CS 473, Fall 2017
Homework 6 (due Oct 25 Wednesday at 8pm)

You may work in a group of at most 3 students. Carefully read http://engr.course.illinois.edu/cs473/policies.html and http://engr.course.illinois.edu/cs473/integrity.html. One member of each group should submit via Gradescope.

1. [15 pts] (This is a tougher version of the last problem from the practice midterm.) We are given an array $A[1, \ldots, n]$ that contains a super-majority element, i.e., an element that occurs $> 2n/3$ times. We want to find an index $i$ such that $A[i]$ is a super-majority element, if it exists. Design and analyze a Monte Carlo algorithm to solve the problem with probability of error less than $1/n^{100}$. Aim for the best running time you can obtain (which should be much better than linear).

[Hint: for the analysis, use Chernoff bound.]

2. [15 pts] We are given a collection of $m$ subsets $A_1, \ldots, A_m \subseteq \{1, \ldots, n\}$, each of even size. We say that a subset $S \subseteq \{1, \ldots, n\}$ is a splitter if $A_i$ intersects both $S$ and its complement for all $i = 1, \ldots, m$, i.e., $A_i \cap S \neq \emptyset$ and $A_i \cap S^c \neq \emptyset$ (where $S^c$ denotes the complement of $S$). The problem of finding a splitter is NP-hard in general, but we will consider a special case:

We say that a subset $P \subseteq \{1, \ldots, n\}$ is a perfect splitter if $|A_i \cap P| = |A_i \cap P^c|$ for all $i = 1, \ldots, m$. For any input instance for which we know the existence of a perfect splitter (but do not know the perfect splitter itself), design and analyze a Monte Carlo algorithm that finds a splitter in polynomial time. (The splitter found does not have to be perfect.)

[Hint: imitate Papadimitriou’s SAT algorithm…]

3. [15 pts] We are given a (unweighted) bipartite graph $G$ with $n$ vertices and $m$ edges. Suppose we have already computed a maximum matching $M$ in $G$. Now suppose we delete a vertex $v$ from $G$ and all its incident edges. Let $G'$ be the new graph (with $n-1$ vertices). Describe how to efficiently compute a new maximum matching $M'$ in $G'$. (Your algorithm should be faster than re-running a matching algorithm from scratch. Remember to prove correctness of your algorithm.)