Ryan DiRezze

Luyao Zhang

CS 162 400

December 2, 2018

Performance Analysis of Recursion vs. Iteration

Before evaluating program runtime differences between iteration and recursion, runtime expectations will be briefly described. When looking at iteration's function, the function accepts an integer value as a function parameter then executes code within the function to return a result. In contrast, the recursive program takes one "step", or "iteration", per function call until calling the recursive function enough times to reach the desired result, which is then returned (#2). Based on the way each program method is structured, I expect the program, which uses iteration, to execute and return a value significantly faster than the recursive function, as the iterative method performs all calculations within one function call, whereas the recursive method requires multiple function calls to return a value, even for a small input parameter.

My expectation is based from the assumption that the number of function calls has an impact on performance, which is reflected by the speed that a program can execute and complete based on different input parameters. Thus, as the Fibonacci number increases, in this example, the runtime difference in duration between iteration and recursion will grow at a non-linear rate, where the marginal difference between the two methods will grow as each function's input parameter grows.

Multiple values of *N* were chosen as input parameters for the iterative and recursive functions, finding the *Nth* Fibonacci number, including: 1, 10, 20, 25, 30, 35, 40, 45, 50. The results show a significant difference in performance between the iterative and recursive functions

after surpassing some threshold value of *N*. Specifically, at *N = 30*, a runtime difference began to emerge for the first time, which grew exponentially as the value of *N* grew. Between *N* values of 30 and 35, the runtime difference was insignificant, reflecting a runtime difference of less than a couple tenths of a second. Once *N* was set to 40, a larger runtime difference was shown, reflecting about a second and a half difference in runtime.

However, once *N* grew to 45 and above, a significant runtime difference was experienced where the iterative function continued to execute and complete at 0 clicks (0 seconds), whereas the recursive function executed and completed with 16,740,000 clicks (16.74 seconds) and 184,900,000 clicks (184.9 seconds) at *N* values of 45 and 50, respectively. One additional item to note is that the returned Fibonacci number for *N = 50* of both the iterative and recursive functions were incorrect, each reflecting a negative number. The iterative function, however, still returned a runtime duration of 0 clicks (0 seconds), and the recursive function returned a runtime duration of 184,900,000 clicks (184.9 seconds). Figures 1 and 2, which are found below in a table and graph, illustrate the seemingly exponential growth relationship of the runtime value for the recursive function versus the stable, and nonexistent, runtime value for the iterative function.

Based on the performed tests, using *clock* to capture and print runtime durations of both iterative and recursive functions with multiple values of *N*, there appears to be a significant runtime penalty with recursive functions that begins to appear after some threshold value (#1,2). After surpassing the threshold value, which is represented by *N = 30* in this case, the marginal difference in runtime grows non-linearly with the value of *N*. The below set of data, reflecting *N* values of 1 through 50, illustrate an exponential growth curve for the runtime duration of recursive functions based on the recursive function's input parameter. Thus, iteration would be the preferred choice when dealing with potentially large values as input parameters.

Comparison Between Iteration and Recursion with Runtime Speed

Value of N	Iteration Runtime	Recursion Runtime
1	0 clicks (0 seconds)	0 clicks (0 seconds)
10	0 clicks (0 seconds)	0 clicks (0 seconds)
20	0 clicks (0 seconds)	0 clicks (0 seconds)
25	0 clicks (0 seconds)	0 clicks (0 seconds)
30	0 clicks (0 seconds)	10,000 clicks (0.01 seconds)
35	0 clicks (0 seconds)	140,000 clicks (0.14 seconds)
40	0 clicks (0 seconds)	1,570,000 clicks (1.57 seconds)
45	0 clicks (0 seconds)	16,740,000 clicks (16.74 seconds)
50	0 clicks (0 seconds)	184,900,000 clicks (184.9 seconds)

Note: *Function output answer was incorrect when N = 50*

Figure 1

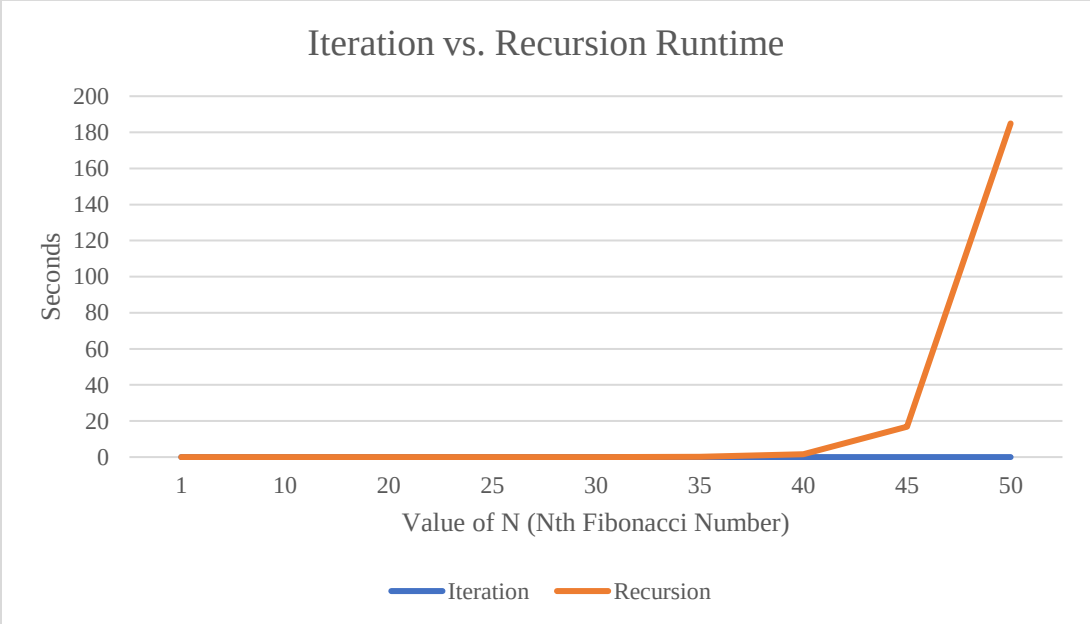

Figure 2

Works Cited

1. Program runtime tool: http://www.cplusplus.com/reference/ctime/clock/

2. Iteration and recursion programs: https://www.codeproject.com/tips/109443/fibonacci-recursive-and-non-recursive-c