Homework 5: Divide & Conquer

Due: October 19, 2023

Problem 1. The recurrence $T(n) = 7T(n/2) + n^2$ describes the running time of an algorithm $A$. A competing algorithm $A'$ has a running time of $T'(n) = aT'(n/4) + n^2$. What is the largest integer value for $a$ such that $A'$ is asymptotically faster than $A$?
Problem 2. An $m \times n$ array $A$ of real numbers is a Gnome array if for all $i, j, k, l$ such that $1 \leq i \leq k \leq m$ and $1 \leq j \leq l \leq n$, we have


That is, whenever we pick two rows and two columns of a Gnome array and consider the four elements at the intersections of these rows and columns, the sum of the upper-left and lower-right elements is less than or equal to the sum of the lower-left and upper-right elements.

(a) Prove that an array is Gnome iff for all $i = 1, 2, \ldots, m - 1$ and $j = 1, 2, \ldots, n - 1$, we have:


**Hint:** For one direction you should use induction separately on rows and columns!

(b) Let $f(i)$ be the index of the column containing the leftmost minimum element of row $i$. Prove that

$$f(1) \leq f(2) \leq \ldots \leq f(m)$$

for an $m \times n$ Gnome array.

(c) Here is a description of a divide-and-conquer algorithm that computes the leftmost minimum element in each row of a $m \times n$ Gnome array

- Construct a sub-array $A'$ of $A$ consisting of the even-numbered rows of $A$.
- Recursively invoke the algorithm to find the leftmost minimum for each row of $A'$.
- Using the information on the leftmost minimum of the even-numbered rows of $A$ you just computed, compute the leftmost minimum in the odd rows of $A$.

Explain how to compute the leftmost minimum of the odd-numbered rows of $A$ given that you have already computed the leftmost minimum for even-numbered rows of $A$. Prove that this can be done in $O(n + m)$ time.

(d) Write the recurrence describing the running time of the algorithm and obtain an explicit closed-form solution for it.

**Note:** You can always assume each of the earlier results on each subsequent part.
Problem 3. While playing with your pile of $n$ rocks, you begin to wonder about the types of rocks you have. A type of rock is *overly common* if more than half of the rocks you have are of this type.

Unfortunately, you don’t have the rock expertise to exactly tell what type a rock is, however, you do have access to a magic machine (i.e. a predicate) that is able to determine if two rocks share the same type.

(a) Describe an algorithm to determine if a collection of $n$ rocks contains an overly common type of rock. If it does, return all of the overly common rocks. Your algorithm should make at most $O(n \log n)$ calls to the magic machine.

(b) Prove the correctness of your algorithm.

(c) Analyze the runtime of your algorithm, justifying that your algorithm makes at most $O(n \log n)$ class to the machine.