Assignment 2 (due Sunday, June 21, midnight EST)

Instructions:

• Hand in your assignment using Crowdmark. Detailed instructions are on the course website.

• Give complete legible solutions to all questions.

• Your answers will be marked for clarity as well as correctness.

• For any algorithm you present, you should justify its correctness (if it is not obvious) and analyze the complexity.

1. [12 marks] Consider the following recurrence:

\[ T(n) = \begin{cases} 
T([n/4]) + T([n/9]) + T([n/36]) + 2016 & \text{if } n \geq 36 \\
341 & \text{if } 1 \leq n \leq 35.
\]

Prove inductively (i.e., with the guess-and-check or substitution method) the upper bound \( T(n) = O(\sqrt{n}) \). [Hint: use \( T(n) \leq c\sqrt{n} - c' \) as the “guess”, where \( c \) and \( c' \) are suitable constants.]

2. [12 marks] Given an array \( A \) of integers. An element in \( A \) is a peak if it is not smaller than its neighbors. For elements at the two ends of the array, we consider them having only one neighbor. For example, in an array of \( \{5, 3, 6, 2, 1\} \), 5 and 6 are peak elements. Give an \( O(\log n) \) time algorithm to find a peak element in \( A \). Show the correctness of the algorithm and analyze its time complexity.

3. [21 marks] Consider the following problem: given a sequence of \( n \times 2 \times 2 \) matrices \( A_1 \ldots A_n \) where each entry in \( A_i \) is either 0 or 1, we want to compute the product \( A_1 A_2 \ldots A_n \). Note that the result is a \( 2 \times 2 \) matrix, where the four entries may be very large integers.

(a) [3 marks] Explain why each entry of the final product requires at most \( O(n) \) bits.

(b) [6 marks] Describe and analyze a simple algorithm to solve this problem, by iteratively multiplying the current product with the next matrix \( A_i \) for \( i \) from 1 to \( n \). Show that this algorithm runs in \( O(n^2) \) time.

(c) [12 marks] Next describe and analyze a recursive algorithm to solve this problem. Show that this algorithm runs in \( O(n^{1.59}) \) time. [Hint: you may use Karatsuba and Ofman’s integer multiplication algorithm as a subroutine.]. You may assume \( n \) is a power of 2.
4. [22 marks] Consider the following problem: We are given a set $R$ of $n_1$ rectangles in 2D, where the sides of the rectangles are parallel to the x- and y-axes (i.e., the rectangles are not rotated) and the rectangles do not overlap. We are also given a set $P$ of $n_2$ points in 2D. Each point $p \in P$, $p$ may or may not be contained inside a rectangle but if it is covered, it is covered with a unique rectangle because of the nonoverlapping assumption. Our goal is to find for every point $p \in P$, the rectangle $r_p \in R$ that contains $p$. Let $n = n_1 + n_2$. This is not necessary but for simplicity, you can assume that all coordinates of points and corners of rectangles are distinct.

![Figure 1: Example of the general case of the problem.](image1)

(a) [10 marks] First consider the special that when all rectangles of $R$ are assumed to intersect a given horizontal line $\ell$. Now design an $O(n \log(n))$-time algorithm for this special case.

![Figure 2: Example of the special case from Q3a.](image2)

(b) [12 marks] Now give an $O(n \log^2(n))$-time algorithm (or better) for the general case of the problem. (Hint: use divide-and-conquer and use part (a) as a subroutine).
5. [15 marks] Your friend has invited you to play a game which goes as follows. There are two sets of sticks: (i) $R$ that consists of $n$ red sticks; and $B$ that consists of $n$ blue sticks. Each stick has a height (you can assume they are distinct for simplicity). Each player pairs red and blue sticks (so creates $n$ pairs of stick, where each pair contains one red and one blue stick) and the winner is the one that minimizes the “sum, across all pairs, the differences in heights. So suppose you have paired the sticks and in your $i$’th pair the height of red stick is $r_i$ and the height of the blue stick be $b_i$, then you incur $|r_i - b_i|$ penalty. Design an algorithm that runs in $O(n \log n)$ time (or better) that pairs the sticks with the minimum possible penalty. In other words, your algorithm minimizes the following sum across all possible pairings:

$$\Sigma_{i=1}^{n} |r_i - b_i|$$