Week 11 — November 22, 2018

Homework

- Read pp. 363-365 ("Functions That Call Themselves: Recursion") in "Beginning C"
- Find the worst-case time complexity for operations (access, search, insertion, and deletion) on arrays and linked lists using the Big-O Cheat Sheet
- Catch up on unfinished exercises

Exercises

1. Recall that the Fibonacci numbers are defined by the recurrence relation

   $$F_n = F_{n-1} + F_{n-2}, \quad n = 2, 3, 4, \ldots$$

   with $F_0 = 0$ and $F_1 = 1$. The Fibonacci numbers can also be expressed in terms of the "golden ratio" $\varphi = (1 + \sqrt{5})/2$ as

   $$F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}, \quad n = 0, 1, 2, 3, \ldots$$

   Analyze the following recursive function that computes the nth Fibonacci number:

```
unsigned long fibonacci(unsigned long n) {
  if ( n == 0 )
     return 0;
  else if ( n == 1 )
     return 1;
  else
     return fibonacci(n-1) + fibonacci(n-2);
}
```

   How many times is the function called if $n = 5$? Write a short program that counts the number of function calls (e.g., using a global variable or static variable).

   Hint: If you let f_n denote the number of function calls required to evaluate F_n, then $f_n = f_{n-1} + f_{n-2} + 1$ for $n \geq 2$ and $f_0 = f_1 = 1$.

2. What is the time complexity of this recursive implementation? Write a short program to verify your result.

   Hint: The time required for each function call is $O(1)$ (i.e., upper bounded by a constant), so it suffices to look at the number of function calls. Is the growth rate linear (i.e., $O(n)$), polynomial (i.e., $O(n^k)$ where the power k is positive and constant), or exponential (i.e., $O(2^n)$)?

3. What is the space complexity of the recursive implementation of the Fibonacci function?

   Hint: What is the maximum number of function calls on the program stack? Write a program that finds the *recursion depth* using global variables. For example, you may use one global variable that keeps track of the depth and another global variable to keep track the *maximum* depth. You may use the following template:

```
int depth=0, max_depth=0;

<type> recursive_func(<type> arg) {
    <type> return_value;
    depth += 1;
    max_depth = depth > max_depth ? depth : max_depth;

    /* body of recursive function */

    depth -= 1;
    return return_value;
}
```

4. Rewrite the Fibonacci function so that you avoid recursive function calls. The return type should be an `unsigned long`. What is the time complexity of your implementation? What is the space complexity?

5. Write a program that measures the CPU time for the recursive variant of the Fibonacci function and your modified version for different values of n. Does the measured time complexity reflect the "Big-O" complexity that you derived in previous exercises?

6. Find the largest Fibonacci number that can be represented as an `unsigned long`.

   Hint: Keep in mind that an `unsigned long` may be either 4 bytes or 8 bytes, depending on your system.