CIS 23: Data Structures and Algorithms

Homework 6
Prof. Sana Vaziri

Cody Vig

Problem 1

In the following function:

```
int recFunc(int num)                    // Line 1
{                                       // Line 2
    if (num == 0)                       // Line 3
        return 0;                       // Line 4
    else if (num < 0)                   // Line 5
        return (-num);                  // Line 6
    else                                // Line 7
        return (num - recFunc(num - 5)); // Line 8
}                                       // Line 9
```

1. Identify the base case(s).
2. Identify the recursive call.
3. What is the depth of the recursion for recFunc(16)?

Solution

1. There are two base cases here. If `num == 0` we return zero, and if `num < 0` we return `-num`. These are the base cases because they can be computed immediately without recursive function calls.
2. The recursive call is written on Line 8. This is where we call the function within its own definition in such a way that brings the argument closer to the base case.
3. Via Line 8, the following is the sequence of function calls required by `recFunc(16)`:

`recFunc(16)` → `recFunc(11)` → `recFunc(6)` → `recFunc(1)` → `recFunc(-4)`

and `recFunc(-4) == 4` by the base case. Hence the depth of the recursion is 4, since there are four function calls within the program.

Problem 2

What is the overhead associated with the execution of a recursive function both in terms of memory space and computer time?

Solution

In general, recursive algorithms are actually much less efficient than their corresponding iterative algorithms. Each new recursive function call requires the creation of a namespace for that function, so the required memory space scales exponentially with the number of recursions. Beyond that, each new function call requires memory to be allocated by the calling routine when the control shifts from one function to another, slowing down computer time.

With modern computers, the loss of computational efficiency associated with recursive functions is almost entirely unnoticeable in most situations, and so the distinction is of little practical importance. From my understanding, recursion is often used over iteration because it is generally easier to read and understand, and it is often easier to write when dealing with certain data structures like trees. Recursion can be seriously optimized by caching (though in fairness, so can iterative methods) if the relevant data you are working with allows for it.