Math F307: Chapter 3 Review Questions

October 23, 2004

The problems listed here are intended to help you study the material from Chapter 3. The later sections of the chapter, in particular, didn't have a great selection of problems. So I've emphasized those problems here. That doesn't mean that there won't be any problems from early parts of the chapter, just that the selection of questions from the text are better. There is no guarantee that because a question is asked here that it will be on the midterm, nor is there a guarantee that every problem from the midterm is represented on this list.

1. Compute $gcd(18, 60)$ with the Euclidean algorithm.

2. Write a recursive algorthm that computes a_n where we define $a_1 = 1$, $a_2 = 2$, $a_3 = 3$, and $a_n = a_{n-1} + a_{n-2} + a_{n-3}$ for $n > 3$. Use induction to prove that your algorithm correctly computes a_n.

3. Write an *iterative* algorithm that computes the same sequence a_n as in the previous question.

4. Compute f_7, the 7^{th} Fibonacci number.

5. Use induction to prove that $1 + f_2 + f_4 + \cdots + f_{2(n-1)} = f_{2n-1}$, where f_k is the k^{th} Fibonacci number.

6. Prove that if $f(x) = O(h(x))$ and $g(x) = O(h(x))$ then $f(x) + g(x) = O(h(x))$.

7. Prove that $\log_{10}(x) = O(\log_2(x))$.

8. Prove that $x^3 = O(x^4)$ but x^4 is not $O(x^3)$.

9. Prove that $f(x) = \Theta(g(x))$ if and only if $f(x) = O(g(x))$ and $g(x) = O(f(x))$.

10. Prove that if $f(x) = O(g(x))$ and $g(x) = O(h(x))$ then $f(x) = O(h(x))$.

11. Prove that $\frac{2x^2+1}{x-1} = O(x)$.

12. Here is an algorithm for determining if the matrix of a relation is antisymmetric.

```
procedure is_anti_symm(A,n)
  for i := 1 to n do
    for j := i+1 to n do
      if a[i][j]=1 and a[j][i]=1 return false
  return true
end is_anti_symm
```

We will use n to denote the size of the input. Find a big-O estimate for the number of times the line `if a[i][j]=1 and a[j][i]=1 return false` is exectued in the worst case. ustify your answer.

13. Here is a slow, recursive algorithm for computing Fibonacci numbers.

```
procedure slow_fib(n)
  if n = 1 then return 1
  if n = 2 then return 2
  return slow_fib(n-1) + slow_fib(n-2)
end slow_fib
```

Use induction to prove that if $n \geq 3$ that the algorithm requires at least 2^{n-2} additions. Hint: in your basis step, show directly that `slow_fib(3)` requires 1 addtion and `slow_fib(4)` requires 2 additions.

Write a big-Ω expression for the time complexity of this algorithm.

Challenge: Prove that this algorithm requires exactly $f_{k-1} - 1$ additions to compute f_k for $k >= 3$.

14. Here is a faster, iterative algorithm for computing Fibonacci numbers.

```
procedure fast_fib(n)
  a := 0
  b := 1
  for i = 1 to n do
  begin
    c:= a
    a:= b
    b:= c + b
  end
  return b
end fast_fib
```

Find a big Θ estimate for the number of additions performed by this algorithm.