4005-800 Algorithms

Homework 1

Christopher Wood

March 23, 2012

PROBLEM 1.

Solution. TODO: explain reasoning here...

$$2^{2^{n+1}}$$

$$2^{2^n}$$

$$(n+1)!$$

$$n!$$

$$TODO : continue$$

PROBLEM 2-a. *Using the definition of O, prove that $n = O(n^2)$.*

Solution. If $n \geq 1$, then $n^2 \geq n$. Further, $0^2 \geq 0$. Therefore, $n^2 \geq n$ for any $n \in \mathbb{N}$. Thus, $cn^2 \geq n$ when $n \geq 0$ and $c \geq 1$. Finally, by definition, this means that $n \in O(n^2)$, or simply $n = O(n^2)$.

PROBLEM 2-b. *Using the definition of O, prove that $n^k = O(n^{k'})$ if $k \leq k'$.*

Solution.

If $k \leq k'$, then it follows that $n^k \leq n^{k'}$ for any $n \in \mathbb{N}$. Thus, for constants $c \geq 1$, it is true that $n^k \leq n^{k'} \leq cn^{k'}$, or simply $n^k \leq cn^{k'}$, for all $n \geq 0$. Therefore, by definition, this means that $n^k \in O(n^{k'})$, or simply $n^k = O(n^{k'})$.

PROBLEM 3. *Write a function fib that implements the recurrence relation for the Fibonacci numbers. What is the smallest n such that you notice fib running slowly?*

Solution. The source code for the fib routine (written in Python) is listed below. It is also included in the electronic submission.

```
1  def fib(n):
2          if (n == 0):
3                  return 0
4          elif (n == 1):
5                  return 1
6          else:
```

```
7          return (fib(n - 1) + fib(n - 2))
```

The smalest value of n that starts to yields long execution times is $n = 32$.

TODO: explain the time complexity of this guy by solving with second order nonlinear homogeneous equations!

PROBLEM 4-a. *Prove using the strong form of mathematical induction that for any $n \in \mathbb{N}$ if $n > 1$ then $f(n; a, b) = f(n - 1; a, b) + f(n - 2; a, b)$.*

Solution. TODO

PROBLEM 4-b. *Prove using the strong form of mathematical induction that for any $n \in \mathbb{N}$, $F_n = f(n; 0, 1)$.*

Solution. Based on the results from the previous problem, $f(n)$ can depend on both $f(n - 1)$ and $f(n - 2)$. Therefore, we have two base cases to cover, as shown below.

Base ($n = 0$)
By definition, $F_0 = 0$, and $f(0; 0, 1) = 0$. Thus, $F_0 = f(0; 0, 1)$.
Base ($n = 1$)
By definition, $F_1 = 1$, and $f(1; 0, 1) = 1$. Thus, $F_1 = f(1; 0, 1)$.
Induction ($n > 1$)
Assume that $F_k = f(k; 0, 1)$ for some $k \in \mathbb{N}$ such that $2 \le k < n$. We now show that $F_n = f(n; 0, 1)$. This result is described below.

$$
\begin{aligned}
F_n &= F_{n-1} + F_{n-2} \text{ (by definition)} \\
&= f(n - 1; 0, 1) + f(n - 2; 0, 1) \text{ (by induction)} \\
&= f(n; 0, 1) \text{ (by definition from problem 4-a)}
\end{aligned}
$$

Thus, $F_n = f(n; 0, 1)$, as desired.

PROBLEM 5. *Write a function $fibIt$ that implements f. Does $fibIt$ also run slowly on the value of n that you found made fib run slowly?*

Solution. The source code for the $fibIt$ routine (written in Python) is listed below. It is also included in the electronic submission.

```
1  def fibIt(n, a, b):
2          if (n == 0):
3                  return a
```

```
4              elif (n == 1):
5                      return b
6              else:
7                      return fibIt(n − 1, b, a + b)
```

Based on empirical observations, $fibIt$ **does not** run slowly on the same value of n that made fib run slowly. In fact, $fibIt$ will execute in a reasonable amount of time up to the point where the size of the recursive call stack is too large for the system to maintain in memory.

TODO: explain why (why oh why is this the case? - put it in a complexity class?)