Noah Caulfield

Dr. Essa Imhmed

CS 451

3/4/2024

PA 01

1. *Does the runtime grow linearly as input number n increases in LinearFibonacci.java?*

   It saves Fibonacci numbers up to the n in an array, iterating from 2 to n, each step calculating the next Fibonacci number as the sum of the two preceding ones. With a runtime complexity of 0(n), the method's efficiency is linear, as it uses a single loop running n times. Consequently, any increase in n leads to one more loop iteration, demonstrating a linear relationship between n and the runtime.

2. *Does the runtime grow exponentially as the input number n increases in ExponentialFibonacci.java?*

   The ExponentialFibonacci.java program's runtime increases very quickly for larger inputs. It uses recursion, making two calls for each number, which significantly multiplies the amount of work needed as the input grows, leading to an exponential increase in runtime, known as 0(2^n).

3. *How often is the linear algorithm faster than the exponential algorithm in ExponentialFibonacci.java on the same input?*

   The linear algorithm is consistently faster than the exponential algorithm for the same input, due to its (O(n)) complexity, compared to the exponential algorithm's (O(2^n)) complexity, which slows down significantly even for small inputs.

4. Write a report that describes your work done in the following sections:
   A. Introduction (define the background and the problem),

      For our assignment, we're tasked with investigating the efficiency of Fibonacci algorithms through Java implementations, assessing their execution times with varying inputs, and contrasting these

statistical findings against their theoretical time complexities. We'll utilize R for precise runtime measurements and visualizations to determine if the actual runtime performance matches the theoretical expectations. This study will allow us to explore the practical applicability of these algorithms and confirm the accuracy of theoretical predictions, specifically focusing on both linear and exponential implementations.

B. Methods (provide the solutions),

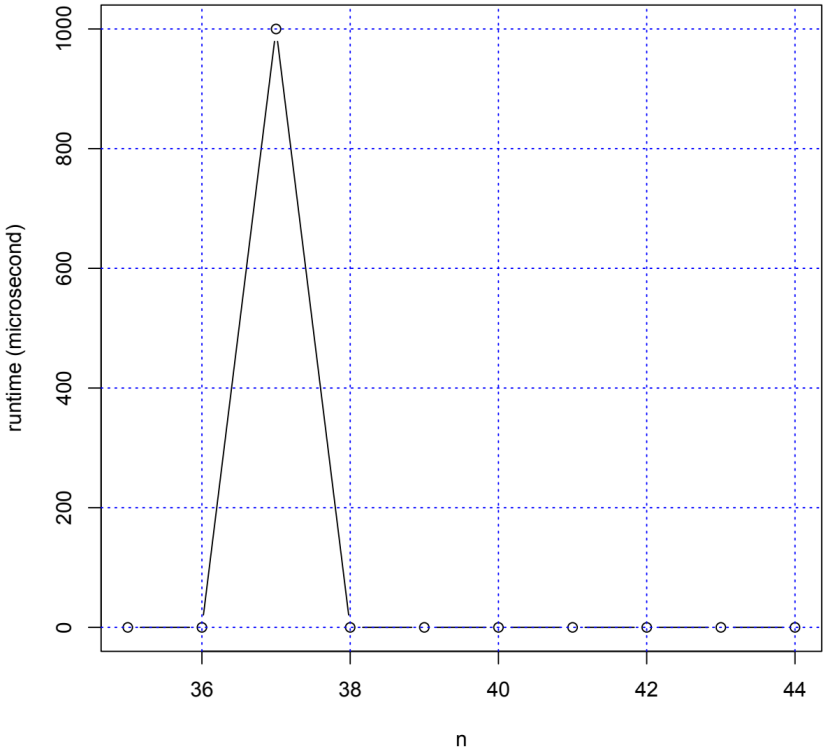
Essentially, calculateFibonacciNumber(int n) for the exponential Fibonacci is all about figuring out Fibonacci numbers in a certain way that's a bit more complex and time-consuming, known as the exponential Fibonacci algorithm. When dealing with base cases, where n is either 0 or 1, the method just gives back 0 or 1, respectively. But for any other number, two questions are asked: "What's the Fibonacci number just before me?" and "What's the one before that?" It calculates both by calling itself with n-1 and n-2, adds those answers together, and produces Fibonacci number for n. This recursive trick allows for it to stick right to the algorithm we're aiming to follow, even though it means the method has to do a lot more work as n gets bigger.

In the `LinearFibonacci.java`, we streamlined the process to figure out Fibonacci numbers without the recursive back-and-forth. Starting off, if `n` is either 0 or 1, it's simple: the answer is `n` itself. For bigger numbers, I use a straight path, calculating each Fibonacci number one after the other and storing them. This way, by the time we reach `n`, we've already got all our answers lined up, making it faster and more efficient, especially for larger `n`.

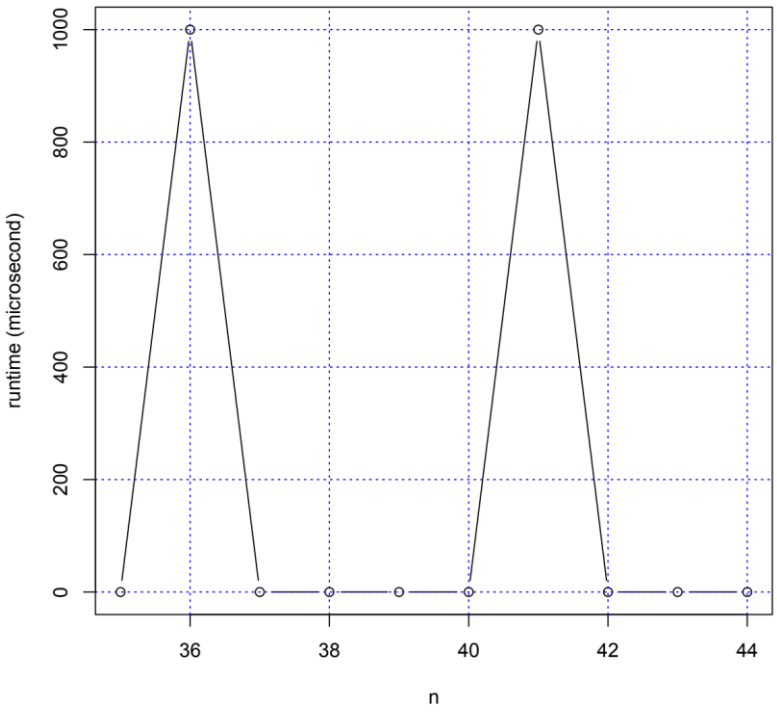
The. R file supplied only required edits to point to the correct Java file/class.

C. Results (show the numbers and figures),

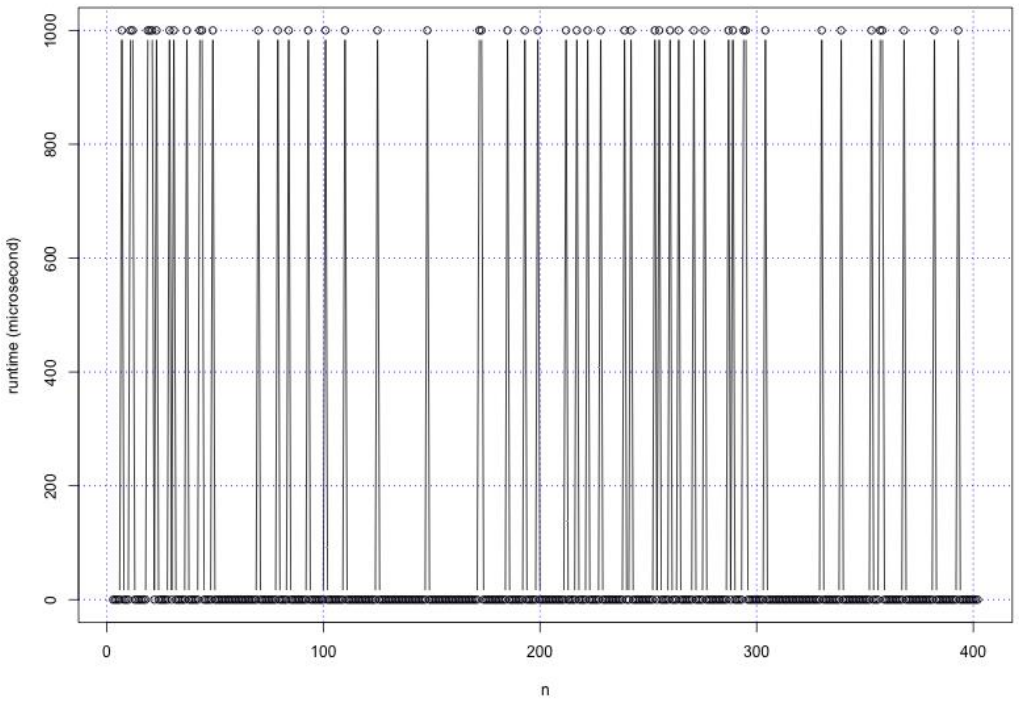
LinearFibonacci with base settings:



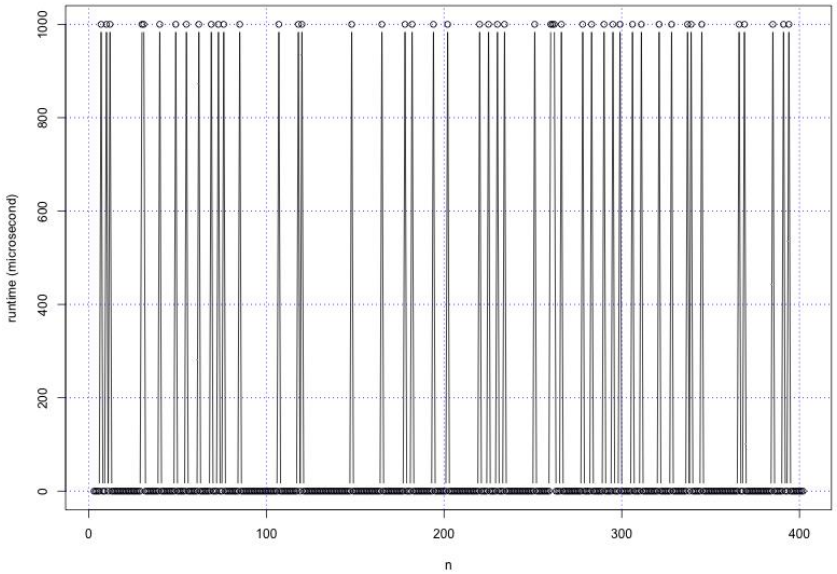
ExponentialFibonacci with base settings:



In .R file, change 10 to 400 and 34 to 2 for LinearFibonacci:



In .R file, change 10 to 400 and 34 to 2 for Exponential Fibonacci:



D. Discussions (implications and issues),

When comparing linear and exponential approaches for calculating Fibonacci numbers, the linear method can be seen as more efficient for larger n values. The exponential method, while simpler and more straightforward within this project, suffers from a significant increase in computation time as n grows due to repeated calculations of the same Fibonacci numbers The exponential method can quickly become impractical for large n, demonstrating the importance of algorithm optimization in software development.

E. Conclusions (summarize the lab and point to a future direction).

This lab's exploration into calculating Fibonacci numbers using linear and exponential methods showed us the importance of algorithm selection in programming. It demonstrated that efficiency can significantly vary depending on the approach, with linear methods offering speed and scalability for larger inputs. The challenge remains to innovate and apply even more efficient algorithms for complex problems, suggesting a time in the future where optimization and computational efficiency continue to be in the forefront of software development and algorithmic research.