Running Time Analysis

Program name : Lab10_Schaefer_Kristin.pdf
Author : Kristin Schaefer
Date : 06-09-2019

N	Recursive Running Time (microseconds)	Iterative Running Time (microseconds)
1	10	2
10	10	2
20	110	2
30	11,301	9
40	1,404,309	6
42	3,655,232	6
45	15,328,161	6

Comparison and Analysis

Program name : Lab10_Schaefer_Kristin.pdf
Author : Kristin Schaefer
Date : 06-09-2019

Sources:
1. Canvas, Week 10 Overview
2. https://cs.stackexchange.com/questions/56867/why-are-loops-faster-than-recursion
3. https://www.cs.cornell.edu/Info/Courses/Spring-98/CS211/lectureNotes/07-Recursion.pdf
4. https://en.wikipedia.org/wiki/Recursion_(computer_science)

When testing the running times of the recursive and iterative Fibonacci functions, I was surprised by the dramatic increase in running time for the recursive function in comparison to the iterative function, which remained relatively stable. Starting with testing the base case of n = 1, the running time for the recursive function was already considerably higher at 10 microseconds in comparison to the iterative function which took only two microseconds. When n was increased to 10, the running time for the recursive and iterative functions remained the same. I then tested n = 20 to see the recursive function increase over 5 times to 110 microseconds, while the iterative function remained stable at 2 microseconds. The first dramatic increase came with the next test of n = 30. The running time increased over 100 times to 11,301 microseconds while the iterative function's running time grew to 9 microseconds. Incrementing to n = 40, I noticed a sizable delay when running the program. This was visible in the running time results with the recursive function's running time increasing once again over 100 times to 1,404,309 microseconds. In contrast, the iterative function surprisingly decreased back to 6 microseconds. After testing n = 40, each additional increment to n produced a notable running time delay in the recursive function. I found that n = 45 produced a long enough delay that I would limit the input range for n to 45. The final comparison of n=45 was shocking, with the recursive function taking 15,328,161 microseconds and the iterative function remaining stable at 6 microseconds.

Prior to this week's module, I would have expected that recursion would take less running time than an iterative function, as recursive functions are typically more compact in terms of length of written code. However, I learned from the Week 10 Overview on Canvas that while iteration "simply executes a new repetition of the code that is in the body of a loop" (1), recursion executes each repetition as a function call (1). Recursion is a memory-intensive operation, and with memory speeds being slower than processor speeds, this leads to a very slow running time (1). To get a better picture in my mind of the difference, I found a diagrammatic example of the difference between an iterative and recursive function in assembly:

Diagrammatic example of iterative function in assembly

```
mov loopcounter,i
dowork:/do work
dec loopcounter
jmp_if_not_zero dowork
```

(2)

Diagrammatic example of recursive function in assembly

```
start_subroutine:
pop parameter1
pop parameter2
dowork://dowork
test something
jmp_if_true done
push parameter1
push parameter2
call start_subroutine
done:ret
```

(2)

These diagrams helped me to see that on a basic level, the iterative function requires less "jumps" and does not need to keep track of parameters, whereas the recursive function uses the run-time stack to hold activation records for each function call (2)(3). Activation records store each function's set of parameters and local variables (3). "After the function call finishes, the AR is popped off the stack and eventually destroyed" (3). It is not a surprise that for the Fibonacci function tests, the recursive function's running time increased dramatically, as with each increase in n, the recursive function calls were added, increasing the number of jumps and parameters to be set up and located (2).