Quinn Roemer

CISP - 430

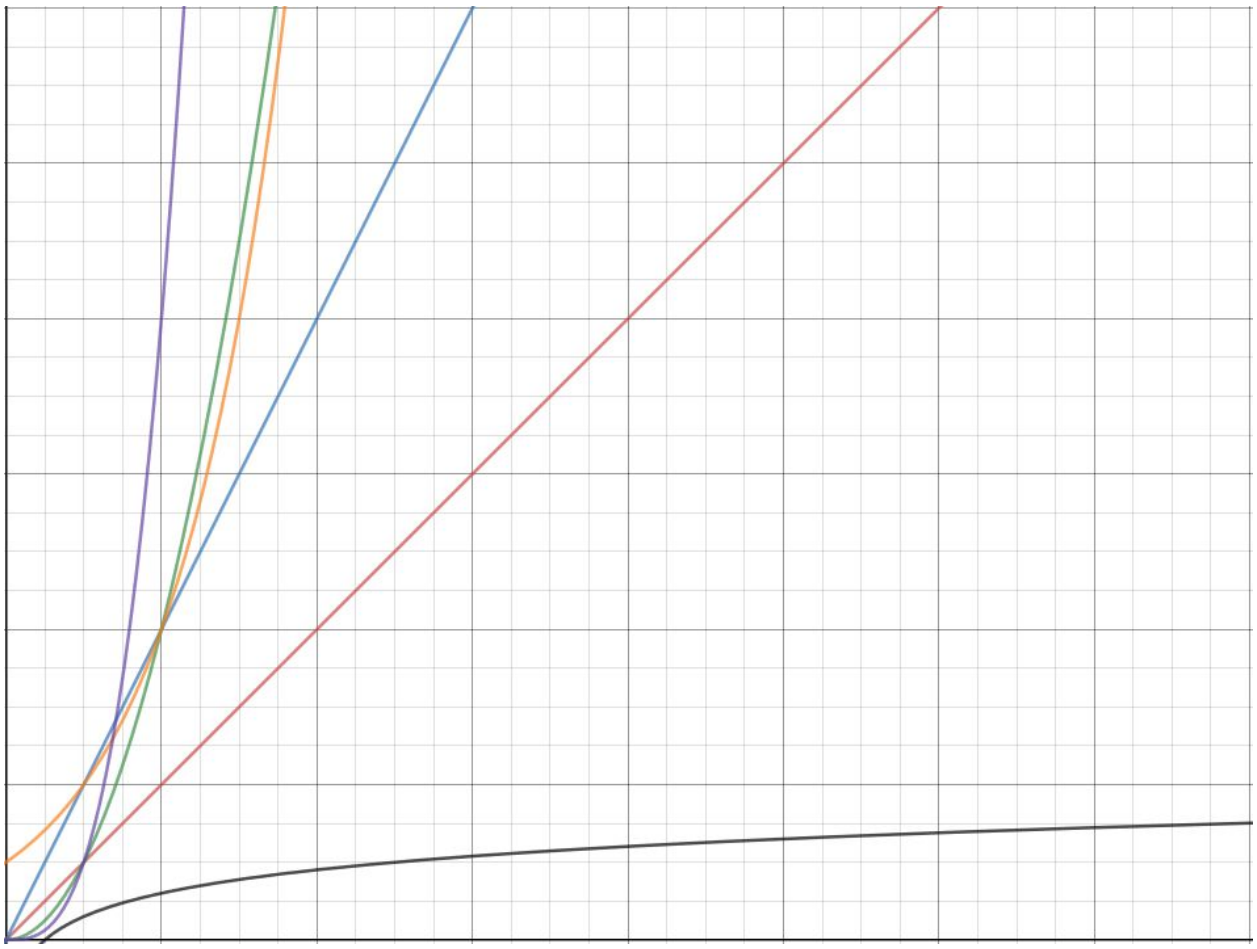
Assignment 1

1/31/2018

Part 0 - Review of Common Math Functions

Description:

The goal of this part of the lab was to graph some common math functions using a graphing utility. I used Desmos.com as my graphing system to get this result.



Red: Y = X
Blue: Y = 2X
Green: Y = X^2
Orange: Y = 2^X
Purple: Y = X^3
Black: Y = log2X

Rank the graphs by rate of growth. Fastest (non-initial) growth first:

1. Y = X^3
2. Y = X^2
3. Y = 2^X
4. Y = 2X
5. Y = X
6. Y = log2X

Match the shape with the closest common Big(0) curve and label them:

Function	Closest Big(O)
Y = X	O(N)
Y = 2X	O(NlogN)
Y = X^2	O(N^2)
Y = 2^X	O(2^N)
Y = X^3	O(N!)
Y = log2X	O(logN)

Part 1 - Empirical Analysis

Description:

In this part of the assignment I was supposed to use the provided source code and determine each algorithms efficiency for N number of times. This data would then be put into a table and also graphed.

Source Code - Foo1:

```
//Original code written by Professor Ross. Modified by Quinn Roemer.

#include <iostream>

using namespace std;

int foo1(int);

int main()
{
   int answer;

   cout << "Please enter desired iterations." << endl;
   cin >> answer;

   answer = foo1(answer);

   cout << "Counter: " << answer << endl;
}

int foo1(int n)
{
   int counter = 0;

   for (int i = 0; i < n; i++)
   {
            counter++;
   }

   return counter;
}
```

Output:

Foo1 Table:

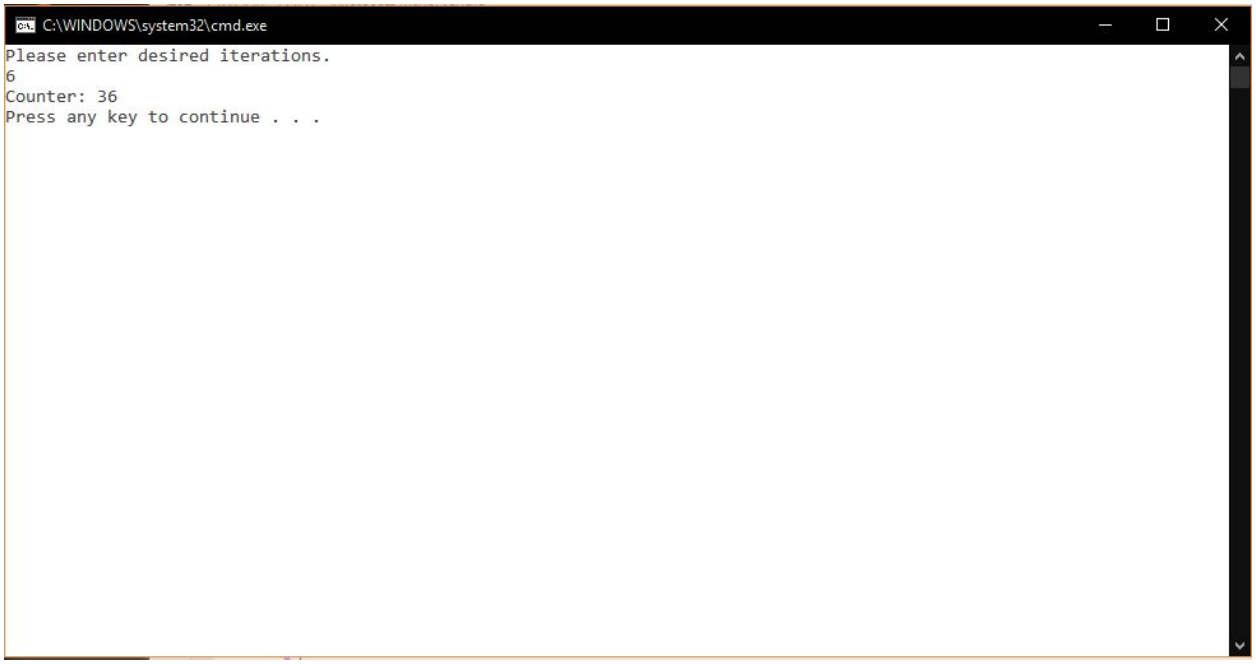
N	Return Value
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16

32	32
64	64

Source Code - Foo2:

```
//Original code written by Professor Ross. Modified by Quinn Roemer.

#include <iostream>

using namespace std;

int foo2(int);

int main()
{
   int answer;

   cout << "Please enter desired iterations." << endl;
   cin >> answer;

   answer = foo2(answer);

   cout << "Counter: " << answer << endl;
}

int foo2(int n)
{
   int counter = 0;

   for (int i = 0; i < n; i++)
   {
            for (int j = 0; j < n; j++)
            {
                     counter++;
            }
   }

   return counter;
}
```

Output:



Foo2 Table:

N	Return Value
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100
11	121
12	144
13	169
14	196
15	225
16	256

32	1024
64	4096

Source Code - Foo3:

```
//Original code written by Professor Ross. Modified by Quinn Roemer.

#include <iostream>

using namespace std;

int foo3(int);

int main()
{
   int answer = 0;
   int counter;

   while (answer <= 64)
   {
            counter = foo3(answer);

            cout << "Counter " << answer << ": " << counter << endl;

            if (answer < 16)
            {
                     answer++;
            }
            else
            {
                     answer = answer * 2;
            }
   }
}

int foo3(int n)
{
   int counter = 0;

   for (int i = n; i > 0; i = i/2)
   {
            counter++;
   }

   return counter;
```

Output:

Foo3 Table:

N	Return Value
0	0
1	1
2	2
3	2
4	3
5	3
6	3
7	3
8	4
9	4
10	4
11	4
12	4
13	4
14	4
15	4
16	5
32	6

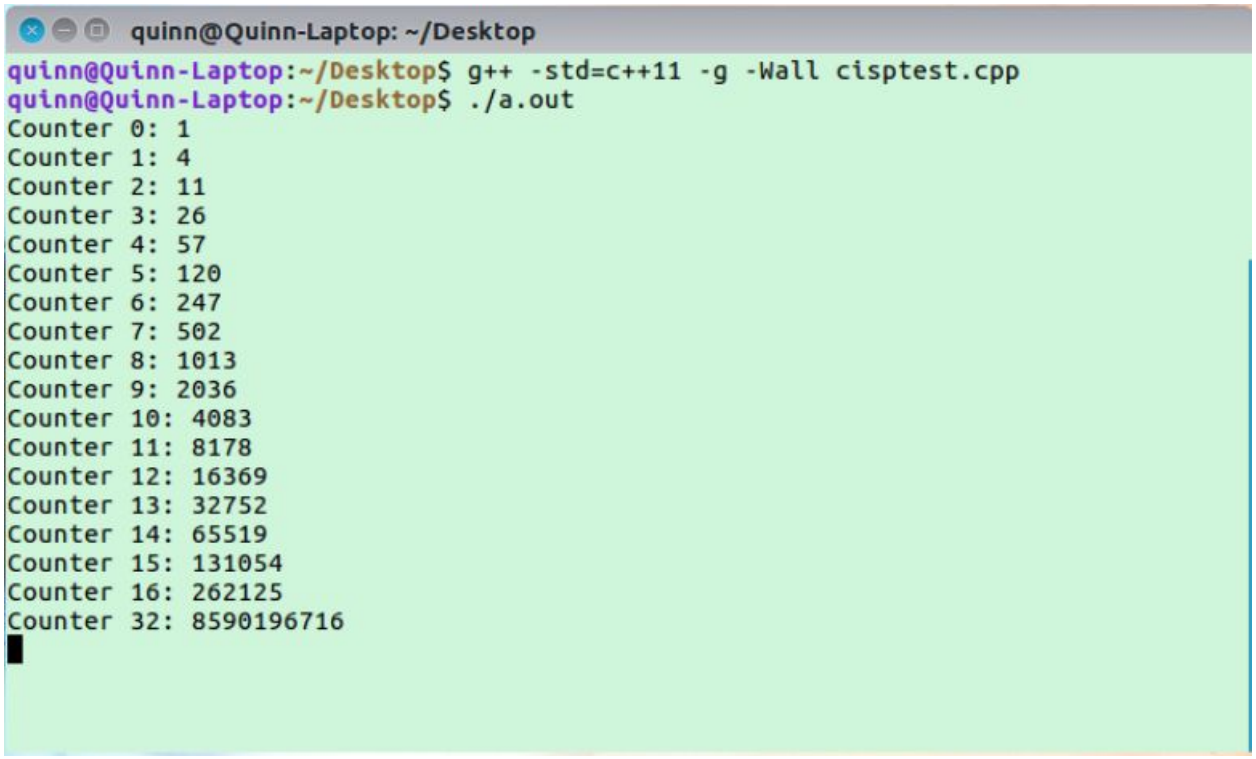
64	7

Source Code - Foo4:

```
//Original code written by Professor Ross. Modified by Quinn Roemer.

#include <iostream>

using namespace std;

long long int foo4(int);

int main()
{

   int answer = 0;
   long long int counter;

   while (answer <= 64)
   {

            counter = foo4(answer);

            cout << "Counter " << answer << ": " << counter << endl;

            if (answer < 16)
            {
                     answer++;
            }
            else
            {
                     answer = answer * 2;
            }
   }
}

long long int foo4(int n)
{
   static long long int counter = 0;
   counter++;

   if (n > 0)
   {
            foo4(n - 1);
            foo4(n - 1);
   }
```

```
    return counter;
}
```

Output:



Note: This code was ran on a Linux machine since my Linux machine can run code much faster than my Windows machine. I allowed my computer to work on this algorithm for 7 hours and it never produced a result for 64. Also, I changed the int variables to long long ints to allow for greater counter values to be stored.

Foo4 Table:

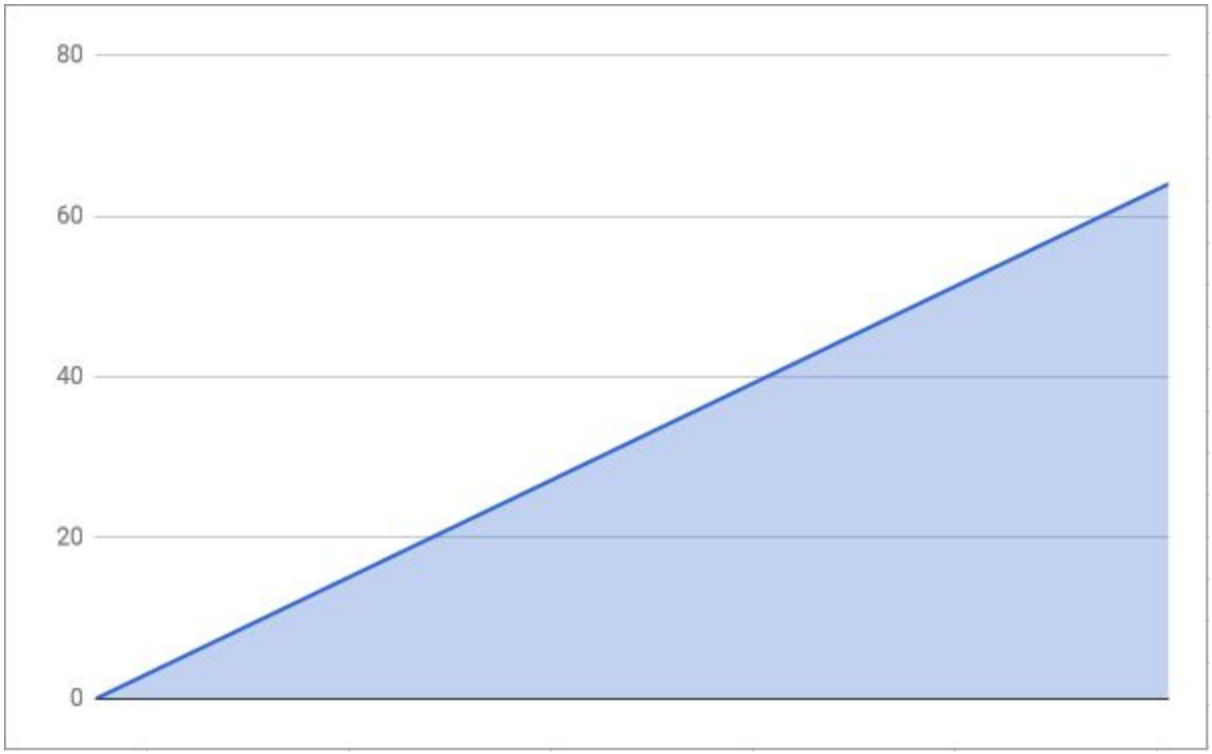
N	Return Value
0	1
1	4
2	11
3	26
4	57
5	120
6	247
7	502

8	1013
9	2036
10	4083
11	8178
12	16369
13	32752
14	65519
15	131054
16	262125
32	8590196716
64	Undefined

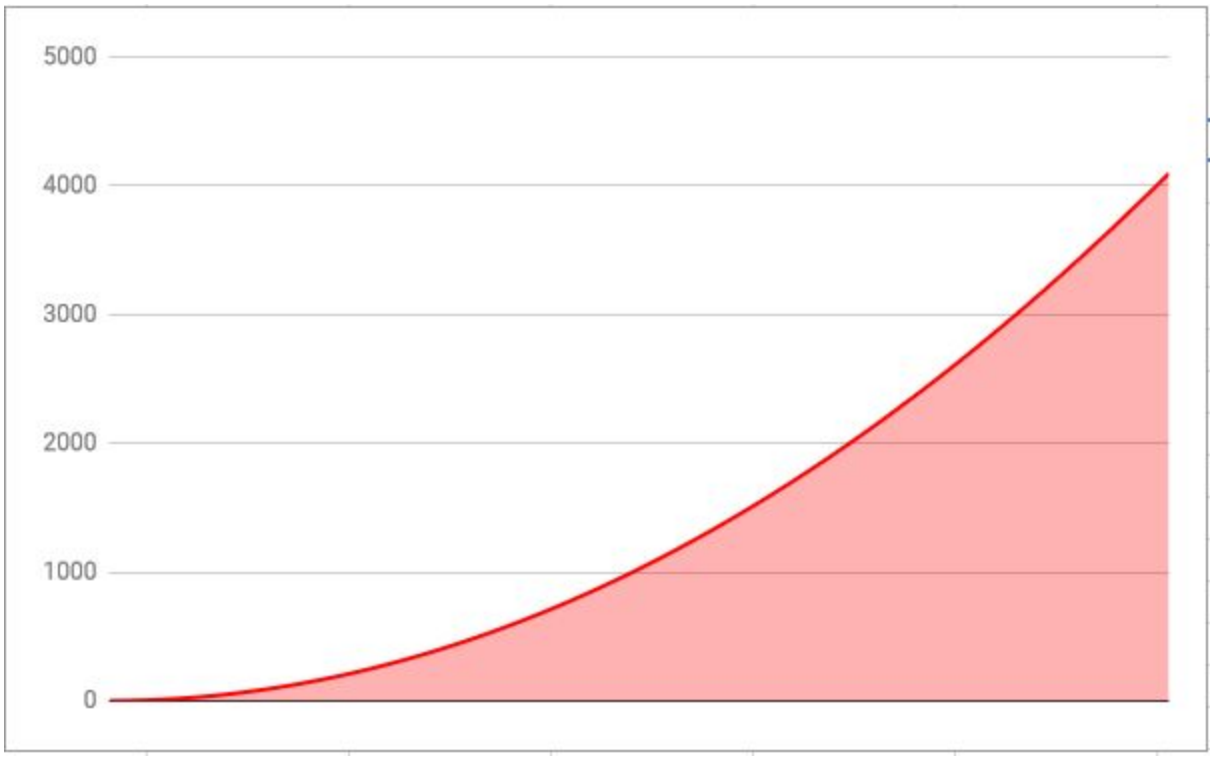
Graphs of all Algorithms:

Note: I did not graph the algorithms in the same graph since the last graph of Foo4 would dwarf every other graph making it hard to read. Thus I have included graphs for each individual algorithm.

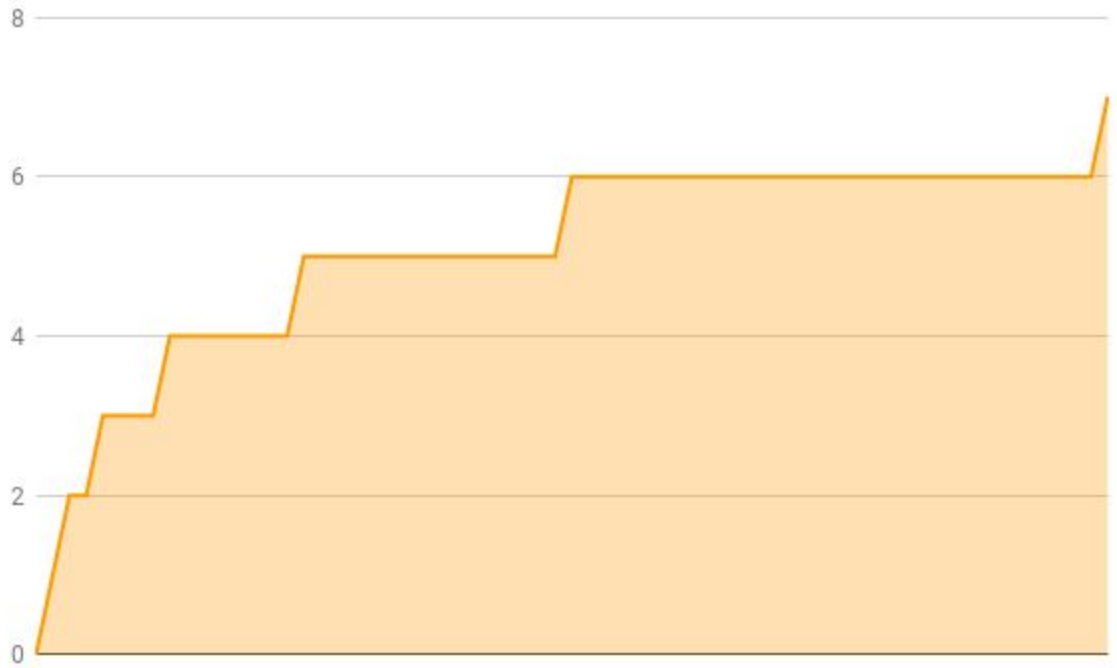
Graph of Foo1:



Graph of Foo2:



Graph of Foo3:



Graph of Foo4:

Graphs ranked by rate of growth, fastest first:

1. Foo4
2. Foo2
3. Foo1
4. Foo3

Part 2 - Code Analysis

Description:

In this part of the assignment I was supposed to test the series given in code by changing the value of N and A. At the end, I was to present my findings in neat tables.

Source Code:

Note: Implements the following series: $\Sigma i = 1 + 2 + 3 + ... + N$

```
//Original code written by Professor Ross. Modified by Quinn Roemer.

#include <iostream>

using namespace std;

int main()
{
  int sum = 0;

  int n = 1;

  while (n <= 10)
  {
        for (int i = 1; i <= n; i++)
        {
              sum = sum + i;

              cout << n << ": " << sum << endl;
        }

        n++;
        sum = 0;
  }
}
```

Output:

Value table:

N Value	Sum Output
1	1
2	1
2	3
3	1
3	3
3	6

4	1
4	3
4	6
4	10
5	1
5	3
5	6
5	10
5	15
6	1
6	3
6	6
6	10
6	15
6	21
7	1
7	3
7	6
7	10
7	15
7	21
7	28
8	1
8	3
8	6
8	10
8	15
8	21
8	28
8	36
9	1
9	3
9	6
9	10

9	15
9	21
9	28
9	36
9	45
10	1
10	3
10	6
10	10
10	15
10	21
10	28
10	36
10	45
10	55

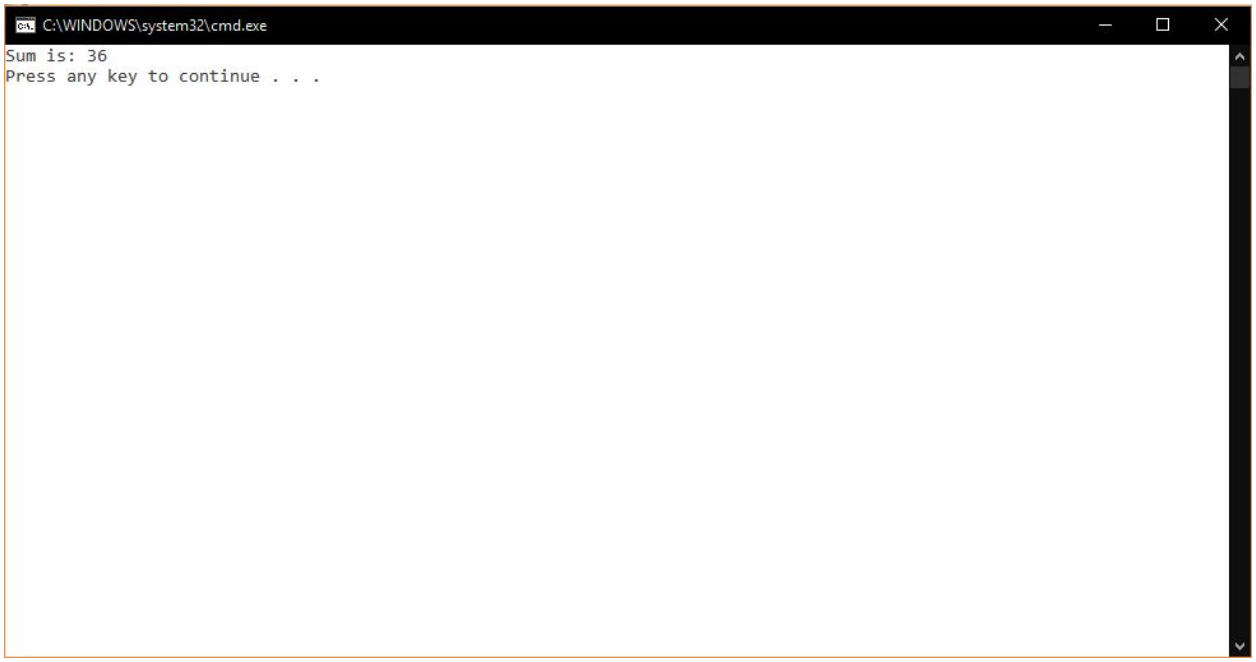
Source Code:

Note: Implements the following series: $\Sigma A^i = A^1 + A^2 + A^3 + ... + A^N$.

```
//Original code written Quinn Roemer

#include <iostream>

using namespace std;

int square(int, int);

int main()
{
   int A = 6;
   int N = 2;
   int sum;

   sum = square(A, N);

   cout << "Sum is: " << sum << endl;

}

int square(int A, int N)
```

```
{
   int temp;
   int sum = 0;

   temp = A;

   for (int count = 1; count < N; count++)
   {
            temp = temp * A;
   }

   sum = temp;
   return sum;
}
```

Output:



Value table:

A: Value	N: Value	Result
1	1	1
1	2	1
1	3	1
1	4	1
1	5	1
2	1	2

2	2	4
2	3	8
2	4	16
2	5	32
3	1	3
3	2	9
3	3	27
3	4	81
3	5	243
4	1	4
4	2	16
4	3	64
4	4	256
4	5	1024
5	1	5
5	2	25
5	3	125
5	4	625
5	5	3125

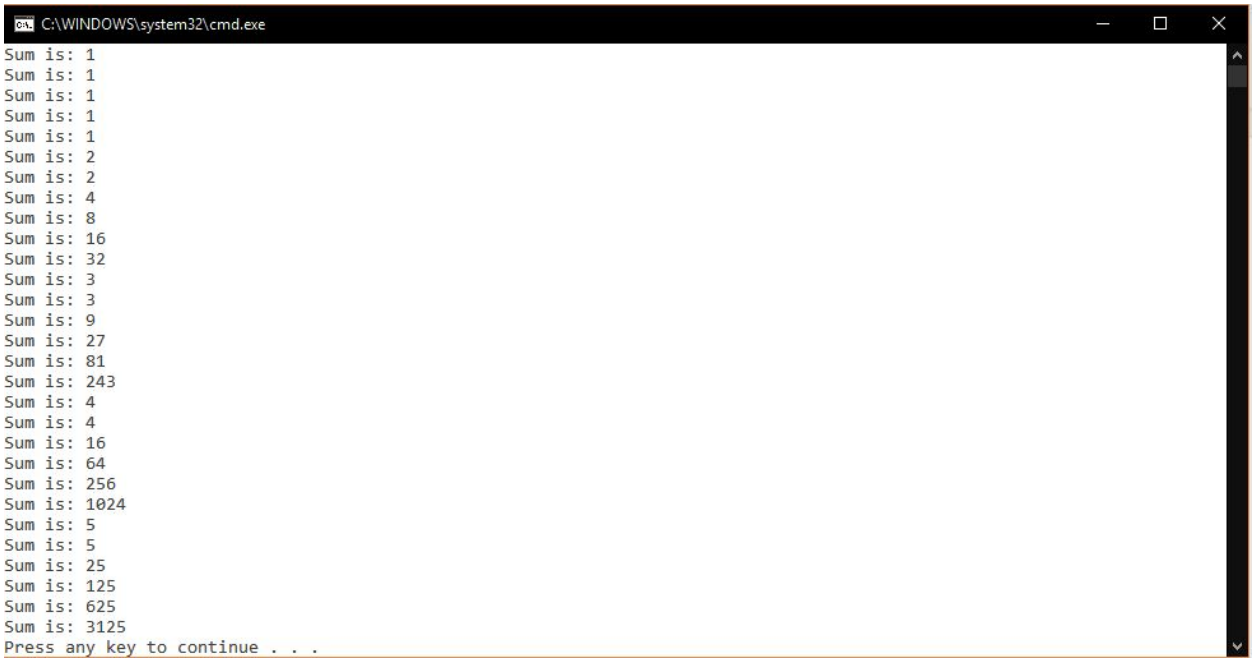
Part 3 - A More efficient Geometric Series

Description:

The goal of this part of the assignment was to create a more efficient algorithm for the geometric series calculation. However, the first algorithm that I created only had one loop to start in it. Thus the following source code will be similar to the previous code.

```
//Original code written by Quinn Roemer.

#include <iostream>
using namespace std;
int square(int, int);
int main()
{
   int A = 1;
   int N = 1;
   int sum;
   while (A <= 5)
   {
            while (N <= 5)
            {
                     sum = square(A, N);
                     cout << "Sum is: " << sum << endl;
                     sum = 0;
                     N++;
            }
            A++;
            N = 1;
   }
}

int square(int A, int N)
{
   int temp;
   int sum = 0;

   temp = A;

   for (int count = 1; count < N; count++)
   {
            temp = temp * A;
   }
   sum = temp;
   return sum;
}
```

Output:


C:\WINDOWS\system32\cmd.exe
Sum is: 1
Sum is: 1
Sum is: 1
Sum is: 1
Sum is: 1
Sum is: 2
Sum is: 2
Sum is: 4
Sum is: 8
Sum is: 16
Sum is: 32
Sum is: 3
Sum is: 3
Sum is: 9
Sum is: 27
Sum is: 81
Sum is: 243
Sum is: 4
Sum is: 4
Sum is: 16
Sum is: 64
Sum is: 256
Sum is: 1024
Sum is: 5
Sum is: 5
Sum is: 25
Sum is: 125
Sum is: 625
Sum is: 3125
Press any key to continue . . .

Part 4 - Another Series

Description:

In this part of the assignment my goal was to implement the following series in code. Also, I was to make a table of outputs with the value of A ranging from 1 to 5 and the value of N ranging from 1 to 5.

$$\Sigma\, iA^i \;=\; 1A^1 \;+\; 2A^2 \;+\; 3A^3 \;+ \ldots \;+\; NA^N$$

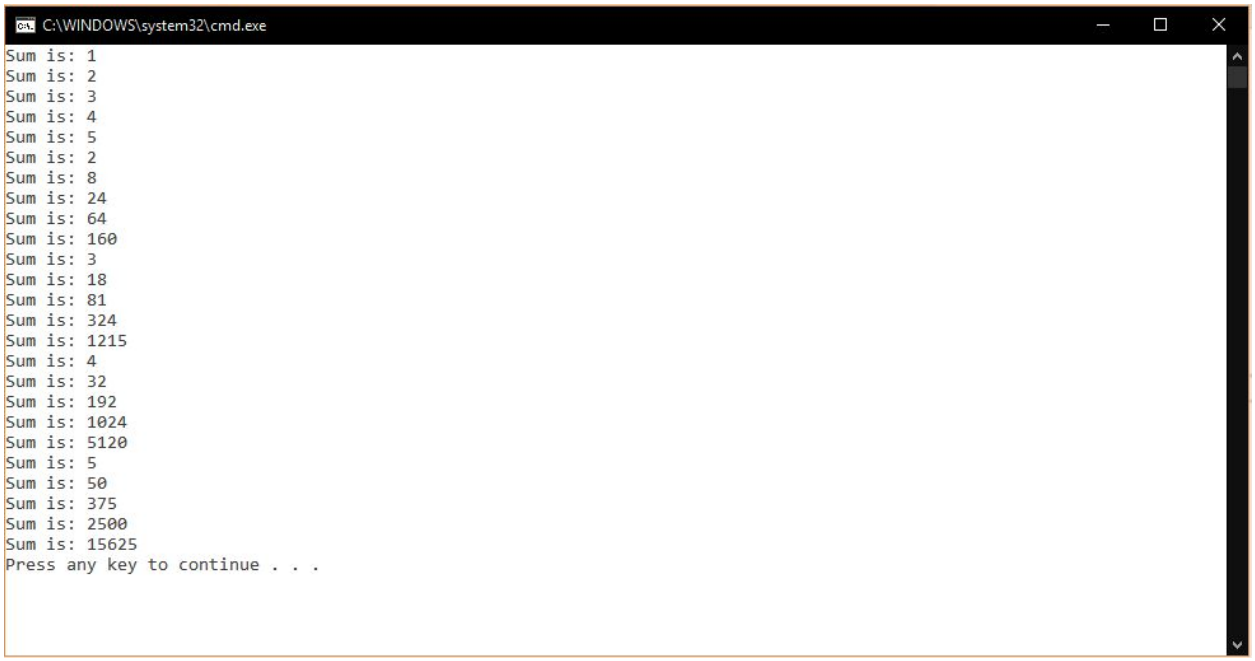
Source Code:

//Original code written by Professor Ross. Modified by Quinn Roemer.

```
#include <iostream>
using namespace std;
int square(int, int);
int main()
{
   int A = 1;
   int N = 1;
   int sum;
   while (A <= 5)
   {
            while (N <= 5)
            {
                     sum = square(A, N);
                     cout << "Sum is: " << sum << endl;
                     sum = 0;
                     N++;
            }
            A++;
            N = 1;
   }
}
int square(int A, int N)
{
   int temp;
   int sum = 0;
   temp = A;

   for (int count = 1; count < N; count++)
   {
            temp = temp * A;
   }
```

```
    sum = temp;
    sum = sum * N;
    return sum;
}
```

Output:



Value Table:

A: Value	N: Value	Result
1	1	1
1	2	2
1	3	3
1	4	4
1	5	5
2	1	2
2	2	8
2	3	24
2	4	64
2	5	160
3	1	3
3	2	18

3	3	81
3	4	324
3	5	1215
4	1	4
4	2	32
4	3	192
4	4	1024
4	5	5120
5	1	5
5	2	50
5	3	375
5	4	2500
5	5	15625

Part 5 - Determine BigO for Previous Series

Description:

In this section of the assignment I needed to examine the previous series I implemented using our “rule of thumb” techniques we learned in class.

Series 1: $\Sigma i = 1 + 2 + 3 + ... + N$

The implementation of this series only involves one loop. The rest of the program only involves items that have a constant time such as assigning values. The work for this series takes place under the loop. Thus the BigO of this series is O(N).

Series 2: $\Sigma A^i = A^1 + A^2 + A^3 + ... + A^N$

The implementation of this series only involves one loop in my case. The rest of the program involves items that have a constant time such as assigning values. The work for this series takes place under the loop. Thus the BigO of this series is O(N).

Note: If I would’ve used the code provided the work would’ve taken place under a nested loop. This means the BigO would’ve been O(N^2).

Series 3 & 4:

$\Sigma A^i = A^1 + A^2 + A^3 + ... + A^N$ & $\Sigma iA^i = 1A^1 + 2A^2 + 3A^3 + ... + NA^N$

The implementation of these series are practically the same. The only addition in the fourth series is a line of code with constant time. Also, the work for these series takes place under one loop. Thus the BigO for these series is O(N).

Conclusion

In this assignment I got to experiment on how different algorithms effect the time complexity of a program in varying ways. This was the first time that I had ever been introduced to the concept of BigO and am very curious about how I can make my programs more efficient. I can’t wait to continue learning more about this.