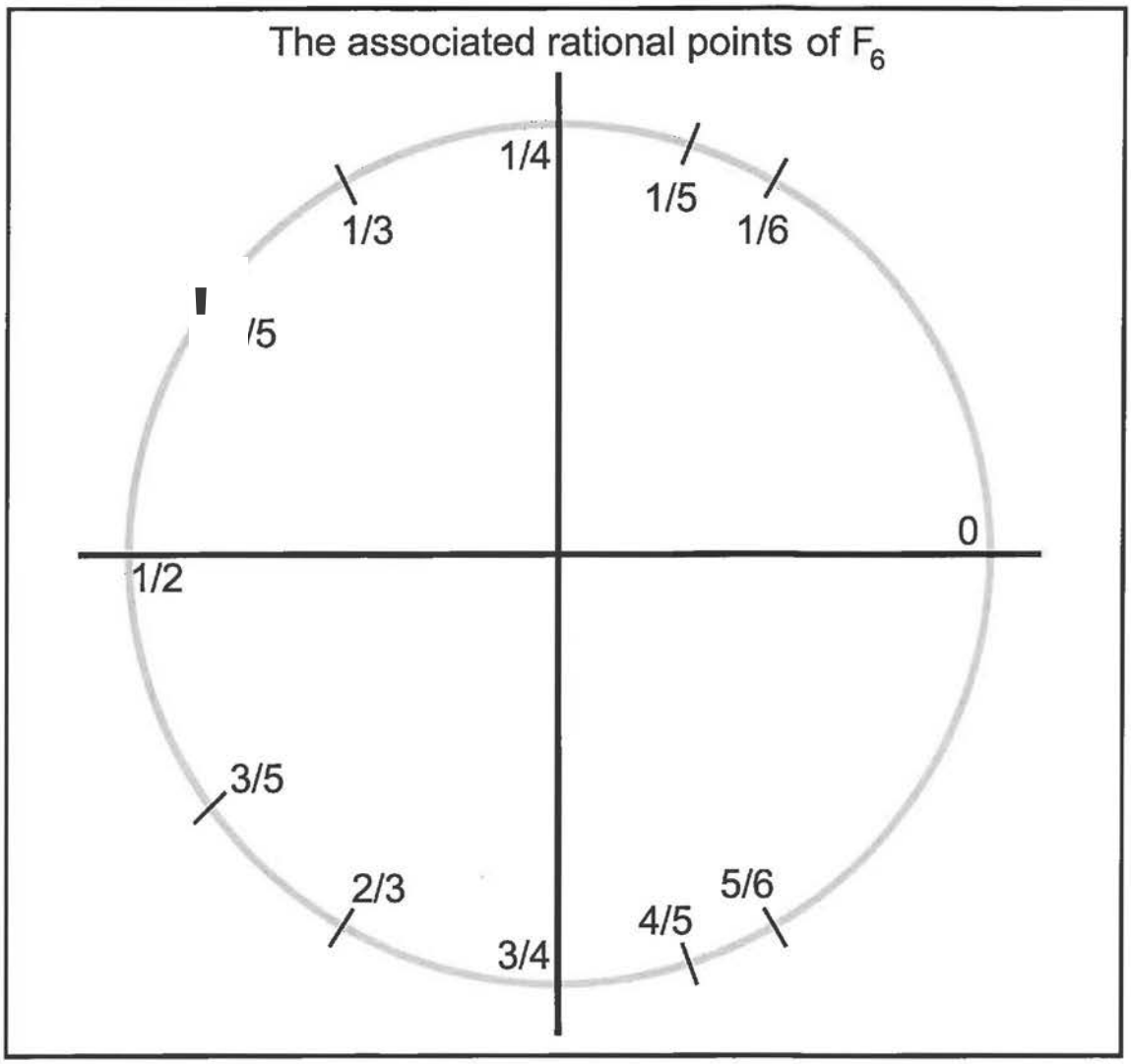
Theorem 17: If $\frac{h}{k}$, and $\frac{h'}{k'}$ are two successive terms of F_{n-1} but $\frac{h''}{k''}$ separates them in F_n, then $h'' = h + h'$ and $k'' = k + k'$, that is, $\frac{h+h'}{k+k'}$ is in reduced form.

Proof: Clearly $\frac{h''}{k''} = \frac{h+h'}{k+k'}$ by theorem 16. Suppose $d > 0$ divides both $h + h'$ and $k + k'$. Then d divides $k(h + h') + (-h)(k + k')$. But

$$k(h + h') - h(k + k') = kh' - hk' = 1$$

by theorem 15, since $\frac{h}{k}$ and $\frac{h'}{k'}$ are two successive terms in F_{n-1}. Hence $d = 1$, and thus $\gcd(h + h', k + k') = 1$. ∎

The curve of integration C will be dissected into arcs which are "centered" on the rational points associated with Farey fractions of order N in the sense that the endpoints of each arc are the rational points associated with the mediants of these Farey fractions. Hence if h_0/k_0, h/k, and h_1/k_1 are three consecutive terms of F_N, then the arc containing the rational point $e^{2\pi i \frac{h}{k}}$ has endpoints $e^{2\pi i\left(\frac{h}{k} - \frac{h_0+h}{k_0+k}\right)}$ and $e^{2\pi i\left(\frac{h_1+h}{k_1+k} - \frac{h}{k}\right)}$.



Evaluating Cauchy's Integral Formula

We continue to evaluate (7). The following divides the a circle into segments as described above: if h_0/k_0, h/k, h_1/k_1 are three consecutive terms of F_N, then define

$$\theta'_{0,1} = \frac{1}{N+1},$$

$$\theta'_{h,k} = \frac{h}{k} - \frac{h_0+h}{k_0+h} \text{ for } h > 0,$$

$$\theta''_{h,k} = \frac{h_1+h}{k_1+h} - \frac{h}{k}.$$

Let C be the curve $f(\phi) = \rho \exp[2\pi i \phi]$, $0 \le \phi \le 1$. (An appropriate ρ will be selected shortly.) Then we have

$$\begin{aligned} p(n) &= \frac{1}{2\pi i} \int_C \frac{P(s)}{s^{n+1}}\, ds \\ &= \frac{1}{2\pi i} \int_0^1 \frac{P(f(\phi))}{f(\phi)^{n+1}} f'(\phi)\, d\phi \\ &= \frac{1}{2\pi i} \int_0^1 \frac{P(\rho \exp[2\pi i \phi])}{\rho^{n+1} \rho \exp[2\pi i \phi (n+1)]} \rho 2\pi i \exp[2\pi i \phi]\, d\phi \\ &= \rho^{-n} \int_0^1 P(\rho \exp[2\pi i \phi]) \exp[-2\pi i n \phi]\, d\phi \\ &= \rho^{-n} \sum_{\substack{k=1 \\ \mathrm{GCD}(h,k)=1 \\ 0 \le h \le k}}^{N} \int_{-\theta'_{h,k}}^{\theta''_{h,k}} P\left(\rho \exp\left[\frac{2\pi i h}{k} + 2\pi i \phi\right]\right) \exp\left[-\frac{2\pi i n h}{k} - 2\pi i n \phi\right] d\phi \quad (10) \end{aligned}$$

where the final sum merely enumerates $\frac{h}{k} \in F_N$, Farey fractions of order N. Now to take advantage of (7), the argument of P must be

$$\exp\left[\frac{2\pi i(h + iz)}{k}\right] = \exp\left[\frac{2\pi i h}{k} - \frac{2\pi z}{k}\right],$$

yet in (10), the argument of P is

$$\rho \exp\left[\frac{2\pi i h}{k} + 2\pi i \phi\right] = \exp\left[\frac{2\pi i h}{k} + 2\pi i \phi + \ln[\rho]\right].$$

By selecting $\rho = \exp[\frac{-2\pi i}{N^2}]$ and $z = \frac{k}{N^2} - k i \phi$, the above becomes

$$\begin{aligned} \exp\left[\frac{2\pi i h}{k} + 2\pi i \phi + \ln[\rho]\right] &= \exp\left[\frac{2\pi i h}{k} - \frac{2\pi}{k}\left(\frac{k}{N^2} - i\phi k\right)\right] \\ &= \exp\left[\frac{2\pi i h}{k} - \frac{2\pi}{k} z\right]. \end{aligned}$$

Rewriting (10), we have

$$p(n) = \exp\left[\frac{2\pi n}{N^2}\right] \sum_{\substack{k=1 \\ \mathrm{GCD}(h,k)=1 \\ 0 \le h \le k}}^{N} \exp\left[\frac{-2\pi i h n}{k}\right] \int_{-\theta'_{h,k}}^{\theta''_{h,k}} P\left(\exp\left[\frac{2\pi i h}{k} - \frac{2\pi}{k} z\right]\right) \exp[-2\pi i n \phi]\, d\phi.$$

Now (7) may finally be applied to obtain

$$p(n) =$$

$$\exp\left[\frac{2\pi n}{N^2}\right] \sum_{\substack{k=1 \\ \mathrm{GCD}(h,k)=1 \\ 0\le h\le k}}^{N} \exp\left[\frac{-2\pi i h n}{k}\right] \omega_{h,k} \int_{-\theta'_{h,k}}^{\theta''_{h,k}} z^{1/2} \exp\left[\frac{\pi(z^{-1}-z)}{12k}\right] P\left(\exp\left[\frac{2\pi i (h'+i z^{-1})}{k}\right]\right) \exp[-2\pi i n \phi]\, d\phi.$$

Recalling that $z = \frac{k}{N^2} - k i \phi$, when $N \to \infty$, $z \to 0$, thus $\exp\left[\frac{2\pi i (h'+i z^{-1})}{k}\right] \to 0$ and hence $P\left(\exp\left[\frac{2\pi i (h'+i z^{-1})}{k}\right]\right) \to 1$. Thus it is expected that

$$\sum_{\substack{k=1 \\ \mathrm{GCD}(h,k)=1 \\ 0\le h\le k}}^{N} \exp\left[\frac{-2\pi i h n}{k}\right] \omega_{h,k} \int_{-\theta'_{h,k}}^{\theta''_{h,k}} z^{1/2} \exp\left[\frac{\pi(z^{-1}-z)}{12k} - 2\pi i n \phi\right] d\phi \tag{11}$$

is an estimate for $p(n)$ with error that approaches 0 as $N \to \infty$.

What is left is to transform the integral in (11) so that Cauchy's theorem can be applied. The integral will then be segmented into managable peices. Let $\omega = N^{-2} - i\phi$. Thus the integral in (11) becomes

$$\begin{aligned} I_{h,k} &= \int_{N^{-2}+i\theta'_{h,k}}^{N^{-2}-i\theta''_{h,k}} (k\omega)^{1/2} \exp\left[\frac{\pi}{12k}\left(\frac{1}{k\omega} - k\omega\right) + 2\pi n(\omega - N^{-2})\right] i\, d\omega \\ &= \exp[-2\pi n N^{-2}]\, k^{1/2}\, i^{-1} \int_{N^{-2}+i\theta'_{h,k}}^{N^{-2}-i\theta''_{h,k}} \omega^{1/2} \exp\left[\frac{\pi}{12k^2\omega}\right] \exp\left[2\pi\left(n - \frac{1}{24}\right)\omega\right] d\omega \\ &= \exp[-2\pi n N^{-2}]\, k^{1/2}\, i^{-1} \int_{N^{-2}+i\theta'_{h,k}}^{N^{-2}-i\theta''_{h,k}} g(\omega)\, d\omega \end{aligned} \tag{12}$$

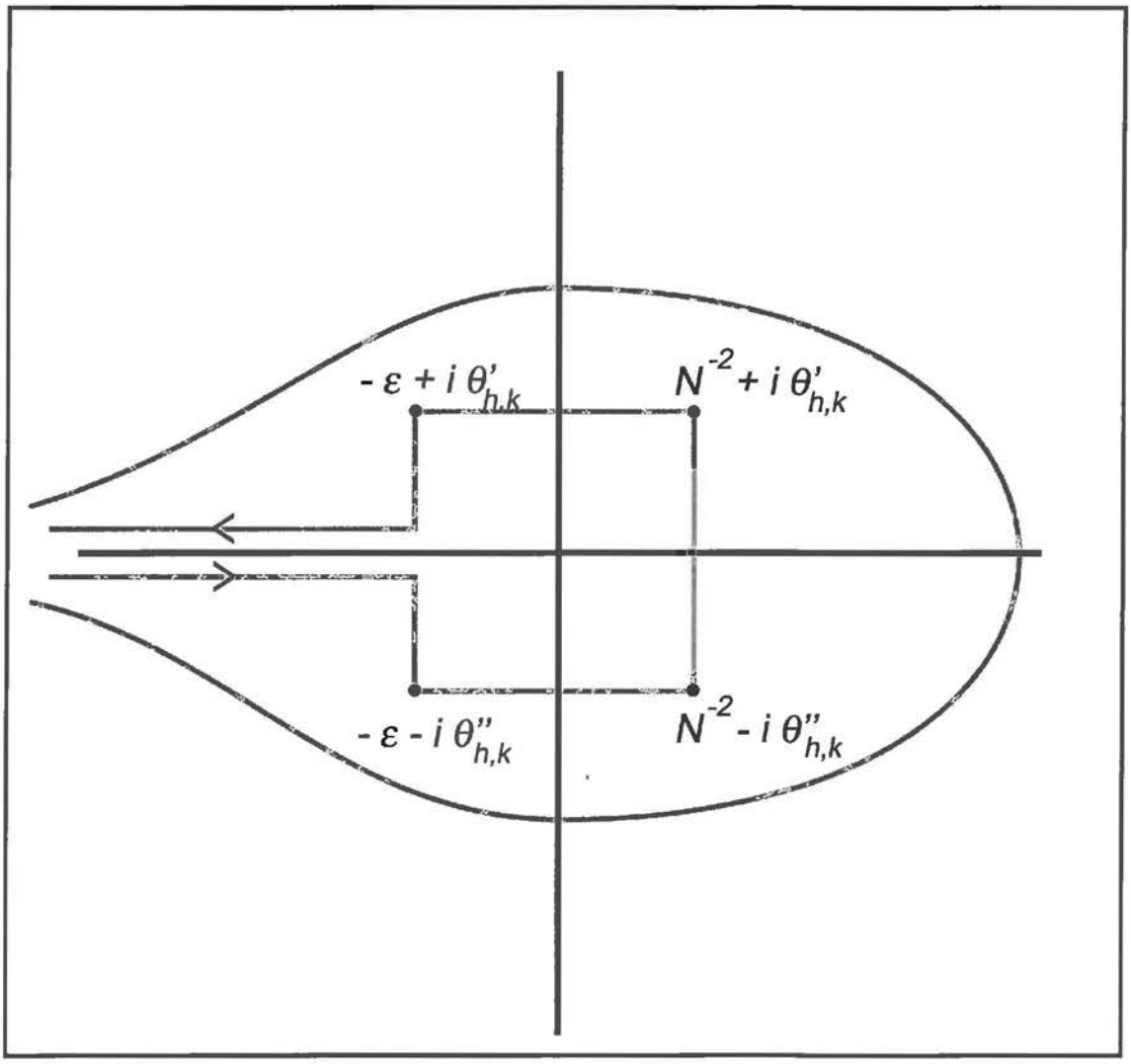
where

$$g(\omega) = \omega^{1/2} \exp\left[2\pi\left(n - \frac{1}{24}\right)\omega + \frac{\pi}{12k^2\omega}\right].$$

Now, g has a branch cut along the negative real axis and is analytic and single valued everywhere. Thus, applying Cauchy's theorem, (12) may be evaluated as

$$\begin{aligned} \exp[2\pi n N^{-2}]\, I_{h,k} &= \frac{k^{1/2}}{i}\left(\int_{-\infty}^{0+} - \int_{-\infty}^{-\epsilon} - \int_{-\epsilon}^{-\epsilon-i\theta''_{h,k}} - \int_{-\epsilon-i\theta''_{h,k}}^{N^{-2}-i\theta''_{h,k}} - \int_{N^{-2}+i\theta'_{h,k}}^{-\epsilon+i\theta'_{h,k}} - \int_{-\epsilon+i\theta'_{h,k}}^{-\epsilon} - \int_{-\epsilon}^{-\infty}\right) g(\omega)\, d\omega \\ &= \frac{k^{1/2}}{i}\,(L_k - I_1 - I_2 - I_3 - I_4 - I_5 - I_6) \end{aligned}$$

with contours according to the following figure:



Fortunately, it can be shown with some work that I_2, I_3, I_4, I_5, and I_6 are all negligible [7]. Moreover, it turns out that L_k and I_1 are able to be evaluated. Specifically, it is shown in [7] that as $N \to \infty$,

$$I_{h,k} = \frac{k^{1/2}}{i}(L_k - I_1) = \frac{k^{1/2}}{\pi\sqrt{2}}\left[\frac{d}{dx}\frac{\sinh\left[\frac{\pi}{k}\sqrt{\frac{2}{3}\left(x-\frac{1}{24}\right)}\right]}{\sqrt{x-\frac{1}{24}}}\right]_{x=n}.$$

Rewriting (11) by evaluating this differential, letting $N \to \infty$ and substituting $A_{h,k}$ yields the HRR in the form presented at the beginning of this section.

■ An Implementation of the Hardy-Ramanujan-Rademacher Formula

The outer most sum of HRR must have a finite number of terms for the algorithm to halt. Thus the sum will be over $1 \le k \le w$, where the value of w will be justified later. Note that k may be regarded as the denominator and h the numerator of Farey fractions of order F_w.

We will now develop a useful theorem. Let f be an arbitrary function over $\mathbb{Q}$. Let a, b, c, and d be in $\mathbb{Z}$ with $\gcd(a, b) = 1$ and $\gcd(c, d) = 1$. Define $S_n(\frac{a}{b}, \frac{c}{d})$ to be

$$S_n\left(\frac{a}{b}, \frac{c}{d}\right) = \begin{cases} S_n\left(\frac{a}{b}, \frac{a+c}{b+d}\right) + S_n\left(\frac{a+c}{b+d}, \frac{c}{d}\right) & \text{if } b+d \le n \\ f\left(\frac{c}{d}\right) & \text{if } b+d > n \end{cases}$$

Theorem 18:

$$S_n\left(\frac{0}{1}, \frac{1}{1}\right) = \sum_{1 \le k \le w} \sum_{\substack{1 \le h < k \\ \gcd(h,k)=1}} f\left(\frac{h}{k}\right).$$

Note: The right hand side is merely a sum over $F_n - \{\frac{0}{1}\}$, all Farey fractions of order n except 0. Our recursive function S_n is just a way to enumerate each Farey fraction without having to determine if $\gcd(h, k) = 1$ for every $1 \le h < k$.

Proof Sketch. Let $\frac{a}{b}$, $\frac{c}{d}$ be successive terms in some Farey series. The fraction with least denominator lying strictly between them is thus $\frac{a+c}{b+d}$ by theorem 17. Repeating this operation for the left interval $[\frac{a}{b}, \frac{a+c}{b+d}]$, we see that $\frac{a+a+c}{b+b+d}$ is the fraction with least denominator in this left interval. Continuing in this fashion for both left and right intervals while the denominator is less than or equal to the order n clearly yields all Farey fractions in the interval (0, 1]. Now consider points $\frac{a}{b}$, $\frac{c}{d}$ such that $b + d > n$. Then there are no more Farey fractions of order n lying strictly between $\frac{a}{b}$, $\frac{c}{d}$. Hence, $S_n(\frac{a}{b}, \frac{c}{d})$ computes $f(\frac{c}{d})$, and $f(\frac{c}{d})$ is only ever computed once.■

■ Pseudocode for the Hardy-Ramanujan-Rademacher Formula

Pseudocode for s(h, k)

```
s(input h, input k)
Begin s:
1)      Initialize result to 0

2)      For m = 1 to k – 1
        Do:
3)              Set result to result + (m/k – 1/2)(hm/k – ⌊hm/k⌋ – 1/2)
        Loop
End s
```

Line (3) clearly executes $k - 1$ times. Thus this algorithm is $O(k - 1)$.

Pseudocode to compute Farey fractions

We ultimately compute a real number for $A_k(n)$. Thus recalling Euler's formula we have

$$\mathrm{Re}(e^{i\theta}) = \mathrm{Re}(\cos[\theta] + i\sin[\theta]) = \cos[\theta],$$

and hence

$$\begin{aligned} A_k(n) &= \sum_{\substack{1\le h<k,\\ \gcd(h,k)=1}} \exp[\pi i\, s(h,k)] \cdot \exp\left[\frac{-2\pi i n h}{k}\right] \\ &= \sum_{\substack{1\le h<k,\\ \gcd(h,k)=1}} \exp\left[\pi i\left(s(h,k) - \frac{2nh}{k}\right)\right] \\ &= \sum_{\substack{1\le h<k,\\ \gcd(h,k)=1}} \cos\left[\pi\left(s(h,k) - \frac{2nh}{k}\right)\right]. \end{aligned}$$

However, k is merely the denominator and h the numerator of an element in F_w. Thus, using the same strategy as in theorem 18 we shall compute the outer sum over each ordered pair $(h,\ k)$ such that $\frac{h}{k} \in F_w$, instead of using a double sum. First, we calculate the ordered pairs $(h,\ k)$ using theorem 18. The following pseudocode, which includes the function `F` and a supporting function `RecurseF`, generates F_{order} in a number of steps proportional to the number of elements in $F_{\text{order+1}}$.

```
F(input order)
Begin F:
1)      Return RecurseF[order, 0, 1, 1, 1]
End F

RecurseF(input order, a, b, c, d)
Begin RecurseF:
2)      If b + d > order then return {}
3)      Otherwise, return RecurseF[order, a, b, a + c, b + d] ∪ {(a+c)/(b+d)} ∪ RecurseF[order, a + c, b + d, c, d]
End RecurseF
```

Line (1) clearly executes once, while lines (2) and (3) each execute once for every call to `RecurseF`. How many times is RecurseF called? Once for each element in F_{order} and once more for each pair of consecutive elements in F_{order} to reach the escape case in line (2). According to [15], if $N(n)$ is the number of terms in F_n, then $\lim_{n\to\infty} N(n) = \frac{3}{\pi^2} n^2$. Hence this algorithm is $O(n^2)$. (*Note*: the n here is not the argument to the partition function. Indeed, we need only compute F_w where w is the number of terms of outer sum in the Hardy-Ramanujan-Rademacher formula we need to take in order to compute $p(n)$. Typically w is very small. For $p(200)$, we may take $w = 5$. Thus perhaps we should say this algorithm is $O(w^2)$.)

Pseudocode to compute $p(n)$

There is one final issue before an algorithm can be written: how many terms of the outer sum must we take? It is trivial to show $A_k(n)$ is $O(k)$, a constant (at least in the context of A_k). Now,

$$\left(1-\frac{k}{\nu(n)}\right)\exp\left[\frac{\nu(n)}{k}\right]+\left(1+\frac{k}{\nu(n)}\right)\exp\left[-\frac{\nu(n)}{k}\right]$$
$$<\left(1-\frac{k}{\nu(n)}\right)\exp\left[\frac{\nu(n)}{k}\right]$$
$$<\exp\left[\frac{\nu(n)}{k}\right]$$

for sufficiently large n. Thus each term is $O\left(\exp\left[\sqrt{2/3}\,\pi\sqrt{n-1/24}\,/k\right]\right)$, or simply $O\left(\exp\left[c\sqrt{n}\,/k\right]\right)$. Now if c and l are constants with $0 \le l < 1$, asymptotically we desire $1 > l\,e^{c\sqrt{n}/k}$, and hence $k > \frac{-c\sqrt{n}}{\ln l}$. Thus we must only take a number of terms proportional to $\sqrt{n}$.

We are ready to put the pieces together.

p(input n)
Begin p:
1) **If $n < 0$ then**
2) **output 0**
3) **halt**
4) **If $n = 0$ then**
5) **output 1**
6) **halt**

7) **Initialize maxK to Max[1, $\lfloor\frac{3}{10}\sqrt{n}\,\rfloor$]**
8) **Initialize FareySet to F[maxK, 0, 1, 1, 1]**
9) **Allocate KCoefficients Array with Length maxK**

10) **For $k = 1$ to maxK**
Do:
11) **Set ν to $\sqrt{2/3}\,\pi\sqrt{n-1/24}$**
12) **Set KCoefficients[k] to**
$$\left\{\left(1-\tfrac{k}{\nu}\right)\exp\left[\tfrac{\nu}{k}\right]+\left(1+\tfrac{k}{\nu}\right)\exp\left[-\tfrac{\nu}{k}\right]\right\}/k$$
Loop

13) **Initialize result to 0**
14) **For each $\frac{h}{k} \in$ FareySet**
Do:
15) **Set result to**
$$\text{result} + \cos\left[\pi\left(s(h,k) - \tfrac{2nh}{k}\right)\right]\cdot \text{KCoefficients}[k]$$
Loop

16) **Set result to Round**$\left(\frac{\sqrt{12}}{24n-1}\cdot\text{result}\right)$
End p

■ Complexity Analysis

Clearly lines (11) and (12) execute $\left\lfloor \frac{3}{10}\sqrt{n} \right\rfloor$ times. Our concern will be line (15). Now there are a number of elements proportional to $\text{maxK}^2 < \left(\frac{3}{10}\sqrt{n}\right)^2 < n$ in FareySet. Moreover, $s(h, k)$ is $O(k)$. To make a simplifying assumption we may let $s(h, k)$ be $O(\text{maxK}) = O(\sqrt{n})$. Hence line (15) represents a number of steps proportional to $\sum_{i=1}^{n} \sqrt{n} = n^{2/3}$. But this is misleading, for the arithmetic operations can be carried out in nearly $O(\log p(n)) = O(\sqrt{n})$ steps, that is, "the number of bit operations is not much larger than the number of bits of p(n)" [19]. After the first few terms, the terms of the series "are of order $k^{-3/2}$ and usually of order k^{-2}. Furthermore, about half of the coefficients $A_k(n)$ turn out to be zero." For example, for $p(10^6)$, 123 of the required 250 $A_k(10^6)$terms are zero [16].

■ Considerations

The advantages of using the Hardy-Ramanujan-Rademacher formula (HRR) are hardly limited to its running time, although its running time is nearly as fast as the fastest alternative algorithms. Indeed, for extremely large values of n, HRR is often the only option, for all other algorithms have significant memory requirements in comparison. Consider, for example, Euler's recurrence, which must hold in memory not only the pentagonal numbers but also every $p(m)$, $0 \le m \le n$. Moreover, HRR can be used to generate symbolic results involving an undefined variable n with k terms of the formula. The symbolic output may be used to calculate $p(n)$ in only the time it takes to evaluate the symbolic expression. Additional terms may be generated on the fly without reference to a fixed n or k. (Remember: It is a simple matter to generated F_{w+1} very quickly from F_w.)

Why, we might ask, would anyone ever use any other aglorithm? For small values of n, HRR tends to have slightly more overhead than Euler's recurrence. Indeed, HRR is significantly harder to understand and implement, as the mathematics behind it are orders of magnitude more sophisticated. In addition, HRR requires high-precision arithmetic sooner than does Euler's recurrence. (See the Java source code listings in the appendices. An implementation of both algorithms using Java's double and long primitive data types has HRR giving inaccurate results for $n = 246$, whereas Euler's recurrence does not fail until $n = 405$.)

■ Other Methods of Determining Arithmetic Properties of $p(n)$

There are a few methods of determining the distribution of $p(n)$ modulus certain prime numbers without actually computing $p(n)$.

Davis and Perez have developed an algorithm that computes $p(n)$ modulus a prime number in $O(N \log\sim N)$ time [13]. Their algorithm begins with the polynomial in theorem 11, then uses a Fast Fourier Transform algorithm to invert it. Thus $p(n)$ can be computed by picking the prime number large enough. The details of the algorithm are yet unpublished, but see [13] for an outline of the technique.

The special case of determining the congruence mod **2** of $p(n)$ is particularly simple. (The proof, omitted here, is found in [7].)

Theorem 19:

$$\begin{aligned}
p(4n) &\equiv p(n) + p(n-7) + p(n-9) + \cdots + p(n-\alpha_i) + \cdots \pmod 2,\\
p(4n+1) &\equiv p(n) + p(n-5) + p(n-11) + \cdots + p(n-\beta_i) + \cdots \pmod 2,\\
p(4n+3) &\equiv p(n) + p(n-3) + p(n-13) + \cdots + p(n-\gamma_i) + \cdots \pmod 2,\\
p(4n+6) &\equiv p(n) + p(n-1) + p(n-15) + \cdots + p(n-\delta_i) + \cdots \pmod 2,
\end{aligned}$$

where $\alpha_i = i(8i \pm 1)$, $\beta_i = i(8i \pm 3)$, $\gamma_i = i(8i \pm 5)$, and $\delta_i = i(8i \pm 7)$.

Note that $O\left(\sqrt{n}\right)$ terms must be taken on the RHS, while each term requires $O(\log_4 k)$ recursive calls. Compare this to Euler's recurrence, where there are also $O\left(\sqrt{n}\right)$ terms on the RHS, but where each term requires $O(k)$ recursive calls. Thus an algorithm that exploits this recurrence is significantly faster than Euler's recurrence. Yet in spite of the simplicity of this special case, there are many unanswered questions about the parity of $p(n)$. (See [20].)

■ Conclusion

With the techniques discussed, the question of the unrestricted partition function's arithmetic properties may be attacked computationally. Indeed, $p(n)$'s distribution mod certain prime numbers has been calculated on super computers for millions of values of n [10], and questions regarding the parity of $p(n)$ have been vigorously investigated [12]. Naturally, however, results tend to be merely suggestive. Ultimately a rigorous proof is desired. Nonetheless, data produced by empirical investigation often reveals patterns which cry out for explanation, and new, groundbreaking questions arise. Thus, these techniques will continue to be invaluable to the study of the partition function.

■ Appendix A: Big-O Notation

Big-O notation is a way of describing an asymptotic upper bound on the running time of an algorithm. For a given function $g(n)$ we denote $O(g(n))$ (that is, "big O of g of n") by

$$O(g(n)) = \{f(n) : \exists\, c > 0\, \exists\, n_0 > 0\, \forall\, n \geq n_0 (0 \leq f(n) \leq c\, g(n))\}.$$

In other words, $O(g(n))$ is the set of all functions that eventually always remain below a fixed positive multiple of $g(n)$. The variable n is interpreted as the *size* of the input to the algorithm. For our purposes, n is the integer for which $p(n)$ is calculated. To take an example, the algorithm based on Euler's recurrence takes time proportional to $n^{3/2}$ to execute. Traditionally, one writes that an algorithm's running time $f(n)$ *is* $O(g(n))$, or alternatively, that $f(n) = O(g(n))$, even though $O(g(n))$ is a set of functions. Note that if $f(n)$ is $O(n)$, then $f(n)$ is also $O(n^2)$.

Most of the algorithms discussed are really asymptotically tightly bounded, that is, bounded above and below by g. We denote $\theta(g(n))$ (that is, "big theta of g of n") by

$$\theta(g(n)) = \{f(n) : \exists\, c_1 > 0\, \exists\, c_2 > 0\, \exists\, n_0 > 0\, \forall\, n \geq n_0 (0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n))\}.$$

In other words, $\theta(g(n))$ is the set of all functions that always remain between fixed positive multiples of $g(n)$. Thus the algorithm based on Euler's recurrence is in fact $\theta(n^{3/2})$. This paper is only really concerned with the upper bound. The asymptotic tightness of the bounds is usually obvious.

Appendix B: Source Code Listing for GeneratingFunct.java

```
 1: /* Robert Jacobson
 2:  * Expands the generating function to compute p(n),
 3:  * the number of unrestricted partitions of n. The
 4:  * generating function is adjusted to exclude any
 5:  * q^i with i > n, since such a term would not
 6:  * contribute to the coefficient of q^n. The new
 7:  * function is
 8:  *
 9:  * Prod[Sum[q^(i m)]]
10:  *
11:  * where the sum goes from i=0 to Floor[n/m] and
12:  * the product goes from m=1 to n.
13:  *
14:  * This algorithm is O( n^2 Log(n) ).
15:  *
16:  * For the sake of comparison, an algorithm that
17:  * does not take advantage the above fact, that is,
18:  * that merely expands the usual generating function,
19:  * is given as p2(n). This inefficient algorithm is
20:  * O( n^5 ).
21:  *
22:  * Both algorithms can calculate up to n= 405 before
23:  * failing due to overflow. However, the O( n^5 )
24:  * algorithm is almost guaranteed to fail far sooner
25:  * since it must allocate a tremendous amount of
26:  * memory.
27:  *
28:  */
29:
30: import java.lang.Math;
31:
32: public class GeneratingFunct{
33:     public static void main(String[] args) {
34:         System.out.println(p(405));
35:     }
36:
37:     //Refined algorithm: O( n^2 Log(n) )
38:     public static long p(long n){
39:     //trivial special cases.
40:     if(n<0) return 0;
41:     if(n==0) return 1;
42:
43:     int i, m;
44:     int j = 0;
45:     long steps = 0;
46:
47:     //This array holds coefficients of q^i in the ith position
48:     long coeffs1[] = new long[(int)n+1];
49:     //Init this array :)
50:     coeffs1[0] = 1;
51:     //This array is initially zeroed
52:     long coeffs2[] = new long[(int)n+1];
53:
54:     //perform the expansion
55:     for(m = 1; m <= n; m++){
56:         //This loop multiplies one factor by another.
57:         //If i went to n instead of n/m, n-i*m would
58:         //become < 0 in the innermost loop and an
59:         //assignment would never be made.
60:         for(i = 0; i <= (n/m); i++){
```

```
61:         //This innermost loop multiplies the intermediate
62:         //polynomial (initially 1) by a single term,
63:         //namely q^i*m, and adds the result to the prev poly.
64:         //We set coeffs=0 for every q^i with i>n.
65:             for(j = 0; j <= (int)n-i*m; j++){
66:             coeffs2[j+i*m] += coeffs1[j];
67:         }
68:         }
69:         //the destination becomes the source for the next iteration.
70:         for(j = 0; j <= n; j++){
71:         coeffs1[j] = coeffs2[j];
72:         coeffs2[j] = 0;
73:         }
74:     }
75:
76:     return coeffs1[(int)n];
77:
78:     }
79:
80:     //Inefficient algorithm: O( n^5 )
81:     public static long p2(long n){
82:     //trivial special cases.
83:     if(n<0) return 0;
84:     if(n==0) return 1;
85:
86:     int i, m;
87:     int j = 0;
88:     long max = (n*n*n + n*n)/2;
89:     long steps = 0;
90:
91:     //This array holds coefficients of q^i in the ith position
92:     long coeffs1[] = new long[(int)(n*n*n + n*n)/2 + 1];
93:     //Init this array :)
94:     coeffs1[0] = 1;
95:     //This array is initially zeroed
96:     long coeffs2[] = new long[(int)(n*n*n + n*n)/2 + 1];
97:
98:     //perform the expansion
99:     for(m = 1; m <= n; m++){
100:        //This loop multiplies one factor by another.
101:        //If i went to n instead of n/m, n-i*m would
102:        //become < 0 in the innermost loop and an
103:        //assignment would never be made.
104:        for(i = 0; i <= n; i++){
105:        //This innermost loop multiplies the intermediate
106:        //polynomial (initially 1) by a single term,
107:        //namely q^i*m, and adds the result to the prev poly.
108:            for(j = 0; j <= n*(m-1)*(m)/2; j++){
109:            coeffs2[j+i*m] += coeffs1[j];
110:            steps++;
111:        }
112:        }
113:        //The destination becomes the source for the next iteration.
114:        //Naturally there are more clever ways to do this.
115:        for(j = 0; j <= n*(m)*(m+1)/2; j++){
116:        coeffs1[j] = coeffs2[j];
117:        coeffs2[j] = 0;
118:        }
119:    }
120:
121:    return coeffs1[j];
122:
123:    }
124: }
```

■ Appendix C: Source Code Listing for SigmaRecurrence.java

```
/* Robert Jacobson
 * Uses a recurrence to compute p(n), the number
 * of unrestricted partitions of n. The recurrence is
 *
 * p(n) = 1/n Sum( sigma(k) p(n-k) )
 *
 * where sigma(k) is the sum of factors of k and the
 * sum is from k=1 to n.
 *
 * This algorithm is O( n^2 ). This implementation can
 * calculate up to p(316) = 28305020340996003 before
 * giving garbage results due to overflow.
 *
 */

public class SigmaRecurrence {
    public static void main(String[] args){
    System.out.println(p(316));
    }

    public static long p(long n){
    //trivial special cases.
    if(n<0) return 0;
    if(n==0) return 1;

    int i = 0;
    int k = 0;
    long partialsum = 0;
    float temp = 0;

    //sigma[k] holds sigma(k+1)
    long sigma[] = new long[(int)n];
    //pv[k] holds p(k)
    long pv[] = new long[(int)n + 1];
    //initialize pv
    pv[0] = 1;

    //Initialize sigma. Perhaps this could be done
    //more efficiently, but it's faster than
    //the rest of the method anyway.
    for(i = 1; i <= n; i++){
        //calculate sigma(i)
        for(k = 1; k <= i; k++){
        if((float)(i)/(float)k - (float)((i)/k) == 0){
            sigma[i-1] += k; //SUM of factors, not COUNT of factors
        }
        }
    }


    //Now calculate p(i) for 1 <= i <= n. (i=0 is already done.)
    for(i = 1; i <= n; i++){
        for(k = 1; k <= i; k++){
        partialsum += sigma[k-1]*pv[i-k];
        }
        pv[i] = partialsum/i;
        partialsum = 0;
    }
```

```
59:
60:     return pv[(int)n];
61:     }
62:
63: }
```

Appendix D: Source Code Listing for EulerRecurrence.java

```
 1: // Robert Jacobson
 2: // Uses Euler's recurrence to compute p(n), the number
 3: // of unrestricted partitions of n. The recurrence is
 4: //
 5: // p(n)=p(n-1)+p(n-2)-p(n-5)-p(n-7)+...
 6: //
 7: // where the numbers subtracted are the generalized
 8: // pentagonal numbers. This algorithm is O( n^(3/2) ).
 9: //
10: // This code can calculate up to p(405) before it fails
11: // due to overflow.
12: //
13: // Future improvements:
14: // Subsequent calls could be significantly expedited
15: // if partNums was cached. The algorithm would then
16: // be O[Sqrt(n)] if (n-1) has already been computed.
17:
18: import java.lang.Math;
19:
20: public class EulerRecurrence {
21:     public static void main(String[] args) {
22:     System.out.println(p(405));
23:     }
24:
25:     public static long p(long n){
26:     //trivial special cases.
27:     if(n<0) return 0;
28:     if(n==0) return 1;
29:
30:     //important things
31:     //K is in 1/2*3k^2+k
32:     //
33:     //we need at least one extra K to stop loops.
34:     long maxK = (long)((Math.sqrt(1+24*(double)n)-1)/6) + 1;
35:     long pentNums[] = new long[2*(int)maxK];
36:     long parts[] = new long[(int)n+1]; //From 0 to n means n+1 elements.
37:     long partialSum = 0;
38:     int i, j;
39:     int sign = 1;
40:
41:     parts[0] = 1;
42:
43:     //generate all the pentagonal numbers we need.
44:     for(i = 1; i<=maxK; i++){
45:         pentNums[2*i -2] = (3*i*i-i)/2; //the -2 in the array index is to zero-base
46:         pentNums[2*i+1 -2] = (3*i*i+i)/2;
47:     }
48:
49:     //calculate p(i) for i<=n.
50:     for(i = 1; i<=n; i++){
51:         partialSum = 0;
52:         //sum p(n-1) + p(n-2) - p(n-5) -...
53:         for(j = 1; pentNums[j-1]<=i; j++){
54:         //determine j's residue class mod 4. Signs come in pairs: +, +, -, -,..
55:         sign =(int)(((float)j/4 - (float)(j/4))*4);
56:         switch (sign){
57:         case 1:
58:             partialSum += parts[i-(int)pentNums[j-1]];
59:             break;
60:         case 2:
```

```
61:             partialSum += parts[i-(int)pentNums[j-1]];
62:             break;
63:         case 3:
64:             partialSum -= parts[i-(int)pentNums[j-1]];
65:             break;
66:         case 0:
67:             partialSum -= parts[i-(int)pentNums[j-1]];
68:         }
69:         }
70:         parts[i] = partialSum;
71:     }
72:     return partialSum;
73:     }
74: }
```

Appendix E: Source Code Listing for HRR.java

```
 1: // Robert Jacobson
 2: // Uses the Hardy-Ramanujan-Rademacher formula to
 3: // compute p(n), the number of unrestricted
 4: // partitions of n.
 5: //
 6: // See paper for the formula and discussion.
 7: //
 8: // This algorithm first fails at p(247) due to
 9: // inadequate precision of Java's primitive double.
10: //
11: // Future improvements:
12: //    The number of terms (maxk) is limited by
13: //    size(int) and accuracy of results is limited
14: //    by the precision of double. Arbitrary
15: //    precision arithmetic would allow arbitrary
16: //    accuracy (up to memory).
17: //    Calculating the terms symbolically would
18: //    allow all p(n) to be calculated in constant
19: //    time.
20: //    Caching symbolic terms would allow the next
21: //    term to be computed very quickly.
22:
23: import java.lang.Math;
24:
25: public class HRR {
26:
27:     public static void main(String[] args) {
28:     System.out.println(p(246));
29:     }
30:
31:     public static long p(long n){
32:     //trivial special cases.
33:     if(n<0) return 0;
34:     if(n==0) return 1;
35:
36:     double nd = (double)n; //convenience
37:     double result = 0; //convenience
38:     double dummy1 = 0;
39:     double dummy2 = 0;
40:     ListNode zero = new ListNode();
41:     ListNode one = new ListNode();
42:
43:     zero.h = 0d;
44:     zero.k = 1d;
45:     one.h = 1d;
46:     one.k = 1d;
47:     zero.next = one;
48:     one.next = null;
49:
50:     //Each term is O( exp[pi*sqrt(2n/3)/k]  ), so we only
51:     //need to take a*sqrt(n) where a is a carefully
52:     //selected constant. We carefully select a = .4
53:     //and take Max[Floor[a*sqrt(n)], 1] terms.
54:
55:     long maxk = (long)(0.4d*Math.sqrt(n));
56:     if(maxk == 0) maxk = 1;
57:
58:     //Compute Farey fractions of order maxk.
59:     ComputeFarey(maxk, zero, one);
60:
61:     //Calculate the coefficients of A_k(n) in the outer sum.
62:     double[] kcoeff = new double[(int)maxk];
63:     for(double k=1; k<=maxk; k++){
64:         dummy1 = Math.sqrt(2d/3d)*Math.PI*Math.sqrt(nd-1d/24d)/k;
65:         dummy2 = Math.exp( dummy1 );
66:         kcoeff[(int)k - 1] = ((1d - 1d/dummy1)*dummy2 + (1d + 1d/dummy1)/dummy2)/Math.sqrt(k);
```

```
67:     }
68:
69:     //We don't want 0/1 in our computations.
70:     ListNode cursor = zero.next;
71:
72:     //enumerate through each (h,k) pair
73:     while(cursor!=null){
74:         result += Math.cos( Math.PI*(s(cursor.h, cursor.k)
75:                 - 2*nd*cursor.h/cursor.k) )
76:               *kcoeff[(int)(cursor.k) -1];
77:         cursor = cursor.next;
78:     }
79:
80:     //Coefficient of the outermost sum.
81:     result = Math.sqrt(12)/(24*nd-1)*result;
82:
83:     return Math.round(result);
84:     }
85:
86:     public static double s(double h, double k){
87:     // This function calculates its result in O( k-1 ) time.
88:     double result = 0;
89:     double temp = 0;
90:     for(double m = 1; m < k; m++){
91:         temp = h*m/k;
92:         result += (m/k - 0.5d)*(temp - Math.floor(temp)-0.5d);
93:     }
94:
95:     return result;
96:     }
97:
98:     public static void ComputeFarey(long order,
99:             ListNode left, ListNode right){
100:    if(left.k + right.k > order) return;
101:
102:    ListNode middle = new ListNode();
103:    middle.k = left.k + right.k;
104:    middle.h = left.h + right.h;
105:
106:    left.next = middle;
107:    middle.next = right;
108:
109:    ComputeFarey(order, left, middle);
110:    ComputeFarey(order, middle, right);
111:
112:    return;
113:    }
114:
115: }
```

■ Bibliography

1. Hardy, G. H. and E. M. Wright, *An Introduction to the Theory of Numbers*, Clarendon Press, Oxford, 1962.

2. S. Ahlgren and K. Ono, *Congruences and conjectures for the partition function*, Contemporary Math. 291 (2001), 1-9

3. Ahlgren, Scott, *The partition function modulo composite integers M*, Mathematische Annalen 318, no.4 (2000), 795-803.

4. Ahlgren, Scott, and Ken Ono, *Congruence Properties for the partition function*, Proceedings of the National Academy of Sciences, U.S.A. 98 no. 9 (2001), 978-984.

5. Ono, Ken, *Distribution of the partition function modulo m*, Annals of Mathematics, 151 (2000), 293-307.

6. S. Ramanujan, *Congruence properties of partitions*, Proc. London Math. Soc. 19
(1919), 207-210, cited in Ono, Ken, *Distribution of the partition function modulo m*, Annals of Mathematics, 151 (2000), 293-307.

7. Andrews, George E., *The Theory of Partition Functions*, Cambridge University Press, Pennsylvania, 2003.

8. Ono, Ken. *Arithmetic of the partition function.*

9. S. Ahlgren and K. Ono, *Congruences and conjectures for the partition function*, Contemporary
Math.291 (2001), 1-9.

10. Swannack, Charles. Personal correspondence.

11. Pemmaraju, Sriram, "Integer Partitions and Generating Functions", lecture notes for Computational Combinatorics, fall 2001, The University of Iowa. Location: <http://www.cs.uiowa.edu/~sriram/196/fall01/>.

12. Davis, Jimena, and Elizabeth Perez, "Computations of the Partition Function, p(n)", July 18, 2002, Clemson University. Available online at: <http://www.math.clemson.edu/~kevja/REU/2002/JDavisAndEPerez.pdf>.

13. Miller, Steven, and Ramin Takloo-Bighash, *The Circle Method*, July 14, 2004, Ohio State University. Available online at: <http://www.math.ohio-state.edu/~sjmiller/reu/handouts/circlemethod.pdf>.

14. Finch, Steven R., *Integer Partitions*, unpublished note, September 22, 2004. Available online at: <http://pauillac.inria.fr/algo/csolve/prt.pdf>

15. Vardi, I. Computational Recreations in Mathematica. Reading, MA: Addison-Wesley, p. 155, 1991.

16. Knuth, D. E., "pre-fascicle" 3b of *Combinatorial Algorithms*, Vol. 4 of *The Art of Computer Programming* (unpublished), Addison-Wesley. Available online at: <http://www.cs-faculty.stanford.edu/~knuth/fasc3b.ps.gz>

17. Littlewood, J. E., *Collected Papers of Srinivasa Ramanujan* in the *Mathematical Gazette,* Vol. 14 (1929, pp. 427-428), as cited in Andrews, George E., *The Theory of Partition Functions*, Cambridge University Press, Pennsylvania, 2003.

18. Rademacher, H., "On the Partition Function p(n)." Proc. London Math. Soc. 43, 241-254, 1937.

19. Odlyzko, Andrew, "Asymptotic enumeration methods," in *Handbook of Combinatorics*, Vol. 2, R. L. Graham, M. Groetschel, and L. Lovasz, eds., Elsevier, 1995, pp. 1063-1229. Available online at: <http://www.dtc.umn.edu/~odlyzko/doc/asymptotic.enum.pdf>.

20. Berndt, Yee, and Zaharescu, "New Theorems on the Parity of Partition Functions," to be published in *J. Reine Angew. Math.* Available online at: <http://www.math.uiuc.edu/~berndt/articles/a2.pdf>.

Dec 2004 = Will Graduate

Southern Scholars Senior Project

Name: Robert Jacobson Date: 9/25/03 Major: Mathematics / Comp. Sci

Senior Project

A significant scholarly project, involving research, writing, or special performance, appropriate to the major in question, is ordinarily completed the senior year. The project is expected to be of sufficiently high quality to warrant a grade of A and to justify public presentation.

Under the guidance of a faculty advisor, the Senior Project should be an original work, should use primary sources when applicable, should have a table of contents and works cited page, should give convincing evidence to support a strong thesis, and should use the methods and writing style appropriate to the discipline.

The completed project, to be turned in in duplicate, must be approved by the Honors Committee in consultation with the student's supervising professor three weeks prior to graduation. Please include the advisor's name on the title page. The 2-3 hours of credit for this project is done as directed study or in a research class.

Keeping in mind the above senior project description, please describe in as much detail as you can the project you will undertake. You may attach a separate sheet if you wish:

I will explore the unrestricted partition function p(n) (which counts the additive partitions of n) and its restrictions and generalizations. Since, in spite of the simplicity of the function, little of its arithmetic properties is known, I will pay special attention to p(n)'s arithmetic properties, using computational tools and creating algorithms for discovering and confirming those properties.

Signature of faculty advisor [signature] Expected date of completion ≈Oct 2004 (Graduating Dec 2004)

Approval to be signed by faculty advisor when completed:

This project has been completed as planned: Yes

This in an "A" project: Yes

This project is worth 2-3 hours of credit: 2 hrs

Advisor's Final Signature [signature]

Chair, Honors Committee _____ Date Approved: _____

Dear Advisor, please write your final evaluation on the project on the reverse side of this page. Comment on the characteristics that make this "A" quality work.

Beverley Self

From: rljacobson [rljacobson@southern.edu]
Sent: Friday, October 31, 2003 11:05 AM
To: Beverley Self
Subject: RE: Senior Southern Scholars Important Information

I will be graduating in December of 2004. I HAVE turned in the proposal form to Dr. McClarty. I assumed that my project would not be due until some time in the fall of 2004, but your email made me realize that nobody actually told me this. Since I will be graduating in December of next year, when is my project due? (The issue of me graduating "late" has already been approved by the honors committee.)

In the case that you want my medallion information now (as opposed to fall '04), it should read: Robert Lawrence Jacobson. If this is too long, have it read: Robert L. Jacobson.

--Robert

>===== Original Message From "Beverley Self" <bdself@southern.edu> =====
>Hello Senior Southern Scholars,
>
>
>
>This message contains some very important information. Please pay
>attention.
>
>
>
>As seniors, I'm sure you realize you must turn in a senior project. I
>need to have you turn in a proposal for your project. This requires
>some thought. There is a form that needs to be completed and your
>advisor needs to sign it before you turn it in to me or Dr. Wilma
>McClarty.
>
>
>
>Your completed proposal MUST be turned in by Wednesday, December 10,
>2003.
>
>
>
>Many of you have spoken with Dr. McClarty and received the necessary
>form. If you have not, please come by her office, Brock Hall 322, as
>soon as possible.
>
>
>
>Just as a reminder:
>
>
>
>Your final project is due, April 16, 2004. You must turn in two (2)
>copies of your final project.
>
>
>
>On Tuesday, April 27, 2004, is the Dean's Luncheon where select
>Southern Scholars present their projects to the faculty.
>Unfortunately, there is not time for all of you to present but four or
>five of you will have an opportunity to share your projects. Just