A Number theory approach to searching

Anshuman Singh

27/01/2025

Abstract

This paper proposes a number theory driven algorithm for fast searching on integers.

1 Introduction

Searching is of utmost importance in computer science, software development, scientific computing and many other spheres of technology, this paper aims to propose an algorithm to do searching fast based on mathematical insights and using inherent mathematical structures.

2 Related Work

Many efficient algorithms exist that serve the purpose of searching. Many of these algorithms are mathematics driven like the Newton-Raphson method and some methods use data structures like binary search tree. Many other intuitive algorithms like linear search are also widely used.

3 methodologies

The algorithm is built upon several mathematical insights which are as follows

3.1 result 1

The square of a positive integer N for all N belonging to the set of all positive integers can be expressed as the sum of the first N consecutive odd integers starting from 1. Formally,

$$sum_{i=1}^{N} 2i - 1 \tag{1}$$

where i belongs to the set of all natural numbers.

3.1.1 proof

For a given positive integer N,

$$N^2 = (1 + (N-1))^2 \tag{2}$$
$$N^2 = 1^2 + (N-1)^2 + 2 * (N-1) \tag{3}$$
$$N^2 = 1 + 2 * N - 2 + (N-1)^2 \tag{4}$$
$$N^2 = (2 * N - 1) + (N-1)^2 \tag{5}$$

The equation 5 proves that for any integer N belonging to the set of all positive integers, N^2 is equal to the sum of Nth odd integer starting from 1 and $(N-1)^2$.
The equation 5 can be generalized as,

$$(N-k)^2 = (2 * (N-k) - 1) + (N - (k-1))^2 \tag{6}$$

Where k belongs to the set of all whole numbers less than N.

From equation 6,the square of N can be written as,

$$(N-0)^2 = 2 * N - 1 + (N-1)^2 \tag{7}$$
$$(N-0)^2 = 2 * N - 1 + 2 * (N-1) - 1 + (N-2)^2 \tag{8}$$
$$(N-0)^2 = 2 * N - 1 + 2 * (N-1) - 1 + 2 * (N-2) - 1 + (N-3)^2 \tag{9}$$

On recursive application of the formula from the equation 6 the equation 9 can be expanded as.

$$N^2 = 2 * N - 1 + 2 * (N-1) - 1 + ... + 1 \tag{10}$$

From equation 10 the consistency of result 1 is verified and N^2 is expressed as the sum of first N consecutive odd integers starting from 1.

3.2 result 2

For a set of consecutive odd integers starting from 1 and the length of the set be of the form 3^k for k belonging to the set of all natural numbers, the sum of all the elements of the set is equal to 9^k. Formally, for the length of the set to be given by X and,

$$X = 3^k \tag{11}$$

Where k belongs to the set of all natural numbers then this specifies that the sum of all the elements of the set given by S is,

$$S = 9^k \tag{12}$$

3.2.1 proof

Let S be the sum of first 3^k consecutive odd integers starting from 1.

$$S = 1 + 3 + ...(1 + (3^k - 1) * 2) \tag{13}$$
$$S = sum_{n=1}^{3^(k-1)}(6n - 1 + 6n - 3 + 6n - 5) \tag{14}$$
$$\tag{15}$$

where n is a natural number.

$$S = sum_{n=1}^{3^{(k-1)}} 9(2n-1) \tag{16}$$

$$S = sum_{n=1}^{3^{(k-2)}} 9^2(2n-1) \tag{17}$$

$$...S = sum_i n = 1^{3^0} 9^k(2n-1) \tag{18}$$

$$\tag{19}$$

As in the equation 18 $3^(k-k)$ or 1 is the maximum value of n and 1 is also the lowest value of n in the summation, the equation 18 reduces to,

$$S = 9^k(2*1-1) \tag{20}$$

$$S = 9^k \tag{21}$$

The equation 21 verifies the consistency of result 2.

4 The algorithm

4.1 step 1

Initialize two variables *low* and *high* for the lower and upper limits satisfying the condition respectively.

4.2 step 2

Make sure *high* and *low* are odd. If not make them odd by subtracting -1 from *low* if it is even or adding 1 to high if it is even or do both if they both are even.

4.3 step 3

Initialize a variable $terms = (high+1)/2 - (low+1)/2$.

4.4 step 4

Initialize a variable $range = (terms * (low + (terms - 1))$.

4.5 step 5

Initialize a temporary variable $temp = range$ and a variable $tripletSize = 1$.

4.6 step 6

Iteratively divide *temp* by 9 until it becomes 0 and for each iteration multiply *tripletSize* by 3.

4.7 step 7

Initialize variables $sum1$ and $sum2$ also initialize a variable $unitSize = tripletSize/3$ and reassign $temp = range$.

4.8 step 8

Initialize a variable $ans = 0$ to hold the answer of search.

4.9 step 9

Assign $sum1 = (unitSize * (low + (unitSize - 1)))$ representing the arithmetic sum of $unitSize$ terms starting from low and having a common difference of 2.

4.10 step 10

Assign $sum2 = ((unitSize*2)*(low+((unitsize*2)-1))$ the arithmetic sum of $unitsize*2$ terms starting from low and having a common difference of 2.

4.11 step 11

Do one of the following as per your requirements.

4.12 searching for maximum integer satisfying the given condition

Check if $sum2$ satisfies the given condition, now two cases arise.

4.12.1 $sum2$ satisfies the condition

In this case increment the value of ans by $sum2$ and decrement the value of $temp$ by $sum2$.Assign $low = unitSize * 2 + 2$.

4.12.2 $sum2$ does not satisfy the condition

In this case check if $sum1$ satisfies the given condition. Now two cases arise

4.12.3 $sum1$ satisfies the given condition

In this case increment the value of ans by $sum1$ and decrement the value of $temp$ by $sum1$.Assign $low = unitSize * 4 + 2$.

4.12.4 $sum2$ does not satisfy the given condition

This means that there is no integer lying between low and $high$ which satisfies the given condition.

4.13 searching for minimum integer satisfying the given condition

Check if $sum1$ satisfies the given condition, now two cases arise.

4.13.1 $sum2$ satisfies the condition

In this case increment the value of ans by $sum1$ and decrement the value of $temp$ by $sum1$.

4.13.2 $sum1$ does not satisfy the condition

In this case check if $sum2$ satisfies the given condition. Now two cases arise

4.13.3 $sum2$ satisfies the given condition

In this case increment the value of ans by $sum2$ and decrement the value of $temp$ by $sum2$.

4.13.4 $sum2$ does not satisfy the given condition

This means that there is no integer lying between low and $high$ which satisfies the given condition.

4.14 step 12

Reassign $unitSize = unitSize/3$.

4.15 step 13

Iteratively repeat each step from step 9 to step 12 until either $temp$ or $unitSize$ falls below 1.

4.16 step 14

4.16.1 Maximum integer satisfying the condition

Check if $ans + 1$ satisfies the given condition, if it does return $ans + 1$ else return ans. Minimum integer satisfying the condition Check if $ans - 1$ is greater than 0 and satisfies the given condition, if it does return $ans - 1$ else return ans.

4.17 C++ Implementation

Repository for C++ implementation

5 Discussion

The proposed algorithm can be used for searching positive integers satisfying a given conditions and minor adjustments can be made to use the algorithm for searching on negative integers.

6 Conclusion

The algorithm uses inherent mathematical structures for searching for positive integers satisfying a given condition Future work will be aimed towards broadening the scope of implementation of the algorithm.