*Introduction to Algorithms* September 8, 2004
Massachusetts Institute of Technology 6.046J/18.410J
Professors Piotr Indyk and Charles E. Leiserson Handout 6

# Problem Set 1

*Reading:* Chapters 1-4, excluding Section 4.4.

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered in the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct which are *which are described clearly*. Convoluted and obtuse descriptions will receive low marks.

---

**Exercise 1-1.** Do Exercise 2.3-7 on page 37 in CLRS.

**Exercise 1-2.** Do Exercise 3.1-3 on page 50 in CLRS.

**Exercise 1-3.** Do Exercise 3.2-6 on page 57 in CLRS.

**Exercise 1-4.** Do Problem 3-2 on page 58 of CLRS.

---

**Problem 1-1. Properties of Asymptotic Notation**

Prove or disprove each of the following properties related to asymptotic notation. In each of the following assume that $f$, $g$, and $h$ are asymptotically nonnegative functions.

**(a)** $f(n) = O(g(n))$ and $g(n) = O(f(n))$ implies that $f(n) = \Theta(g(n))$.

**(b)** $f(n) + g(n) = \Theta(\max(f(n), g(n)))$.

**(c)** Transitivity: $f(n) = O(g(n))$ and $g(n) = O(h(n))$ implies that $f(n) = O(h(n))$.

**(d)** $f(n) = O(g(n))$ implies that $h(f(n)) = O(h(g(n))$.

**(e)** $f(n) + o(f(n)) = \Theta(f(n))$.

**(f)** $f(n) \neq o(g(n))$ and $g(n) \neq o(f(n))$ implies $f(n) = \Theta(g(n))$.

**Problem 1-2. Computing Fibonacci Numbers**

The Fibonacci numbers are defined on page 56 of CLRS as

$$\begin{aligned} F_0 &= 0 , \\ F_1 &= 1 , \\ F_n &= F_{n-1} + F_{n-2} \quad \text{for } n \geq 2 . \end{aligned}$$

In Exercise 1-3, of this problem set, you showed that the $n$th Fibonacci number is

$$F_n = \frac{\phi^n - \widehat{\phi}^n}{\sqrt{5}} ,$$

where $\phi$ is the golden ratio and $\widehat{\phi}$ is its conjugate.

A fellow 6.046 student comes to you with the following simple recursive algorithm for computing the $n$th Fibonacci number.

```
FIB(n)
 1  if n = 0
 2     then return 0
 3  elseif n = 1
 4     then return 1
 5  return FIB(n − 1) + FIB(n − 2)
```

This algorithm is correct, since it directly implements the definition of the Fibonacci numbers. Let's analyze its running time. Let $T(n)$ be the worst-case running time of FIB$(n)$.[1]

**(a)** Give a recurrence for $T(n)$, and use the substitution method to show that $T(n) = O(F_n)$.

**(b)** Similarly, show that $T(n) = \Omega(F_n)$, and hence, that $T(n) = \Theta(F_n)$.

Professor Grigori Potemkin has recently published an improved algorithm for computing the $n$th Fibonacci number which uses a cleverly constructed loop to get rid of one of the recursive calls. Professor Potemkin has staked his reputation on this new algorithm, and his tenure committee has asked you to review his algorithm.

[1] In this problem, please assume that all operations take unit time. In reality, the time it takes to add two numbers depends on the number of bits in the numbers being added (more precisely, on the number of memory words). However, for the purpose of this problem, the approximation of unit time addition will suffice.

```
FIB′(n)
 1  if n = 0
 2     then return 0
 3  elseif n = 1
 4     then return 1
 5   sum ← 1
 6   for k ← 1 to n − 2
 7        do  sum ← sum + FIB′(k)
 8  return sum
```

Since it is not at all clear that this algorithm actually computes the $n$th Fibonacci number, let's prove that the algorithm is correct. We'll prove this by induction over $n$, using a loop invariant in the inductive step of the proof.

**(c)** State the induction hypothesis and the base case of your correctness proof.

**(d)** State a loop invariant for the loop in lines 6-7. Prove, using induction over $k$, that your "invariant" is indeed invariant.

**(e)** Use your loop invariant to complete the inductive step of your correctness proof.

**(f)** What is the asymptotic running time, $T'(n)$, of FIB′$(n)$? Would you recommend tenure for Professor Potemkin?

**Problem 1-3. Polynomial multiplication**

One can represent a polynomial, in a symbolic variable $x$, with degree-bound $n$ as an array $P[0\,..\,n]$ of coefficients. Consider two linear polynomials, $A(x) = a_1x + a_0$ and $B(x) = b_1x + b_0$, where $a_1$, $a_0$, $b_1$, and $b_0$ are numerical coefficients, which can be represented by the arrays $[a_0, a_1]$ and $[b_0, b_1]$, respectively. We can multiply $A$ and $B$ using the four coefficient multiplications

$$\begin{aligned} m_1 &= a_1 \cdot b_1 \,, \\ m_2 &= a_1 \cdot b_0 \,, \\ m_3 &= a_0 \cdot b_1 \,, \\ m_4 &= a_0 \cdot b_0 \,, \end{aligned}$$

as well as one numerical addition, to form the polynomial

$$C(x) = m_1x^2 + (m_2 + m_3)x + m_4 \,,$$

which can be represented by the array

$$[c_0, c_1, c_2] = [m_4, m_3 + m_2, m_1] \,.$$

**(a)** Give a divide-and-conquer algorithm for multiplying two polynomials of degree-bound $n$, represented as coefficient arrays, based on this formula.

**(b)** Give and solve a recurrence for the worst-case running time of your algorithm.

**(c)** Show how to multiply two linear polynomials $A(x) = a_1 x + a_0$ and $B(x) = b_1 x + b_0$ using only *three* coefficient multiplications.

**(d)** Give a divide-and-conquer algorithm for multiplying two polynomials of degree-bound $n$ based on your formula from part (c).

**(e)** Give and solve a recurrence for the worst-case running time of your algorithm.