CIS 303 Analysis of Algorithms
Current Semester Here

Trevor Butler
Assignment 3b

10/07/20

Problem

Provided with the `FibTest.java` source file, implement a client to run and record times of two different methods implementing iterative and recursive Fibonacci generation (`fibi(int n)` and `fibr(int n)` respectively). The recorded data will show the compared times in nanoseconds, where they will be plotted to a graph to help demonstrate which method is the best case.

Hypothesis

Being that time is an important consideration when writing code, it's imperative to understand which algorithms result in the best case. While looking through the provided `FibTest.java` source file, there are two different functions which I believe will have very different outcomes regarding time. I believe this is true as the function `fibr(int n)` is equivalent to 2^n being it calls and returns the function `fibr(int n)` twice through every iteration of the for loop in `FibClient.java`. But `fibi(int n)` on the other hand appears to be n as it runs through a single for loop n times. Thus, `fibi(int n)` would result in the best case.

Methods

1. The implementation:

```
public static void main (String [] args) throws Exception{

    int maxFib = 65;
    PrintStream fibiPrintStream = new PrintStream(new File("fibiData.txt"));
    fibiPrintStream.println("n\tTime(ns)");
    for(int fi = 3; fi <= maxFib; fi++){
        long startFibi = System.nanoTime();
        FibTest.fibi(fi);
        long stopFibi = System.nanoTime();
        long timeFibi = stopFibi-startFibi;
```

```
        fibiPrintStream.println(fi + "\t" +   timeFibi);
    }

    PrintStream fibrPrintStream = new PrintStream(new File("fibrData.txt"));
    fibrPrintStream.println("n\tTime(ns)");
    for(int fr = 3; fr <= maxFib; fr++){
        long startFibr = System.nanoTime();
        FibTest.fibr(fr);
        long stopFibr = System.nanoTime();
        long timeFibr = stopFibr-startFibr;
        fibrPrintStream.println(fr + "\t" +   timeFibr);
    }
}
```

2. General approach:
   The approach to this program was to implement a for loop for each function (`fibr(int n)` and `fibi(int n)`) that will iterate through n times while recording the time it took for each iteration of the loop. This way we can create and observe a graph of the time taken by each n in both the iterative and recursive functions.

3. Data structures:
   Within this assignment the main structure that was implemented was a for loop for each iterative and recursive function call. While doing this it was also extremely important to record our data thus PrintStream was used to create a text file which would later be used to create a graph to analyze.

4. Experiment details.

   (a) Size of N you used:
   65

   (b) Case(s) tested (best, worst, random, etc.):
   Two cases were tested: Best-case (iterative) and worst-case (recursive)

   (c) Number of iterations/copies of the experiment:
   There were 65 iterations (n) of both the recursive and iterative functions

   (d) Metrics.
   Run-time in nanoseconds

Results and Discussion

After implementing and running the newly created code there were a few things to take note of right off the bat. The first was that fact that milliseconds was not sufficient enough while logging data as it was impossible to see change from number to number in the iterative text file

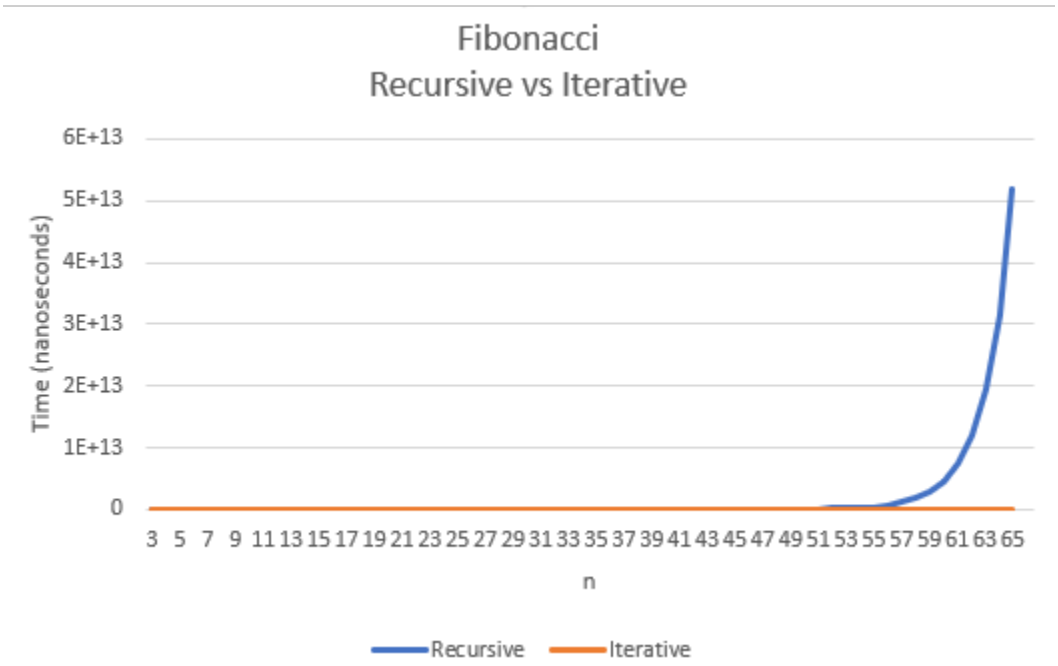

Figure 1: Graph of recorded numbers in time test between recursion and iteration

as they were all zeros. After the change to nanoseconds and increasing the n value past high 50's you could see just how much time began to increase (Figure 1 illustrates this perfectly). Initially puzzled when reading *"I encourage you to go beyond* $n = 50$ *if you have the time to invest"*, it began to make sense when hours would pass without a new log in the recursive text file. When $n = 59$, recursion took about a 45 minutes to complete then as n increased only 3 more times to 62 it was about 3.3 hours this is when the time investment statement started to make sense. Ending the experiment at 65 and looking back was a huge eye opener as $n = 65$ the record time was about 14.4 hours; just one n back to 64 the record time was about 8.7 hours.

Seeing these increases of time helps illustrate the difference between iteration and recursion as iteration never went above 399,700 nanoseconds ($n = 1$). When $n = 65$ the iterative text file logged 1,100 nanoseconds, comparatively to the recursive log of 51,803,163,885,400 nanoseconds (14.4 hours). This is obviously a major difference, the fact that the timestamp had to be changed to nanoseconds from milliseconds because the iteration text file showed only zeros expresses how much faster iteration is.

As previously stated, the highest iteration time logged was when $n = 1$ at 399,700 nanoseconds which seemed odd so it was ran a few more times to see if it was some sort of error. After running it a few more times, n = 1 was always the largest. This is because on the first iteration all of the code before the for loop must be ran, but after that is ran and gets into the for loop time gets cut significantly as it just iterates through the for loop with no previous code to execute.

Conclusion

My hypothesis was valid as recursion was much more expensive than iteration showed to be. The graph and data backs this claim up as it it very simple to see the difference between these two results. Just by looking at the graph you can see the exponential increase of time in recursion when n gets to the low 50s. This also does not mean that iterative and recursive are the same up until $n = 50$ but rather this is where you can visually see this in Figure 1. When $n = 7$, recursion clocked at 1,100 nanoseconds, the same time of iteration at $n = 65$ thus further helping support the validity of my hypothesis.

A follow-up I would like to consider would be to see when the iterative times being to climb exponentially like the recursive times. I am interested to see at what point does iteration take hours on end to compute. It seems as if the number would have to be extremely large to take hours considering at $n = 65$ it only took 1,100 nanoseconds which is roughly 0.0000000003 hours