1 Jug Matching (33 points)

Suppose that you are given \( n \) red and \( n \) blue water jugs, all of different shapes and sizes. All red jugs hold different amounts of water, as do all the blue ones. Moreover, for every red jug there is a blue jug that holds exactly the same amount of water (and vice versa).

Your job is to find a matching between red jugs and blue jugs that hold the same amount of water. To do this, you are only allowed to use the following operation: pick a red jug and a blue jug, fill the red jug, and pour it into the blue jug. This will tell you whether the volume of the red jug is less than, equal to, or greater than the volume of the blue jug. In other words, you can compare any red jug and any blue jug. But you **cannot** compare two red jugs or two blue jugs.

Give a randomized algorithm that uses \( O(n \log n) \) comparisons in expectation, and prove that this bound is correct.

2 Median of Sorted Arrays (33 points)

Let \( A \) and \( B \) be sorted arrays of \( n \) elements each. We can easily find the median of \( A \) or the median of \( B \), since they are already sorted – it will be at index \( \left\lfloor \frac{n+1}{2} \right\rfloor \). But what if we want to find the median element of \( A \cup B \)? We could just concatenate them and use the \( O(n) \) time median algorithm, but is it possible to do better? In this problem you should give a better algorithm: find a function \( f(n) = o(n) \) and design a deterministic algorithm whose running time (measured in terms of the number of comparisons) is \( O(f(n)) \). The smaller the function \( f(n) \) the better – to get full credit you must use the smallest possible function \( f(n) \) (asymptotically), but for partial credit any function that is \( o(n) \) will suffice.

As always, prove the running time and correctness of your algorithm.

3 Linear Time Algorithms (34 points)

(a) Let \( A \) be a list of \( n \) (not necessarily distinct) integers. Design an algorithm to test whether any item occurs more than \( \lceil n/2 \rceil \) times in \( A \) that runs in \( O(n) \) time (and prove correctness and running time).

(b) Design an \( O(n) \)-time algorithm that, given a set \( S \) of \( n \) distinct numbers and a positive integer \( k \leq n \), determines the \( k \) numbers in \( S \) that are closest to the median of \( S \) but are not the median. In other words, you should return all values that in the sorted array would be in
locations \((n - k)/2\) through \((n + k)/2\), but not the median itself. If it helps, feel free to assume \(n\) or \(k\) is even or odd. Prove correctness and running time.