CSC236 – Problem Set 6

There are two components of this problem set: a preliminary question designed to check your understanding of the basic topics covered this week, and a set of more challenging questions designed to make you think critically about the material and apply it in new contexts. Get in the habit of starting work early – the less time you give yourself, the most stressed you’ll find yourself each week!

Caution: you must submit two separate files in two separate locations on MarkUs, one for the Preliminary and one for the Challenge.

To avoid suspicions of plagiarism, clearly state any resources (people, print, electronic) outside of your group, the course notes, and the course staff, you consulted at the beginning of your assignment submission.

---

**Preliminary: due February 25, 2014 8:00 pm**

This question is an opportunity for you to check your understanding of the topics and practice writing formal solutions. This is a valuable learning opportunity – if you see that you’re at a loss, get help quickly!

Your goal should not be to get the right answer, but to convince the marker that you know what you’re doing. This question is marked on the following 3-point scale:

| 3: You’ve mastered this topic | 1: You don’t really know what you’re doing |
| 2: You’re almost there, but missing something | 0: You didn’t submit/had absolutely no clue |

This question must be completed INDIVIDUALLY.

Consider the following code for a recursive exponentiation function.

```python
1 def rec_exp(a, b):
2     if b == 0:
3         return 1
4     else if b mod 2 == 0:
5         x = rec_exp(a, b / 2)
6         return x * x
7     else:
8         x = rec_exp(a, (b-1) / 2)
9         return x * x * a
```

State preconditions and postconditions for this algorithm (your postcondition must involve exponentiation). Then, prove that your algorithm is correct according to your specifications.

---

**Challenges: due March 1, 2014 noon**

Answer each question completely, always justifying your claims and reasoning. Your solution will not just be graded on correctness, but on its clarity as well. Technically correct answers that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets].

You may work in groups of up to THREE to complete these questions.

1. When designing programs, we can often take a top-down approach: starting with a high-level idea of an algorithm we’d like to write to solve a problem, we use specifications (pre- and postconditions) for individual steps to argue why our algorithm should be correct. Then after being satisfied that our main algorithm is correct, all we need to do is implement the steps according the specifications we set.

   (a) [5] Here’s a simple example. Consider the following code for (a variant of) mergesort, with the natural pre- and postconditions.
def mergesort3(A):
    '''
    Pre: A is a list of numbers; its length is a power of 3.
    Post: Returns a list with the same elements in A, in non-decreasing order.
    '''
    if len(A) == 1:
        return A
    else:
        m = len(A) // 3
        L1 = mergesort(A[0..m-1])
        L2 = mergesort(A[m..2*m-1])
        L3 = mergesort(A[2*m..len(A)-1])
        return merge3(L1, L2, L3)

Write pre- and postconditions for the merge3 function. Then, prove that mergesort3 is correct according to the given specifications, assuming that merge3 is correct (according to your provided specifications).

Note that you are not required to provide an implementation of merge3; treat it as a black box that would be implemented later.

(b) [6] Here’s a more unfamiliar example. A palindrome is a string that is equal to its reversal: examples are ‘a’, ‘wow’, and ‘abcdeedcba’. Consider the following algorithm.

def longestPalindrome(s):
    '''
    Pre: s is a non-empty string
    Post: returns the longest palindrome that is a substring of s.
    If there is more than one palindrome in s of maximum length,
    return <YOU FIGURE THIS OUT>.
    '''
    if len(s) == 1:
        return s
    else:
        palindrome1 = longestPalindrome(s[1..len(s)-1])
        palindrome2 = firstPalindrome(s)
        if len(palindrome1) > len(palindrome2):
            return palindrome1
        else:
            return palindrome2

You have two tasks here, which should be accomplished together.

- As is often the case in real life, the client (David) has failed to consider an edge case in the provided specification.
  By studying the given the algorithm, you must complete the specification.
- Once again, write pre- and postconditions for the helper function firstPalindrome, and then prove that longestPalindrome(s) is correct, assuming that firstPalindrome is correct.

Note that you cannot prove that longestPalindrome is correct without completing its specification; but in order to complete its specification, you can carefully trace through the code, as you would when actually proving correctness (so the two tasks can be accomplished together).
2. You saw on Problem Set 4 that the naive recursive implementation of computing the Fibonacci numbers run in exponential time.

(a) [3] Give a recursive implementation here that runs in linear time. Your algorithm must be recursive, and may not contain any loops. It also may not use the closed form expression for the Fibonacci numbers from the Lecture Notes.

How to do this? Your algorithm does not need to return just the $n$-th Fibonacci number. This exercise is for you to practice the principle of “doing more is easier.” Write a recursive function that returns a bit more than just the $n$-th Fibonacci number, but from which the $n$-th Fibonacci number can be extracted in constant time, or even in a single operation.

Then write the main Fibonacci function, which uses your recursive code as a helper function. The code template is given below (including specifications for the main function).

```
def fib_helper(...):
    '''
    Pre: ???
    Post: ???
    '''
    <YOUR CODE GOES HERE>
    <Return something more than just f_n>

def main_fib(...):
    '''
    Pre: n is a natural number >= 1
    Post: returns the n-th Fibonacci number
    '''
    # result = call fib_helper(...)  
    # from result, extract f_n and return, IN CONSTANT TIME 
    <YOUR CODE GOES HERE>
```

For full marks your code should run in $O(n)$ (linear) time, although you do not need to analyse the runtime (it’s good practice, though).

(b) [2] Write pre and postconditions for your helper function from part (a).

(c) [3] First prove that your helper function is correct, according to your specifications.

(d) [1] Prove that your main_fib is correct.