**CECS 328: Program #1**

For each of the tasks, I have included the algorithms in C# as well as a screenshot of the outputs. The source code for the project is available here: <https://github.com/Vardominator/CSULBProjects/blob/master/CECS328_DataStructuresAlgorithms/Project1/Project1/Project1/Program.cs>

1. Write a program to calculate S(n) by calculating the values of the Fibonacci sequence recursively.

**Source:**



**Sample run:**



1. Write a non-recursive program to calculate S(n).

**Source:**

**Sample run:** 

1. Algebraically verify that g(n) is a solution of Equ 1 by substituting g(n) in Equ 1.

(This portion is attached to the assignment packet)

1. Write a third iterative program by summing g(n) over n.

**Source:**



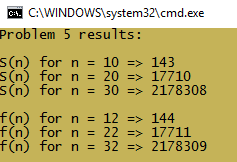
**Sample run:**



1. Calculate these values of S for n = 10, 20, 30. Calculates values of f for n = 12, 22, 32.

**Sample run:**





1. Prove the identity S(n) = f(n + 2) – 1 using induction.

(This portion is attached to the assignment packet)

1. Example run using the alternate sum from #6.

**Source:**

****

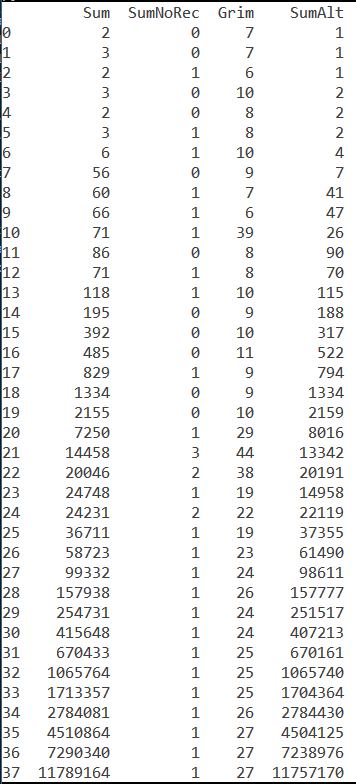
**Sample run:**

****

1. Since I am using Int64 (long in C#) to calculate the sums, the largest n that can be used is such that S(n) does not exceed 9,223,372,036,854,775,807. Another limitation would be the possibility of running out of stack space. Since the Fibonacci calls are recursive, a sufficiently large n would cause the program to crash.
2. Compare the running times of the 4 methods.

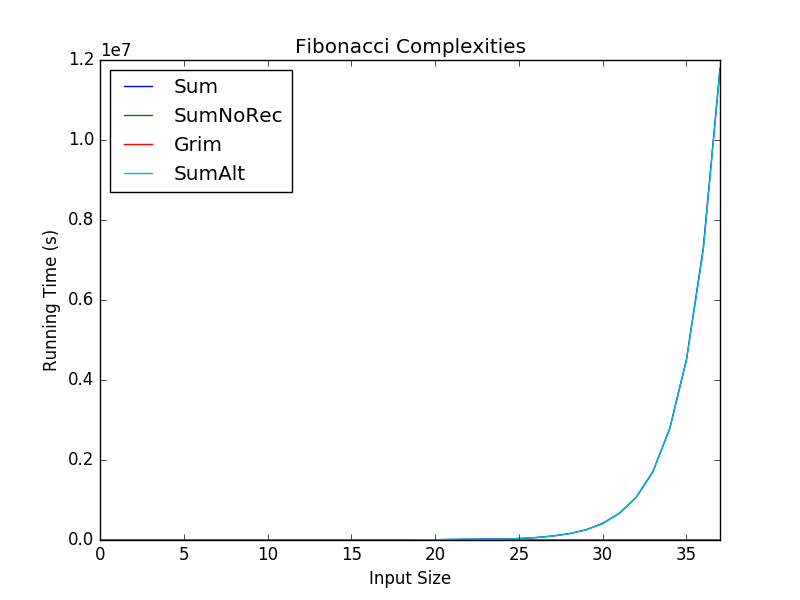
I computed the sum with each method for values of n starting at 1 and ending at 37. I recorded the running times for each. Finally, I ran a python script to plot the growth of the running times versus the input size.

1. Summary and conclusions.



Non-recursive algorithms

Recursive algorithms



We see in the plots above that the 4th method grows the fastest by a considerable amount. However, the difference is not much. The first drawback of the recursive methods is that they have linear growth in space complexity: , as opposed to the non-recursive methods that have a constant, , space complexity.

Mathematically the recurrence relation for the Fibonacci sequence is . Using a recursion tree, we can prove that the time complexity is . This means that the recursive Fibonacci sequence running time grows very fast. In fact, a trial run using a recursive method with an input size of 50 took me about 30 minutes to run. This is incredibly inefficient and its growth can be seen in the table to the top-right. I have displayed here the input size on the left most column, the results of the recursive methods on the second and fifth columns and the non-recursive methods in the middle columns (their values represent the elapsed ticks of the timer in C#). We can clearly see that after an input size of about **13**, the incredibly different growth rates become evident.

All in all, it is very clear that the recursive methods dominate in regards to computational time (in addition to computational space). By looking at the plot we see that the **Sum** and **SumAlt** lines, which represent the recursive growths, overlap because their computation times are nearly identical, and **SumNoRec** and **Grim**, the non-recursive running times, are so insignificant they are not even visible.