CS 170 Algorithms
Fall 2015 Prasad Raghavendra and Sanjam Garg

# HW 2

## Due September 11, 5:00pm

**Instructions:** You are welcome to form small groups (up to 4 people total) to work through the homework, but you **must** write up all solutions by yourself. List your study partners for homework on the first page, or "none" if you had no partners.

Begin each problem on a new page. Clearly label where each problem (and subproblems, if any) begins. The pages of your homework submissions must be in order (all pages of problem 1 followed by all pages of problem 2, etc.).

For questions asking you to give an algorithm, you must respond in what we will refer to as the "four-part format" for algorithms:

1. High-level description
2. Pseudocode
3. Running time analysis
4. Proof of correctness

Read the relevant Piazza post to understand what these mean.

You risk receiving no credit for any homework that does not adhere to these guidelines.

No late homeworks will be accepted. **No exceptions.** Do not ask for extensions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. Out of a total of approximately 11 homework assignments, the lowest two scores will be dropped.

This homework is due Friday, September 4, at 5:00pm via Gradescope. Please submit via PDF, not images.

1. **(10 pts.) Recurrence Relations**
   Solve the following recurrence relations and give a $\Theta$ bound for each of them.

   (a) $T(n) = 3T(n/4) + 4n$

   (b) $T(n) = 45T(n/3) + .1n^3$

   (c) $T(n) = T(n-1) + c^n$, where $c$ is a constant.

   (d) $T(n) = 2T(\sqrt{n}) + 3$, and $T(2) = 3$. (Hint: this means the recursion tree stops when the problem size is 2)

2. **(15 pts.) Goldilocks' Problem**
   In this problem, we are given two distinct arrays, one of $n$ distinct *Goldilocks*, $G[1\dots n]$, and one of $n$ distinct *soups* $S[1\dots n]$. For every distinct Goldilocks, there is exactly one soup that is neither too hot nor too cold, but just right. Furthermore, every soup is just right for exactly one Goldilocks. The computational task is to match each Goldilocks with their correct soup. However, the only comparison allowed is "Is soup $j$ too hot, too cold, or just right for Goldilocks $i$?", so you cannot compare a Goldilocks to a different Goldilocks, and you cannot compare a soup to a different soup. Design an efficient randomized algorithm to solve this problem.

3. **(15 pts.) Stock Market Hindsight**
   In this problem, we are given a single array $A[1\dots n]$ of (possibly negative) integers. $A[i]$ represents the change in price of a stock $A$ after day $i$ ends. As stock analysts, we are looking back on this data and trying to decide when the best time to buy and sell the stock would have been, assuming we could only buy once and sell once. In other words, if the stock was bought on day $i$ and sold on day $j$, the *profit* would be $\sum_{k=i}^{j-1} A[k]$, and we seek to maximize this quantity over all choices of $i$ and $j$.

   (a) Given the array $A$, show how to construct an array $B$ such that $B[j] - B[i] = \sum_{k=i}^{j-1} A[k]$.
   *For this part, give only a clear description; full algorithm response format not needed.*

   (b) Using the array $B$, design an algorithm to solve the Stock Market Hindsight problem. What is the total runtime of your algorithm, counting construction of the array $B$?
   *Hint:* Does $B$ remind you of anything from last week?
   *Note:* When citing algorithms we have already studied, do not use the full response format—just a short paragraph explaining why it solves the original problem, and a few sentences for runtime analysis.

4. **(20 pts.) Majority Elements**
   An array $A[1\dots n]$ is said to have a *majority element* if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be **no** comparisons of the form "is $A[i] > A[j]$?". (Think of the array elements as GIF files, say.) However you *can* answer questions of the form: "is $A[i] = A[j]$?" in constant time.

   (a) Show how to solve this problem in $O(n \log n)$ time. (Hint: Split the array $A$ into two arrays $A_1$ and $A_2$ of half the size. Does knowing the majority elements of $A_1$ and $A_2$ help you figure out the majority element of $A$? If so, you can use a divide-and-conquer approach.)

   (b) Can you give a linear-time algorithm? (Hint: Here's another divide-and-conquer approach:
   - Pair up the elements of $A$ arbitrarily, to get about $n/2$ pairs
   - Look at each pair: if the two elements are different, discard both of them; if they are the same, keep just one of them
   - If $|A|$ is odd, there will be one unpaired element. What should you do with this element?

Show that after this procedure there are at most $n/2$ elements left, and that they have a majority element if $A$ does.)

**5. (20 pts.) Local Maxima in Matrices**

Consider an $n \times n$ matrix $M$ with distinct integer entries. Call an entry $M_{ij}$ a *local maximum* if it is greater than all of its neighbors. More precisely, $M_{ij}$ is a local maximum if for all $a, b$ with $|i-a| \leq 1$ and $|j-b| \leq 1$, $M_{ij} \geq M_{ab}$.

Suppose that $M$ is guaranteed to have exactly one local maximum. Show that the local maximum can be found in $O(\log^2 n)$ time.

**6. (20 pts.) Upper Triangular Matrix Multiplication**

A matrix $M$ is called *upper-triangular* if $M(i, j) = 0$ whenever $i > j$. The following is an incorrect proof that upper-triangular matrices can be multiplied in $O(n^2 \log n)$ time:

"An upper-triangular matrix $M$ can be decomposed as $M = \begin{bmatrix} A & B \\ 0 & C \end{bmatrix}$ with $A$, $B$, and $C$ being $n/2 \times n/2$ matrices, and the multiplication of two upper-triangular matrices $M_1$ and $M_2$ can be written as

$$\begin{bmatrix} A_1 & B_1 \\ 0 & C_1 \end{bmatrix} \cdot \begin{bmatrix} A_2 & B_2 \\ 0 & C_2 \end{bmatrix} = \begin{bmatrix} A_1A_2 & A_1B_2 + B_1C_2 \\ 0 & C_1C_2 \end{bmatrix},$$

and thus only four multiplications are needed, $A_1A_2$, $A_1B_2$, $B_1C_2$, and $C_1C_2$. Then if we recurse on these four subproblems and solve them, we can solve the original matrix multiplication problem. This obeys a recurrence of $T(n) = 4T(n/2) + O(n^2)$, which by the master theorem has solution $T(n) = \Theta(n^2 \log n)$."

What is wrong with the above proof?

**7. (1 pts.) Optional bonus problem: More medians**

(This is an *optional* challenge problem. It is not the most effective way to raise your grade in the course. Only solve it if you want an extra challenge.)

We saw in class a randomized algorithm for computing the median, where the expected running time was $O(n)$. Design a algorithm for computing the median where the *worst-case* running time is $O(n)$.

Hint: It's all in finding a good pivot. If you divide the array into small groups of $c$ elements, can you use that to help you a good pivot?