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)**

February 1, 2019

The Largest and the Smallest Pythagorean Number from randomly generated 1000 Natural Numbers

Akshay Gupta IIT2017505, Naman Deept IIT2017507 Snigdha Dobhal IIT2017506

February 1, 2019

1 Abstract

This paper introduces several algorithms to find the largest and the smallest **Pythagorean** Numbers from the randomly generated array of **N**atural Numbers and then the most optimal approach to the problem is considered to efficiently solve the problem.
Keywords: *Pythagorean*, *Pythagorean Triples*, *Primes*, *Euclid's Formula*, *Pythagorean prime*

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

Input: Given 1000 randomly generated Natural numbers in an array , the aim is to find the largest and the smallest Pythagorean Number from this array of natural numbers ranging in between 10^4 to 10^6.

3.1 Naive approach

First method could be to check whether there exist any number **i** less than the given number **n** such that $n^2 - i^2$ is a perfect square . If such a number exist then the given number is a Pythagorean Number .

3.1.1 Algorithm for the Naive Approach

```
1: procedure CHECKPYTHAGOREAN(n)
2:     for i ← 3 to n do
3:         k ← √(n² − i²)
4:         l ← k²
5:         if l equals n² − i² then
6:             return 1
7:         end if
8:     end for
9:     return 0
10: end procedure
```

3.1.2 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 Ω(n) .
Average Case : In this case the complexity is Θ(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).

3.2 Better Approach

A better method could be implementing **Euclid's** formula which is a general formula for generating Pythagorean numbers, given any arbitrary pair of integers m and n where $m > n > 0$, if $c^2 = a^2 + b^2$ then

$$c = m^2 + n^2, b = 2mn, a = m^2 - n^2$$

The triple generated by Euclid's formula is primitive if and only if m and n are coprime and not both odd. When both m and n are odd, then a, b, and c will be even, and the triple will not be primitive; however, dividing a, b, and c by 2 will yield a primitive triple when m and n are coprime and both odd.

3.2.1 Algorithm for the better approach

```
1: procedure CHECKPYTHAGOREAN(n)
2:     for i ← 1 to √n do
3:         k ← √(n − i²)
4:         l ← k²
5:         if l equals n − i² 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 only 60 iterations. Consider the time taken for the single iteration to be x units .So the total time will be approximately

$$t_\Omega \propto 60.x$$

and hence the complexity becomes overall closer to $\sqrt{n}$ which is 100.So the notation will be $\Omega(\sqrt{n})$.
Average Case : In this case the complexity is $\Theta(\sqrt{n})$.
Worst Case : The worst case complexity can occur if the number is not indeed a Pythagorean number .So the complexity will be O($\sqrt{n}$).

3.3 Optimal Approach

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 Approach

```
1: procedure CHECKPYTHAGOREAN(n)
2:     while 2 divides n do
3:         n ← n/2
4:     end while
5:     for i ← 3 to √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 ← 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 $o(\sqrt{n})$.

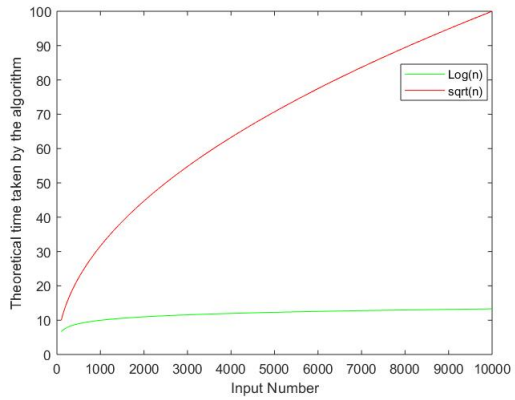

Figure 1: Comparing best case and worst case complexity in optimal approach

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 .

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 ← 0 to n do
        if checkPythagorean(a[i])equals 1 then
            if min > a[i] then
                min ← a[i]
            end if
            if max < a[i] then
                max ← a[i]
            end if
        end if
    end for
end procedure
```

4 Experimental Results

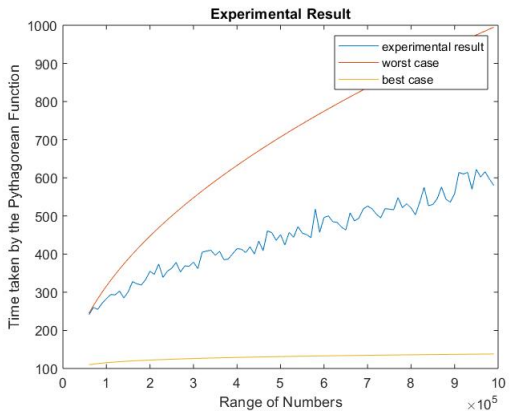

Figure 2: Graph showing the comparisons of the experimental performance of the algorithm with the theoretical analysis .

4.1 Complexity Analysis and Explanation

: The overall complexity in given problem turns out to be $o(k.\sqrt{n}/2 + k.\log(n))$ 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 $2^n.p$ where p is prime, Then the time taken in this case will be $k.(n + \sqrt{p}/2)$ which will be the worst case situation and the best case will be k.$\dot{l}og_2 n$ 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

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

where k is the length of the array.

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 . However , pre-computation of the problem to generate Pythagorean primes or with sieve method , the given problem may be more efficient if memory-time complexity trade-off is considered in favor of memory.

7 References

1. IDAA432C (Design and Analysis of Algorithm) class lecture
2. Pythagorean triples reference - $https://en.wikipedia.org/wiki/Pythagorean_triple$
3. Euclid's formula reference $http://demonstrations.wolfram.com/EuclidsFormulaAndPropertiesOfPythagoreanTriples/$
4. Introduction to Algorithms by Cormen