Group Seven

Number Theory Final Presentation
Spring 2020

Carlos Galvan
B.S. Computer Science

Jackeline Escalante
B.S. Computer Science

September 5, 2020

Introduction

Introduction

This project is meant to show how addition and multiplication interact to generate positive integers. For a positive integer n is to be the smallest number of ones required to write the number n using only addition, multiplication, and grouping symbols. Our task as a group was to learn as much about the sequence of numbers C_n

Background, Definitions, and Motivation

Outline

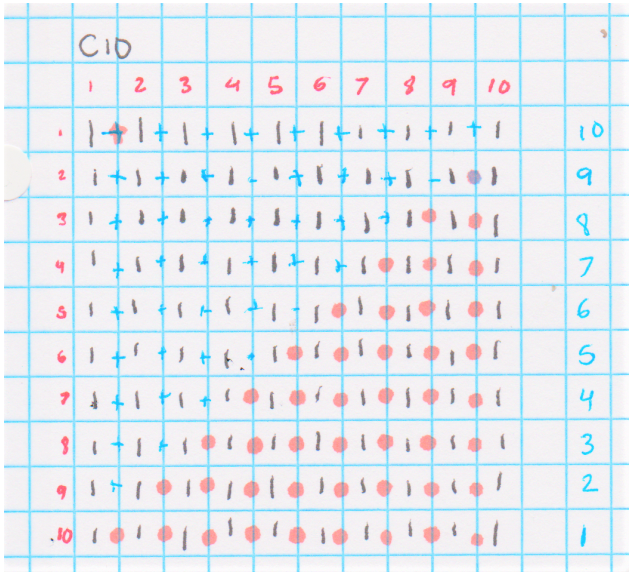
Step 1: We will begin talking about our initial findings of n amount of ones being generated with the minimal information given in project information.

Step 2: Steps about finding our basic formula needed to find the lowest amounts of ones to generate n. And trying to come up an alternative way.

Step 3: Generating code and scatter plot with our basic formula.

Week 1

To start our project, we wanted to see how many values, n amounts of ones can be generated with addition and/or multiplication



Week 1

Conclusion: The sum of n amount of ones, results in n; The product of n amount of ones, will result in one.

This approach did not include grouping, which can drastically alter the equation's value.

Week 2

We discovered that we can drastically lower the amounts of ones needed to generate n by dividing n by 2

The sum of 10 ones is equal to...

$10 = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1$

The sum of 5 ones times the sum 2 ones

$10 = (1 + 1 + 1 + 1 + 1)\ \ (1 + 1)$

Week 2

This can only be applied to even numbers only since integers cannot be divided. To overcome this, we just subtract one from n to make it even.

The sum of 11 ones is equal to...

$11 = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1$

The sum of 5 ones times the sum 2 ones plus one

$11 = ((1 + 1 + 1 + 1 + 1)\ \ (1 + 1)) + 1$

Week 2

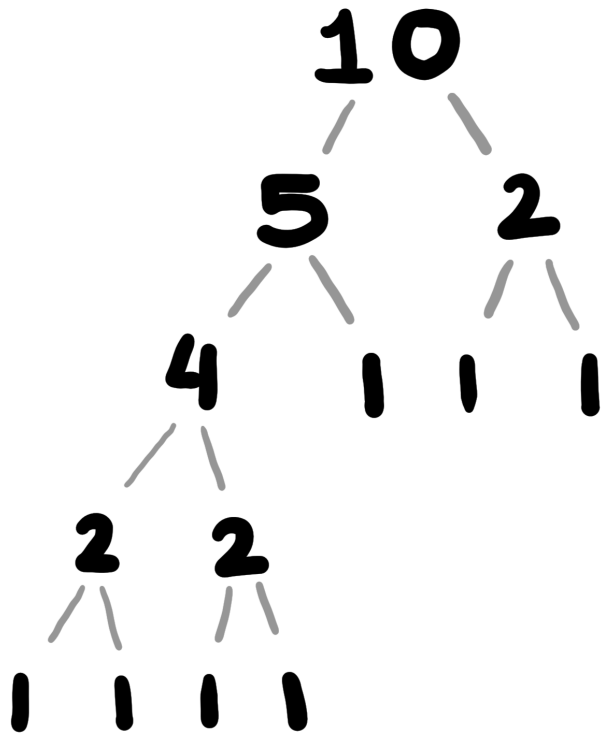
Conclusion: We found our basic formula needed to find the lowest amounts of ones to generate n

If n is even then **n / 2**

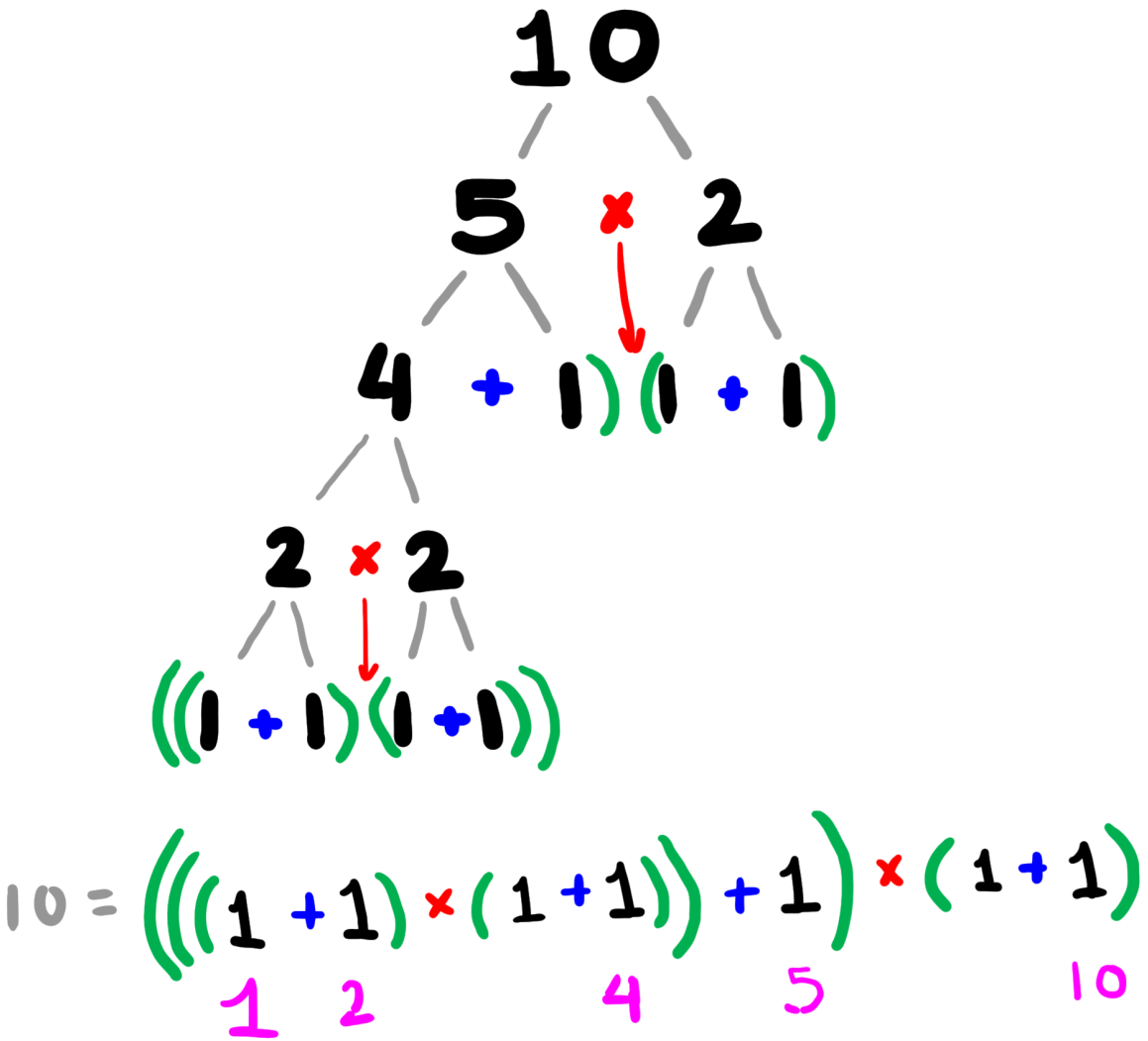
If n is odd then **(n – 1) / 2**

Week 3

We discovered, that we can apply the formula over and over. Using this strategy we created a binary tree (inspired by a factor tree).



Week 3



Week 3

```
def recursionPrinting(n, str):
    if n == 0:  # if no number, then 0
        str = "0"
        return str
    elif n == 1:  # once n has been broken down to 1
        str.append(n)
        return str
    elif (n % 2) == 0:  # is even
        str.append(n)
        n /= 2  # n/2
        return recursionPrinting(n,str)
    elif (n % 2) == 1:  # is odd
        str.append(n)
        n = (n - 1)  # (n - 1) + 2
        return recursionPrinting(n, str)
    else:
        print("Not a valid input")
```

```
>print(recursionPrinting(10,[]))

[10, 5.0, 4.0, 2.0, 1.0]
```

Week 3

```
def eqFormat(n):
    eq = ""
    for i in range(len(n)):
        if n[i] == 1: #starting point
            eq = "1"
        elif n[i] == 2: #the second step, requiered to
    keep things neat
            eq = "( " + eq + " + 1 )"
        elif (n[i] % 2) == 0: #if the number is even,
   then opposite of /2 is ( X 2 )
            eq = "( " + eq + " * ( 1 + 1 ) )"
        elif (n[i] % 2) == 1:# if the number is odd,
   the opposite of - 1 is + 1
            eq = "( " + eq + " + 1 )"
        else: #basic else statement
            print("something went wrong")
    return eq
```

```
> n = [1, 2, 4, 5, 10]
> print(eqFormat(n))

( ( ( ( 1 + 1 ) * ( 1 + 1 ) ) + 1 ) * ( 1 + 1 ) )
```

Week 3

10

```
Enter the number(enter 0 to quit): 10
C 10  <=  7

 10  =   ( ( ( 1 + 1 ) * ( 1 + 1 ) ) + 1 ) * ( 1 + 1 )
```

7

100

```
Enter the number(enter 0 to quit): 100
C 100  <=  14

 100  =   ( ( ( ( ( ( 1 + 1 ) + 1 ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) * ( 1 +
1 ) ) + 1 ) * ( 1 + 1 ) ) * ( 1 + 1 )
```

14

1,000

```
Enter the number(enter 0 to quit): 1000
C 1000  <=  23

 1000  =   ( ( ( ( ( ( ( ( ( ( 1 + 1 ) + 1 ) * ( 1 + 1 ) ) + 1 ) * (
 1 + 1 ) ) + 1 ) * ( 1 + 1 ) ) + 1 ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) + 1 ) * (
 1 + 1 ) ) * ( 1 + 1 ) ) * ( 1 + 1 )
```

23

5,000

```
Enter the number(enter 0 to quit): 5000
C 5000  <=  28

 5000  =   ( ( ( ( ( ( ( ( ( ( ( ( 1 + 1 ) * ( 1 + 1 ) ) * ( 1 + 1 )
 ) + 1 ) * ( 1 + 1 ) ) + 1 ) * ( 1 + 1 ) ) + 1 ) * ( 1 + 1 ) ) * ( 1 + 1 )
 ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) + 1 ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) * ( 1 + 1
 )
```

28

1,000,000

```
Enter the number(enter 0 to quit): 1000000
C 1000000  <=  44

 1000000  =   ( ( ( ( ( ( ( ( ( ( ( ( ( ( ( ( ( ( 1 + 1 ) + 1
) * ( 1 + 1 ) ) + 1 ) * ( 1 + 1 ) ) + 1 ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) + 1
) * ( 1 + 1 ) ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) +
1 ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) + 1 ) * ( 1 + 1 ) ) * ( 1 +
1 ) ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) * ( 1 + 1 ) ) * ( 1 + 1 )
```

44

Week 4

Heading in a different direction with what we've already proved in previous approach. We wanted to find a new approach finding least number of ones and see if it's more efficient. We did so by forming factor families breaking down to ones.

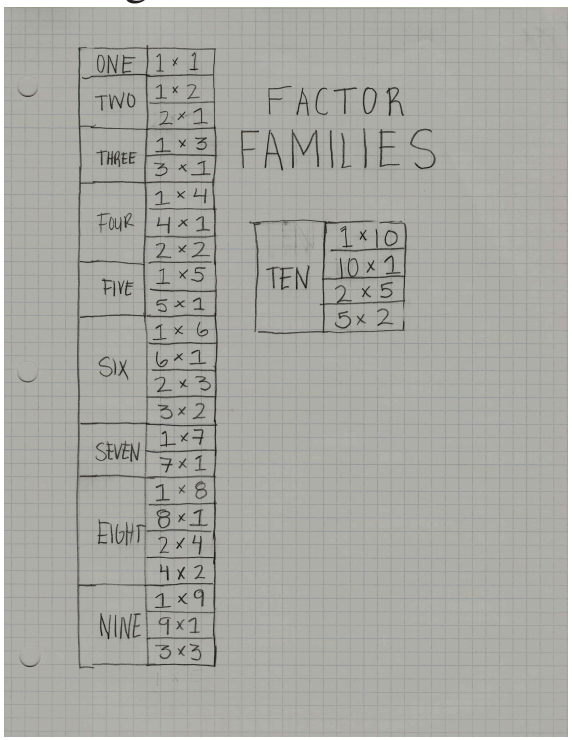


Figure: Factor Families

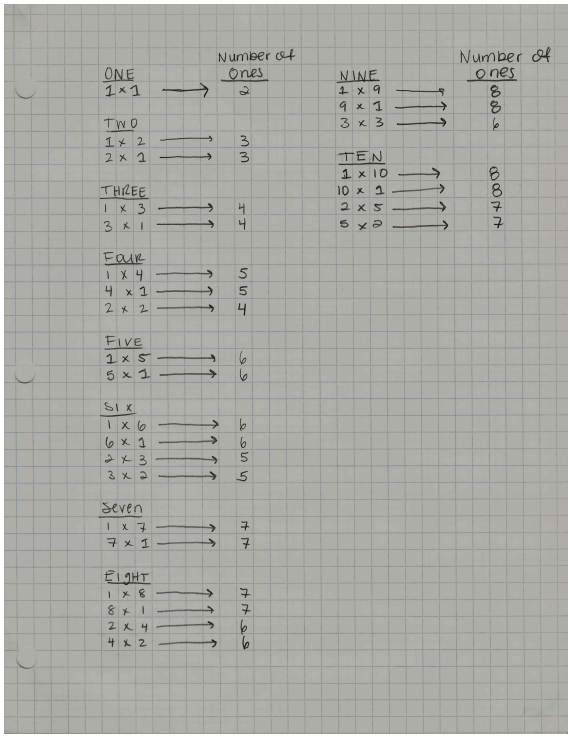


Figure: Number of Ones

Week4

We then came up with eleven ways in which we add numbers that would equal ten.

- 0 + 10
- 10 + 0
- 1 + 9
- 9 + 1
- 2 + 8
- 8 + 2
- 3 + 7
- 7 + 3
- 4 + 6
- 6 + 4
- 5 + 5

Seeing from the amount of ones each equation generated, order at this point did not matter.

Week 4

We then created steps in order to find the least number of ones.

Step 1: Use table 1 to break down the equation, use factor families corresponding to the number with the least number of ones.

Step 2: Avoid choosing the factor family where the number multiplies itself and 1, from what we saw it just generated a lot more ones.(E.g. 1*2,1*3,1*4, etc.)

Step 3: When breaking down the equation and you get 2. Don't use the factor family of (1*2), because this generates 3 ones. Instead just do (1+1), because this generates 2 ones.

Step 4: If the number is odd, use its closest lower even number plus one. For example if you have...

- 3 = 2+1
- 5 = 4+1
- 7 = 6+1
- 9 = 8+1

Week 4

Conclusion: This method was not as efficient because it would take longer for larger n values.

Week 5

With the program created, we've used it to create a scatter plot. The main function recursion() is what generates m (least amount of 1s needed to generate n).

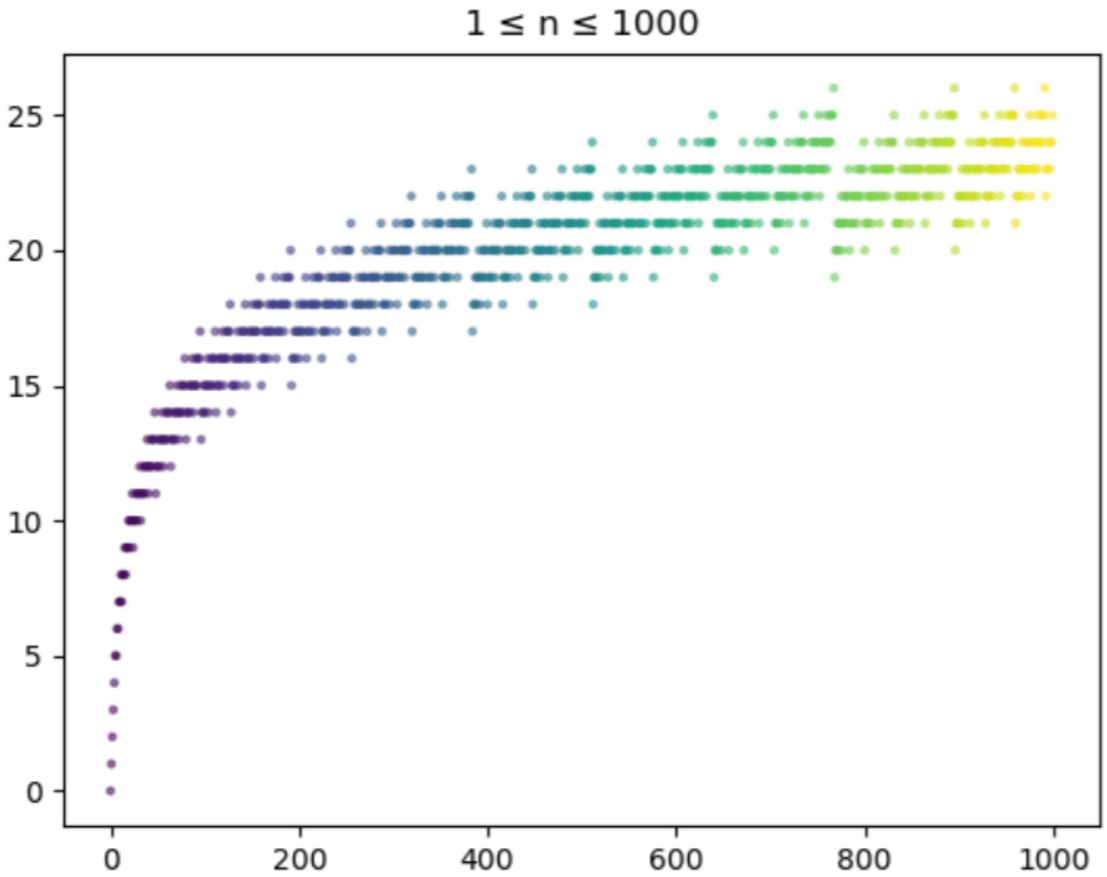
```
def recursion(n, c):
    if n == 0:  # if no number, then 0
        c += 0
        return c
    elif n == 1:  # once n has been broken down to 1
        if n > c:  # this only applies to 1 other wise
    C1 --> 0
            c += 1
        return c
    elif (n % 2) == 0:  # is even
        n /= 2  # n/2
        c += 2
        return recursion(n, c)
    elif (n % 2) == 1:  # is odd
        n = (n - 1) / 2  # (n - 1) + 2
        c += 3  # (-1) and /2 all result to the use of
    3 Ones
        return recursion(n, c)
    else:
        print("Not a valid input")
```

Week 5

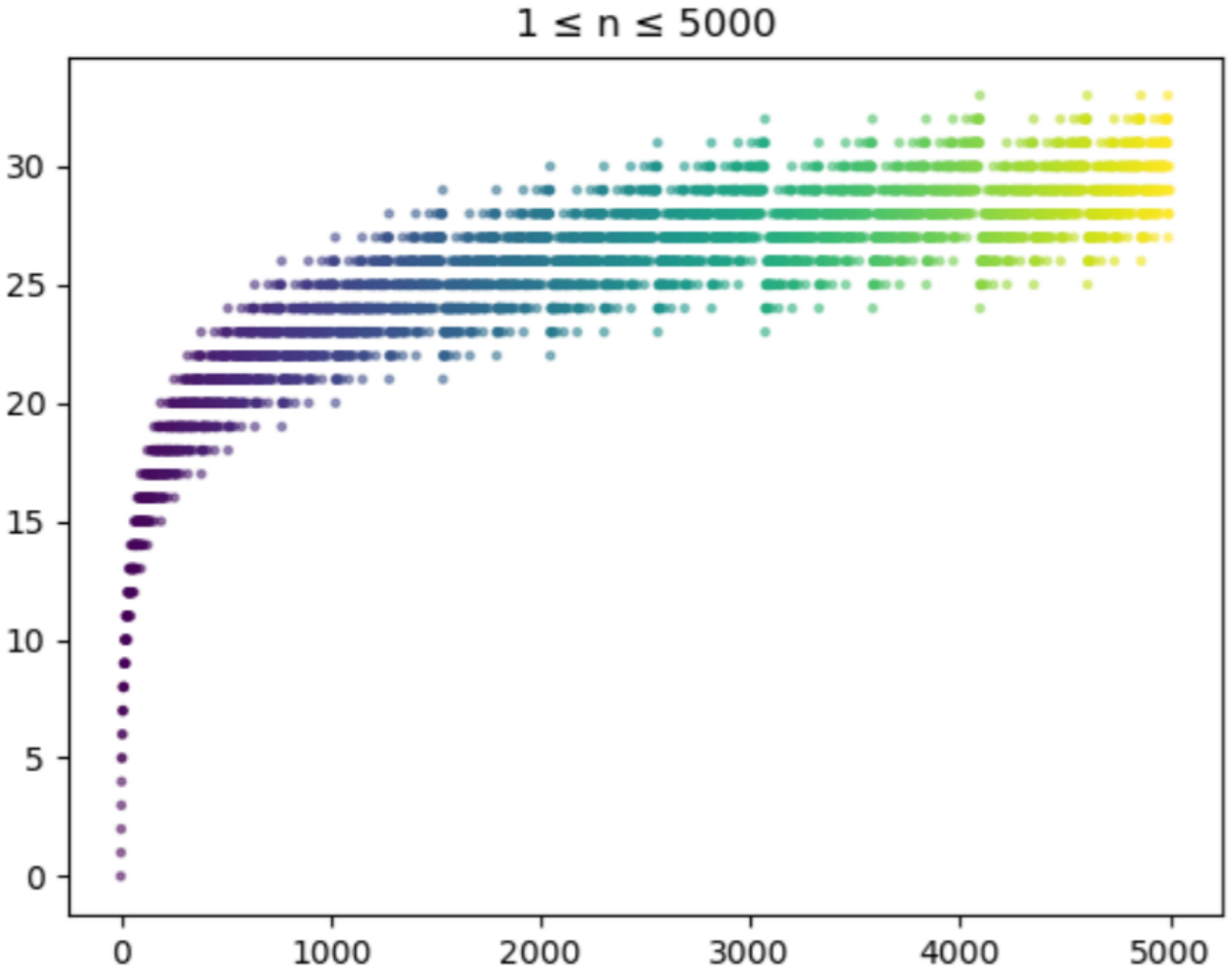
```
x = []
y = []
max = int(input("Enter Number: "))
N = max
colors = []
area = []
diction = {}

for i in range(max):
    x.append(i)
    y.append(recursion(i, 0))
    colors.append(i)
    area.append(i / 10)

title = "1      n      " + str(max)
plt.title(title)
plt.scatter(x, y, s=5, c=colors, alpha=0.5)
plt.show()
```

Week 5

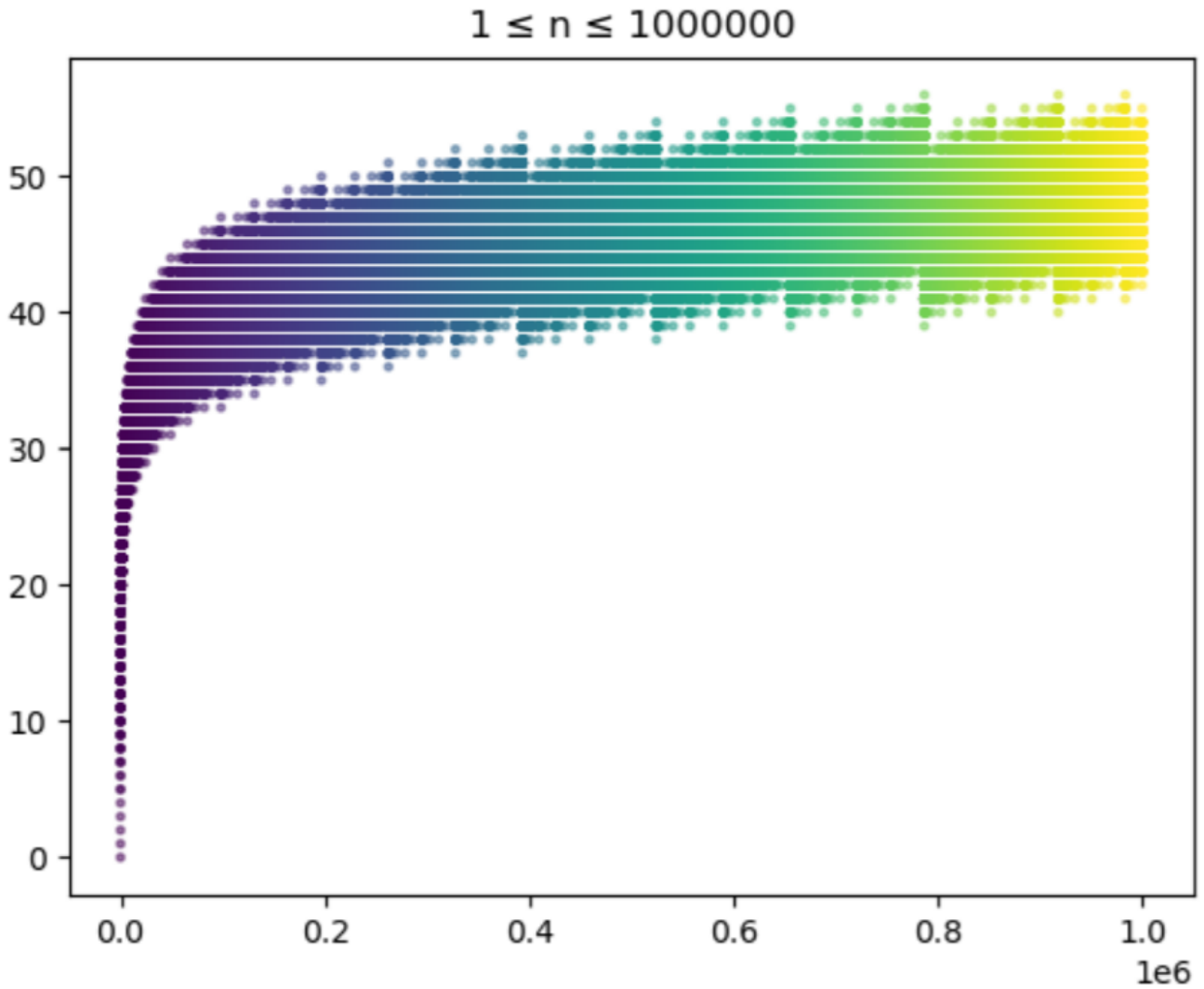
The Results



Week 5



Week 5



Summary

Summary

Week by week we found important information that led us to the end of our project, finding smallest number of ones required to write the number n using only addition, multiplication, and grouping symbols.

Week 1: The sum of n amount of ones, results in n; The product of n amount of ones, will result in one. This approach did not include grouping, which can drastically alter the equation's value.

Week 2: We found our basic formula needed to find the lowest amounts of ones to generate n.

Week 3: We discovered, that we can apply the formula over and over. Using this strategy we created a binary tree. As well as generating code using recursive function to do so.

Week 4: We found an alternative way to find the sum of n amounts of ones. But this method was not as efficient because it would take longer for larger n values.

Week 5: Used code to generate a scatter plot that would help us generate m(least amount of 1s needed to generate n).