NOTE: from now on **do not** use sources beyond the materials linked to on the course website to solve these problems.

Remember that you must write up your solutions **independently**, and **list your collaborators** by name clearly at top of your submission (or "no collaborators" if none).
50 points total, with an additional 12 bonus points.

1. (16 pts total, **plus 4 bonus pts**) Implement the exponential weights / Hedge algorithm in the "action" setting (see the right-hand side of the table on page 1 of these notes).

   a. (8 pts) Use your Hedge implementation to write a game AI. Write a moderator program which take as input a number of rounds n and a matrix M which encodes the payoffs of a zero-sum game. For each round, the moderator will solicit an action from the user (that's you, the human) and the AI: when the player chooses action i and the AI chooses action j, the player gets a reward of $M[i,j]$, and the AI gets $-M[i,j]$. Both the player and AI know M. The moderator will report the actions and payoffs to both player and AI at the end of the round.

   Your AI should use Hedge: at round t, sample j_t with propability distribution given by p_t (you can use built-in random calls for this, no need to write from scratch), and upon observing (i_t, j_t) from the moderator, sets the loss vector $\ell_t = M[i_t, :]$, the i_tth row of M — this tells Hedge what the loss would have been had some other j been chosen. Then Hedge updates the weights and is ready with the distribution p_{t+1} to sample from for the next round. (Choose η wisely!)

   Display some sample output (the actions chosen and the resulting payoff, as well as the totals at the end) for a few randomly-generated matrices. You should run each for $T \geq 20$ rounds. Submit your code separately.

   b. (8 pts) Now moving beyond the zero-sum case, consider the case where Nature can choose *any* loss vector $\ell_t \in [0,1]^N$, meaning each component of the loss vector must be between 0 and 1. Use your Hedge implementation to empirically find a *worst-case* sequence of loss vectors when there are only $N = 2$ actions. Show a plot which clearly exhibits $\Theta(\sqrt{T})$ regret, and submit your code separately. How close to the $\sqrt{(T/2)\ln N}$ bound can you get?

   c. (**Optional: 4 bonus pts**) Give a specific bad sequence, which could be your answer to (b), and *prove* that it forces Hedge to have $\Theta(\sqrt{T})$ regret. (In fact, for *any* algorithm, there is such a sequence!)

2. (4 pts, **plus 4 bonus**) We discussed several loss functions $L : [0,1] \times \{0,1\} \to \mathbb{R}$ in class, such as squared loss $L(x,y) = (x-y)^2$ and absolute loss $L(x,y) = |x-y|$.

Suppose that you knew the outcome y was going to be 1 with probability p and 0 with probability $1-p$; show that squared loss guarantees that the best prediction (in terms of expected loss) is going to be $x = p$, but give a distribution where this is not the case for absolute loss.

Bonus (4 pts): Show that $x = p$ would also be the best prediction for any loss of the form $L(x, y) = \phi(y) - \phi(x) - \phi'(x)(y - x)$ where ϕ is any (differentiable) strictly convex function. You may look up (and **CITE**) the definition of a "subgradient" for this problem. (Interestingly, these are the *only* loss functions with this property.)

3. (8 pts) Suppose you do not have any idea how long your learning algorithm will need to run. Use Hedge as a subroutine, which takes both $\hat{T}$ and η as parameters (the time horizon guess and learning rate), to build a meta-algorithm which will always achieve $O(\sqrt{T})$ regret even without knowing T ahead of time. (Hint: draw inspiration from the array doubling technique we used in amortized analysis.)

4. (16 total pts) Suppose you are running an altruistic hardware store, but only have exactly one each of the n different products available. One by one, n customers arrive; when customer i arrives, she identifies a set of S_i parts any of which would solve her problem, and you can choose to sell her one of them if you still have one available. This is the only chance you have to help her out; once she leaves she will not come back. The goal is to design an online selling algorithm which will maximize the number of customers helped.

a. (6 pts) Give a deterministic algorithm which is 2-competitive for this problem, meaning the competitive ratio $\mathsf{OPT}/\mathsf{Alg}$ is at most 2: the algorithm always sells at least half as many items as the optimal offline solution. (Hint: think about the vertex cover approximation algorithm we covered in class.)

b. (4 pts) Show that *every* deterministic algorithm has competitive ratio at least 2.

c. (6 pts) Now consider the randomized algorithm that, upon the arrival of a customer, sells her an item chosen uniformly at random from those still available in S_i (so if U_i are the unsold items at time i, a random one from $S_i \cap U_i$ is selected and sold). Prove that the competitive ratio of this algorithm (in expectation) is strictly less than 2 for all n.

5. (6 pts) Take your student ID digit by digit and fill out a 5×5 matrix M, going back to the beginning when you run out of digits. For example, if my ID were 1435264, the

matrix would be

$$M = \begin{bmatrix} 1 & 4 & 3 & 5 & 2 \\ 6 & 4 & 1 & 4 & 3 \\ 5 & 2 & 6 & 4 & 1 \\ 4 & 3 & 5 & 2 & 6 \\ 4 & 1 & 4 & 3 & 5 \end{bmatrix} .$$

Find a (mixed) Nash equilibrium of the resulting zero-sum game. Give both strategies and the value of the game, and certify your solution by showing the vector of expected payoffs for player 2 under player 1's strategy, and vice versa. (Hint: we will see in class how to use Hedge here!)

6. (**Optional: 4 bonus pts**) Give a 6-node stable matching example (3 on each side) where the L-optimal stable matching is the worst for R, and vice versa. Generalize this to $2n$ nodes.