CSC 545 **Homework 1 (version 1)** **Spring 2015**

This homework is due Thursday, February 5, at the start of class. The questions are drawn from Chapters 2, 3, and 4 of the text and from the material given in class on *asymptotics*, *recurrences*, and *divide-and-conquer*.

The homework is worth a total of 100 points. When questions with several parts do not specify the points for each part, each part has equal weight.

For questions that ask you to design an algorithm, you should:

(i) describe an algorithm for the problem using *prose* and *pictures*,
(ii) argue that your algorithm is correct, and
(iii) analyze the running time of your algorithm.

Providing pseudocode is not necessary, but do explain the *ideas* behind your algorithm and its correctness. Only give high-level pseudocode if it aids the time analysis of your algorithm.

Remember to write on just one side of a page, do not use scrap paper, put your answers in the correct order, and staple your pages together. If you can't solve a problem, state this, and write only what you know to be correct. Neatness and conciseness count.

(1) **(Asymptotic relationships)** (15 points) Prove each of the following statements.

(a) (5 points) Let $f(n) = \sum_{0 \le i \le k} a_i \, n^i$ be a degree-k polyomial with $a_k > 0$. Then

$$f(n) \;=\; \Theta(n^k).$$

(Hint: Take a limit.)

(b) (5 points) For all constants b and c with $b > 1$,

$$n^c \;=\; o(b^n).$$

(Hint: Take a limit and repeatedly use L'Hospital's Rule from calculus.)

(c) (5 points) For all constants $\epsilon > 0$ and k,

$$(\lg n)^k \;=\; o(n^\epsilon).$$

(Hint: Use the rule on composition of functions from the class handouts.)

(2) **(bonus) (Superpolynomial growth)** (10 points) Prove or disprove the following.

> **Conjecture** *Let $f(n)$ be an asymptotically positive function. Suppose $f(n) = \omega(n^c)$ for every c. Then $f(n) = \Omega(b^n)$ for some $b > 1$.*

(Note: The above conjecture says that if f is growing faster than a polynomial, then it is growing exponentially fast.)

(3) **(Solving recurrences)** (15 points) Derive a Θ-bound on the solution to each of the following recurrences. Use the Master Theorem (including the specializations given in class). Do not worry about taking floors or ceilings of arguments.

(a) $T(n) \;=\; 4\,T(n/2) \;+\; n^2\sqrt{n}.$
(b) $T(n) \;=\; 3\,T(n/2) \;+\; n \lg n.$
(c) $T(n) \;=\; 2\,T(n/2) \;+\; n \lg n.$

(4) **(Finding the missing integer)** (15 points) Suppose you have a 2-dimensional array $A[1:n, 1:k]$ of bits, where $k = \Theta(\log n)$. Each row of A represents an integer in the range $[0, n]$, written in binary. All integers 0, 1, ..., n are represented by the rows of A, *except one*, which is missing.

Using divide-and-conquer, design an algorithm that finds the missing integer in $\Theta(n)$ time, assuming the only operation you can perform on A is to read a bit $A[i, j]$. Reading a bit takes $\Theta(1)$ time.

(Note: Since the array contains $\Theta(n \log n)$ entries, you cannot examine all the entries and run in $O(n)$ time. Also keep in mind that copying a row of A will take $\Theta(\log n)$ time, which is expensive.)

(5) **(Minimum positive-sum subarray)** (25 points) In this variation of the Maximum-Sum Subarray Problem, you are given a one-dimensional array $A[1:n]$ of positive and negative numbers, and you are asked to find a subarray $A[i:j]$ such that the sum of its elements is (1) strictly greater than zero, and (2) minimum. In other words, you want to find a subarray of smallest positive sum.

(a) (25 points) Using the *divide-and-conquer* strategy, design a $\Theta(n \log^2 n)$ time algorithm for this problem.

(Hint: Remember that n numbers can be sorted in $\Theta(n \log n)$ worst-case time. To analyze the time for your algorithm, you may need to use the extended form of the Master Theorem for divide-and-conquer recurrences that was given in class.)

(b) **(bonus)** (10 points) Using an *incremental strategy*, design an algorithm that runs in $O(n \log n)$ time. The incremental strategy proceeds by solving a subproblem of size k, adding one element, and updating the solution to solve a problem of size $k+1$.

(Hint: You may find a balanced search tree useful.)

(6) **(Identifying good chips by pairwise tests)** (30 points) Suppose we have n computer chips, some of which are good and some of which are faulty. We want to identify which chips are good by performing pairwise tests. In a pairwise test, we choose a pair of chips, and have each chip test the other. (So in a pairwise test of chips A and B, chip A tests B and reports the result, and chip B also tests A and reports the result.) Good chips always report correctly whether the tested chip is good or faulty, but faulty chips can report anything.

(a) (5 points) Show that if at most $n/2$ chips are good, then no algorithm can correctly identify the good chips by pairwise tests.

(Hint: Show how to construct a set of test results that will fool any procedure that claims to solve the problem.)

(b) (20 points) Suppose more than $n/2$ chips are good. Show how to reduce the problem of finding 1 good chip from among the n chips to a problem of half the size, using $\Theta(n)$ pairwise tests.

(c) (5 points) Using Part (b), design an algorithm that correctly identifies all the good chips using $\Theta(n)$ pairwise tests, assuming that more than $n/2$ chips are good.

Note that Problems (2) and (5)(b) are *bonus* questions. They are not required, and their points are not added to regular points.