Welcome to Problem Set 5. A few notes before you begin.

- You may collaborate with up to 4 other people. All work must be written up individually. You may collaborate, but you cannot copy. E.g. you can talk to your friends about ideas for a problem together, but you cannot copy their solution and just change some words around. **List your collaborators at the top of your submission.**
- Googling or looking up solutions in any way is not allowed.
- Cheating is not worth it! Your grade in this course does not define your worth as a person, and in 10 years you will not care about your bad grade on a homework assignment. But you will care if you are caught plagiarizing: plagiarizing has serious consequences, including the potential of expulsion.
- These problems are designed to challenge you. Start them early; if we’ve done our job well writing them, you will have to chew on them for a while before finding the solutions, and that means you will do best if you can sleep on your solutions rather than starting them the night before the due date.
- You do not need to implement anything in code. In fact, please do not. Pseudocode or a clear English explanation of your algorithm are both acceptable: in some cases pseudocode may be clearer than plain English, and in others the plain English might be better.
- If a question asks you to come up with an algorithm with a certain target runtime (say $O(n \log n)$), and you don’t know how to achieve this runtime but know how to solve the problem less efficiently (in, say, $\Theta(n^2)$ time), write that down! Depending on how inefficient your solution is compared to the benchmark, it will receive a varying amount of credit.
- For each of our algorithm design questions, you should come up with a deterministic algorithm unless the question specifies that a randomized algorithm is wanted.
- All analysis must be mathematically rigorous. Any answer you provide should be proven.
- Remember, we can’t evaluate your work if we can’t understand it. Communicating mathematical and/or complex ideas is an important skill in computer science; treat your problem sets as practice.
- You should use LaTeX to typeset your solutions.
- Have fun!!

CIS 320 Problem Set 5

Due: Wednesday 11/24/21
Problem 1 (Regretfully Random). In class, we derived the polynomial weights algorithm, which guarantees that against any sequence of loss vectors, even a sequence chosen by an adversary the loss experienced by the algorithm is no more than the loss of the best expert in hindsight, up to low order terms. But the polynomial weights algorithm was randomized. In this problem, you will show that it is impossible to get the same guarantees with any deterministic algorithm. Given a sequence of loss vectors, let OPT be the loss of the best expert in hindsight. Show that for every deterministic algorithm operating in the experts setting, there is a sequence of loss vectors that causes the algorithm to experience loss at least $2 \cdot OPT$.

Problem 2. Recall that the (unweighted) $s-t$ min-cut problem is to find the partition of the vertices of a graph $G = (V, E)$, $V = A \cup B$ such that $s \in A$ and $t \in B$, so that the number of edges crossing between $A$ and $B$ is minimized. Given a graph $G$, let $P$ be the set of all paths between $s$ and $t$, and consider the following integer linear program (i.e. a linear program in which the decision variables $x_e$ are constrained to be integers):

\[
\begin{align*}
\text{Minimize} & \quad \sum_{e \in E} x_e \\
\text{subject to} & \quad \sum_{e \in p} x_e \geq 1, \quad \forall p \in P \\
& \quad x_e \in \{0, 1\}, \quad \forall e \in E
\end{align*}
\]

(a) Prove that $I$ solves the min-cut problem, and show how to efficiently recover the partition $(A, B)$ from the solution to $I$.

(b) In general, we can’t solve integer linear programs efficiently because of the constraint that the variables take integer values, so consider the relaxation to the following linear program $L$, which no longer requires that the decision variables $x_e$ be integers. It turns out that in this case, the optimal solution to $L$ is the same as the optimal solution to $I$ (you’ll have to take our word for it, proving this is beyond the scope of this class), so solving $L$ is sufficient to solve the min-cut problem:

\[
\begin{align*}
\text{minimize} & \quad \sum_{e \in E} x_e \\
\text{subject to} & \quad \sum_{e \in p} x_e \geq 1, \quad \forall p \in P \\
& \quad x_e \geq 0, \quad \forall e \in E
\end{align*}
\]

We know how to solve linear programs efficiently, but this one might take awhile to write down: In terms of the number of vertices $n$, how many constraints can $L$ have in the worst case?

(c) Fortunately, to solve a linear program with the polynomial weights algorithm, we don’t need to write down all of the constraints! We only need to be able to find the most violated constraint, given a candidate solution. Give a polynomial time algorithm which given a candidate solution $\{x_e\}$ finds the most violated constraint $p$ — i.e. finds the $p \in P$ that maximizes $1 - \sum_{e \in p} x_e$.

Problem 3. In this problem, we will apply the Minimax Theorem to derive a powerful technique for proving lower bounds on randomized algorithms called Yao’s Minimax Principle.

A) Consider some class of problems $\Pi$. Suppose that $A_1$ is a finite collection of potential deterministic algorithms for solving $\Pi$, suppose that $A_2$ is a finite collection of possible inputs to an algorithm for $\Pi$. Given an algorithm $a \in A_1$ and an input $b \in A_2$, let $C(a, b)$ denote the cost of running algorithm $a$ on input $b$ (the cost can be running time, some proxy for this like the number of comparisons made by a sorting algorithm, or anything else). Consider a zero sum game in which the minimization player has action set $A_1$ and the maximization player has action set $A_2$, and the cost function is $C$. What does the min max value of this game represent? How about the max min value?

b) Suppose that there is a distribution $q$ over inputs $A_2$ so that for every deterministic algorithm $a \in A_1$ the expected cost of running $a$ on an input $b$ sampled from $q$ is at least $R$. Prove that for every distribution $q$ the expected cost of running a deterministic algorithm $a \in A_1$ on an input $b$ sampled from $q$ is at least $R$.
over algorithms in $A_1$, there is an input $b \in A_2$ that causes the expected cost to be at least $R$, over the randomness of sampling an algorithm from $p$.

c) Use this principle to show that our $\Omega(n \log n)$ lower bound for comparison based sorting algorithms extends to randomized algorithms: For every randomized comparison based sorting algorithm, there is an input that causes it to make at least $\Omega(n \log n)$ comparisons in expectation. You can assume here that a randomized algorithm for sorting must always return the correct solution, the only thing that is uncertain is its running time/number of comparisons made.

**Problem 4.** In this problem, we’ll use the minimax theorem to prove a basic result from decision theory. Suppose there are some finite number of actions $A$ you can take, and some finite number of “states” $S$ that the world can be in. Each action $a_i \in A$ has some payoff $U(a_i, s_j)$ depending on which state $s_j \in S$ the world is in. The problem is you don’t know the state of the world. You might have some belief about the state of the world, in the form of a probability distribution $q$ over states: in this case, you would pick an action $a^*(q)$ that would maximize your expected payoff given your belief about the world. Lets write

$$U(a, q) = \sum_{s_j \in S} q(s_j) \cdot U(a, s_j)$$

for your expected payoff for playing action $a_i$ given your belief $q$ about the states, and similarly, given a distribution $p \in \Delta A$ over your actions, we’ll write:

$$U(p, s) = \sum_{a_i \in A} p(a_i) \cdot U(a_i, s_j)$$

for your expected utility of playing an action at random from $p$ if the state of the world is $s_j$. In this notation:

$$a^*(q) \in \arg \max_{a_i \in A} U(a_i, q).$$

Here $q(s_j)$ is the probability that the world is in state $s_j$ according to your belief, and $p(a_i)$ is the probability that your distribution $p$ places on action $a_i$.

There might be some actions that are never a good idea to play, no matter what the state of the world. Such actions are called strictly dominated actions. Formally, an action $a_i$ is strictly dominated if there exists some other distribution over actions $p$ that you could play that is always better than $a_i$, no matter what the state of the world. Formally, $a_i$ is strictly dominated if there exists a distribution over actions $p \in \Delta A$ such that for every state $s_j \in S$:

$$U(p, s_j) > U(a_i, s_j).$$

Clearly there is no circumstance in which you should ever play a strictly dominated action. If $a_i$ is strictly dominated by $p$, then no matter what the state of the world is, you would do better to play $p$ instead of $a_i$ — so even without information about the state of the world, you can eliminate $a_i$ from consideration.

Prove that this is the only case in which you would ever want to eliminate an action from consideration without knowledge of the state of the world. Specifically, show that if an action $a_i$ is not strictly dominated, then there exists a distribution over states $q \in \Delta S$ such that if your belief is $q$, then playing $a_i$ is optimal:

$$a_i \in \arg \max_{a \in A} U(a, q).$$