Monte Carlo Technique for Prime Factorization

S Shashank
2022H1030067H
Computer Science and Information Systems
Birla Institute of Technology and Science Pilani, Hyderabad Campus
h20221030067@hyderabad.bits-pilani.ac.in

Ajinkya Medhekar
2022H1030099H
Computer Science and Information Systems
Birla Institute of Technology and Science Pilani, Hyderabad Campus
h20221030099@hyderabad.bits-pilani.ac.in

***Abstract*—There have been various approaches for finding prime factors of an integer N. One of these is Fermat's Factorization Algorithm (FFA) which finds two prime factors of an integer. FFA works best if the two prime factors are close to each other and fail if the integer N has more than two factors. Pollard presented a Monte Carlo factorization algorithm called the Pollard rho algorithm that finds a prime factor of a composite integer N and is faster than FFA. Pollard rho algorithm uses concepts of Floyd's cycle finding algorithm, Birthday Paradox, and Greatest Common Divisor (GCD). Later, Brent provided a prime factorization algorithm that uses his cycle finding algorithm and is faster than the Pollard rho algorithm. We have implemented and analyzed these two Monte Carlo algorithms for prime factorization and summarized the results.**

***Index Terms*—Monte Carlo, integer factorization, cycle-finding**

I. Introduction

This section discusses various approaches to solving the prime factorization problem.

A. Trial Division Technique

This technique relies on the property that all prime factors of a number n are less than $\sqrt{n}$. To determine the prime factorization of n, we divide the number by every divisor in the range [2, $\sqrt{n}$].[1]

B. Fermat's Factorization Method

This approach is suitable for determining factors close to each other. Fermat's equation states that an odd composite integer n is of the form:

$$n = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2 \tag{1}$$

Two flavors, FFA-1 and FFA-2, are proposed with improvements made to FFA.[2]

C. Pollard's Rho Algorithm

This algorithm is a randomized method that uses Floyd's cycle finding algorithm and the birthday paradox. Section III discusses this algorithm.

D. Brent's Algorithm

It is another randomized algorithm proposed by Brent that improves Pollard's algorithm and uses a cycle-finding algorithm proposed by Brent. Section III discusses this algorithm.

II. Theory

A. Greatest Common Divisor

The greatest common divisor, or GCD(m,n), is the largest common number that divides both m and n. An algorithm proposed by Euclid finds the GCD of two numbers in O(log n) time. For example, $GCD(42, 90) = 6$.
The algorithm by Euclid is given in Algorithm 1.[3]

Algorithm 1 Euclid GCD(m,n)

Input: m,n
Output: GCD of m and n
1: **if** $(m == 0)$ **then**
2: **return** n
3: **end if**
4: **return** $GCD(n\%m, m)$

B. Birthday Paradox

The birthday paradox is a widespread problem in probability theory that asks the following: What is the likelihood that at least two people in a room of n randomly chosen individuals have the same birthday? The probability of such an occurrence is surprisingly high for a small number of people. The odds of two people having the same birthday reach about 50 percent for only 23 people.[4]

C. Floyd's Cycle Finding Algorithm

This algorithm uses a two-pointer approach to find a cycle in a sequence. The first pointer moves once, but the second pointer advances two elements in each iteration. As a result, if a cycle exists, the hare makes at least one cycle before it meets the tortoise. The algorithm is described in Algorithm 2. [5]

Algorithm 2 Floyd's Cycle Detection

$x = x_0$
$y = x_0$
$j = 0$
repeat
  $j = j + 1$; $x = f(x)$; $y = f(f(y))$
until $x == y$

D. A Family of Cycle Finding Algorithms

Brent modified a general family of cycle-finding algorithms to speed up the prime factorization process.[5]

III. Monte Carlo Algorithms

A. Pollard rho algorithm

The Pollard rho algorithm[6] uses concepts like GCD, Floyd's Cycle Detection algorithm, and the Birthday Problem explained in section II. It takes a composite integer N as input and outputs d, one of the prime factors of N. To find the remaining factors of N, perform $N' = \frac{N}{d}$, and if N' is not prime and not divisible by the prime factors already found, pass N' to the algorithm to find its prime factors. To calculate all prime factors of N, we have to repeat this process till N' is itself prime. As suggested in [5], f evaluations are considered to measure the work done by the Pollard rho algorithm. Since we know that any prime factor p of an integer N cannot be greater than $N^{\frac{1}{2}}$, the expected number of f evaluations required by the Pollard rho algorithm will be O($N^{\frac{1}{4}}$) [5]. The Pollard rho algorithm for finding one of the prime factors of an integer N is described in Algorithm 3.

Algorithm 3 Pollard_Rho(N)

Input: N
Output: a prime factor of N
if $(N == 1)$ **then**
  return N
end if
if $(N\%2 == 0)$ **then**
  return 2
end if
x = random number in range [2,N-1]
y = x
c = random number in range [1,N-1]
d = 1
while $(d == 1)$ **do**
  $x = (x^2 \ \% \ N + c) \ \% \ N$
  $y = (y^2 \ \% \ N + c) \ \% \ N$
  $y = (y^2 \ \% \ N + c) \ \% \ N$
  $d = GCD(|x - y|, N)$
  if $(d == N)$ **then**
    return Pollard_Rho(N)
  end if
end while
return d

B. Brent's Prime Factorization Algorithm

The Brent prime factorization algorithm uses a modified cycle-finding algorithm from a family of cycle-finding algorithms[2][5]. Brent used his cycle-finding algorithm instead of Floyd's cycle-finding algorithm in the Pollard rho algorithm. On average, Brent's cycle-finding algorithm is about 36 percent faster than Floyd's [5]. The Brent prime factorization algorithm takes an integer N as input and outputs g, one of the prime factors of N. To find the remaining factors of N, perform $N' = N/g$, and pass N' directly to the algorithm again to obtain further factors. The Brent prime factorization algorithm for finding one of the prime factors of an integer N is described in Algorithm 4.

Algorithm 4 Brent(N)

Input: N
Output: a prime factor of N
if $(N == 1)$ **then**
  return N
end if
if $(N\%2 == 0)$ **then**
  return 2
end if
y = random number in range [1,N-1]
c = random number in range [1,N-1]
m = random number in range [1,N-1]
g = 1; r = 1; q = 1; ys = y; x = y
while $(g == 1)$ **do**
  x = y
  for i = 1 to r **do**
    $y = (y^2 \ \% \ N + c) \ \% \ N$
  end for
  k = 0
  while $(k < r \ and \ g == 1)$ **do**
    ys = y
    for i = 1 to min(m,r-k) **do**
      $y = (y^2 \ \% \ N + c) \ \% \ N$
      $q = q * |x - y| \ \% \ N$
    end for
    $g = GCD(q, N)$
    $k = k + m$
  end while
  $r = r * 2$
end while
if $(g == n)$ **then**
  while (True) **do**
    $ys = (ys^2 \ \% \ N + c) \ \% \ N$
    $g = GCD(|x - ys|, N)$
    if $(g > 1)$ **then**
      break
    end if
  end while
end if
return g

IV. Implementation

We implemented Algorithm 3 and Algorithm 4 on a system with Intel i3 7^{th} Gen 64-bit processor with 12 GB RAM and 1 TB HDD. These algorithms were implemented with modifications to find all prime factors of N. The respective code written in Python 3.9 can be found on GitHub[7].

V. Results

The average running times for Pollard and Brent's algorithms are tabulated in Table 1 and Table 2.

TABLE I
AVERAGE RUNNING TIME FOR N WITH TWO PRIME FACTORS

Runtime measured in milliseconds		
Number of digits in prime factor	Pollard-Rho Algorithm	Brent's Algorithm
10	210.74161	99.23044
11	927.94813	445.24462
12	2365.18395	905.07001
13	5780.29275	4571.04415
14	19254.78268	10496.45120
15	100551.48474	43384.81275

Table I shows a comparison between the running times of Pollard Rho Algorithm and Brent's Algorithm for a number N with 2 prime factors. We have considered different lengths of the prime factor for the analysis. These observations are visualised in Fig 1.

TABLE II
AVERAGE RUNNING TIME FOR N WITH MULTIPLE PRIME FACTORS

Runtime measured in milliseconds		
Number of factors of N	Pollard-Rho Algorithm	Brent's Algorithm
3	139.23307	106.11415
4	144.11473	122.17343
5	223.39928	153.58758

Table II shows a comparison between the running times of both the algorithms for a number N with multiple prime factors of length nine. We have considered different cases for the analysis. These observations are plotted in Fig 2.

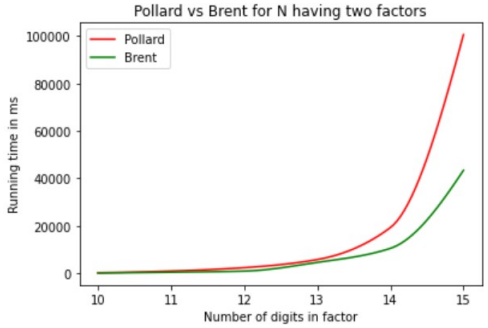

Fig. 1. Pollard vs. Brent for N having two factors

Figure 1 shows that the running times for both methods rise as the number of digits in the factor increases. Brent's algorithm is substantially faster than the Pollard-Rho algorithm. Brent has mathematically proved it.[5]

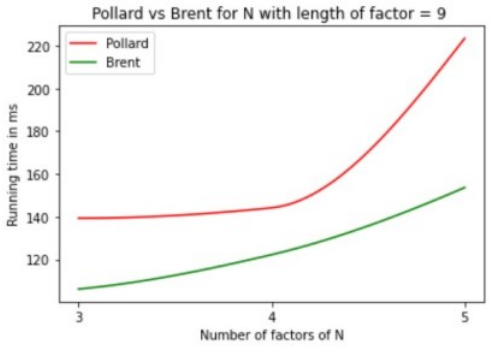

Fig. 2. Pollard vs. Brent for N having multiple factors

From Fig. 2, we observe that Brent's algorithm is more efficient than Pollard Rho algorithm for numbers with multiple factors as well.

VI. Conclusion

In this paper, we presented two Monte Carlo algorithms for prime factorization. Implementation and analysis of these algorithms reveal that both are more efficient than the traditional trial division technique and Fermat's Factorization Algorithm. Further, among the two Monte Carlo algorithms, the one proposed by Brent is faster than the Pollard Rho algorithm.

References

[1] Wunderlich, Marvin C., and John L. Selfridge. "A design for a number theory package with an optimized trial division routine." Communications of the ACM 17.5 (1974): 272-276.

[2] Somsuk, Kritsanapong. "The new integer factorization algorithm based on fermat's factorization algorithm and euler's theorem." Int. J. Electr. Comput. Eng 10 (2020): 1469-1476.

[3] D. E. Knuth, The Art of Computer Programming, vol. 2, Addison-Wesley, Reading, Mass., 1969.

[4] Mitzenmacher, Michael, and Eli Upfal. Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis. Cambridge university press, 2017.

[5] Brent, Richard P. "An improved Monte Carlo factorization algorithm." BIT Numerical Mathematics 20.2 (1980): 176-184.

[6] Pollard, J.M. A monte carlo method for factorization. BIT 15, 331–334 (1975).

[7] https://github.com/shashank-samavedula/aac-prime.git