CS 222A Winter 2011, HW 2 Due Jan 18 Gusfield

1. Read Section 4.1 of the Kleinberg and Tardos book, That material covers algorithms for the interval scheduling problem and the problem of scheduling all intervals (also called the classroom scheduling problem). Here we ask whether the algorithm for the interval scheduling problem can be used to solve the classroom scheduling problem.

Consider the following algorithm:

> Let L be the set of input intervals.
>
> While L is not empty {
>
> Solve the interval scheduling problem on L, and let S be the set of selected intervals in the solution.
>
> Assign the classes (intervals) in S to a classroom that has not been assigned yet. Remove the set of intervals S from L.
>
> }

1a. Is the classroom scheduling problem (to minimize the number of classrooms used) solved by the algorithm above? If you think yes, give a proof, and if you think no, give an example showing that it fails.

1b. How about the modification where we consider the intervals in L by their starting times, rather than their ending times, i.e., using the following algorithm:

> While L is not empty {
>
> Set $S = \emptyset$
>
> Until no longer possible {
>
> Find the interval i in L with earliest starting time that doesn't overlap any intervals already in S.
>
> Place interval i into S.
>
> }
>
> Assign the classes (intervals) in S to a classroom that has not been assigned yet. Remove the set of intervals S from L.
>
> }

If you think this algorithm solves the classroom scheduling problem using the minimum number of classrooms, give a proof, and if you think it does not, give a counterexample.

2. We proved in class that every binary decision tree algorithm for sorting n numbers has a worst case running time of $\Omega(n \log n)$. As stated during the proof, we assumed that the algorithm is correct for any list it is given. That is, no matter what permutation of the sorted order is given to us in the input, the algorithm will work.

Now suppose that you want to create a sorting "algorithm" that is correct only half of the time. That is, it will be correct for half of the possible permutations describing the correct way to sort the input list, but it may report the wrong answer for the other half. Is there a decision tree algorithm to solves this problem whose worst case running time is not $\Omega(n \log n)$? What about if the algorithm must be right for only $1/n$ of the possible permutations? What about if it has to be correct for only $\frac{1}{2^n}$ of the permutations?

3. Do problem 3.14 in the notes for the adversary lower bound.

4. Do problem 3.17 in the notes for the adversary lower bound, just for the cases of n = 4 and 5.