T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

INTEGER FACTORIZATION

MEHMET ACAR

SUPERVISOR
DR. TÜLAY AYYILDIZ AKOĞLU

GEBZE
2023

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

INTEGER FACTORIZATION

MEHMET ACAR

SUPERVISOR
DR. TÜLAY AYYILDIZ AKOĞLU

2023
GEBZE

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 31/08/2021 by the following jury.

JURY

Member
(Supervisor) : Dr. Tülay Ayyıldız Akoğlu

Member : Dr. Tülay Ayyıldız Akoğlu

Member : Prof Dr. Didem Gözüpek

ABSTRACT

In this report, used integer factorization algorithms for improving is presented.Then, advantages of used algorithms from other algorithms is described. Later, and what is my contribution on the subject is described. Finally, code part and gui part of the project is given.

Keywords: keyword1, keyword2.

ÖZET

In this report, used integer factorization algorithms for improving is presented.Then, advantages of used algorithms from other algorithms is described. Later, and what is my contribution on the subject is described. Finally, code part and gui part of the project is given.

Anahtar Kelimeler: anahtar kelime1, anahtar kelime2.

ACKNOWLEDGEMENT

Thank you mate!!

Name Surname

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
ABC	:	The first three letters in the English alphabet
μ	:	Small mu

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In number theory, integer factorization is the decomposition, when possible, of a positive integer into a product of smaller integers. If the factors are further restricted to be prime numbers, the process is called prime factorization, and includes the test whether the given integer is prime (in this case, one has a ”product” of a single factor).

Prime factorization can be described by a graph as it is presented below, in Figure 1.1

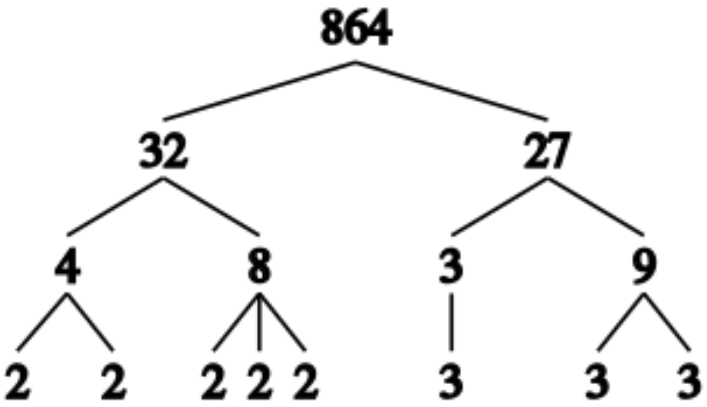


Figure 1.1: Prime Factorization

2. USED ALGORITHM FOR IMPROVING AND ADVANTAGES OF THIS ALGORITHM

I am going to use Pollard's Rho method for factoring a number into its prime divisors with below algorithm in Figure 2.1

1. Let n be the number we aim to factor. Start with random x and c. Define $f(x) = x^2 + c \pmod{n}$. Begin with y = x, and let d = 1.
2. While d == 1:
    a. Update x to f(x) [x_i, the tortoise]
    b. Update y to f(f(y)) [x_{2i}, the hare]
    c. Update d to gcd(|x-y|, n)
    d. If d != 1:
        i. If d == n, start again with a new set of x, y and c.
        ii. Else: d is the answer!

Figure 2.1: Pollard's Rho Algorithm

Pollard's Rho algorithm is used in this project. Because Pollard's Rho is a prime factorization algorithm, particularly fast for a large composite number with small prime factors.Its running time complexity is proportional to the square root of the size of the smallest prime factor and the amount of space used to execute the algorithm is much less than others.

We can prefer Pollard's Rho because we don't have to test all possible integers until a divisor is found, which means it will improve the time complexity. This algorithm is more efficient than Fermat's and wheel factorizations when comparing time and space complexity.

Pollard's Rho algorithm is a probabilistic algorithm for integer factorization, meaning that it may not always find the factors of a given number, but it has a high probability of success. Some advantages of Pollard's Rho algorithm over other integer factorization algorithms are:

1- Speed: Pollard's Rho algorithm is faster than some other factorization algorithms, such as trial division and Fermat's factorization method. Also it can be faster than quadratic sieve and the number field sieve, for smaller integers. It can handle larger numbers than these methods in reasonable time.

2- Memory efficiency: Pollard's Rho algorithm uses a relatively small amount of memory compared to other factorization algorithms, such as the general number field sieve. This makes it useful for factorizing large numbers on computers with limited memory.

3- Parallelizable: Pollard's Rho algorithm is highly parallelizable, which means that it can be run on multiple processors or cores at the same time. This makes it very fast for factorizing large numbers on machines with multiple processors or cores.

4- Suitable for composite numbers with small factors: Pollard's Rho algorithm is effective at factoring composite numbers with small factors. This makes it useful for factoring numbers that are the product of two relatively small primes.

5- Randomized Approach: The algorithm uses a randomized approach, which means that it has a chance of finding the factors of a composite number in a relatively short amount of time.

3. POLLARD'S RHO ALGORITHM EXAMPLE

Let us suppose n = 187 and consider different cases for different random values.

An Example of random values such that algorithm finds result: y = x = 2 and c = 1, Hence, our $f(x) = x^2 + 1$.

In the first step, we calculate f(x) and f(f(y)) values

$f(x) = 2^2 + 1 = 5$

$f(f(y)) = f(2^2 + 1) = f(5) = 5^2 + 1 = 26$

Then, we applying module operation to the f(x) and f(f(y)) numbers.

f(x) (mod n) = 5 (mod 187) = 5

f(f(y)) (mod n) = 26 (mod 187) = 26

Later, we update our x and y values with f(x) (mod n) and f(f(y)) (mod n) values. So, our new x value is 5 and our new y value is 26.

Then we calculate GCD value. In order to find GCD value, calculate the GCD of |x-y| and n.

GCD(|x-y|,n) = GCD(|5-26|,187) = GCD(21,187) = 1

Because of GCD value is equal to 1, we continue to the algorithm. In order to find the one prime factor of the input number(n), our GCD value should not be 1 or n.

Now, we calculate f(x) and f(f(y)) values again with new x and y values

$f(x) = 5^2 + 1 = 26$

$f(f(y)) = f(26^2 + 1) = f(677) = 677^2 + 1 = 458330$

Then, we applying module operation to the f(x) and f(f(y)) numbers.

f(x) (mod n) = 26 (mod 187) = 26

f(f(y)) (mod n) = 458330 (mod 187) = 180

Later, we update our x and y values with f(x) (mod n) and f(f(y)) (mod n) values. So, our new x value is 26 and our new y value is 180.

Then we calculate GCD value. In order to find GCD value, calculate the GCD of |x-y| and n.

GCD(|x-y|,n) = GCD(|26-180|,187) = GCD(154,187) = 11

Because of GCD value is equal to 11, we find the one prime factor of input number(n). Other prime factor of n is calculated with division of n with founded first prime factor. So, other prime factor is equal to 187/11 which is 17.

4. WHAT IS MY CONTRIBUTION ON THE SELECTED ALGORITHM

It is important to note that Pollard's Rho algorithm is a probabilistic algorithm, which means that it does not always give the correct result. In some cases, it may fail to find the factors of a composite number or give an incorrect factorization. Therefore, it is often used in combination with other factorization algorithms to increase the likelihood of finding the correct factors. In this project, Pollard's Rho algorithm is combined with trial division algorithm. With this contribution, this algorithm can make prime factorization of integers.

5. IMPROVED CODE OF POLLARD'S RHO ALGORITHM

```python
import random

def factorize(n):

    factors = []
    d = 2

    while n > 1:
        # Try to divide by small primes using trial division
        while n % d == 0:
            factors.append(d)
            n //= d

        # If n is still not 1, use Pollard's Rho algorithm
        if n > 1:
            x = random.randint(2, n-1)
            y = x
            c = random.randint(1, n-1)
            g = 1

            while g == 1:
                x = (x*x + c) % n
                y = (y*y + c) % n
                y = (y*y + c) % n
                g = gcd(abs(x-y), n)
                #print("g",g)

            # If Pollard's Rho found a non-trivial factor, add it to the list of factors
            if g != n:
                factors += factorize(g)
                factors += factorize(n//g)
                break

            # If Pollard's Rho failed, increment d and try trial division again
            else:
                d += 1

    return sorted(factors)

def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

Figure 5.1: Improved Code of Pollard's Rho Algorithm

This code implements a function called factorize which takes an integer n as input and returns a sorted list of its prime factors. The function first tries to divide n by small primes using trial division, and then uses Pollard's Rho algorithm if n is still not 1. The implementation of the factorize function relies on another helper function called gcd, which computes the greatest common divisor of two numbers.

The factorize function first initializes an empty list called factors to store the prime factors. It then initializes a variable d to 2 and enters a loop that runs while n is greater than 1. Inside the loop, it uses trial division to check if n is divisible by d. If it is, it appends d to the list of factors, updates n by dividing it by d, and continues trying to divide n by d until it is no longer divisible.

If n is still greater than 1 after the trial division step, Pollard's Rho algorithm is used to find a non-trivial factor of n. The algorithm generates two random numbers x and y and calculates their modular squares until a greatest common divisor g is found

that is greater than 1 and less than n. If g is not equal to n, the function recursively calls factorize() on g and n//g to find their prime factors and appends them to the list of factors.

If Pollard's Rho algorithm fails to find a non-trivial factor, d is incremented and trial division is tried again.

The function finally returns a sorted list of prime factors of n.

6. TESTING THE IMPROVED CODE OF POLLARD'S RHO ALGORITHM

The code is tested with different integers and results are given in the below figures.

```
factors = factorize(30)
print(factors)
```

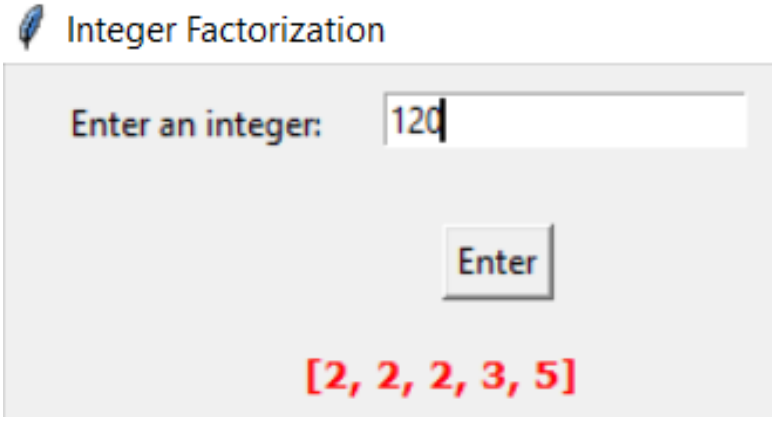
```
[2, 3, 5]
```

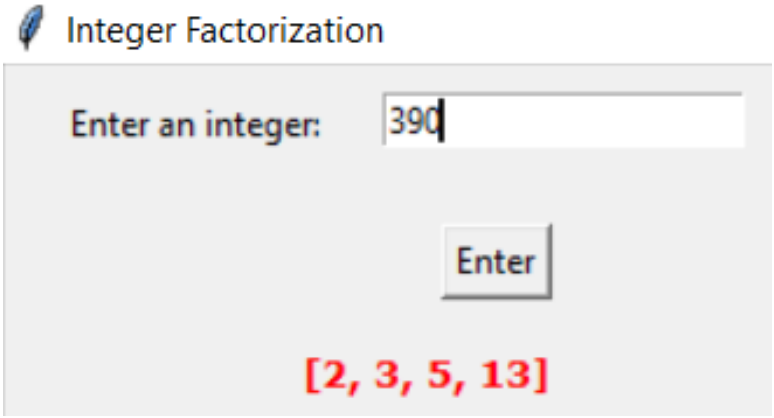
```
factors = factorize(390)
print(factors)
```

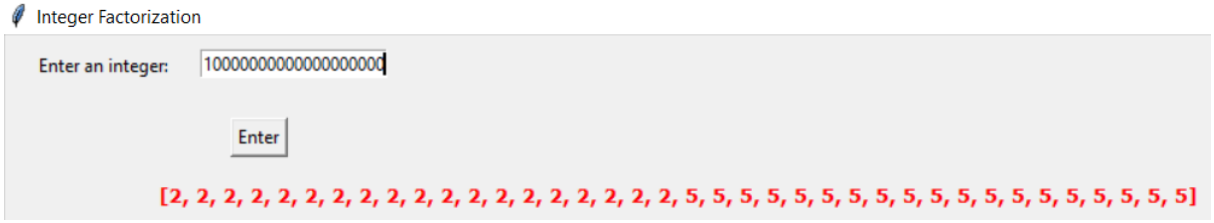
```
[2, 3, 5, 13]
```

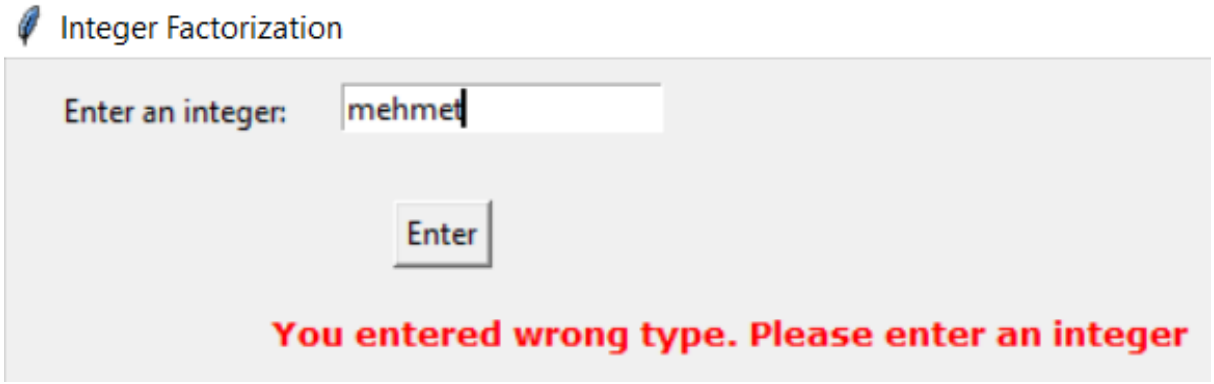
7. GUI FOR IMPROVED CODE OF POLLARD'S RHO ALGORITHM

For preparing GUI, tkinter library of Python is used. In the GUI, entry part is created for taking integer from user. Also, button is created in order to see prime factors of integer. When user writes a integer to the entry part and enters "Enter" button, user can see prime factors of integer. Also, in the GUI code, user entry is checked in order to give true results to the user. Only integer numbers are allowed for user input. If user writes other type than integer, GUI warns to the user with printing "You entered wrong type. Please enter an integer" message.




Integer Factorization
Enter an integer:
390
Enter
[2, 3, 5, 13]


Integer Factorization
Enter an integer:
10000000000000000000
Enter
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]


Integer Factorization
Enter an integer:
mehmet
Enter
You entered wrong type. Please enter an integer

BIBLIOGRAPHY

[1] zurGathen and Gerhard, *Modern Computer Algebra*. 1999, pp. 541–573.

[2] P. L. Jensen, "Integer factorization," pp. 1–85, 2005.

[3] J. Nilsson. "Integer factorization algorithms." (2020), [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1460632/FULLTEXT01.pdf.

[4] S. Babu. "Integer factorization algorithms." (), [Online]. Available: https://iq.opengenus.org/integer-factorization-algorithms/.

[5] J. Samuel S. Wagstaff. "History of integer factorization." (), [Online]. Available: https://www.cs.purdue.edu/homes/ssw/chapter3.pdf.

[1] [2] [3] [4] [5]

CV

XXX.

APPENDICES

Appendix 1: Some publications

No one significant, in AAA.

Appendix 2: Some explanation

None needed mate!