CS161 HOMEWORK 2

SUBMIT ON GRADESCOPE (OUT: 6/30/2016 (THURS), DUE: 7/7/2016 (THURS))

(1) **Recurrence Party** Please show your work.
   (a) Consider the recurrence $T(n) = T(n/2) + T(n/4) + T(n/8) + n$, where $T(1) = 1$. You may assume that n is a power of 2.
      (i) Use a recursion tree to generate a "guess" for the runtime $T(n)$. The guess should be in the form of a Θ bound.
      (ii) Use the substitution method to prove that your guess reflects a correct upper bound.
      (iii) Prove that the bound is tight (i.e. that it is a Θ bound as opposed to just an O bound).
   (b) Use the master theorem to solve for $T(n)$ in the following recurrences.
      (i) $T(n) = 4T(\lceil\sqrt{n}\rceil) + 2(\log n)^2$.
      (ii) $T(n) = 4T(\lfloor n/2 \rfloor) + n^2\sqrt{n}$.
   (c) Determine H_n, the length of the n^{th} Hilbert curve (suppose $H_1 = 1 + 1 + 1 = 3$):

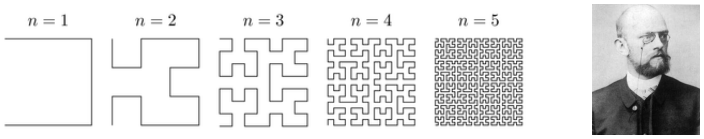

FIGURE 1. The first 5 Hilbert curves (from left to right) and the creator, David Hilbert

What is $\lim_{n\to\infty} H_n$?

   (d) Determine A_n, the area of the n^{th} Sierpinski carpet (suppose $A_1 = 1$):


FIGURE 2. The first 5 Sierpinski carpets (from left to right) and the creator, Waclaw Sierpinski

What is $\lim_{n\to\infty} A_n$?

(2) **Integer Multiplication** In this problem, assume that it takes $O(1)$ time to multiply two 1-digit numbers, and $O(k)$ time to add (or subtract) two k-digit numbers. (Note that this differs from the word-RAM assumption used in the rest of the class, where we assume that all numbers can be added and multiplied in constant time, regardless of their length.)
   (a) Consider two polynomials $a(x) = \sum_{i=0}^{n} a_i x^i$ and $b(x) = \sum_{i=0}^{n} b_j x^j$. Fast Fourier Transform (FFT) is a really neat algorithm that computes the products of these polynomials in $O(n \log n)$ time. How can we extend this algorithm to multiply two n-digit integers in $O(n \log n)$ time? Please justify your answer.
   (b) Give an algorithm that multiplies two n-digit integers in $\Theta(n^{\log_2 3})$ time. Please prove the runtime of your algorithm and informally justify why your algorithm is correct. There is no need for a formal loop invariant proof here, but please explain why your algorithm in fact

generates the correct answer. (Hint: Any n-digit integer X can be written as $X_1 \cdot 10^{\lceil n/2 \rceil} + X_2$, where X_1 and X_2 are smaller integers.)

(3) **Trough Finding** Water flow to the troughs in a range of mountains. Consider a Lego-like (discrete) toy-model:

(a) (1D) Given an array of n real numbers (for simplicity assume that any pair of adjacent integers are different from each other). Please design an algorithm to find a local trough (an element smaller than all its neighbors) in $O(\log n)$ time. (For example, in the array $A = \begin{bmatrix} 1 & 2 & 0 & 3 \end{bmatrix}$, the local troughs are $A[1] = 1$ and $A[3] = 0$). We are looking for a formal correctness proof, and a proof that the algorithm runs in $O(\log n)$ time.

(b) (2D) Given a grid of nonnegative real numbers (for simplicity assume the grid is a square of width n). Please design an algorithm to find a local trough (an element smaller than all its neighbors) in $O(n)$ time. (For example, in the grid $G = \begin{bmatrix} 5 & 6 & 3 \\ 6 & 1 & 4 \\ 3 & 2 & 3 \end{bmatrix}$, the local troughs are $G[1][1] = 5$ and $G[2][2] = 1$.) We are not looking for a formal correctness proof, but please explain why your algorithm is correct, and prove that your algorithm runs in $O(n)$ time.

(4) **d-ary heaps** Solve Problem 6-2 in the textbook. No correctness proofs are needed for this problem, but you should informally explain why your algorithms work.

6-2 Analysis of d-ary heaps

A ***d-ary heap*** is like a binary heap, but (with one possible exception) non-leaf nodes have d children instead of 2 children.

a. How would you represent a d-ary heap in an array?

b. What is the height of a d-ary heap of n elements in terms of n and d?

c. Give an efficient implementation of EXTRACT-MAX in a d-ary max-heap. Analyze its running time in terms of d and n.

d. Give an efficient implementation of INSERT in a d-ary max-heap. Analyze its running time in terms of d and n.

e. Give an efficient implementation of INCREASE-KEY(A, i, k), which flags an error if $k < A[i]$, but otherwise sets $A[i] = k$ and then updates the d-ary max-heap structure appropriately. Analyze its running time in terms of d and n.

(5) **Extra credit** Don't forget to fill out the feedback form!
It's at http://goo.gl/forms/Hvv9ypG5erxbRjCk2
There will also be extra credit for extremely well-written solutions.