## Groundrules

- Homeworks will generally consist of *exercises*, easier problems designed to give you practice, and *problems*, that may be harder, trickier, and/or somewhat open-ended. You should do the exercises by yourself, but you may work with a friend on the harder problems if you want. One exception: no fair working with someone who has already figured out (or already knows) the answer. If you work with a friend, then write down who you are working with.
- If you've seen a problem before (sometimes we'll give problems that are "famous"), then say that in your solution (it won't affect your score, we just want to know). Also, if you use any sources other than the textbook, write that down too (it's fine to look up a complicated sum or inequality or whatever, but don't look up an entire solution).

## Exercises

1. **(We're on the same page.)** Show that for paging, the algorithm "just throw out a random page from the cache" has competitive ratio $\Omega(k)$, even for $n = k + 1$.
2. **(Smart and Select.)** In this problem we give a randomized algorithm for selection/median-finding that is different from the QuickSelect algorithm, and uses only $(1.5+o(1))n$ comparisons in expectation.
   (a) Given a set of $n$ distinct numbers $A = \{a_1, a_2, \ldots, a_n\}$, consider the following algorithm to find an approximate median: *Choose $2m + 1$ elements $S$ uniformly at random from $A$. Return the median $M$ of the elements in $S$.* (You pick the elements independently with replacement, say.) For $\epsilon, \delta < 1/2$, show that if $m = \frac{1}{\epsilon^2} \log \frac{1}{\delta}$, then
   $$\Pr[rank(M) \in n(1/2 \pm \epsilon)] \geq 1 - \delta.$$
   (Remember: the smallest number has rank 1, largest rank $n$. Assume distinct numbers if needed.)
   (b) Show that given any $k \in [n]$, the above algorithm can be modified to find an element with rank in $k \pm \epsilon n$. (Call this the *approximate selection* algorithm.)
   (c) Now consider an algorithm to find the element of rank $K$ of the original set of numbers $A$ in near-linear time. This algorithm will return the exact answer (not just an approximate one), but will have low expected running time.
   > *Set $\epsilon = n^{-1/3}$, and $\delta = 1/n^2$. Run the approximate selection algorithm with parameters $k_\ell = K - \epsilon n$ and $k_h = K + \epsilon n$ to find elements $a_\ell$ and $a_h$ respectively. By comparing every element to both $a_\ell$ and $a_h$, place all elements into buckets $L$, $M$, and $H$, where $L = \{a \in A : a < a_\ell\}$, $M = \{a \in A : a_\ell \leq a \leq a_h\}$ and $H = \{a \in A : a > a_h\}$. Sort the elements in $M$, and return the element of rank $K - |L|$ in $M$. (If $a_\ell > a_h$ or $K \notin [|L| + 1, |L| + |M|]$, then restart from scratch.)*
   i. Show that the algorithm always returns the element of rank $K$.
   ii. Show that the size of $M$ is $O(n^{2/3})$ whp.
   iii. Show that the total number of comparisons performed by the algorithm, including those in the approximate selection subroutine, is at most $2n + O(n^{2/3} \log n)$ whp, and hence the expected runtime of the algorithm is also $2n + O(n^{2/3} \log n)$.
   iv. Show how to improve the expected running time to $1.5n + o(n)$ by using fewer comparisons to form $L$, $M$, and $H$.

*Note: There is a lower bound of $(2+\epsilon)n$ comparisons for any deterministic median finding algorithm (Dor and Zwick, FOCS '96), so we've just shown an algorithm where we can do strictly better using randomization. Can we do even better using randomization? It turns out that we can't—Cunto and Munro (JACM '89) showed that any randomized median finding algorithm must perform $1.5n - o(n)$ comparisons on average. So our algorithm is best possible, up to lower order terms.*

## Problems

1. **(On approximate Nash equilibria.)** Recall that a Nash Equilibrium in a 2-player general-sum game is a pair of distributions $P$ and $Q$ (one for each player) such that neither player has any incentive to deviate from its distribution assuming that the other player doesn't deviate from its distribution either. In particular, under $(P, Q)$, the expected payoff to the row player for each row $r$ with $P(r) > 0$ is equal to the maximum payoff out of all the rows, and the expected payoff to the column player for each column $c$ with $Q(c) > 0$ is equal to the maximum payoff out of all the columns.

   Now, assume we have a game in which all payoffs are in the range $[0, 1]$. Define a pair of distributions $P, Q$ to be an "$\epsilon$-Nash" equilibrium if each player has *at most* $\epsilon$ incentive to deviate. That is, the expected payoff to the row player for each row $r$ with $P(r) > 0$ is within $\epsilon$ of the maximum payoff out of all the rows, and vice-versa for the column player.

   Using the fact that Nash equilibria must exist, show that there must exist an $\epsilon$-Nash equilibrium in which each player has positive probability on at most $O(\frac{1}{\epsilon^2} \log n)$ actions (rows or columns), where $n$ is the total number of rows and columns. Hint (if you are a machine learning person): think of "$P$" as a distribution over "examples" that are the rows, with the columns as different "hypotheses", and vice-versa.

   Note: this fact immediately yields an $n^{O(\frac{1}{\epsilon^2} \log n)}$-time algorithm for finding an $\epsilon$-Nash equilibrium. No PTAS (algorithm running in time polynomial in $n$ for any fixed $\epsilon > 0$) is known, however.

2. **(Catching Robbers, Again.)** In HW#2, problem 2(b) you came up with an algorithm, that no matter what the probability distribution $\vec{p} = (p_1, \ldots, p_n)$ over the $n$ locations was, you'd catch the thief with probability at least $v \geq 1 - 1/e$. The solution was to write an LP and sample from the solution, so that each position was covered with probability at least $v$. Then linearity of expectations implies that the probability of catching the thief is at least $v$.

   Now suppose I gave you an algorithm $\mathcal{A}$ that, *given the fixed vector $\vec{p}$*, it outputs a deterministic collection of set indices $C(\vec{p}) \subseteq [m]$ such that $|C(\vec{p})| \leq k$ and

   $$\sum_{i \in [n]} p_i \cdot \mathbf{1}(\text{there exists an index } j \in C(\vec{p}) \text{ such that point } i \in S_j) \geq v. \tag{1}$$

   In other words, *if we knew the robber's probability vector $\vec{p}$*, we could cover at least $v$ probability mass using the $k$ points in $C(p)$.

   Using this algorithm $\mathcal{A}$ as a subroutine, we'll now develop a different algorithm that solves HW#2, problem 2(b). Given $\delta > 0$, our algorithm will generate a (multi)set of potential solutions $\mathcal{C} = \{C_1, C_2, \ldots, C_T\}$, where each $C_i \subseteq [m]$ and $|C_i| \leq k$, such that $T = O(\delta^{-2} \log n)$, and the strategy "choose a uniformly random $C_i$ from $\mathcal{C}$ and cover the sets with indices in $C_i$" has at least $v - \delta$ probability of catching the thief, for any fixed unknown $\vec{p}$. (In other words, each position $j \in [n]$ will be contained in at least a $v - \delta$ fraction of the sets $C_i \in \mathcal{C}$.)

   Consider the following algorithm, given any $\delta < 1$:

   - Start with $\vec{w}^{(0)} = (1, 1, \ldots, 1)$; set $\epsilon = \delta/4$ and $T = \frac{2 \log n}{\epsilon \delta}$.
   - At each step $t \in [1, T]$, use $\mathcal{A}$ to find the set $C_t := C(\frac{\vec{w}^{(t-1)}}{W_{t-1}})$, where $W_{t-1} := \sum_j \vec{w}_j^{(t-1)}$.

     For all elements $j \in [n] \setminus C_t$, set $w_j^{(t)} \leftarrow w_j^{(t-1)} \cdot (1 + \epsilon)$. For all $j \in C_t$, set $w_j^{(t)} \leftarrow w_j^{(t-1)}$.

Finally, let $\mathcal{C}$ be the sets $C_1, C_2, \ldots, C_T$ we found in the above steps.

(a) Give an upper bound of the form $W_t \leq n(1 + \_\_\_)^t$ for all $t \in [1, T]$.

(b) After $T$ steps, if there is an element that is contained in at most $v - \delta$ fraction of the sets $C_i$ (i.e., the algorithm fails), show that $W_T \geq (1 + \epsilon)—^T$.

(c) Combine these two bounds, and use algebra to obtain a contradiction for the settings of $T$ and $\epsilon$; hence show the algorithm is successful when it terminates. (I.e., each position $j \in [n]$ is contained in at least a $v - \delta$ fraction of the sets $C_i \in \mathcal{C}$.)

*Note: Given an algorithm that catches the robber with probability at least $v$ for an unknown $\vec{p}$, clearly we get an algorithm that works if we know $\vec{p}$ as well. What we've shown is the other way: given an algorithm that works for every fixed known $\vec{p}$, we developed an algorithm that works for unknown $\vec{p}$ as well!*

3. **(Project Proposal.)** Think about your course project and write 1-2 paragraphs on what you plan to do. A typical project might involve reading and explaining some advanced topic (e.g., Talagrand's inequality, FKG inequality, ...), reading 1-2 papers and thinking about the open questions or how the papers relate, or thinking about a problem of your own design related to course material. Feel free to come talk with us before March 16 about your ideas. (You can work in pairs if you want.)