15-451/651 Algorithm Design & Analysis, Spring 2026

Oral Homework #4

Oral Homework #4, Due: Week of Apr 20-24

Guidelines:

1. This is an oral homework.
2. You will work in groups of size two or three.
3. A scheduling link is up at `https://docs.google.com/forms/d/e/1FAIpQLSeWxDCilgJ7YPbpofRi2kq2Fh-ZMAHE0R viewform`, and is due midnight Saturday, Apr 18.
4. You will be asked to present solutions to each of the three problems below. Groups can decide who presents which, but:
   - for groups of size 3, one person per problem.
   - for groups of size 2, one person presents 2, while the other presents 1.
5. Your presentations will be graded on correctness. There may be back and forth, where the TAs will help guide you through solutions, etc.
6. **You will have 30 total minutes to present 3 problems, one problem presented per group member.** Please make sure your presentations are concise to fit in this time limit.
7. Notes are allowed during presentations. However, the grading TAs will ask you questions.

Problems:

1. (**) Consider the even more limited query oracle of SAME(i, j), which returns true if $A[i] = A[j]$, false otherwise. Show that:
   (a) one needs at least $\Omega(n^2)$ calls to SAME to decide whether n numbers are all distinct. (Hint: it's much easier to build an adversary that constructs a ground truth based on the queries asked)
   (b) one can decide whether there is a majority element, i.e., one that appears more than $n/2$ times, using $O(n)$ queries to SAME.
2. (**) Recall that an increasing subsequence of an array $A[1 \ldots n]$ is a subsequences of indices $i_1 < i_2 < \ldots < i_l$ such that $A[i_1] < A[i_2] < \ldots < A[i_l]$. Show that the number of comparisons needed to sort a length n array whose elements are all distinct, but with longest increasing subsequence at most t is $\Theta(n(1+\log t))$ (both asymptotic upper and lower bounds).

   You may assume the algorithm has access to t, and the output of the sort is defined to be a permutation under which the input is in non-decreasing order.

   Note that you only need to optimize the number of comparisons, not the runtime of your algorithm.

   Hint: Sequences with longest increasing subsequence of length t can be partitioned into t decreasing sequences. You will need on algorithmic version of this that's within the cost bounds: one way to do so is by extending a LIS algorithm.

3. (***) There is an $n \times n$ grid with all cells set to 0, except non-touching bars of 1s, one horizontal and one vertical. Formally, the two bars some continuous group of at least two 1s in some row, and some continuous group of at least two 1s in some column, and the non-touching condition means that no 1s in either bar touch (share a side, or a corner).

   You can query the grid using queries of 'does this sub-rectangle of the grid contain some 1', and need to find the precise locations of the two bars.

   Show upper/lower bounds of $6 \log n \pm \Theta(1)$ queries.