THE CITY COLLEGE OF NEW YORK

Revolutionizing the Division Operation:

Analysis and Implementation of Common Division Algorithms

Georgios Ioannou

CSC 30100

Professor Erik K. Grimmelmann

14 December 2021

TABLE OF CONTENTS

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

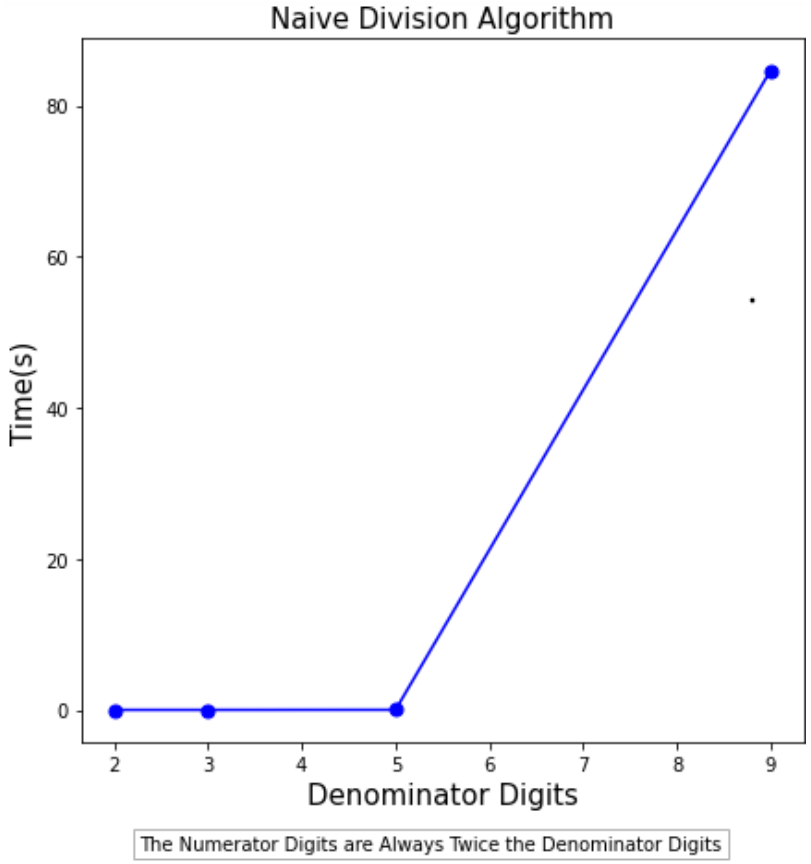
APPENDIX

```python
def naive_division_algorithm(n, d):
    r = n
    q = 0.0
    while r >= d:
        r -= d
        q += 1.0
    return q, r
```

Code A1. Naive Division Algorithm Python Code

	Numerator Digits	Denominator Digits	Time(s)
1	3	2	0.000000
2	5	3	0.000000
3	9	5	0.010991
4	17	9	84.627516

Table A1. Naive Division Algorithm Results Table



Graph A1. Naive Division Algorithm Results Graph

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

```
def first(quotient_multiply_by_divisor, numerator_list):
    size = len(str(quotient_multiply_by_divisor))
    subtraction_term_string = ""
    for i in range(size):
        subtraction_term_string += str(numerator_list[i])
    if int(subtraction_term_string) < quotient_multiply_by_divisor:
        size += 1
    subtraction_term_string = ""
    for i in range(size):
        subtraction_term = numerator_list.pop(0)
        subtraction_term_string += str(subtraction_term)
        remainder = int(subtraction_term_string) - quotient_multiply_by_divisor
    return remainder


def long_division_algorithm(n, d):
    numerator_list = [int(x) for x in str(n)]
    i = 0
    quotient = ""
    first_turn = True

    valid_dividend = numerator_list[i]
    valid_dividend_string = str(valid_dividend)

    while int(valid_dividend_string) < d:
        i += 1
        valid_dividend = numerator_list[i]
        valid_dividend_string += str(valid_dividend)
    while len(numerator_list) != 0:
        j = 0
        while (d * j) <= int(valid_dividend_string):
            j += 1
        quotient += str((j - 1))

        quotient_multiply_by_divisor = int(quotient) % 10 * d
        if first_turn:
            remainder = first(quotient_multiply_by_divisor, numerator_list)
            first_turn = False
        else:
            remainder = int(valid_dividend_string) - quotient_multiply_by_divisor
            numerator_list.pop(0)
        if len(numerator_list) == 0:
            q = int(quotient)
            r = n - (q * d)
            return q, r
        else:
            valid_dividend_string = str(remainder) + str(numerator_list[0])
    return int(quotient), n - (int(quotient) * d)
```

Code A2. Long Division Algorithm Python Code

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

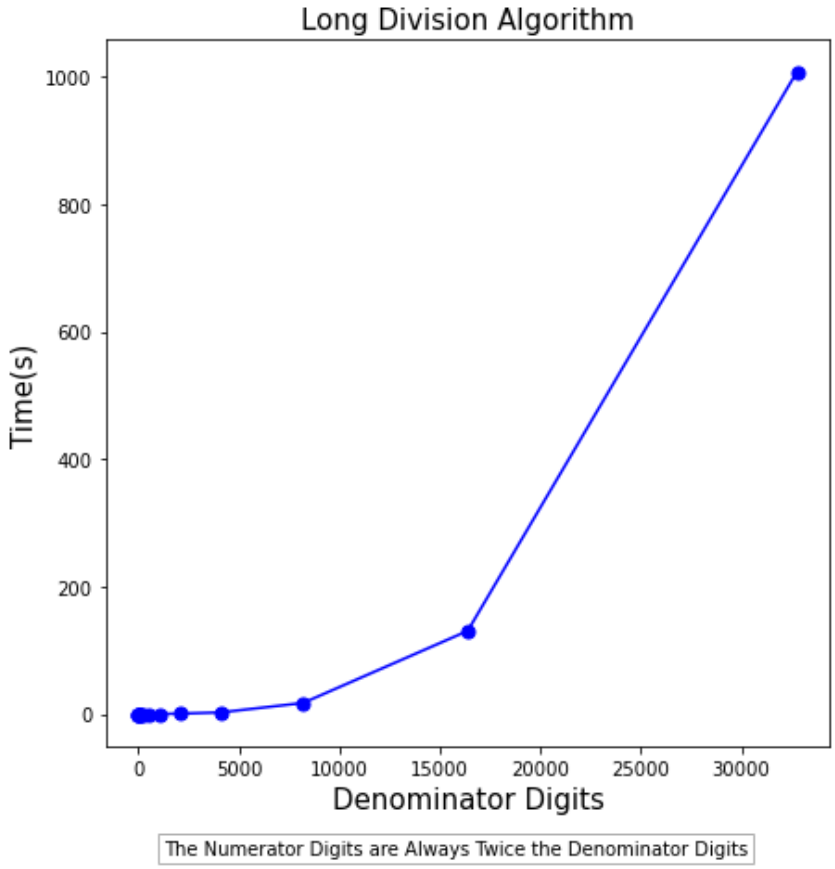
	Numerator Digits	Denominator Digits	Time(s)
1	3	2	0.000000
2	5	3	0.000000
3	9	5	0.001000
4	17	9	0.000000
5	33	17	0.000000
6	65	33	0.000999
7	129	65	0.001997
8	257	129	0.003997
9	513	257	0.007992
10	1025	513	0.022979
11	2049	1025	0.084921
12	4097	2049	0.405623
13	8193	4097	2.424751
14	16385	8193	17.200045
15	32769	16385	130.346088
16	65537	32769	1007.628282

Table A2. Long Division Algorithm Results Table

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021



Graph A2. Long Division Algorithm Results Graph

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

```
def complement(m):
    d = {'0': '1', '1': '0'}
    e = ''.join(d[x] for x in m)
    l = len(e)
    sum = bin(int(e, 2) + int('1', 2))
    sum = sum[2:]
    return sum.zfill(l)


def shift_left(s):
    s = s[1:]
    s += '0'
    return s


def add_zero_to_string(x):
    x = list(x)
    x[-1] = '0'
    return ''.join(x)


def add_one_to_string(x):
    x = list(x)
    x[-1] = '1'
    return ''.join(x)


def restoring_division_algorithm(n, d):
    q = bin(n)
    m = bin(d)

    q = q[2:]
    m = m[2:]

    if len(m) > len(q):
        q = q.zfill(len(m))
    else:
        m = m.zfill(len(q))

    l = len(q)
    count = l
    mc = complement(m)

    a = '0'
    for i in range(0, l - 1):
        a = a + '0'

    while count > 0:
        s = a + q
        value = shift_left(s)
        a = value[0:l]
        q = value[l:]
        a = bin(int(a, 2) + int(mc, 2))
        if len(a[2:]) == l + 1:
            a = a[3:]
        else:
            a = a[2:]
        if a[0] == '1':
            a = bin(int(a, 2) + int(m, 2))
            if len(a[2:]) == l + 1:
                a = a[3:]
            else:
                a = a[2:]
            q = add_zero_to_string(q)
        else:
            q = add_one_to_string(q)
        count -= 1
    return q, a
```

Code A3. Restoring Division Algorithm Python Code

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

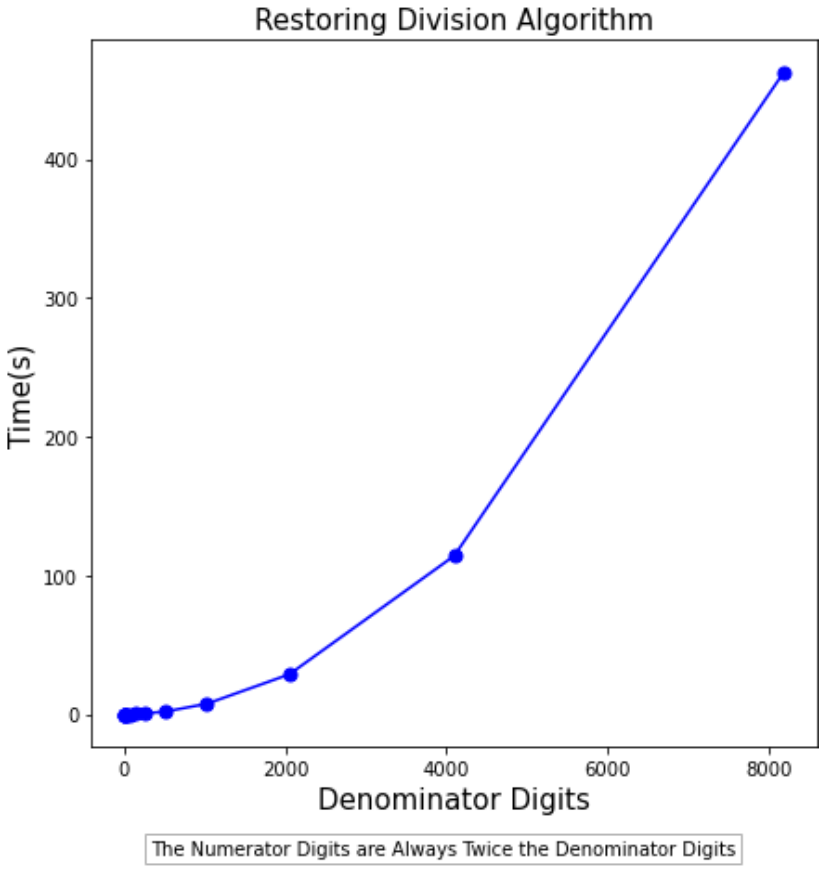
	Numerator Digits	Denominator Digits	Time(s)
1	3	2	0.001000
2	5	3	0.000000
3	9	5	0.000998
4	17	9	0.000999
5	33	17	0.003997
6	65	33	0.011989
7	129	65	0.036967
8	257	129	0.133874
9	513	257	0.483552
10	1025	513	1.853281
11	2049	1025	7.444095
12	4097	2049	28.663396
13	8193	4097	114.108163
14	16385	8193	462.880590

Table A3. Restoring Division Algorithm Results Table

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021



Graph A3. Restoring Division Algorithm Results Graph

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

```
def add(a, m):
    carry = 0
    sum = ""

    for i in range(len(a) - 1, -1, -1):
        tmp = int(a[i]) + int(m[i]) + carry
        if tmp > 1:
            sum += str(tmp % 2)
            carry = 1
        else:
            sum += str(tmp)
            carry = 0
    return sum[::-1]


def complement_non_restoring_division(d):
    return complement(d)


def non_restoring_division_algorithm(n, d, accumulator):
    q = n
    count = len(d)
    complement_m = complement_non_restoring_division(d)
    flag = "successful"
    while count:
        accumulator = accumulator[1:] + q[0]
        if flag == "successful":
            accumulator = add(accumulator, complement_m)
        else:
            accumulator = add(accumulator, d)
        if accumulator[0] == '1':
            q = q[1:] + '0'
            flag = "unsuccessful"
        else:
            q = q[1:] + '1'
            flag = "successful"
        count -= 1
    return q, accumulator
```

Code A4.Non- Restoring Division Algorithm Python Code

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

	Numerator Digits	Denominator Digits	Time(s)
1	3	2	86.672612
2	5	3	83.805259
3	9	5	83.946112
4	17	9	84.373748
5	33	17	84.718411
6	65	33	84.425683
7	129	65	84.488609
8	257	129	83.954136
9	513	257	93.925871
10	1025	513	87.674854

Table A4.1. Non-Restoring Division Algorithm Results Table 1

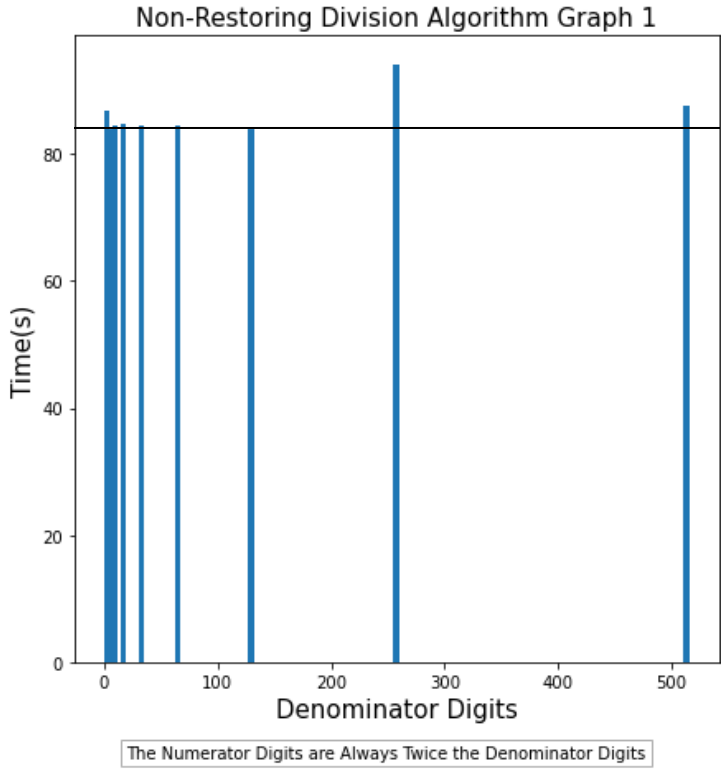
	Numerator Digits	Denominator Digits	Time
1	3	2	370.069701
2	5	3	368.594196
3	9	5	376.286671
4	17	9	370.354443
5	33	17	368.197445
6	65	33	375.336798
7	129	65	380.020501
8	257	129	366.533988
9	513	257	371.276589
10	1025	513	370.073704
11	2049	1025	373.608426

Table A4.2. Non - Restoring Division Algorithm Results Table 2

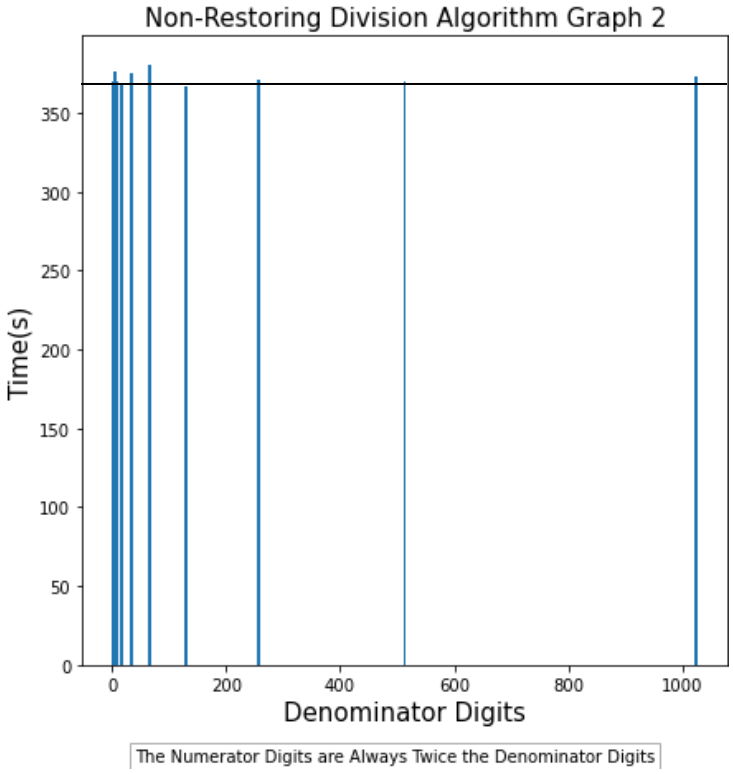
Professor Erik K. Grimmelmann

CSC 30100

14 December 2021



Graph A4. Non-Restoring Division Algorithm Results Graph 1



Graph A4.2. Non-Restoring Division Algorithm Results Graph 2

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

```
def size(x):
    if isinstance(x, int):
        return int(log(x, 2))
    return x.numdigits(2)


def newton_raphson_division_algorithm(n, d):
    precision = 10**300
    size_n = size(n)
    size_d = size(d)
    size_r = size_n - size_d
    if min(size_n, size_d, size_r) < (2 * precision):
        return n // d
    r = (1 << (2 * precision)) // (d >> (size_d - precision))
    last_precision = precision
    for prec in range(precision, size_r + 1):
        a = r << (prec - last_precision + 1)
        b = (r**2 * (d >> size_d - prec)) >> (2 * last_precision)
        r = a - b
        last_precision = prec
    return ((n >> size_d) * r) >> size_r
```

Code A5. Newton Raphson Division Algorithm Python Code

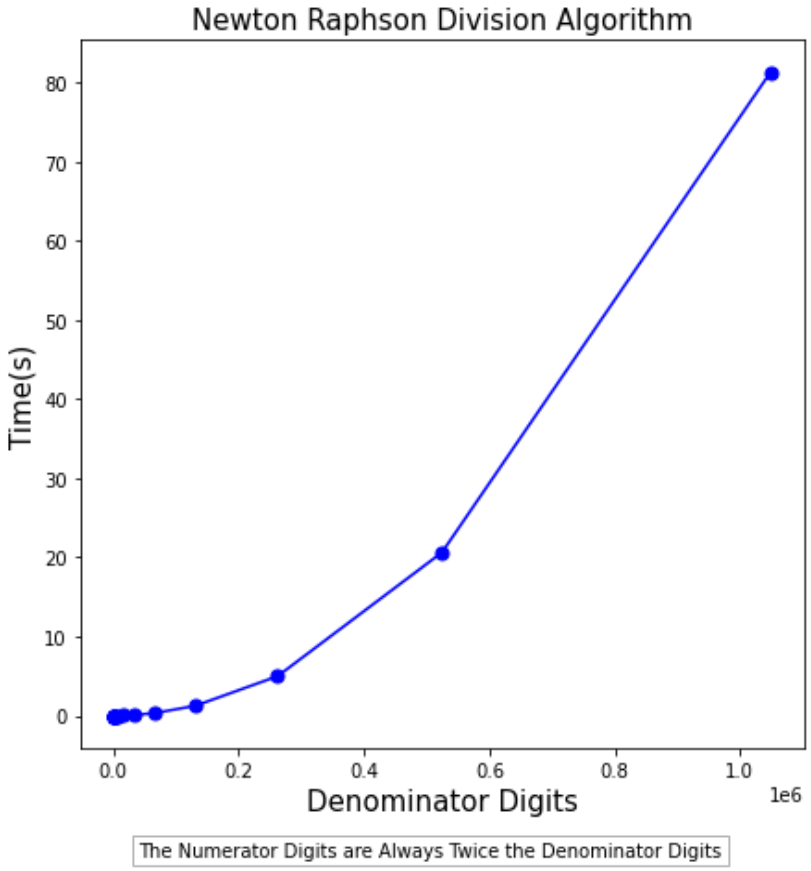
	Numerator Digits	Denominator Digits	Time(s)
1	3	2	0.000999
2	5	3	0.000000
3	9	5	0.000000
4	17	9	0.000000
5	33	17	0.000000
6	65	33	0.000000
7	129	65	0.000000
8	257	129	0.000000
9	513	257	0.000000
10	1025	513	0.000000
11	2049	1025	0.000000
12	4097	2049	0.000999
13	8193	4097	0.001998
14	16385	8193	0.004995
15	32769	16385	0.019983
16	65537	32769	0.077929
17	131073	65537	0.310712
18	262145	131073	1.243822
19	524289	262145	4.985398
20	1048577	524289	20.624876
21	2097153	1048577	81.322561

Table A5. Newton Raphson Division Algorithm Results Table

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021



Graph A5. Newton Raphson Division Algorithm Results Graph

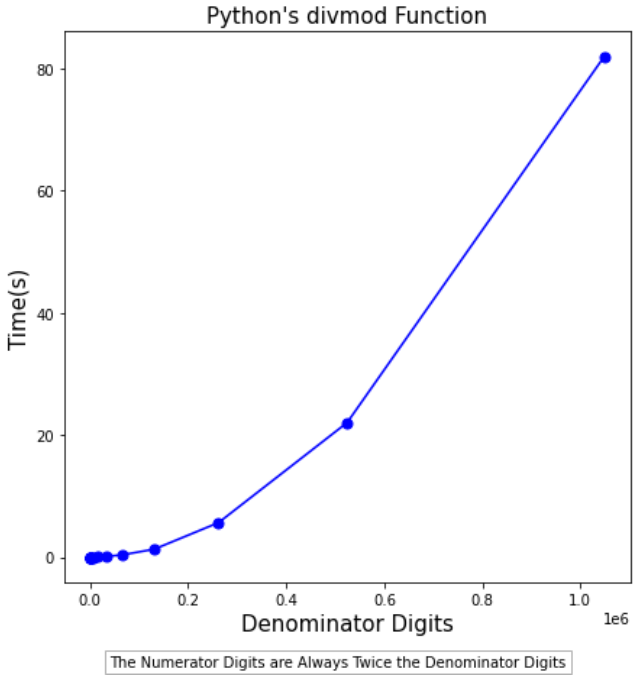
Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

	Numerator Digits	Denominator Digits	Time(s)
1	3	2	0.000000
2	5	3	0.000000
3	9	5	0.000000
4	17	9	0.000000
5	33	17	0.000000
6	65	33	0.000000
7	129	65	0.000000
8	257	129	0.000000
9	513	257	0.000000
10	1025	513	0.000000
11	2049	1025	0.000000
12	4097	2049	0.000000
13	8193	4097	0.001996
14	16385	8193	0.005996
15	32769	16385	0.024977
16	65537	32769	0.104904
17	131073	65537	0.361669
18	262145	131073	1.296821
19	524289	262145	5.649780
20	1048577	524289	22.032585
21	2097153	1048577	82.013895

Table A6. Python's divmod Function Results Table



Graph A6. Python's divmod Function Results Graph

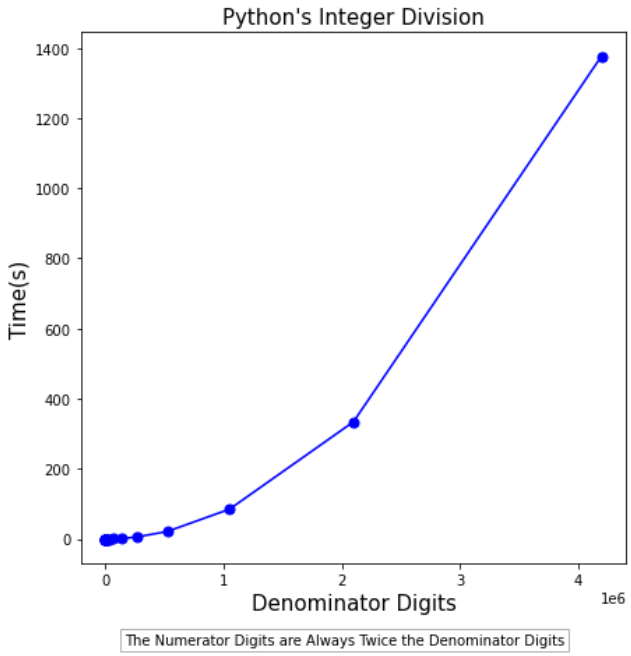
Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

	Numerator Digits	Denominator Digits	Time(s)
1	3	2	0.000000
2	5	3	0.000000
3	9	5	0.000000
4	17	9	0.000000
5	33	17	0.000000
6	65	33	0.000000
7	129	65	0.000000
8	257	129	0.000000
9	513	257	0.000000
10	1025	513	0.000000
11	2049	1025	0.000000
12	4097	2049	0.000000
13	8193	4097	0.002991
14	16385	8193	0.007985
15	32769	16385	0.025974
16	65537	32769	0.079931
17	131073	65537	0.339664
18	262145	131073	1.276824
19	524289	262145	5.358030
20	1048577	524289	21.516048
21	2097153	1048577	84.886254
22	4194305	2097153	334.145011
23	8388609	4194305	1376.462213

Table A7. Python's Integer Division Results Table



Graph A7. Python's Integer Division Results Graph

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

```python
def degree_of_function(polynomial):
    while polynomial[0] == 0:
        polynomial.pop(0)
    return len(polynomial) - 1


def polynomial_long_division_algorithm(dividend, divisor):
    dn = degree_of_function(dividend)
    dd = degree_of_function(divisor)

    q = []
    r = dividend
    dr = dn
    i = 0
    l = 0
    while (len(r) != 0) and (dr >= dd):
        leading_term_division = r[i] / divisor[0]
        q.append(leading_term_division)

        tmp = []
        for j in range(len(divisor)):
            tmp.append(leading_term_division * divisor[j])

        for k in range(len(tmp)):
            r[k + l] = r[k + l] - tmp[k]
        dr -= 1
        i += 1
        l += 1
    return q, r
```

Code A8. Polynomial Long Division Algorithm Python Code

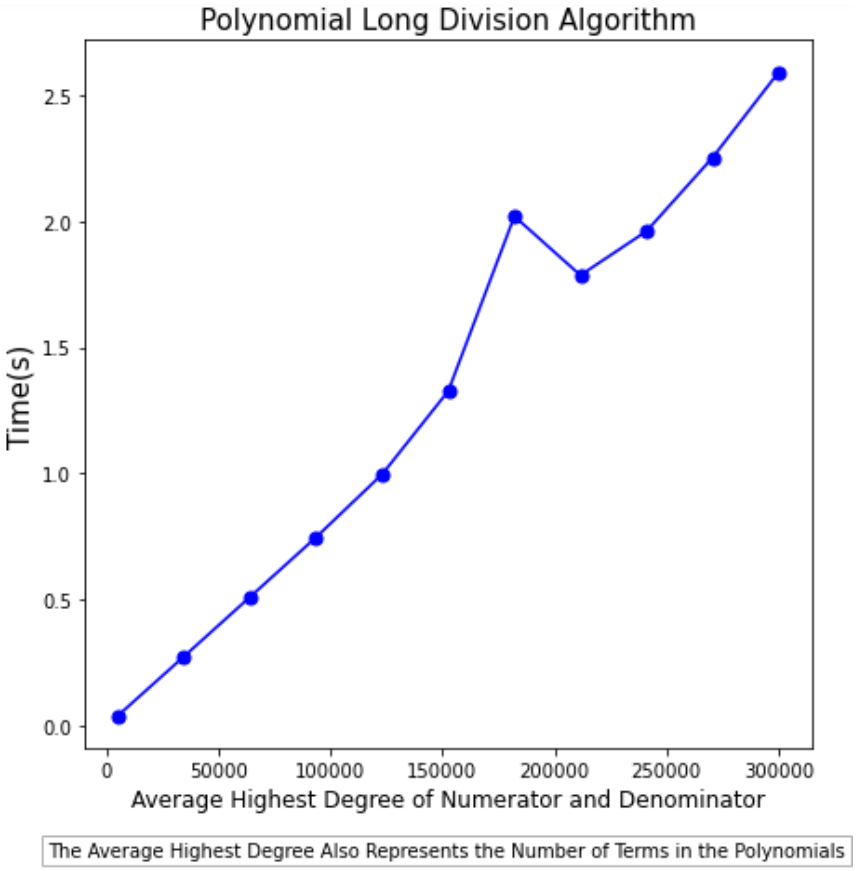
	Numerator Highest Degree	Denominator Highest Degree	Time(s)
1	5002	5000	0.039962
2	34502	34500	0.274746
3	64002	64000	0.510526
4	93502	93500	0.747307
5	123002	123000	0.997053
6	152502	152500	1.326780
7	182002	182000	2.019138
8	211502	211500	1.785330
9	241002	241000	1.960184
10	270502	270500	2.251888
11	300002	300000	2.590598

Table A8. Polynomial Long Division Algorithm Results Table

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021



Graph A8. Polynomial Long Division Algorithm Results Graph

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021

```python
def polynomial_extended_synthetic_division_algorithm(dividend, divisor):
    normalizer = divisor[0]
    for i in range(len(dividend) - len(divisor) + 1):
        dividend[i] /= normalizer
        coefficient = dividend[i]
        if coefficient != 0:
            for j in range(1, len(divisor)):
                dividend[i + j] += -divisor[j] * coefficient
    separator = 1 - len(divisor)
    return dividend[:separator], dividend[separator:]
```

Code A9. Polynomial Extended Synthetic Division Algorithm Python Code

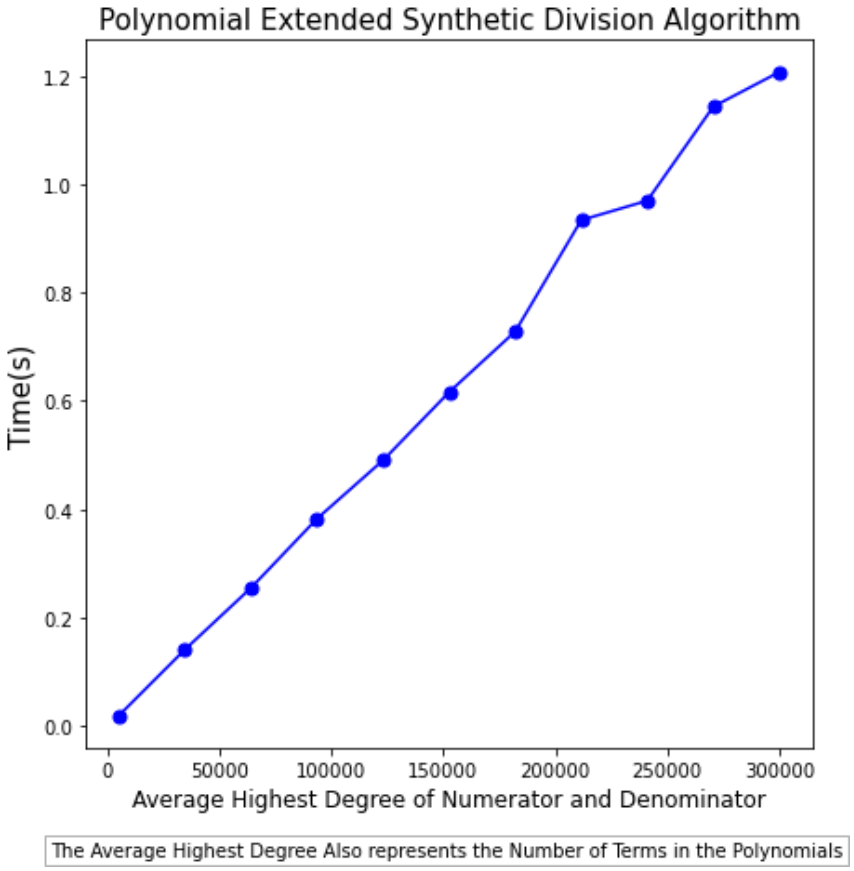
	Numerator Highest Degree	Denominator Highest Degree	Time(s)
1	5002	5000	0.018982
2	34502	34500	0.140893
3	64002	64000	0.254762
4	93502	93500	0.382647
5	123002	123000	0.490546
6	152502	152500	0.616430
7	182002	182000	0.728325
8	211502	211500	0.934136
9	241002	241000	0.970101
10	270502	270500	1.143923
11	300002	300000	1.207880

Table A9. Polynomial Extended Synthetic Division Algorithm Results Table

Professor Erik K. Grimmelmann

CSC 30100

14 December 2021



Graph A9. Polynomial Extended Synthetic Division Algorithm Results Graph