Automating Polynomial Division

Mohd Mohtashim Nawaz* , Harsh Aryan†, Mayank Taksande‡ and Sneha Mishra§
Indian Institute of Information Technology, Allahabad
Allahabad-211012
Email: *itm2017005@iiita.ac.in, †itm2017003@iiita.ac.in, ‡ itm2017008@iiita.ac.in, § itm2017004@iiita.ac.in

***Abstract*—This Paper introduces an algorithm to automate the polynomial division of form p(x) by x-r. Here p(x) is any polynomial in x which acts as dividend and x-r is the divisor. The idea is to find an efficient algorithm in terms of time and space complexity.**

I. Introduction

To solve the problem we will use the classic approach of long division. In this approach, the polynomial in the numerator is simply divided by denominator using classic division approach. The user is prompted to input all coefficients of the polynomial p(x) i.e. the polynomial in the numerator. Since according to problem denominator is always of form x-r, so we ask the user to input r. The algorithm automates the division.

Long division is a standard division algorithm suitable for dividing multi-digit numbers and polynomials. It breaks down a division problem into a series of easier steps. The polynomial being divided is called dividend, the polynomial which divides is called divisor while the result obtained is termed as the quotient. The quantity that remains indivisible and separates out, at last, is called remainder.

For example,

For given polynomial division,

$$\frac{-13x^2+4x^3+2x-7}{x^2+3x-2}$$

The result comes out as:

Quotient = 4x-25
Remainder = 85x-57

Here we have tried to minimize the time and space complexity of the algorithm.

This algorithm is tested for different sample test cases for time complexity, space complexity and accuracy. In our case, we tend to focus on establishing a rule or a relationship between the time and input data.

This report further contains -

II. Algorithm Design and Analysis

III.Illustration

IV.Alternate Approach

V. Result

VI. Conclusion

VII. References

II. Algorithm Design

An efficient algorithm can only be created by considering every possibility. So we have to consider every possibility while designing the algorithm.

Steps for designing the algorithm include:

1) Inputs are taken from the user and stored in the array. Inputs include polynomial coefficients and value of r.

2) After that, the degree of the polynomial is calculated.

3) In the next step, the degree of the multiplying factor is calculated by taking the difference between the degree of the polynomial and the degree of the divisor.

4). Next, algorithm loops till the degree of the numerator is greater than or equal to the denominator.

5). In the loop, the denominator is multiplied by a suitable factor which is the difference of degrees of polynomial and divisor with a constant (if needed) multiplied.

6). New numerator and quotient are updated. At the last, remaining factor constitutes the remainder.

Algorithm to Subtract every coefficient of the polynomial

```
def polySubtract(p: double[], s :double[])
        len←p.length
        For i=0 to len do
                p[i] ←  p[i]-s[i]
end
```

Time Analysis:

Best case: 6*len + 2

= Ω(len)

Worst case: 6*len + 2

=O (len)

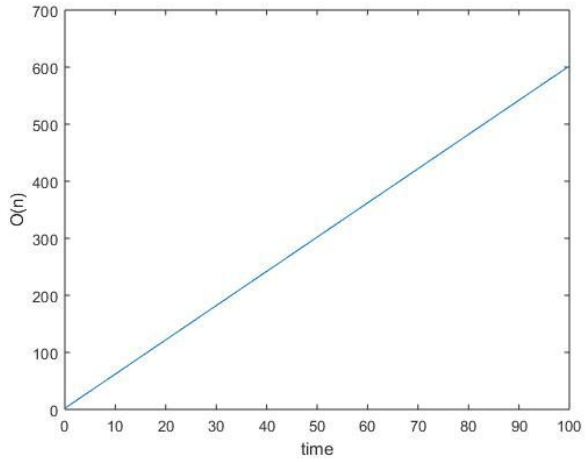

Fig 1.1: Graph depicting linear time complexity

Algorithm to Multiply every coefficient of polynomial

```
def polyMultiply(p: double[], m :double)
        len←p.length
        For i=0 to len do
                p[i] ←  p[i]*m
end
```

Time Analysis:

Best case: 5*len+2

= Ω(len)

Worst case: 5*len+2

= O(len)

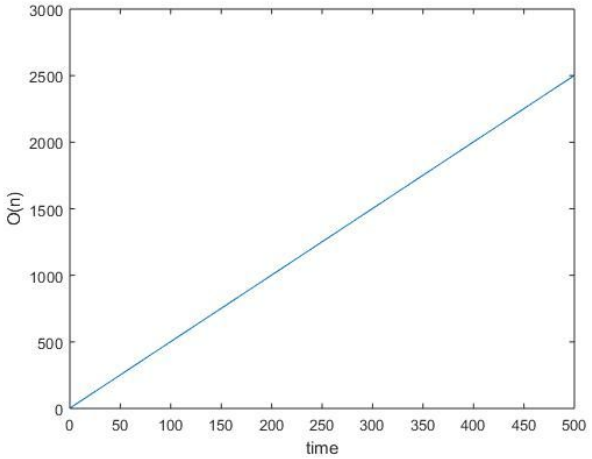

Fig 1.2: Graph depicting linear time complexity

Algorithm to Shift Polynomial to Right

```
def polyShiftRight(p: double[],places : int)

        IF places<=0 then return p;
        pd : int
        pd = polyDegree(p)
        IF pd+places >= p.length then throw exception

        d : double []
        d ← p
        For i=pd to 0 do
                d[i+places] ← d[i]
                d[i]=0.0
        return d

end
```

Time Analysis:

Best case: 1

= Ω(1)

Worst case: 9* leng + 10

= O(len)

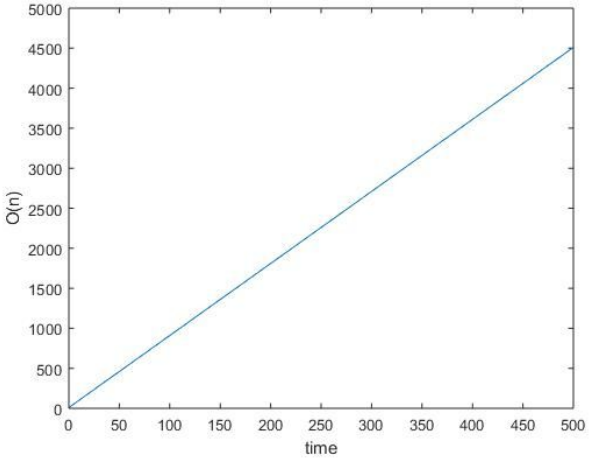

Fig 1.3: Graph depicting linear time complexity

Algorithm to find Degree of Polynomial

```
def polyDegree(p : double[])
        int:i
        leng← p.length -1
        For i=leng to 0 do
                IF p[i] !=0.0 then
                        return i;
```

return Integer.MIN_VALUE

end

Time Analysis:
Best case: 5 units
= Ω(1)

Worst case: 3 + 3*leng
= O(leng)

Main Algorithm

```
degree(P):
  return the index of the last non-zero element of P;
        if all elements are 0, return -∞

polynomial_long_division(N, D) returns (q, r):
  // N, D, q, r are vectors
  IF degree(D) < 0 then error
  q ← 0
  while degree(N) ≥ degree(D)
    d ← D shifted right by (degree(N) - degree(D))
    q(degree(N) - degree(D)) ← N(degree(N)) / d(degree(d))
    // by construction, degree(d) = degree(N)
    d ← d * q(degree(N) - degree(D))
    N ← N - d
  endwhile
  r ← N
  return (q, r)

end
```

Time Analysis:
Best case: Ω(1)

Worst case: (len-1)*(9*len + 10 + 5*len + 2)
= 14*len*len+ 12*len
= O(len*len)

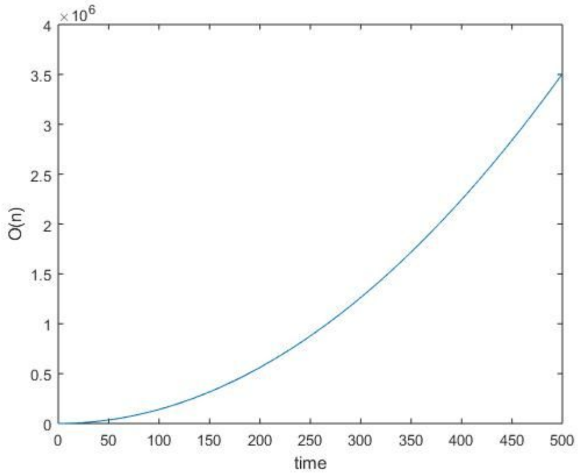

Fig 1.4: Graph depicting time complexity of overall code

III. *Illustration*

An example is taken to illustrate the algorithm used here to perform the division of the polynomial by the divisor of the form x-r.

Here polynomial is = x^3-2x^2-4
And the divisor is = x-3

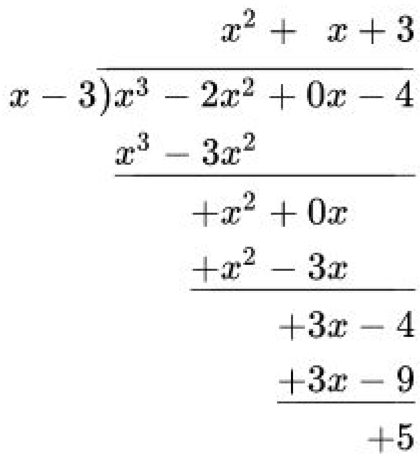

Fig 3.1: Illustration depicting division of polynomial

After performing the division according to the algorithm, the value of the quotient and remainder comes out as follows,

Quotient = x^2 + x + 3

Remainder = 5

The procedure here can be visualized to be directed from the algorithm.

IV. *Alternate Approach: Synthetic Division Method*

Synthetic division is a method of performing Euclidean division of polynomials, with less writing and fewer calculations than occur with polynomial long division. It is mostly taught for division by binomials of the for x - a but

the method generalizes to division by any monic polynomial, and to any polynomial. The advantages of synthetic division are that it allows one to calculate without writing variables, it uses few calculations, and it takes significantly less space on paper than long division. Also, the subtractions in long division are converted to additions by switching the signs at the very beginning, preventing sign errors.

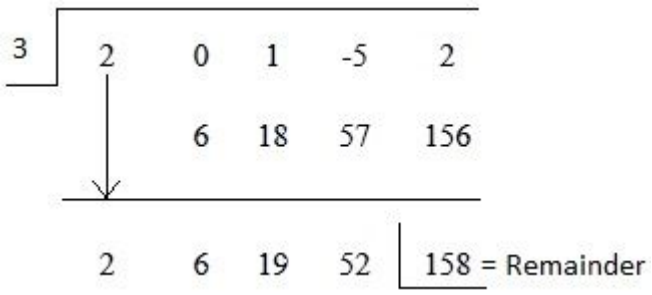

Fig 4.1: Illustration depicting synthetic division method

Synthetic division is a simple and more efficient but is a totally unorthodox method. Thus it is not selected to be the main approach.

V. RESULT

Overall Time Complexity

Best case: $\Omega(1)$

Worst case: (len-1)*(9*len + 10 + 5*len + 2)
= 14*len*len+ 12*len
= O(len*len)

Overall Space Complexity

Best case : $\Omega(4 * len)$

Worst case: O(4 * len)

VI. CONCLUSION

We have developed an algorithm to solve the given problem, that is the division of polynomials. We performed the same using long division method, which could take n^2 time complexity in the worst case, where n is the highest power of numerator.

There could have been some other approaches including the synthetic division approach which works well if the denominator is a linear equation, however the one used is simple and can be applied for all cases. Thus we have come up with the best possible approach of the given problem.

VII. REFERENCES

[1] "Long Division", en.wikipedia.org/wiki/Long_division [Online]
Referred : [23 Mar 2019]

[2] "Synthetic Division"
en.wikipedia.org/wiki/Synthetic_division
Referred :[23 Mar 2019]
[Online]

[3] Thomas H. Corman, Introduction to Algorithms, 3rd Edition 2003

[4] Ellis Horowitz,Sartaj Sahni,Sanguthevar Rajasekaran,Fundamentals of Computer Algorithms, 2nd Edition