

4005-800 ALGORITHMS

HOMEWORK 1

Christopher Wood

March 27, 2012

PROBLEM 1.

Solution.

The following table ranks the provided functions based on their order of growth in descending order and groups asymptotically tight functions $f(n)$ and $g(n)$ together in an equivalence class if and only if $f(n) = \Theta(g(n))$.

Rank	Functions
1	$2^{2^{n+1}}$
2	2^{2^n}
3	$(n+1)!$
4	$n!$
5	e^n
6	$n * 2^n$
7	2^n
8	$(\frac{3}{2})^n$
9	$n^{\lg \lg n}, (\lg n)^{\lg n}$
10	$(\lg n)!$
11	n^3
12	$n^2, 4^{\lg n}$
13	$\lg(n!), n \lg n$
14	$n, 2^{\lg n}$
15	$(\sqrt{2})^{\lg n}$
16	$2^{\sqrt{2} \lg n}$
17	$\lg^2 n$
18	$\ln n$
19	$\sqrt{\lg n}$
20	$\ln \ln n$
21	$2^{\lg^* n}$
22	$\lg^*(\lg n), \lg^* n$
23	$\lg(\lg^* n)$
24	$n^{\frac{1}{\lg n}}, 1$

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 $c \geq 1$. 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 smallest value of n that starts to yield long execution times is $n = 31$.

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. Based on the definition for f and the problem constraints, we need only consider $n = 2$ as the base case for induction since it is the first valid value of n for which f is true.

Base ($n = 2$)

By definition, we know that $f(2; a, b) = f(1; b, a + b) = (a + b)$. Further, by definition we know that $f(1; a, b) + f(0; a, b) = b + a = (a + b)$. Thus, $f(2; a, b) = f(1; a, b) + f(0; a, b)$, as desired.

Induction ($n > 2$)

Assume that $f(k; a, b) = f(k - 1; a, b) + f(k - 2; a, b)$ for some $k \in \mathbb{N}$ such that $3 \leq k < n$. We now show that $f(n; a, b) = f(n - 1; a, b) + f(n - 2; a, b)$. This result is described below.

$$\begin{aligned}
f(n; a, b) &= f(n-1; b, a+b) \text{ (by definition)} \\
&= f(n-2; b, a+b) + f(n-3; b, a+b) \text{ (by induction)} \\
&= f(n-1; a, b) + f(n-2; a, b) \text{ (by definition of } f)
\end{aligned}$$

Thus, $f(n; a, b) = f(n-1; a, b) + f(n-2; a, b)$, as desired.

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 $2 \leq k < n$. 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, at which point the Python interpreter throws a run-time exception and exits abnormally.