Jeffrey Lansford

Lab 1

CS 372

08-28-2019

Introduction:

The Fibonacci Algorithm is one of the first algorithms a Computer Science student learns. It could be the first recursive function they learn as well, but what we first learn is not always efficient. The recursive Fibonacci Algorithm is incredibly slow with higher input, so we need to improve on this algorithm with creating one that has a faster runtime.

Methods:

In our first lecture, we went over pseudo-code of implementing a Fibonacci Algorithm that has a linear runtime instead of the exponential runtime of the recursive algorithm. Instead of recursive calling a function to get previous numbers, we can use a data structure to store the previous results and call that data structure to get previous numbers. We used an array as it is the simplest solution and start the array with 0 and 1 for the first two elements. Then we can simply add the two numbers together to get the Fibonacci number we want. Here is the linear function in C++:

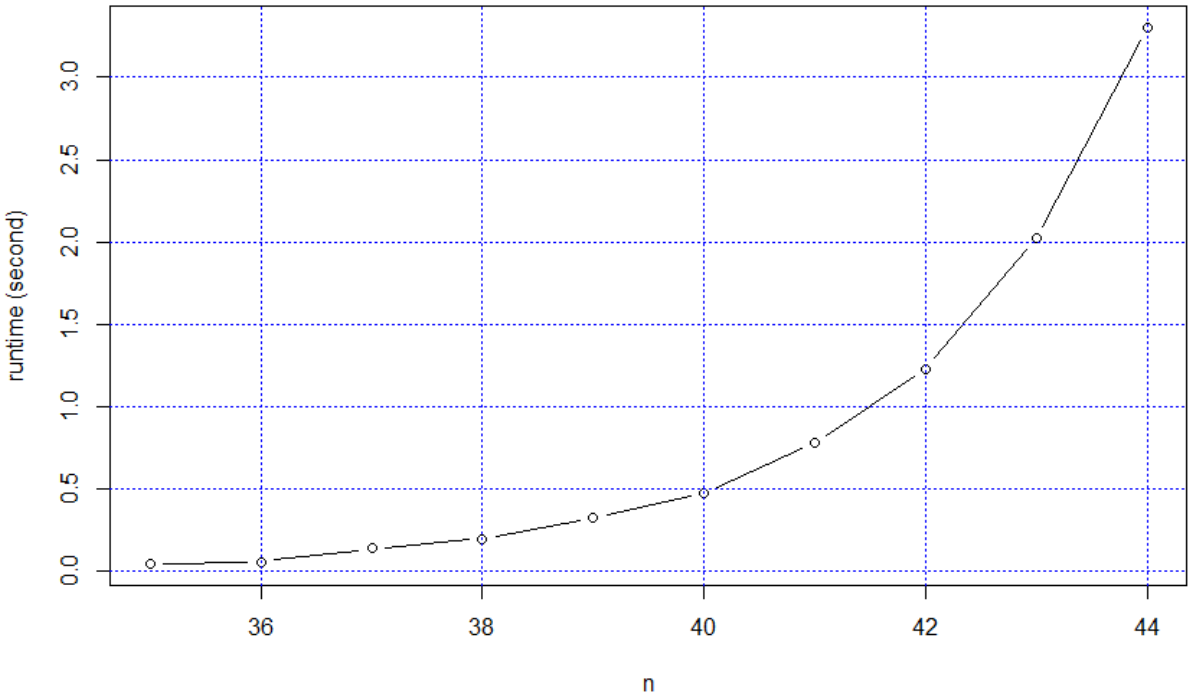
```
// Fibonacci algorithm with a linear runtime
int fib2(int n) {
  // creates a dynamic array
  int *history;
  history = new int[n+1];
  history[0] = 0;
  history[1] = 1;
  // gets 2 previous results from history array and adds to new value
  for (int i = 2; i < n+1; i++) {
    history[i] = history[i-1] + history[i-2];
  }
  // get last value of sequence
  int returnValue = history[n];
  // delete array for garbage removal
  delete [] history;
  return returnValue;

}
```

Using R we can measure the runtime and plot out the points on a graph to visual see the runtime:

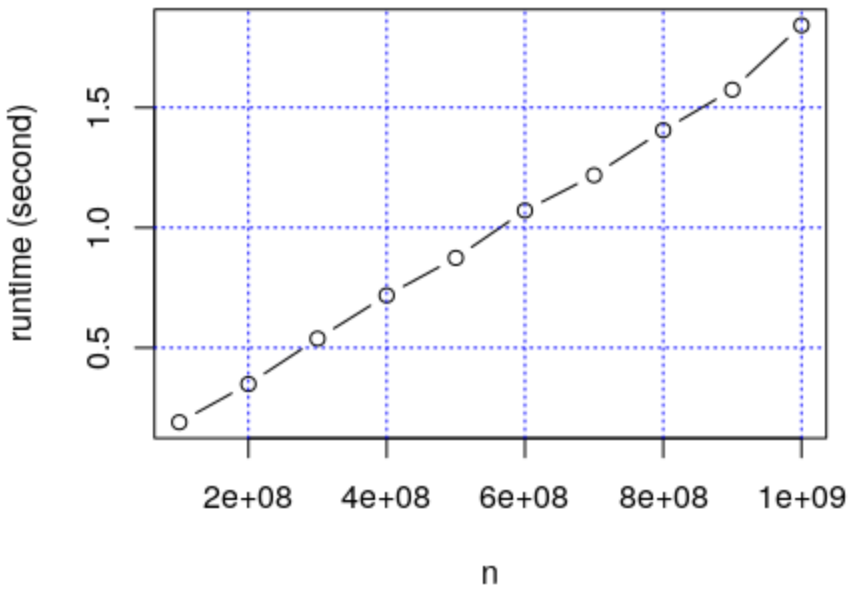
```
k <- 10
ns <- (1:k) * 100000000
runtime <- vector(length=k)

for ( i in 1:k) {
  n <- ns[i]
  runtime[i] <- system.time(fib2(n))["user.self"]
}
plot(ns, runtime, type="b", xlab="n", ylab="runtime (second)")
grid(col="blue")
```

Results:

The first graph is the time the recursive Fibonacci algorithm with different sizes of input. As you can see it is in a shape of an exponential function proving that the recursive function has an exponential runtime.



This graph is the result from the linear runtime algorithm I showed earlier. We can also clearly see the that the runtime is a linear function. To note, I had to use a much larger data set compared to the recursive algorithm to achieve it looking like a linear function since this algorithm is much faster.



Discussions:

As we can see, the runtime does grow linearly as the input increases. This is good for our algorithm as we can verify that it is running the way we are expecting it to be. To see how faster the linear algorithm is to the exponential algorithm, we must estimate a little to have the same input. Creating a regression line with the data from the linear algorithm, we can create a function: $1.76e-9x+6.66e-4$ to help estimate the runtime will be for the input 44. Plugging in 44 gets us $6.67\mathrm{e}-4$ seconds. The runtime for the exponential algorithm with the input of 44 gives 3.3 seconds. The linear algorithm is very much faster than the exponential algorithm by 5000%. The difference is astounding of using a data structure can improve an algorithm to be more efficient.

Conclusions:

In this lab, we wrote a linear Fibonacci algorithm to improve the exponential Fibonacci algorithm we are used to writing. We used the language R to plot out the different algorithms to clearly see the difference in runtimes. What this lab showed me is how effective a data structure can improve your program. Using the linear Fibonacci algorithm makes the program more robust and least likely to encounter a fault if it was being used in a bigger application.