Assignment 2

Design and Analysis of Algorithm

IDAA432C

To Find the reciprocal of a given Number without using Multiplication, Division and Modulus Operator

---

Submitted By:
Akshay Gupta **(IIT2017505)**
Naman Deept **(IIT2017507)**

February 4, 2019

Computing the reciprocal (Multiplicative inverse) of any given number

Akshay Gupta IIT2017505, Naman Deept IIT2017507

Dept. of Information Technology
Indian Institute of Information Technology Allahabad

February 4, 2019

1 Abstract

This paper introduces an efficient algorithm for any integer ,to find the reciprocal of the number and then goes on to find the reciprocal of any real number to a required precision without using multiplication , division or modulus operator.
Keywords: *Multiplicative Inverse*, *Reciprocal*, *Inverse*, *Newton Raphson*, *Bit Manipulation*

2 Introduction

In mathematics, a multiplicative inverse or reciprocal for a number x, denoted by $1/x$ or x^{-1}, is a number which when multiplied by x yields the multiplicative identity, 1. The multiplicative inverse of a fraction a/b is b/a. For the multiplicative inverse of a real number, divide 1 by the number. For example, the reciprocal of 5 is one fifth (1/5 or 0.2), and the reciprocal of 0.25 is 1 divided by 0.25, or 4. The reciprocal function, the function $f(x)$ that maps x to $1/x$, is one of the simplest examples of a function which is its own inverse (an involution).
To find the reciprocal of a given number without using multiplication , division and modulus operators we can either use either bit manipulation , iterative subtraction or Newton-Raphson method.

3 Proposed Method

Input : Given any number,to find the multiplicative inverse of the number to any given precision using various approaches.The number can be any number integer and any rational number.

3.1 Bit Manipulation

The algorithm of Bit Manipulation can be able to compute only those multiplicative inverses where the value of the desired number is only an integer.

3.1.1 Algorithm

```
1:  procedure DIVIDEMOD(dividend, num)
2:      quotient ← 0
3:      temp ← 0
4:      for i ← 31 to 0
5:          if temp + (divisor << i) ≤ 1
6:              temp ← temp + divisor << 1
7:              quotient ← quotient|1 << i
8:              quotientresult ← quotient
9:          end if
10:     end for
11:     return divisionresult ,quotientresult
12: end procedure
```

```
1: procedure DIVIDE(dividend, num)
2:     NoDecimal ← 1
3:     quotient ← divisionresult
4:     dividend ← quotientresult
5:     if dividend! = 0
6:         quotient ← quotient + "."
7:     end if
8:     while NoDecimal ≤ 10 and dividend ≠ 0
9:         dividend ← dividend<<3 + dividend<<1
10:        dividemod(dividend, divisor)
11:        quotient ← quotient + quotientresult
12:        dividend ← dividendresult
13:        NoDecimal ← NoDecimal + 1
14:    end while
15:    return quotient
16: end procedure
```

```
procedure GENERATEZERO(num)
    temp ← emptyString
    numZero ← EmptyString
    temp ← num ToString
    for i ← 1 to length(temp)
        if temp.charAt(i) equals '.'
            continue
            numZero ← numZero + '0'
        end if
    end for
    return numZero
```

```
procedure ADDZERO(num)
    Result ← emptyString
    for i ← 1 to num
        Result ← Result + 0
    end for
    return Result
```

```
procedure RETURNBACK(num)
    str ← emptyString
    res ← emptyString
    str ← num ToString
    for i ← 1 to length(str)
        if temp.charAt(i) equals '.'
            continue
            res ← res + str.charAt(i)
        end if
    end for
    return res To Integer
```

3.1.2 Time Complexity Analysis

Best Case- Given algorithm in the best case scenario works on $\Omega(\log(n))$.These are those numbers who divide numbers which are power of 10 perfectly.
Worst Case- Given algorithm in the worst case scenario will be proportional to $O(log(n) \times b)$.Which will be very large if non - terminating , non -repeating number decimal producing or repeating decimals point generating numbers are there.Where b is the desired precision level.

3.2 Iterative Subtraction

This method works for all real number n . Here first for any desired precision, given number is first converted to a corresponding integer and then smallest power of 10 larger than the corresponding integer is subtracted from it to generate the quotient . The same procedure is done via iteration to generate till the desired precision and then added to quotient through string manipulation .

```
1: procedure FINDRECIPROCAL(num)
2:     Result ← Empty String
3:     if num < 0
4:         Result ← " − 0."
5:     end if
6:     if num < 0 and num notEquals 1
7:         Result ← "0."
8:         num ← −num
9:     end if
10:    if num equals 1
11:        Result ← Empty String
12:    end if
13:    string ← EmptyString
14:    Zeroes ← EmptyString
15:    string ← num(ToString)
16:    Zeroes ← 1 + GenerateZeroes(num)
17:    Dividend ← 0
18:    Divisor ← 0
19:    Dividend ← Zeroes (To Integer)
20:    Divisor ← Returnback(num)
21:    Precision ← 0
22:    Quotient ← 0
23:    Result ← Result + generateAZero(num)
24:    while Precision ≤ 10
25:        Quotient ← 0
26:        while Dividend ≥ Divisor
27:            Dividend ← Dividend − Divisor
28:            Quotient ← Quotient + 1
29:        end while
30:        if Dividend equals 0
31:            Break
32:        end if
33:        DividendString ← EmptyString
34:        DividendString ← Dividend ToString
35:        Counter ← 0
36:        while true
37:            Dividend ← DividendString ToInt
38:            DividendStr ← DividendStr + "0"
39:            if Dividend > Divisor
40:                Break
41:            end if
42:            Precision ← Precision + 1
43:            Counter ← Counter + 1
44:        end while
45:        Result ← Result + addZero(Counter)
46:        Precision ← Precision + 1
47:        Result ← Result + Quotient
48:    end while
49:    return Result
50: end procedure=0
```

3.2.1 Time Complexity Analysis

Best Case Analysis : The best case for the Iterative Subtraction holds when the numbers are perfect powers of 2 or 5 or both. i.e of the form 2^n ,5^n or 10^n. When we consider the analysis for this case, we observe that for the given number of this form, the initial code is calling several methods that is taking $2.\log_{10} n$ times.But if we consider the best case analysis, we observe that the out of three given while loops,The outer loop for the precision will execute only once and the time taken by the inner loops will be constant. So we can say $t_\omega \propto \log_{10} n + k$,Where k is some proper constant.So the best case complexity will be of the order $\Omega(\log n)$.

Average Case Analysis : The average case complexity will be of the order of $\Theta(\log n)$

Worst Case Analysis : The Worst Case Complexity will be of the Order $O(\log n)$

4 Experimental results

x	time (in ns)
5	34130
10	34140
20	61020
11	119673
13	116053
37	219022

For the given number if any number is of the form of power of 2 or 5 then the reciprocal of the number will be found as in the best case scenario . For a number which is prime or generates non - repeating , non - terminating decimal numbers then iteration inside loop will be iterated for the given value of desired precision upto which accuracy is desired . Here as we can see , the given number if it is prime or any such number which produces non-termination non-repeating decimal points or repeating decimals then the given number will produce peaks in graph

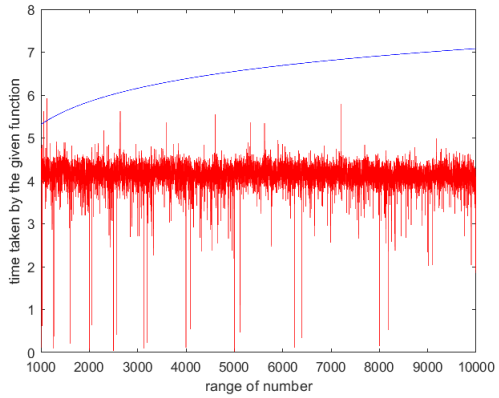
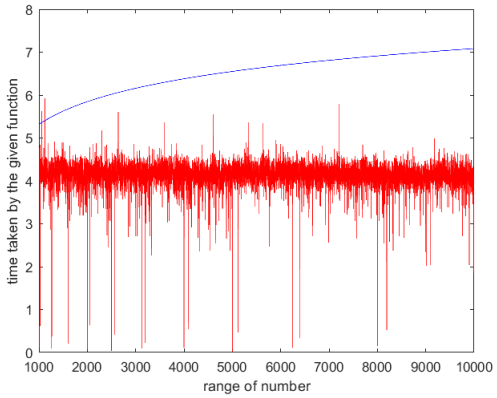


Figure 1: Figure: Graph showing the experimental analysis for integers vs the asymptotic analysis



.

Figure: Graph showing the experimental analysis for any real number vs asymptotic analysis.
This graph shows particularly chosen such numbers and their analysis of time taken by the given functions.

5 Discussion and Future Work

Here we used string manipulation to generate reciprocal of any real number and bit manipulation to generate reciprocal of any integer . Both the method uses o($logn$) to solve the given problem . Another method could be to convert division problem into multiplication problem .One such method is to use Newton-Raphson method to find the multiplicative inverse without using division . Here let,
$f(x) = 1/x - a$, $f'(x) = -1/x^2$
According to newton raphson method
$x_{n+1} = x_n - f(x_n)/f'(x_n)$ which yields ,
$x_{n+1} = x_n(2 - ax_n)$
Now , instead of division , multiplication of numbers should be performed . Another method could be to use binary exponentiation method which also changes the algorithm from division to multiplication problem .

6 Conclusion

The given problem was solved using bit manipulation and iterative subtraction to perform division without using division operator to generate multiplicative inverse of the given number . Other method could be to solve using Newton Raphson method or through binary exponentiation method through multiplication .

7 References

1. IDAA432C (Design and Analysis of Algorithm) class lecture
2. Multiplicative inverse
3. Newton Raphson method
4. Introduction to Algorithms by Cormen