# CS 473G: Graduate Algorithms, Spring 2007
# Homework 2

Due Tuesday, February 20, 2007

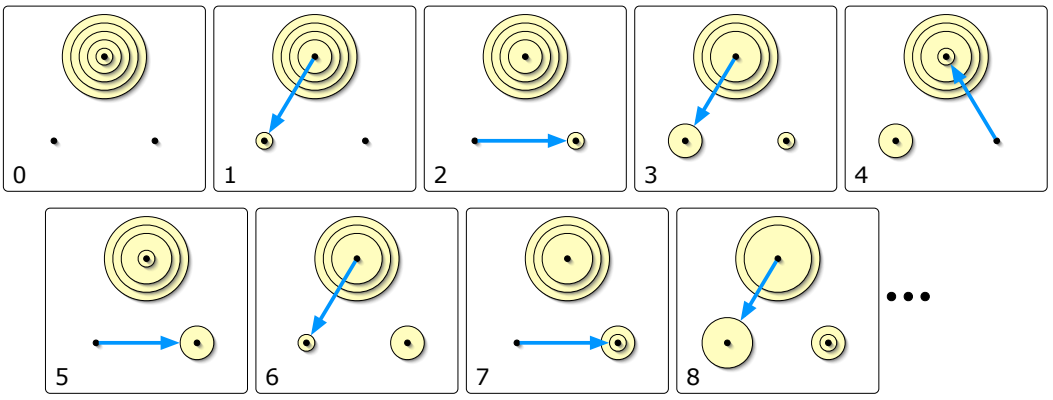
---

Remember to submit **separate, individually stapled** solutions to each problem.

As a general rule, a complete full-credit solution to any homework problem should fit into two typeset pages (or five hand-written pages). If your solution is significantly longer than this, you may be including too much detail.

---

1. Consider a restricted variant of the Tower of Hanoi puzzle, where the three needles are arranged in a triangle, and you are required to move each disk *counterclockwise*. Describe an algorithm to move a stack of $n$ disks from one needle to another. *Exactly* how many moves does your algorithm perform? To receive full credit, your algorithm must perform the minimum possible number of moves. *[Hint: Your answer will depend on whether you are moving the stack clockwise or counterclockwise.]*




A top view of the first eight moves in a counterclockwise Towers of Hanoi solution

*2. You find yourself working for The Negation Company ("We Contradict Everything... Not!"), the world's largest producer of multi-bit Boolean inverters. Thanks to a recent mining discovery, the market prices for amphigen and opoterium, the key elements used in AND and OR gates, have plummeted to almost nothing. Unfortunately, the market price of inverton, the essential element required to build NOT gates, has recently risen sharply as natural supplies are almost exhausted. Your boss is counting on you to radically redesign the company's only product in response to these radically new market prices.

   Design a Boolean circuit that inverts $n = 2^k - 1$ bits, using only $k$ NOT gates but *any* number of AND and OR gates. The input to your circuit consists of $n$ bits $x_1, x_2, \ldots, x_n$, and the output consists of $n$ bits $y_1, y_2, \ldots, y_n$, where each output bit $y_i$ is the inverse of the corresponding input bit $x_i$. *[Hint: Solve the case $k = 2$ first.]*

3. (a) Let $X[1..m]$ and $Y[1..n]$ be two arbitrary arrays. A *common supersequence* of $X$ and $Y$ is another sequence that contains both $X$ and $Y$ as subsequences. Give a simple recursive definition for the function $scs(X, Y)$, which gives the length of the *shortest* common supersequence of $X$ and $Y$.

   (b) Call a sequence $X[1..n]$ *oscillating* if $X[i] < X[i+1]$ for all even $i$, and $X[i] > X[i+1]$ for all odd $i$. Give a simple recursive definition for the function $los(X)$, which gives the length of the longest oscillating subsequence of an arbitrary array $X$ of integers.

   (c) Call a sequence $X[1..n]$ of integers *accelerating* if $2 \cdot X[i] < X[i-1] + X[i+1]$ for all $i$. Give a simple recursive definition for the function $lxs(X)$, which gives the length of the longest accelerating subsequence of an arbitrary array $X$ of integers.

   Each recursive definition should translate directly into a recursive algorithm, *but you do **not** need to analyze these algorithms.* We are looking for correctness and *simplicity*, not algorithmic efficiency. Not yet, anyway.

4. Describe an algorithm to solve 3SAT in time $O(\phi^n \operatorname{poly}(n))$, where $\phi = (1+\sqrt{5})/2 \approx 1.618034$. *[Hint: Prove that in each recursive call, either you have just eliminated a pure literal, or the formula has a clause with at most two literals. What recurrence leads to this running time?]*

5. (a) Describe an algorithm that determines whether a given set of $n$ integers contains two distinct elements that sum to zero, in $O(n \log n)$ time.

   (b) Describe an algorithm that determines whether a given set of $n$ integers contains *three* distinct elements that sum to zero, in $O(n^2)$ time.

   (c) Now suppose the input set $X$ contains $n$ integers between $-10000n$ and $10000n$. Describe an algorithm that determines whether $X$ contains three ***distinct*** elements that sum to zero, in $O(n \log n)$ time.

   For example, if the input set is $\{-10, -9, -7, -3, 1, 3, 5, 11\}$, your algorithm for part (a) should return TRUE, because $(-3) + 3 = 0$, and your algorithms for parts (b) and (c) should return FALSE, even though $(-10) + 5 + 5 = 0$.