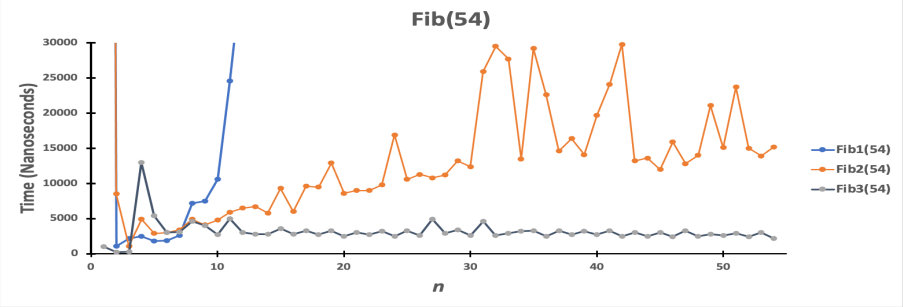
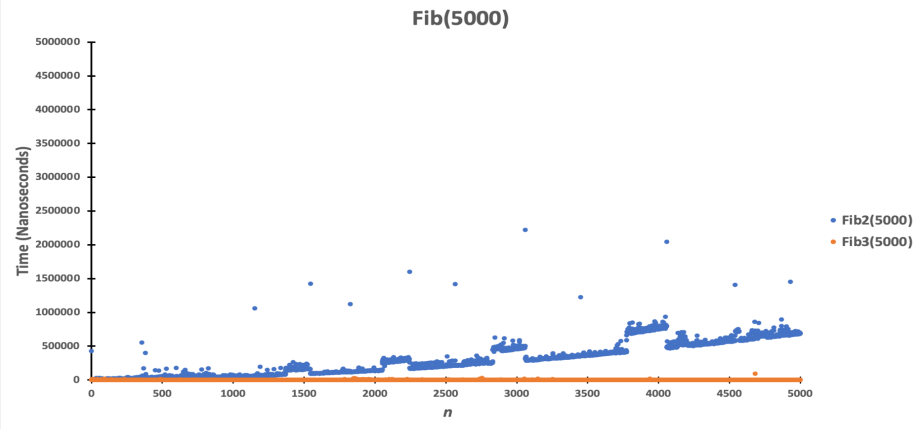
CPSC 331 Assignment One - Part I and II Names: Hassan Sohail, Mellisa Phongsa



Graph 1: Plot of Algorithms 1, 2, and 3 generating the first 54 fibonacci numbers



Graph 2: Plot of Algorithms 2 and 3 generating the first 5000 fibonacci numbers

Analysis: By examining graph 1, we could immediately tell that the fib1 algorithm had the slowest running time. Looking at the curve for the fib1 algorithm, we see that the running time values recorded are increasing at an exponential rate. The fib2 algorithm seems to have a much faster running time than the fib1 algorithm. We can immediately recognize this as the running time values for the fib2 algorithm are much smaller and the curve for the algorithm is growing at a linear rate. Finally, we can infer from the plot that the algorithm with the fastest running time is the fib3 algorithm. The graph shows that the fib3 algorithm starts at very small time values and it increases until n is about equal to 15 at which point the running time values don't seem to change much. The time values in the fib3 algorithm are much smaller than the fib1 and fib2 algorithm values. To explain the differences in the running time of these algorithms, we took a look at the code. Firstly, we believe that the fib1 algorithm has the slowest time complexity because every recursive call in this algorithm ends up doing 2 more additional recursive calls with the previous 2 fibonacci numbers. As a result of this, we believe that the function calls in this fib1 algorithm grow exponentially by 2^n. Looking at the fib2 algorithm, it seems that this algorithm is mostly dependent on the *for loop* which compares the k value with the n value. This *for loop* will run an n number of times and since the rest of the algorithm is based on constant time functions, the algorithm should have linear growth, making it much faster than the fib1 algorithm. Finally, the fib3 algorithm seems to halve the input value n by 2 when n is even and by $\frac{n+1}{2}$ when it is odd, using this to then calculate the fibonacci numbers. We believe that this halving of the input value gives us a logarithmic function for this algorithm. This would explain why fib3 is the fastest of the algorithms, as the logarithmic function will result in a faster time complexity then the exponential and linear functions of algorithms fib1 and fib2 respectively. This explanation is also evident in graph 2 which has a large input value of fib(5000). The graph shows that the running time values for the fib2 algorithm are increasing linearly. The running time values of the fib3 algorithm are so small compared to the fib2 algorithm that the graph shows the curve of the fib3 algorithm as just a straight constant line along the x-axis. We believe that the fib3 curve looks like a straight line on the graph because of its logarithmic functionality resulting in these very small and somewhat constant running time values. This graph shows that on a larger scale a logarithmic function will just appear to be a constant line when compared with a linear function such as the fib3 and fib2 algorithms respectively. With all of this information and mainly going off the curves seen in both the graphs, we were able to figure out the big Oh of each algorithm. **The big Oh of the fib1 algorithm is O(2^n). The big Oh of the fib2 algorithm is O(n) and finally the big Oh of the fib3 algorithm is O(log n).** After doing this analysis, we realized that we would prefer to use the fib3 algorithm when calculating the fibonacci especially under the circumstances of working with very large inputs. If the input values are small such as fib(10) or fib(15) then any of the functions can be used without any problems, but when working with much larger input values such as fib(10^6), then the fib3 algorithm would be the most convenient choice.

The Loop Invariant:

(a) n is a non-negative integer input such that $n \geq 0$

(b) k is a non-negative integer variable such that $0 \leq k \leq n$

(c) i is a non-negative integer variable such that $i \geq 0$, where i represents the (k - 1)th Fibonacci number.

(d) j is a non-negative integer variable such that $j \geq 0$, where j represents the k-th Fibonacci number.

(e) f(m) = j for every integer m such that $0 \leq m \leq k$ and j is returned as output

Claim: Consider the Fib2() algorithm and the for loop in this algorithm. Suppose that this algorithm is executed when the precondition is satisfied. If ℓ is a positive integer such that the body of the for loop is executed at least ℓ times when the for loop is being executed, then the loop invariant is satisfied at both the beginning and the end of this ℓth execution of the loop body and generates the correct output.

Proof by Mathematical Induction: This will be established by induction on ℓ. The standard form of mathematical induction will be used

Basis (ℓ = 1): suppose that n = 1 and there is at least $\ell = 1$ executions of the body of the for loop. Since there is no step in this algorithm that changes the value of n, part (a) of the loop invariant is satisfied when the for loop is reached. Since there are no statements in the for loop that change the value of n, it is still true n is an integer input such that $n \geq 0$. Thus part (a) of the loop invariant is also satisfied at the end of this execution of the loop body as well. The variable k is declared to be an integer variable with value 1, so that $1 = k \leq n$, and part (b) of the loop invariant is satisfied when the for loop is reached as well. Furthermore, the loop test is checked and passed so that $k \leq n$. Since k and n are both integers it follows that k is an input such that $1 \leq k \leq n - 1$ at the beginning of this execution of the loop body. In the loop body n is unchanged, but the value of k is increased by one. It follows that, at the end of the execution of the loop body, k is an integer variable such that $2 \leq k \leq n$, as needed to establish that part (b) of the loop invariant is satisfied at the end of this execution of the loop body. Finally, Since i and j are declared to be integer variables with values 1 and 0, respectively. Since j holds the k-th Fibonacci number, when k = 0, f(k) = f(0) = 0, therefore part (c), (d) and (e) of the loop invariant also hold when the loop is reached. At the end of this execution of the loop body j becomes the sum of its previous value, 0, and the value of i = 1. Thus j becomes the k-th fibonacci number, where k = 1, f(k) = f(1) = j = 1. The value of i becomes j minus its previous value, and thus i = f(k - 1) = k(1 - 1) = f(0) = 0. Which is needed to establish that part (c), (d), and (e) of the loop invariant is satisfied at the end of the execution of the loop body. Thus the loop invariant is satisfied at both the beginning and end of first execution of the body of the for loop. Furthermore, at the end of the execution of the for loop j is the k-th Fibonacci number and returned as output, as needed to establish the claim when $\ell = 1$, and complete the basis

Inductive Step: let k be an integer such that $k \geq 1$. It is necessary and sufficient to use the following

Inductive Hypothesis: If the body of the for loop is executed k times then the loop invariant is satisfied at both the beginning and the end of the k-th execution of the loop body. To prove the following

Inductive Claim: If the body of the for loop is executed k + 1 times then the loop invariant is satisfied at both the beginning and the end of the (k + 1)th execution of the loop body. Suppose that the loop body is executed at least k + 1 times then it is true that the loop body has been executed at least k times, and it follows by the inductive hypothesis that the loop invariant is satisfied at both the beginning and the end of the k-th execution of the body of the for loop. We know that i and j represent the (k - 1)th and k-th Fibonacci numbers respectively. Thus at the end of this k-th iteration, j contains the k-th Fibonacci number and i contains the (k - 1)th Fibonacci number. j = f(k) = f(k - 2) + f (k - 1), and j is returned as output. In the (k + 1)th iteration, j becomes the sum of the previous two Fibonacci numbers, which was i, and j in the previous iteration (kth iteration) j = f(k + 1) = f((k + 1) - 2) + f((k + 1) - 1) = f(k - 1) + f(k). Therefore at the end of the (k + 1)th iteration i and j represent the k-th and (k + 1)th Fibonacci numbers, respectively. Thus the loop invariant is satisfied at both the beginning and end of this (k + 1)th execution of the body of the for loop and f(k + 1) = j is returned as output, as needed to establish the inductive claim and proof of the claim that the loop invariant is satisfied at both the beginning and end of the body of the for loop and the correct Fibonacci number is returned as output.