COMP 271 Design and Analysis of Algorithms
2003 Spring Semester
Questions for Second Tutorial – 21 Feb, 2003

Problem 1.
```
float unknown(int n)
{
 if (n <= 1)
      return(1.0);
 else
      return(unknown(n-1) + unknown(n-2));
}
```

(a) What does the above function compute?

(b) Executing the function for $n = 6$ results in the function being recursively invoked with the argument $n = 1, 2, 3, 4$, and 5. Draw a recurrence tree to illustrate this fact. How many times is `unknown(i)` executed for $1 <= i <= 5$?

(c) How many additions are performed to compute `unknown(6)`?

(d) Assuming that each addition takes constant time, write a recurrence relation for the running time of `unknown(n)`.

(e) Show that the time to compute `unknown(n)` is at least as bad as $\Omega(1.5^n)$. (Hint: Use induction.)

(f) Using the result of part (e), give a lower bound on the time it would take to compute `unknown(100)` on a computer that can do 1 million additions per second? (For this question, just consider additions in determining the running time; thus you may ignore costs such as the cost of testing whether $n \leq 1$ in the *if* statement.)

(g) Can you suggest a more efficient way of computing the same function? How long does it take your program to compute `unknown(n)`? Is your algorithm faster than the above recursive program? If yes, what design idea did you exploit to achieve this speedup?

Problem 2. (modified problem 9.3-1, p192, CLRS).
Assume that you are given a "black-box" worst case linear time median finding algorithm. Show how *Quicksort* can be modified to run in $O(n \log n)$ *worst case* time.

Problem 3. (problem 9.3-5, p192, CLRS).
Assume again that you are given a "black-box" worst case linear time median finding algorithm. Give a simple linear-time algorithm that solves the selection problem for an arbitrary order statistic. That is, given k, your algorithm should find the k smallest item.

Problem 4.

You have just been hired as the quality-control engineer for a company that makes coins. The coins must all have identical weight. You are given a set of n coins and are told that *at most one* (possibly none) of the n coins is lighter than the others. Your task is to develop an efficient test procedure to determine which of the n coins is defective or report that none is defective. To do this test you have a scale. For each measurement you place some of the coins on the left side of the scale and some of the coins on the right side. The scale indicates either that (1) the left side is heavier, (2) the right side is heavier or (3) both subsets have the same weight. It does not indicate how much heavier or lighter.

(A) Design an algorithm for solving this problem that works in $\log_2 n + c$ time, for some constant c (try to make c as small as possible).
Hint: try a divide and conquer approach.

(B) Design an algorithm for solving this problem that works in $\log_3 n + c$ time, for some constant c (try to make c as small as possible).

(C) The problem now changes in that the defective coin, if it exists, may be lighter or *heavier* than the other coins. Modify the algorithm from part (A) or (B) to work in this case. How fast is your algorithm?