Homework 5: Divide & Conquer

Due: October 19th, 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 $A$?

Solution. By the Master Theorem, $T(n) \in \Theta(n^{\log_2 7})$ and $T'(n) \in \Theta(n^{\log_4 a})$. When $n^{\log_2 7} = n^{\log_4 a}$, algorithm $A$ is asymptotically as fast as $A'$. Therefore, we want to find $a$ such that $n^{\log_2 7} > n^{\log_4 a}$. Solving, $a < 49$.

Intuitively, for the two algorithms to be the same asymptotic speed, they should fall under the same Master Theorem case. \qed
**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 $1 \leq j \leq \ell \leq n$, 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, computed 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 a explicit closed-form solution for it.

**Note:** You can always assume each of the earlier results on each subsequent part.
Solution.

(a) We want to prove:


Trivially, by definition of Gnome array, (1) is true. To prove the converse, we first induct on rows and use the result to induct on columns.

Rows: We would like to show that $A[i, j] + A[i + 1, j + 1] \leq A[i, j + 1] + A[i + 1, j]$ implies $A[i, j] + A[k, j + 1] \leq A[i, j + 1] + A[k, j]$ for $i \leq k \leq m$. The base case $k = i + 1, l = j + 1$ is given by the problem statement. Let us assume that this implication holds for some $k = i + n$; we want to show that it must therefore also hold for $k = i + n + 1$. We have the following:

$$A[i + n, j] + A[i + n + 1, j + 1] \leq A[i + n, j + 1] + A[i + n + 1, j] \quad \text{(base case)}$$


Summing the inequalities (i.e. $a \leq b, c \leq d \Rightarrow a + b \leq c + d$), we can conclude that $A[i, j] + A[i + n + 1, j + 1] \leq A[i, j + 1] + A[i + n + 1, j]$.


(b) Define $a_i$ as the index of the leftmost minimal element in row $a$, and $b_j$ as the index of the leftmost minimal element in row $b$, where $i < j$. Assume $a_i > b_j$. By Gnome array property:


We have reached a contradiction because $A[i, a_i]$ and $A[j, b_j]$ are the leftmost minimal elements in their respective rows. Therefore, $a_i \leq b_j$ for $i < j$.

(c) If $\mu_i$ is the index of the $i$-th row’s leftmost minimum, then we know $\mu_{i-1} \leq \mu_i \leq \mu_{i+1}$. For $i = 2k + 1, k \geq 0$, finding $\mu_i$ takes $\mu_{i+1} - \mu_{i-1} + 1$ steps at most, since we only need to compare those elements. Thus:

$$T(m, n) = \sum_{i=0}^{m/2-1} (\mu_{2i+2} - \mu_{2i} + 1)$$

$$= \sum_{i=0}^{m/2-1} \mu_{2i+2} - \sum_{i=0}^{m/2-1} \mu_{2i} + m/2$$

$$= \sum_{i=1}^{m/2} \mu_{2i} - \sum_{i=0}^{m/2-1} \mu_{2i} + m/2$$

$$= \mu_m - \mu_0 + m/2$$

$$= n + m/2$$

$$\Rightarrow O(n + m)$$
(d) The divide time is \( O(1) \), the conquer part is \( T(m/2) \), and the merge part is \( O(n + m) \). Using substitution:

\[
T(m) = T(m/2) + cn + dm
\]

\[
= cn + dm + cn + dm/2 + cn + dm/4 + \ldots
\]

\[
= \sum_{i=0}^{\log_2 m - 1} cn + \sum_{i=0}^{\log_2 m - 1} \frac{dm}{2^i}
\]

\[
= cn \log_2 m + dm \sum_{i=0}^{\log_2 m - 1} \frac{1}{2^i}
\]

\[
\leq cn \log_2 m + 2dm
\]

Since

\[
\sum_{i=0}^{\log_2 m - 1} \frac{1}{2^i} \leq \sum_{i=0}^{\infty} \frac{1}{2^i} = 2
\]  

(1)
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.

Solution. Suppose the pile is split into two halves, $P_1$ and $P_2$. If neither of these two subpiles have an overly common rock type, it must be true that there is no overly common type in the entire pile: indeed, for any type $t$, there are at most $|P_i|/2$ rocks of type $t$ in pile $i$, so the total number of rocks is $\leq |P_1|/2 + |P_2|/2 = n/2$.

Therefore, we must determine if overly common types exist in each of these halves, suggesting a divide and conquer approach: given two subpiles and whether or not an overly common type is in either pile, we must combine the result to determine if the merged pile has an overly common type.

If neither subpile does, then the answer is no by the argument above. Now suppose at least one of the two subpiles has an overly common rock. Note no subpile can have more than one overly common type. The key idea is that these types are the only candidates for an overly common type in the merged pile. Indeed, suppose the overly common type in the merged pile is not an overly common type in either interval (call this type $T$). Then, applying similar logic to the first paragraph, at most half of the rocks will be of type $T$, contradiction.

Thus, we can take a rock of one of the candidate types and compare with every other rock to count how many rocks are of that type. If an overly common type is found, then return all these rocks. Otherwise, there is no overly common type.

To determine time complexity, the recursion comes out to

$$T(n) \leq 2T(n/2) + 2n$$

since a linear search is applied to count the number of rocks resulting in $O(n)$ calls, and there are at most 2 candidates for the overly common type. By Master Theorem, then, the number of calls made is $O(n \log n)$.

Remark: There is a solution that uses $O(n)$ calls: as a start, take pairs of rocks make a call to the machine for each. Depending on the result, what should be done to reduce the search space by at least half?