

ASSIGNMENT 1



DESIGN AND ANALYSIS OF ALGORITHM

IDAA432C

**Largest and Smallest Pythagorean Numbers from a Randomly Generated
Array of Natural Numbers.**

Submitted By:

Akshay Gupta (IIT2017505)

Naman Deept (IIT2017507)

Snigdha Dobhal (IIT2017506)

January 27, 2019

IDAA432C Assignment 1

Akshay Gupta IIT2017505, Naman Deepthi IIT2017507, Snigdha Dobhal IIT2017506

January 27, 2019

1 Abstract

This paper introduces several algorithms to find the largest and the smallest **Pythagorean** Numbers from the randomly generated array of Natural Numbers and then the most optimal approach to the problem is considered to efficiently solve the problem.

Keywords: *Pythagorean, Pythagorean Triplets, Primes, Largest, Smallest*

2 Introduction

Pythagorean Number: A Natural number **a** is said to be a Pythagorean number if it can be written as sum of the square of two smaller Natural numbers **b** and **c**, such that ,

$$a^2 = b^2 + c^2$$

Such a triple is commonly written (**a, b, c**), and a well-known example is (3, 4, 5). If (a, b, c) is a Pythagorean triple, then so is (**ka, kb, kc**) for any positive integer k. A primitive Pythagorean triple is one in which a, b and c are co-prime (that is, they have no common divisor larger than 1). A triangle whose sides form a Pythagorean triple is called a Pythagorean triangle, and is necessarily a right triangle. Any Pythagorean Number is a multiple of a **Pythagorean prime** (*n*) which is prime as well as of the form of $4n+1$.

When searching for integer solutions, the equation $a^2 + b^2 = c^2$ is a Diophantine equation. Thus Pythagorean triples are among the oldest known solutions of a nonlinear Diophantine equation.

3 Proposed Method

3.1 Naive approach

First method could be to check for all the numbers less than the value of **n** for a given number **n** if it can be written as sum of squares of such numbers. For this we need to find a pair of integer numbers **i** and **j** such that if $i^2 + j^2 = n^2$ then we can clearly say that our number is a Pythagorean number, else if the condition fails to hold for every possible integers **i** and **j** then our number is not Pythagorean.

3.1.1 Algorithm for the Naive Approach

```
1: procedure CHECKPYTHAGOREAN(n)
2:   for i ← 3 to n-1 do
3:     for j ← i+1 to n do
4:       if  $i^2 + j^2$  equals  $n^2$  then
5:         return 1
6:       end if
7:     end for
8:   end for
9:   return 0
10: end procedure
```

3.1.2 Time analysis

Best Case: The best case complexity will hold here if the value of *n* equals 5. For the value 5 both the loops will run for 1 time. Hence for the best case complexity the total time taken will be (3+3) units (1 for each loop computation). So for our best case complexity the time taken will be of the order of 1, so the time

taken in this case will be $t_\omega = 6$ But in our case the numbers are in the range of 10^4 to 10^6 .So the complexity will be increased for sure.The best case will be computed for 10^4 .Since $10^4 = 6.10^3 + 8.10^3$.So we can split i and j such that $i^2 + j^2 = 10^8$. So the minimum time taken in this case will be (assuming every operation takes a unit time) which is given by

$$t_\omega \propto \sum_{i=3}^{6.10^3} (n - i - 1)$$

which approximates to $\sim \omega(n^2)$ for the given range of n . Hence the complexity will be $\omega(n^2)$.

Average Case : The average case complexity will hold for the value where the loops are running by prerequisite greater time than the best case but lesser time than the worst case .Average case complexity is denoted by θ notation when the complexities in the best case and the worst case are the same.In this case it is same. So complexity is $\theta(n^2)$.

Worst Case : The worst case complexity can occur if the number is not indeed a Pythagorean number .So the entire loop iteration will run for maximum times and so the complexity will be $o(n^2)$.

3.2 Better Approach

A better method could be to check whether for every number from i to given number n if $n^2 - i^2$ is a perfect square .

3.2.1 Algorithm for the better approach

```

1: procedure CHECKPYTHAGOREAN( $n$ )
2:   for  $i \leftarrow 3$  to  $n$  do
3:      $k \leftarrow \sqrt{(n^2 - i^2)}$ 
4:      $l \leftarrow k^2$ 
5:     if  $l$  equals  $n^2 - i^2$  then
6:       return 1
7:     end if
8:   end for
9:   return 0
10: end procedure

```

3.2.2 Time Analysis

Best Case: For the best case of the algorithm we need to consider the smallest possible value in the range of the number ,and the value is 10^4 which itself takes $6.10^3 - 2$ iterations. Consider the time taken for the single iteration to be x units .So the total time will be approximately $t_\omega \propto (6.10^3 - 2).x$ and hence the complexity becomes overall closer to n .So the notation will be $\omega(n)$.

Average Case : In this case the complexity is $\theta(n)$.

Worst Case : The worst case complexity can occur if the number is not indeed a Pythagorean number .So the complexity will be $o(n)$.

3.3 Optimal solution

We know that all Pythagorean number are multiples of Pythagorean prime which are prime number of the form of $4n + 1$. Now to check whether the given number is Pythagorean number or not we can check if any of the prime factor of the given number is a Pythagorean prime or not .

3.3.1 Algorithm for the Optimal Solution

```

1: procedure CHECKPYTHAGOREAN( $n$ )
2:   while 2 divides  $n$  do
3:      $n \leftarrow n/2$ 
4:   end while
5:   for  $i \leftarrow 3$  to  $\sqrt{n}$  in step of 2 do
6:     if  $i$  divides  $n$  and 4 divides  $(n-1)$  then
7:       return 1
8:     end if
9:     while  $i$  divides  $n$  do
10:       $n \leftarrow n/i$ 
11:    end while
12:  end for
13:  if  $n > 2$  and 4 divides  $(n-1)$  then
14:    return 1
15:  end if
16:  return 0
17: end procedure

```

3.3.2 Time Analysis

Best Case: Best case holds for the least possible time to be taken from the number in the given range. In our case consider the smallest number say 10^4 in this case the complexity will be given as

$$t_{\omega} \propto \log_2 10^4 + v$$

where v is a lesser quantity than $\log_2 10^4$. So the complexity is $\omega(\log_2 n)$. This complexity will hold perfect for the number which are perfect powers of 2 that is of the form 2^n .

Worst Case : The worst case complexity can occur if the number is not indeed a Pythagorean number. So the loop iteration will run for $\log_2 k + \sqrt{z}$ times such that $z = (n/2^k)$. Out of both \sqrt{n} is greater and so the time will heavily be determined by this factor. So the complexity here will be $o(\sqrt{n})$.

2) **min** is an integer to store the minimum Pythagorean number from the given array of numbers and is initialized to 0.

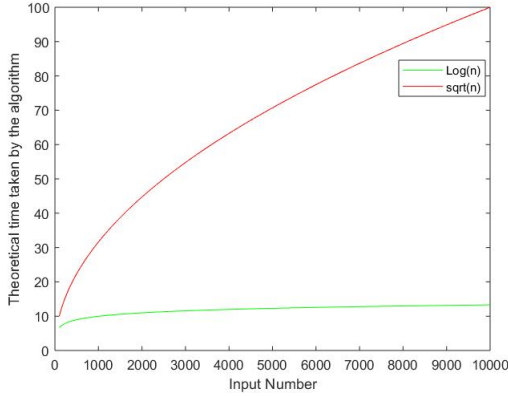
3) **max** is an integer to store the maximum Pythagorean number from the given array of numbers and is initialized to 1000000.

```

procedure MAIN
  for  $i \leftarrow 0$  to  $n$  do
    if checkPythagorean( $a[i]$ ) equals 1 then
      if  $min > a[i]$  then
         $min \leftarrow a[i]$ 
      end if
      if  $max < a[i]$  then
         $max \leftarrow a[i]$ 
      end if
    end if
  end for
end procedure

```

Figure 1: Comparing best case and worst case complexity in optimal approach

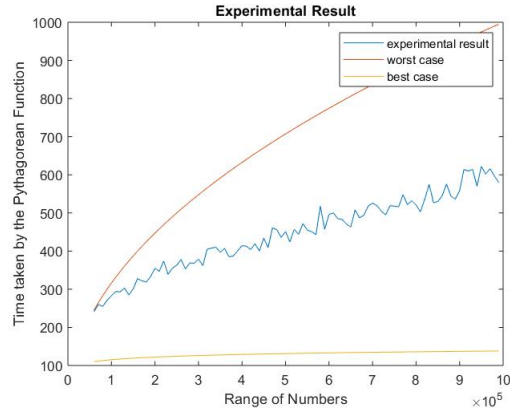


3.4 Main Function

3.4.1 Variables:

1) **a[]** is the given array of randomly generated natural numbers of size 1000 which is in the range from 10^4 to 10^6 .

4 Experimental Results



4.1 Complexity Analysis and Explanation

: The overall complexity in our problem turns out to be $O(k \cdot \sqrt{n}/2 + k \cdot \log(n))$ where k denotes the length

of the array. In our experimental analysis, we considered an array of k integers and then generated random numbers from 10^4 to 10^6 with the gap of 10^4 to a specific value. The plot for the graph can be explained as follows : Consider a value from the randomly generated numbers if it is of the form of $2^n \cdot p$ where p is prime, Then the time taken in this case will be $k \cdot (n + \sqrt{p}/2)$ which will be the worst case situation and the best case will be $k \cdot \log_2 n$ provided the loop terminates in the while block itself. Hence the overall complexity in the situation will lie in between those two values, that is

$$k \cdot \log_2 n < t_{avg} < k \cdot \sqrt{n} + k \cdot p$$

5 Discussion and Future Work

We have used the fact that all the Pythagorean numbers are multiples of Pythagorean prime, so we only checked if the smallest Pythagorean prime divides the given number. Other way could be, as we have been given array ranging from 10^4 to 10^6 , the largest prime factor of such number will be less than 10^3 if such a number is not prime else we need to check the given number is prime and of the form $4n+1$. We can store all the Pythagorean primes less than 10^6 in an array, and check the given number if divisible by any Pythagorean prime stored in this array. Only 80 such primes exist under 10^3 , so a lot of computations will be minimized. Also one could use sieve method to save all the Pythagorean primes and then use the same method to check if the number is Pythagorean number or not.

6 Conclusion

We analyzed various algorithms to find the largest and the smallest Pythagorean number and developed optimal approach to find the desired number from the randomly generated array of numbers. We also analyzed how Pythagorean prime factors were related to the problem.

7 References

1. IDAA432C (Design and Analysis of Algorithm) class lecture
2. Pythagorean triples reference - https://en.wikipedia.org/wiki/Pythagorean_triple
3. Introduction to Algorithms by Cormen