CS 261 Problem Set 3

Luis A. Perez

Problem 13

Solution:

(a)

Lemma 0.1. *The minimum worst-case number of mistakes that a deterministic algorithm can make is precisely* $\log_2 n$.

Proof. Let us fix some deterministic algorithm $\mathcal{A}$. We will show that there exists a sequence of expert preditions and event realizations for this algorithm such that it makes precisely $\log_2 n$ mistakes. First, note that for any algorithm, once an expert is incorrect, the algorithm will ignore all future predictions by the expert. This is because we know that there is at least one omniscient expert that is always correct.

As such, let E^t be the set of experts the algorithm is currently considering at time t, with $E^1 = \{1, \cdots, n\}$. At time t, let precisely half of these experts predict 1 and the other half predict 0 (any remaining experts don't matter since no algorithm will take them into consideration). This is possible by induction, since $|E^1| = n$ is a power of 2 so we will always be able to half it (until we reach $E^{\log_2 n+1} = |1|$).

Let $p^t \in \{0, 1\}$ be our algorithm's pick given the expert predictions as constructed above.

Then let the event realization be $r^t = 1 - p^t$. Why can we do this? Note that all E^t experts are distinguished only by their prediction at time t (they have agreed on all previous predictions up to this point, since if they had not, then they would have been incorrect and any algorithm would ignore them). As such, re-assigning the omniscient expert from p^t to $1 - p^t$ is indistinguishable to the algorithm $\mathcal{A}$.

Then note that with the above sequence of expert predictions and realizations, $\mathcal{A}$ will make precisely $\log_2 n$ mistakes, since by construction, a mistake occurs at each timestep $t = 1, 2, \cdots, \log_n$. □

For an algorithm that achieves the lower bound proposed above, see the following:

- We begin by considering the advice of all experts. We denote this by having $n^0 = n$.

- At timestep t, let n_c^t be the number of experts suggesting the value $c \in \{0, 1\}$. Note that we have $n^t = n_0^t + n_1^t$.
- Let the deterministic policy p^t be given by:

$$p^t(v) = \begin{cases} 1 & n_v^t \geq n_{1-v}^t \\ 0 & \text{otherwise} \end{cases}$$

- Drop all $n_{1-r_t}^t$ experts who were wrong, and from now on, no longer consider their suggestion. We denote this by seeting $n^{t+1} = n_{r_t}^t$ (we will only look at the expert that were right).

Lemma 0.2. *The algorithm above achives the minimum worst-case number of mistakes,* $\log_2 n$.

Proof. The proof is straight-foward. The algorithm always agrees with the majority predictions of the current set E^t of never-before-wrong experts. As such, if the algorithm is incorrect (a mistake is made), this implies the majority of the never-before-wrong experts were also incorrect, which implies that $|E^{t+1}| \leq \frac{E^t}{2}$. As such, in the worst-possible case, the algorithm can make $\log_2 n$ mistakes. □

(b)

Lemma 0.3. *The minimum worst-case number of mistakes that a randomized algorithm can make is precisely* $\frac{1}{2}\log_2 n$.

Proof. Let us fix some randomized algorithm $\mathcal{A}$. We will show that there exists a sequence of expert preditions and event realizations for this algorithm such that it makes precisely $\frac{1}{2}\log_2 n$ mistakes. First, note that for any algorithm, once an expert is incorrect, the algorithm will ignore all future predictions by the expert. This is because we know that there is at least one omniscient expert that is always correct.

As such, let E^t be the set of experts the algorithm is currently considering at time t, with $E^1 = \{1, \cdots, n\}$. At time t, let precisely half of these experts predict 1 and the other half predict 0 (any remaining experts don't matter since no algorithm will take them into consideration). This is possible by induction, since $|E^1| = n$ is a power of 2 so we will always be able to half it (until we reach $E^{\log_2 n+1} = |1|$).

Let $p^t = [p^t(1), 1-p^t(1)]$ be our randomized algorithm's distribution given the expert predictions as constructed above.

Then let the event realization be:

$$r^t = \begin{cases} 1 & p^t(1) \leq \frac{1}{2} \\ 0 & p^t(1) > \frac{1}{2} \end{cases}$$

Why can we do this? Note that all E^t experts are distinguished only by their prediction at time t (they have agreed on all previous predictions up to this point,

since if they had not, then they would have been incorrect and any algorithm would ignore them). As such, re-assigning the omniscient expert from predicting 0 to 1 (or vice versa) is indistinguishable to the algorithm $\mathcal{A}$.

Given the above, let us compute the minimum expected number of mistakes:

$$\sum_{i=1}^{\log_2 n} \Pr[\text{mistake at timestep } i] = \sum_{i=1}^{\log_2 n} \max\{p^i(1), 1 - p^i(1)\}$$

(We have at most $\log_2 n$ tries before the expert set is a singletone)

$$\geq \sum_{i=1}^{\log_2 n} \frac{1}{2}$$

(Probability of mistake is always at least $\frac{1}{2}$)

$$\geq \frac{1}{2} \log_2 n$$

□

For an algorithm that achieves the lower bound proposed above, see the following:

- We begin by considering the advice of all experts. We denote this by having $n^0 = n$.
- At timestep t, let n_c^t be the number of experts suggesting the value $c \in \{0, 1\}$. Note that we have $n^t = n_0^t + n_1^t$.
- Let the randomized policy p^t be given by:

$$p^t(v) = \frac{n_v^t}{n^t}$$

- Drop all $n_{1-r_t}^t$ experts who were wrong, and from now on, no longer consider their suggestion. We denote this by seeting $n^{t+1} = n_{r_t}^t$ (we will only look at the expert that were right).

Lemma 0.4. *The algorithm above achives the minimum worst-case number of mistakes,* $\frac{1}{2} \log_2 n$.

Proof. The proof is straight-foward. We simply calculated the expected number of mistakes with the worst-case input sequence and rewards constructed in our proof

above. In that case, we have:

$$\sum_{i=1}^{\log_2 n} \Pr[\text{mistake at timestep } i] = \sum_{i=1}^{\log_2 n} \max\{p^i(1), 1 - p^i(1)\}$$

(We have at most $\log_2 n$ tries before the expert set is a singletone)

$$= \sum_{i=1}^{\log_2 n} \max\left\{\frac{n_1^t}{n^t}, \frac{1 - n_1^t}{n^t}\right\}$$

(Using our proposed algorithm)

$$= \sum_{i=1}^{\log_2 n} \max\left\{\frac{1}{2}, \frac{1}{2}\right\}$$

(Using the worst-case sequence of predictions)

$$= \frac{1}{2}\log_2 n$$

As such, our algorithm achives the given lower bound. □

Problem 14

Solution:

(a)

Lemma 0.5. *At each timestep $t = 1, 2, \cdots, T$, the largest perturbed cumulative reward of an action prior to t is more than one larger than the second-largest such perturbed reward with probability $1 - \eta$.*

Proof. We proof this directly. Let us consider the timestep t. In this case, the perturbed cumulative reward of an action a prior to t is given simply by $r_a = X_a + \sum_{u=1}^{t-1} r^u(a)$. Let us sort the r_a from smallest to largest, and label them as $r_a^1, \cdots, r_a^n$. Using this set-up, we are therefore interested in showing that $\Pr[r_a^n - r_a^{n-1} > 1] \geq 1 - \eta$. We can simplify this as follows:

$$\begin{aligned}
\Pr[r_a^n - r_a^{n-1} > 1] &= \Pr[(X_{a^n} + \sum_{u=1}^{t-1} r^u(a^n)) - (X_{a^{n-1}} + \sum_{u=1}^{t-1} r^u(a^{n-1})) > 1] \\
&\qquad \text{(Definition)} \\
&= \Pr[X_{a^n} - X_{a^{n-1}} > 1 + (\sum_{u=1}^{t-1} r^u(a^{n-1}) - \sum_{u=1}^{t-1} r^u(a^n))] \\
&\qquad \text{(Re-arranging terms)} \\
&\geq \Pr[X_{a^n} - X_{a^{n-1}} > 1] \qquad \text{(See below)} \\
&\geq 1 - \eta \qquad \text{(See below)}
\end{aligned}$$

To see why the second to last line applies, we know that the difference between the unperturbed cumulative rewards so-far must be in the interval $(0, 1)$. As such the RHS of the inequality must be in the interval $(0, 2)$ (at most add/subtract 1). Since all X_a are non-negative integers, there difference (LHS) must also be an integer, which means a lower bound on this probability is just $\Pr[X_{a^n} - X_{a^{n-1}} > 1]$.

To see why the last line is the case, consider the process of determining the values X_a^n and X_a^{n-1} through the flipping of coins. Note that we must have $X_{a^n} \geq X_{a^{n-1}}$, since otherwise r_{a^n} would be larger than $r_{a^{n-1}}$. As such, by construction, X_a^{n-1} will be realized before X_a^n (since its coin will hit heads first). Now, the only way for $X_a^n - X_a^{n-1} > 1$ is for the next coin flip for X_a^n to **not be heads**. This is given exactly by the probaility $1 - \eta$, as such this establishes a lower bound on the probability of $\Pr[X_{a^n} - X_{a^{n-1}} > 1]$. □

(b) We follow the hint as proposed, but proof using arbitrary X_a (fixed at the start). Then we claim the following:

Lemma 0.6. *The reward of our algorithm, given by* $\sum_{t=1}^{T} r^t(a_t)$ *where:*

$$a_t = \arg\max \left\{ X_a + \sum_{u=1}^{t} r^u(a) \right\}$$

, is at least the reward of the best fixed-action that maximized the perturbed reward, given by

$$a^* = \max_{a \in A} \left\{ X_a + \sum_{t=1}^{T} r^t(a) \right\}$$

Proof. To see why, consider the action sequence $\{a_t\}$ where

$$a_t = a^* = \arg\max \left\{ X_a + \sum_{t=1}^{T} r^t(a) \right\}$$

We will iteratively transform this sequence into that of our algorithm, arguing that at each step, the transformation cannot decrease our total reward. We do this by induction on the number of actions that match our algorithm's sequence, beginning from the last action.

For the base case, consider the action taken on the last step, $t = T$, and construct the sequence where $a'_i = \arg\max_{a \in A}\{X_a + \sum_{u=1}^{T} r^u(a)\}$ for all $1 \leq i \leq T$. Then the total reward of this sequence is given by:

$$\begin{aligned} \sum_{t=1}^{T} r^t(a'_t) &= \sum_{t=1}^{T-1} r^t \left(\arg\max_{a \in A}\{X_a + \sum_{u=1}^{T} r^u(a)\} \right) \text{ (Subtituting our new sequence } a') \\ &= \sum_{t=1}^{T} r^t(a^*) \end{aligned}$$

(Our selected action is just the best fixed action that maximized the total perturbed reward)

From the above, it's clear that the reward provided by our modified action sequence is at least that of the best fixed-action sequence that maximized the perturbed total reward.

For our inductive hypothesis, suppose we have a sequence

$$\{a_1, \cdots, a_{T-k-1}, a_{T-k}, a_{T-k+1}, \cdots, a_T\}$$

where the last k actions match those of our algorithm (actions $a_{T-k+1}, \cdots, a_T$), and the first $T-k$ match the $(T-k+1)$-th action. Our inductive hypothesis is that the reward of such a sequence is at least as much as the reward of the best fixed-action sequence which maximized the perturbed reward.

With the above, we show how to construct a sequence of the same form, but where the last $k+1$ actions match those of our algorithm. The sequence is given by:

$$a'_t = a_t \quad \text{for } t > T-k \qquad \text{(The last } k \text{ actions match)}$$

$$a'_{T-k} = \arg\max_{a \in A}\{X_a + \sum_{u=1}^{T-k} r^u(a)\}$$

(The $k+1$-th action from the end matches our algorithm)

$$a'_t = a_{T-k} \quad \text{(All other actions are the same and match the } (T+k)\text{-th action)}$$

Then note that the total reward for this sequence is given as:

$$\begin{aligned}
\sum_{t=1}^{T} r^t(a'_t) &= \sum_{t=1}^{T-k} r^t(a'_t) + \sum_{t=T-k+1}^{T} r^t(a'_t) && \text{(Definition of total reward)} \\
&= \sum_{t=1}^{T-k} r^t\left(\arg\max_{a \in A}\left\{X_a + \sum_{u=1}^{T-k} r^u(a)\right\}\right) + \sum_{t=T-K+1}^{T} r^t(a_t) && \text{(Our new sequence)} \\
&\geq \sum_{t=1}^{T-k} r^t(a_{T-k+1}) + \sum_{t=T-K+1}^{T} r^t(a_t) && \text{(See below)} \\
&= \sum_{i=1}^{T} r^t(a_t) && \text{(We've recovered the previous sequence)} \\
&\geq \sum_{i=1}^{T} r^t(a^*) && \text{(By the inductive hypothesis)}
\end{aligned}$$

To see why we can go from line (2) to (3), note that we are selecting the best fixed-action maximizing the perturbed reward up to step $T-k$. This reward must therefore by the largest perturbed reward possible for any fixed-action sequence, include the fixed-action sequence given by a_{T-k+1} (matching our previous sequence).

As such, the above shows how to contruct a sequence of actions $\{a'_t\}$ where the last $k+1$ actions match our algorithm's sequence and where the remaining actions match a'_{T-k} such that the total reward is no worse then the reward of the best fixed-action sequence maximizing the total perturbed reward. □

From the above, we can conclude that our algorithm's regret is bounded above by the regret of the best-fixed action sequence maximizing the total perturbed reward.

Lemma 0.7. *The best fixed-action sequence maximizing the total perturbed reward has regret bounded above by* $\max_{a \in A} X_a$.

Proof. We can proof this directly. The regret is given as:

$$\underbrace{\max_{a\in A}\sum_{t=1}^{T} r^t(a)}_{\text{best fixed action}} - \underbrace{\sum_{t=1}^{T} r^t\left(\arg\max_{a\in A}\{X_a + \sum_{u=1}^{T} r^t(a)\}\right)}_{\text{reward for best fixed action maximizing perturbed reward}} = \max_{a\in A}\sum_{t=1}^{T} r^t(a) - \max_{a\in A}\{X_a + \sum_{u=1}^{T} r^t(a)$$

$$\leq \max_{a\in A} X_a$$

□

Putting the two results above together, we arrive at the conclusions that:

Lemma 0.8. *The regret of our proposed algorithm is bounded above by* $\max_{a\in A} X_a$.

(c) We were not able to follow the hint, so this is probably more complicated than it needs to be. However, we nonetheless arrived at the desired bounds. Note that we proof that $E[\max_{a\in A} X_a] \leq \frac{1}{\eta}[b \ln n + 1]$ (as suggested in Piazza). We have:

$$\begin{aligned}
E[\max_{a\in A} X_a] &= \sum_{k=0}^{\infty} P[\max_{a\in A} X_a \geq k] && \text{(Definition of expectation for non-negative r.v)}\\
&= \sum_{k=0}^{\infty}(1 - P[\max_{a\in A} X_a < k]) && \text{(Complement)}\\
&= \sum_{k=0}^{\infty}(1 - P[X_a < k]^n) && \text{(All } X_a \text{ must be less than } k\text{, and they are IID)}\\
&= \sum_{k=0}^{\infty}(1 - (1 - P[X_a \geq k])^n) && \text{(Complement)}\\
&= \sum_{k=0}^{\infty}(1 - (1 - (1-\eta)^k)^n) && \text{(only if first } k \text{ flips are tails)}
\end{aligned}$$

Now, there's no nice close-formed solution for the above, so we take a pretty heavy approach and instead seek to upper-bound it by the corresponding integral, as fol-

lows:

$$\begin{aligned}
\sum_{k=0}^{\infty}(1-(1-(1-\eta)^k)^n) &\leq 1+\int_0^{\infty}(1-(1-(1-\eta)^x)^n)dx \\
&= 1+\frac{1}{\ln\frac{1}{(1-\eta)}}\int_0^1\frac{(1-u^n)}{1-u}du \quad \text{(Using } u=1-(1-\eta)^x\text{)} \\
&= 1+\frac{1}{\ln\frac{1}{(1-\eta)}}\int_0^1\sum_{k=0}^{n-1}u^k du \\
&\qquad \text{(Replacing fraction by corresponding geometric series given bounded } u\text{)} \\
&= 1+\frac{1}{\ln\frac{1}{(1-\eta)}}\sum_{k=0}^{n-1}\int_0^1 u^k du \\
&\qquad \text{(Nice functions, can swap sum/integration)} \\
&= 1+\frac{1}{\ln\frac{1}{(1-\eta)}}\sum_{k=0}^{n-1}\frac{1}{k+1} \\
&= 1+\frac{1}{\ln\frac{1}{(1-\eta)}}\sum_{k=1}^{n}\frac{1}{k} \qquad \text{(Re-index)} \\
&\leq 1+\frac{1}{\ln\frac{1}{(1-\eta)}}\ln n \\
&\qquad \text{(Harmonic series is upper-bounded by } \ln n\text{)}
\end{aligned}$$

We now further simplify the above. The first claim we make is that $\frac{1}{\ln\frac{1}{1-\eta}} \leq \frac{1}{\eta}$ for $\eta \in [0,1]$. This follows from the fact that $\eta \leq \ln\frac{1}{1-\eta}$ for $\eta \in [0,1]$. Note that we can exponentiate and re-arrange to give us the inequality $(1-\eta)e^{\eta} \leq 1$ on $\eta \in [0,1]$, which holds since looking at the boundaries and any minimum/maximum, we have the extra at $\eta = 0$ which gives 1 and $\eta = 1$ which gives 0. As such, we can rewrite the above as:

$$\begin{aligned}
E[\max_{a\in A} X_a] &\leq 1+\frac{1}{\eta}\ln n \\
&= \frac{1}{\eta}[\eta+\ln n] \\
&\leq \frac{1}{\eta}(1+\ln n)
\end{aligned}$$

which is precisely the inequality we wanted to show, where we have $b = 1$ as a constant indepdent of η and n.

(d) For this problem, we assume that we know T in advance (though this is not necessary). Note that at each time-step, with probability at least $1-\eta$, the action chosen which maximises the perturbed reward up to $t-1$ will be the same as the action that

maximized the perturbed reward up to t, assuming that our rewards are in $[0, 1]$. This is because, as per (a), at each timestep the maximal perturbed reward and the second maximal perturbed reward, with probaility at least $1 - \eta$, vary by more than 1. Since the rewards are limited to $[0, 1]$, this difference cannot be overcome in a single step. As such, the action leading to the maximal perturbed reward will not change from $t - 1$ to t.

For these timesteps, FTPL will therefore produce an action equivalent to the algorithm in (b) (e.g, we get lucky and the optimal action stays the same). Otherwise, we can't say much about the action produced.

Putting this together, we can calculated the worst-case expected regret of the FTLP algorithm as:

$$\begin{aligned} E[\text{FTLP regret}] &= E[(b)\text{ regret}](1-\eta) + E[\text{otherwise regret}]\eta \\ &\leq \underbrace{\frac{1}{\eta}(1+\ln n)}_{\text{upper bound regret on (b)}}(1-\eta) + \underbrace{2T}_{\text{worst possible regret}}\eta \\ &\leq \frac{1}{\eta}(1+\ln n) + 2T\eta \qquad \text{(Simple upperbound)} \end{aligned}$$

Since we know T in advance, we can pick a good choice of η in order to achive the bound we want.

In fact, let us select:

$$\eta = \sqrt{\frac{\ln n}{T}}$$

Note that this is valid selection of η for sufficiently large T. If not, simply pick some value larger than T to satisfy the constraints on η. Then our regret is given as:

$$\begin{aligned} E[\text{FTLP regret}] &\leq \frac{1}{\eta}(1+\ln n) + 2T\eta \\ &= \sqrt{\frac{T}{\ln n}} + \sqrt{T\ln n} + 2\sqrt{T\ln n} \\ &\leq \sqrt{T\ln n} + \sqrt{T\ln n} + 2\sqrt{T\ln n} \\ &= 4\sqrt{T\ln n} \end{aligned}$$

As such, we have $b = 4 > 0$ and the bounds as desired.

Problem 15

Solution:

(a) Let us consider the deterministic algorithms A_i as described in the problem. What is the competitive ratio for each? If $i \leq B$, then the competitive ratio is given by:

$$\frac{B + (i-1)}{i} = 1 + \frac{B-1}{i}$$

Since in the worst case, we get exactly i assignments, so we end-up buying the on the i-th assinment after renting $i-1$ times where the optimal would have been to just rent i times. If instead we have $i > B$, then the competitive ration is given by:

$$\frac{B + (i-1)}{B} = 1 + \frac{i-1}{B}$$

Since in the worst case, we get exactly i assignments, so we end-up buying on the i-th assignment after renting $i-1$ times where the optimal would have been to just buy immediately.

From the above, we can see by inspection that the optimal deterministic algorithm is given by A_B, which will have a competitive ratio of:

$$1 + \frac{B-1}{B} = 2 - \frac{1}{B} < 2$$

as desired.

(b) We basically did this when working on (a), so please refer to that. We've seen that for A_i, the competitive ration is given by:

$$\frac{ALG(A_i)}{OPT} = \begin{cases} 1 + \frac{B-1}{i} & i \leq B \\ 1 + \frac{i-1}{B} & i > B \end{cases}$$

Note that the first case is minimized when $i = B$, and similarly the second case as well. As such, a lower bound on the above function occurs at $i = B$ and is given by $2 - \frac{1}{B}$.

(c) As described, let us suppose that Alice gets j assignments. Then the expected cost of the randomized algorithm is given by:

$$\begin{aligned} E[\text{cost}] &= \sum_{i=1}^{\infty} p_i \text{Cost}(A_i) \\ &= \sum_{i=1}^{j} p_i[B + (i-1)] + j \sum_{i=j+1}^{\infty} p_i j \end{aligned}$$

The first line just comes from the definition of expected value. In the second line, we simply consider two sets of cases. In the first, we consider all algorithm which end-up buying the camerate, since $i \leq j$. These have cost B (to purchase the camera) plus $(i-1)$ for the number of rentals. The second set of cases considers all algorithm which never buy, and instead just rent, since $i > j$. These algorithm just have cost j for renting j times.

We now let c denote the competitive ratio of this randomized algorithm. First, we note that since OPT is a constant (it is completely defined by the value of j, since if $j > B$, then OPT is B, otherwise OPT is j), the competitive ratio will be a linear compation of the p_i as shown in the above expression computing the expected cost of the algorithm.

As such, we can write the LP P directly as:

$$\text{minimize} \quad c \tag{1}$$

subject to

$$\sum_{i=1}^{j} p_i[B + (i-1)] + j \sum_{i=j+1}^{\infty} p_i \leq c \min\{j, B\} \quad \text{for all } j \geq 1 \tag{2}$$

$$\sum_{i \geq 1} p_i = 1 \tag{3}$$

$$p_i \geq 0 \quad \text{for all } i \geq 1 \tag{4}$$

The constraints for $j \geq 1$ exists to guarantee that whatever minimum optimality ratio we find c, this ratio is the largest ratio for all possible assignments which Alice could receive. This is exactly what the definition of finding the optimal competitive ration requires.

(d) WLOG, an optimal randomized strategy has $p_i = 0$ for all $i > B$. This is because there exists no scenario (value of j) where these algorithms offer a lower cost. If Alice is ever in a situation where she buys the camera at time $i > B$, then it would have been strictly better for her to have bought the camerate at time B. As such, for $i > B$, A_B is a strictly dominant algorithm over A_i and so no optimal randomized algorithm would place any probability distribution on A_i.

Similarly, we only need to write constraints for assignments $j \leq B$. Note that all constraints where $j > B$ will be bounding the expected cost of our algorithm by cB. Furthermore, using the fact that $p_i = 0$ for al l$i > B$, the second summation in constraints with $j > B$ will disappear, meaning that all constrains where $j > B$ are of the form:

$$\sum_{i=1}^{D} p_i[B + (i-1)] \leq cB$$

Which is exactly the from we expect for the constraint $j = B$. As such, we don't need any of these additional constraints.

Putting the above together, we have that our linear program is bounded, on the variables $\{p_i\} \cup \{c\}$ for $1 \leq i \leq B$ given:

$$\text{minimize} \quad c \tag{5}$$

subject to

$$\sum_{i=1}^{j} p_i[B + (i-1)] + j \sum_{i=j+1}^{B} p_i \leq cj \quad \text{for all } 1 \leq j \leq B \tag{6}$$

$$\sum_{i=1}^{B} p_i = 1 \tag{7}$$

$$p_i \geq 0 \quad \text{for all } 1 \leq i \leq B \tag{8}$$

$$c \in \mathbb{R} \tag{9}$$

From the above we can read-off that we have $B + 1$ variables given by $\mathbf{y}^T = [p_1, \cdots, p_B, c] \in \mathbb{R}^{B+1}$ with $B + 1$ constraints ($B \leq$ and one $=$). Our right hand-side is given by $\mathbf{c}^T = [0, \cdots, 0, 1] \in \mathbb{R}^{B+1}$, with our objective function $\mathbf{b}^T = [0, \cdots, 0, 1] \in \mathbb{R}^{B+1}$, and our constrain matrix $\mathbb{A} \in \mathbb{R}^{(B+1)\times(B+1)}$ is given by:

$$\mathbf{A} = \begin{bmatrix} B & 1 & 1 & 1 & \cdots & 1 & 1 & -1 \\ B & B+1 & 2 & 2 & \cdots & 2 & 2 & -2 \\ B & B+1 & B+2 & 3 & \cdots & 3 & 3 & -3 \\ B & B+1 & B+2 & B+3 & \cdots & 4 & 4 & -4 \\ & & & & \vdots & & & \\ B & B+1 & B+2 & B+3 & \cdots & B+B-2 & B-1 & -(B-1) \\ B & B+1 & B+2 & B+3 & \cdots & B+B-2 & B+B-1 & -B \\ 1 & 1 & 1 & 1 & \cdots & 1 & 1 & 0 \end{bmatrix}$$

(e) Taking the dual using the formula, we have the following program with $B + 1$ constraints and $B + 1$ variables $\{q_j\} \cup \{r\}$.

$$\text{maximize} \quad r \tag{10}$$

subject to

$$\sum_{j=1}^{i-1} jq_j + (B + i - 1) \sum_{j=i}^{B} q_j \leq -r \quad \text{for all } 1 \leq i \leq B \tag{11}$$

$$\sum_{j=1}^{B} -jq_j = 1 \tag{12}$$

$$r \in \mathbb{R} \tag{13}$$

$$q_j \leq 0 \quad \text{for all } 1 \leq j \leq B \tag{14}$$

For interpretability, we rewrite by subtituing all the variables q_j with their negation as $y_j = -q_j$.

$$\text{maximize} \quad r \tag{15}$$

subject to

$$\sum_{j=1}^{i-1} j y_j + (B + i - 1) \sum_{j=i}^{B} y_j \geq r \quad \text{for all } 1 \leq i \leq B \tag{16}$$

$$\sum_{j=1}^{B} j y_j = 1 \tag{17}$$

$$r \in \mathbb{R} \tag{18}$$

$$y_j \geq 0 \quad \text{for all } 1 \leq j \leq B \tag{19}$$

(f) In order to argue that the dual has the provided interpretation, we first construct the probability distribution $\mathcal{D}$ on the number of assignments from the solutions to the dual, $\{y_j\}$. If fact, the probability q_j that Alice gets exactly j assignments for $1 \leq j \leq B$ is given by:

$$q_j = j y_j$$

where all other assignments are given 0 probability. Note that this is a valid probability distribution since by 17, we have that $\sum_{j=1}^{B} q_j = 1$. Let us now consider the expected values of the ratio of the cost of A_i to the optimal cost in hindsight of some fixed algorithm A_i over this distribution of values number of assignments.

$$\begin{aligned}
E[\text{ratio for algorithm } A_i] &= \sum_{j=1}^{B} q_j \frac{\text{Cost}(A_i(j))}{OPT(j)} \\
&\qquad \text{(Taking expectation over realized values for } j\text{)} \\
&= \sum_{j=1}^{B} q_j \frac{\text{Cost}(A_i(j))}{j} \\
&\qquad \text{(The optimal in hindsigh is always } j\text{)} \\
&= \underbrace{\sum_{j=1}^{i-1} q_j \frac{\text{Cost}(A_i(j))}{j}}_{\text{always rent}} + \underbrace{\sum_{j=i}^{B} q_j \frac{\text{Cost}(A_i(j))}{j}}_{\text{rent for } i-1 \text{ then buy}} \\
&= \sum_{j=1}^{i-1} q_j \frac{j}{j} + \sum_{j=i}^{B} q_j \frac{(B + i - 1)}{j} \\
&= \sum_{j=1}^{i-1} j y_j + (B + i - 1) \sum_{j=1}^{B} y_j
\end{aligned}$$

Note that the above equation exactly matches the LHS of the i-th constraint in our dual (see 11). Combined with the RHS inequality that the above must be $\geq r$ for all i, our dual is solving for a lower bound on all possible expected ratios. As such, the resulting value of the dual given by r will be the minimum of these ratios for all algorithms A_i.

(g) We handle the solution in it's full generality using the matrix A, as given below, which contains the first B constraints.

$$\mathbf{A} = \begin{bmatrix} B & 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ B & B+1 & 2 & 2 & \cdots & 2 & 2 & 2 \\ B & B+1 & B+2 & 3 & \cdots & 3 & 3 & 3 \\ B & B+1 & B+2 & B+3 & \cdots & 4 & 4 & 4 \\ & & & & \vdots & & & \\ B & B+1 & B+2 & B+3 & \cdots & B+B-2 & B-1 & (B-1) \\ B & B+1 & B+2 & B+3 & \cdots & B+B-2 & B+B-1 & B \end{bmatrix}$$

First, subtract each row from the row below. This gives:

$$\mathbf{A} = \begin{bmatrix} B & 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ 0 & B & 1 & 1 & \cdots & 1 & 1 & 1 \\ 0 & 0 & B & 1 & \cdots & 1 & 1 & 1 \\ 0 & 0 & 0 & B & \cdots & 1 & 1 & 1 \\ & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \cdots & B & 1 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & B & 1 \end{bmatrix}$$

Then subtract each from the the row above, giving:

$$\mathbf{A} = \begin{bmatrix} B & 1-B & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & B & 1-B & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & B & 1-B & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & B & \cdots & 0 & 0 & 0 \\ & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \cdots & B & 1-B & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & B & 1 \end{bmatrix}$$

The above gives the following system of equalities:

$$\begin{aligned} Bp_1 &= (B-1)p_2 \\ Bp_2 &= (B-1)p_3 \\ &\vdots \\ Bp_B &= (B-1)p_{B-1} \\ Bp_B &= c \end{aligned}$$

Noticing the recursion and subtituting untl the base case, we have:

$$p_i = \left(\frac{B-1}{B}\right)^{B-i} \frac{c}{B}$$

Recalling the final constraint where $\sum_{i=1}^{B} p_i = 1$, we can take the sum along both sides to get that:

$$\begin{aligned}
1 &= \frac{c}{B}\sum_{i=1}^{B}\left(\frac{B-1}{B}\right)^{B-i} \\
\implies c &= \frac{B}{\sum_{i=1}^{B}\left(\frac{B-1}{B}\right)^{B-i}} \\
&= \frac{B}{\sum_{i=0}^{B-1}\left(\frac{B-1}{B}\right)^{i}} && \text{(Re-indexing to geometric series)} \\
&= \frac{B}{\frac{1-(1-\frac{1}{B})^{B}}{1-(1-\frac{1}{B})}} && \text{(Closed form of geometric series)} \\
&= \frac{B}{\frac{1-(1-\frac{1}{B})^{B}}{\frac{1}{B}}} \\
&= \frac{1}{1-(1-\frac{1}{B})^{B}} \\
&\approx \frac{1}{1-e^{-1}} && \text{(For large enough } B \text{ since } (1+\tfrac{1}{x})^{x} = e^{x} \text{ for large } x) \\
&= \frac{e}{e-1}
\end{aligned}$$

From the above, the best possible competitive ratio is bounded by:

$$c \geq \frac{e}{e-1} \approx 1.582$$

and is achived when we set p_i as:

$$p_i = \left(\frac{B-1}{B}\right)^{B-i} \frac{e}{B(e-1)}$$

Problem 16

Solution:

(a) $OPT = n$ for every such instance since we can match the i-th vertex of R with the i-th vertex of L. Note that this edge always exists, since we know that the i-th vertex of R is connected to the last $n - i + 1$ vertices of L. Furthermore, the i-th vertex in the LHS will only ever be matched once, since the i-th vertex of R does not have any edges to the first i vertices of L.

As such, $OPT = n$ since the matching achieved has size n.

(b) This is straight-forward to show. Consider the i-th vertex of L (according to π). Define the event E_{ij} as occuring if the j-th vertex of R matches to the i-th vertex of L. Then we have:

$$\begin{aligned}
\Pr[i \text{ is matched}] &= \Pr[\cup_{j=1}^{n} E_{ij}] && \text{(Any one matching means the vertex matched)} \\
&\leq \sum_{j=1}^{n} \Pr[E_{ij}] && \text{(Union bound)} \\
&= \sum_{j=1}^{i} \Pr[E_{ij}] + \sum_{j=i+1}^{n} \Pr[E_{ij}] \\
& && \text{(Splitting into vertices before } i \text{ and after } i\text{)} \\
&= \sum_{j=1}^{i} \Pr[E_{ij}] \\
& \text{(All vertices after the } j = i\text{-th only have vertices to nodes in } L \text{ that are not } i\text{)} \\
&\leq \sum_{j=1}^{i} \frac{1}{n-j+1}
\end{aligned}$$

The last line is due to the fact that the j-th vertex is connected to $n - j + 1$ vertices in L, one of which we know is i. As such, the probability of E_{ij} is at most $\frac{1}{n-j+1}$ (there could be a case where i is already taken). Noting the other trivial bound on a probability, we have:

$$\Pr[i \text{ is matched}] \leq \min\left\{\sum_{j=1}^{i} \frac{1}{n-j+1}, 1\right\}$$

(c) We can prove this directly by taking the expected value. Note that the expected size of the matching can be counted by letting L_i be a random indicator variable, which is 1 if the i-th vertex (in some random ordering π) is matched. The expectation is

taken over all possible orderings of π. Then the size of the matching is given simply by: $\sum_{i=1}^{n} L_i$:

$$\begin{aligned} E_\pi[\sum_{i=1}^{n} L_i] &= \sum_{i=1}^{n} E_\pi[L_i] && \text{(Linearity of expectation)} \\ &= \sum_{i=1}^{n} \Pr_\pi[L_i = 1] && \text{(Expectation of indicator variable)} \\ &\leq \sum_{i=1}^{n} \min\left\{\sum_{j=1}^{i} \frac{1}{n-j+1}, 1\right\} && \text{(From results in (b))} \end{aligned}$$

(d) We extend following the hint. Suppose we are given some randomized online algorithm **A**. Consider the set of all possible **deterministic** online algorithms, $D = \{A_d\}$. Now, our randomized algorithm **A** is simply a distribution over all the deterministic algorithms A_d (we can flip all the necessary randomness for A in advance, which yields a deterministic algorithm). Let p_d be the probability (over A's internal coin flips) that A_d is the selected deterministic algorithm (if we flip coins in advance).

Let M be a random variable distributed over π and A's internal coin flips that specifies the size of the matching produced by A. Then we simply have:

$$\begin{aligned} E_{\pi,A}[M] &= E_A[E_\pi[M \mid A = A_d]] && \text{(Law of Total Expectation)} \\ &= \sum_{A_d \in D} p_d E_\pi[M \mid A = A_d] && \text{(Definition of expectation)} \\ &\leq \sum_{A_d \in D} p_d \sum_{i=1}^{n} \min\left\{\sum_{j=1}^{i} \frac{1}{n-j+1}, 1\right\} \\ & && \text{(Using results from (c) since } A_d \text{ is some deterministic algorithm)} \\ &= \sum_{i=1}^{n} \min\left\{\sum_{j=1}^{i} \frac{1}{n-j+1}, 1\right\} \sum_{A_d \in D} p_d \\ & && \text{(Note the inner summation did not depend on } A_d) \\ &= \sum_{i=1}^{n} \min\left\{\sum_{j=1}^{i} \frac{1}{n-j+1}, 1\right\} \\ & && (\textstyle\sum_{A_d \in D} p_d = 1 \text{ since it is a valid probability distribution}) \end{aligned}$$

As such, we have shown that:

$$E_{\pi,A}[M] \leq \sum_{i=1}^{n} \min\left\{\sum_{j=1}^{i} \frac{1}{n-j+1}, 1\right\}$$

as desired.

(e) Let A be some (possibly randomized) online algorithm. From part (d) above, we've shown that the expected value (over π and the internal coin flips of A) of matches (the size of the output of A) is at most $n(1 - \frac{1}{e})$ (for sufficiently large n, see part (c) for the limit).

TODO

Problem 17

Solution:

(a) We show that such a strong statement is not true by providing a clear counter-example. In our case, the counter-example consists of a request sequence $\sigma = (\sigma_1, \cdots, \sigma_m)$ where and $1 \leq k \leq m$ such that $C_{MTF}(\sigma_k) > 2C_{OPT}(\sigma_k)$.

Proof. Let our request sequence be $\{\sigma_1, \sigma_2, \sigma_3, \sigma_1\}$ □

(b) We can show this directly. See:

$$\begin{aligned}
C_{MFT}(\sigma) &= \sum_{k=1}^{m} C_{MFT}(\sigma_k) && \text{(Definition)} \\
&\leq 2\sum_{k=1}^{m} C_{OPT}(\sigma_k) + \Phi_{k-1} - \Phi_k && \text{(Equation (2) given)} \\
&= 2\sum_{k=1}^{m} C_{OPT}(\sigma_k) + 2\sum_{k=1}^{m} \Phi_{k-1} - \Phi_k \\
&= 2\sum_{k=1}^{m} C_{OPT}(\sigma_k) + 2(\Phi_0 - \Phi_m) && \text{(Telescoping sum)} \\
&= 2\sum_{k=1}^{m} C_{OPT}(\sigma_k) - 2\Phi_m && (Phi_0 = 0) \\
&\leq 2\sum_{k=1}^{m} C_{OPT}(\sigma_k) && (\Phi_m \geq 0) \\
&= 2C_{OPT}(\sigma) && \text{(Definition)}
\end{aligned}$$

This is precisely what we desired to show.

(c) The fact that $\Phi_0 = 0$ follows from the fact that MFT and OPT begin with the same list, and as such there are no inverstions. The fact that $\Phi \geq 0$ follows from the fact that Φ counts inversions, and the count can never be negative.

Given the provided definitions, this is actually rather straight forward. We let S and T be defined as in the hint. Then we proof the inequality directly. First note that:

$$C_{MFT}(\sigma_k) = |S| + |T| + 1$$

This is because the cost of servicing the request σ_k is determined by the current position of σ_k in MFT's list. Note that this is given by $|S|$ (the number of elements preceeding σ_k in both lists) plus $|T|$ (the number of elements preceeding σ_k in only the MFT list) plus 1 (the cost to access the element itself).

We also have:

$$1 + |S| \leq C_{OPT}(\sigma_k)$$

This is because $|S|$ counts the number of elements preceesding σ_k in both lists, which is a lower bound on the cost of accessing σ_k. The $+1$ handles the fact that we access the element itself.

Lastly, we have:

$$\Phi_k - \Phi_{k-1} = |S| - |T|$$

To see why, note that $\Phi_k - \Phi_{k-1}$ counts the change in the number of inversions introduced by servicing request σ_k. In servicing σ_k, our algorithm will move the element to the front of the list. As such, it will now have $|S|$ more inversions (for $y \in S$, we have that y preceeds σ_k in OPT but will no longer in MFT since σ_k is now at the head). We will also decrease the number of inversions by $|T|$ since for each $y \in T$, y will no longer preceed σ_k in MFT which is the same situation it finds itself in OPT.

Putting the above together, we have:

$$\begin{aligned} C_{MFT}(\sigma_k) + (\Phi_k - \Phi_{k-1}) &= |S| + |T| + 1 + |S| - |T| && \text{(See above)} \\ &= 2|S| + 1 \\ &\leq 2(|S| + 1) \\ &\leq 2C_{OPT}(\sigma_k) && \text{(See above)} \end{aligned}$$

This proofs equation (2). From the work in (b), this establishes that our algorithm is 2-competitive.

(d) We show that for any $\epsilon > 0$, the competative ratio of A is at least $\frac{2n}{n+1} - \epsilon$ on a request sequence of length n.

Proof. Fix some deterministic algorithm A. Then construct the sequence of lenght m as follows. At each k, note that there is some element that is at the end of the algorithm's current list (we can know which element since we have access to A and A is deterministic). As such, simply set the request σ_k to be the last element.

It is immediate that the total cost of the algorithm A is mn, since on each request, we must access the last element at cost n.

Now let us consider the optimal algorithm. Note for some random permutation of the list, the expected cost would be given by $\frac{m(n+1)}{2}$, since we expect each element to be at position $\frac{n+1}{2}$ and will cost as much to access on each request. Since the permutation described above achieves a value higher than $\frac{n+1}{2}$ (our expected value), then we know there must exists **some** permutation that achives a value less than $\frac{n+1}{2}$.

As such, the competative ration is given by:

$$\begin{aligned}\frac{ALG(A)}{OPT} &= \frac{mn}{m^{\frac{(n+1)}{2}}} \\ &= \frac{2n}{n+1} \\ &> \frac{2n}{n+1} - \epsilon\end{aligned}$$

For any $\epsilon > 0$, as desired. □

Problem 18

Solution:

(a) We deterministic algorithm is given as follows.

- For timesteps $t = 0, 1, \cdots$, submit bid $b_t = 2^t$.
- If the bid b_t is rejected, continue to $t = t + 1$ and attempt again. Otherwise, stop.

Note that it is clear that the algorithm always buys the painting (the bids keep strictly increasing until a bid is accepted).

Lemma 0.9. *The algorithm provided above buys the painting for at most* $4p$.

Proof. Suppose the target price is fixed as $p \geq 1$. Then note that our algorithm will succeed ($b_t \geq p$) at timestep $t = \lceil \log_2 p \rceil$, since we will have $b^t = 2^{\lceil \log_2 p \rceil} \geq 2^{\log_2 p} = p$. As such, the amount our algorithm will pay will be:

$$\begin{aligned}
\sum_{t=0}^{\lceil \log_2 p \rceil} b_t &= \sum_{t=0}^{\lceil \log_2 p \rceil} 2^t \\
&= 2^{\lceil \log_2 p \rceil + 1} - 1 && \text{(Summation of geometric series)} \\
&\leq 2^{2 + \log_2 p} - 1 && (\lceil \log_2 p \rceil \leq \log_2 p + 1) \\
&= 2^2 2^{\log_2 p} - 1 \\
&= 4p - 1
\end{aligned}$$

As such, we conclude that:

$$\sum_{t=0}^{\lceil \log_2 p \rceil} b_t \leq 4p$$

□

(b) Let us consider the deterministic algorithms of the from A_c where instead of starting our initial bid at 1, we instead start our initial bid at c. Note that the algorithm in (a) is simply A_1.

Lemma 0.10. *The deterministic algorithm* A_c *buys the painting paying at most* $c + 4p\left(\frac{c}{2^{\lceil \log_2 c \rceil}}\right)$.

Proof. Note that the above holds for A_1, since we showed in (a) that this algorithm pays at most $4p \leq 1 + 4p$. In the more general case, the argument is as follows:

$$\text{amound paid by } A_c = c + 2c + 2^2c + 2^3c + \cdots + 2^{\lceil \log_2 \frac{p}{c} \rceil} c$$

(This are just the bids the algorithm submits)

$$= c + 2c \sum_{k=0}^{\lceil \log_2 \frac{p}{c} \rceil - 1} 2^k$$

(We alway bid c, and the rest is just writing in summation notation)

Now that we know the amount we paid for an algorith A_c, let us define a distribution over the values of c where each value is assigned $p_c \in [0, 1]$ for all $c \geq 1$. Taking the expectaion over this distribution, we have the expected cost of our randomized algorithm as:

$$E[\text{cost of randomized algorithm}] = \sum_{c=1}^{\infty} p_c \text{Cost}(A_c) \quad \text{(Definition of expected value)}$$

$$= \sum_{c=1}^{\infty} p_c \left[c + 2c \sum_{k=0}^{\lceil \log_2 \frac{p}{c} \rceil - 1} 2^k \right]$$

$$= \sum_{c=1}^{\infty} p_c c + 2 \sum_{c=1}^{\infty} \left[p_c c \sum_{k=0}^{\lceil \log_2 \frac{p}{c} \rceil - 1} 2^k \right]$$

$$= \sum_{c=1}^{\infty} p_c c + 2 \sum_{c=1}^{p} \left[p_c c \sum_{k=0}^{\lceil \log_2 \frac{p}{c} \rceil - 1} 2^k \right]$$

(The inner summation is 0 for $c > p$ since we win on the first bid)

The goal is to have the expression above be $\leq 3p$ for some choice of p_c. Let M be some sufficiently large integer value which we will describe in more detail below. Then define let $p_c = \frac{1}{M}$ for $c \in \{1, 2, \cdots, M\}$. Plugging this into the above, we have

the expression:

$$\begin{aligned}
E[\text{cost of randomized algorithm}] &= \sum_{c=1}^{\infty} p_c c + 2\sum_{c=1}^{p}\left[p_c c \sum_{k=0}^{\lceil \log_2 \frac{p}{c} \rceil - 1} 2^k\right] \\
&= \frac{1}{M}\sum_{c=1}^{M} c + \frac{2}{M}\sum_{c=1}^{p} c(2^{\lceil \log_2 \frac{p}{c} \rceil} - 1) \\
&\leq \frac{1}{M}\sum_{c=1}^{M} c + \frac{4p}{M}\sum_{c=1}^{p} 1) \\
&= \frac{1}{M}\frac{M(M+1)}{2} + \frac{4p^2}{M} \\
&= \frac{M+1}{2} + \frac{4p^2}{M}
\end{aligned}$$

We therefore pick $M = 2p\sqrt{2} >> p$, which gives us an upper bound of:

$$\begin{aligned}
E[\text{cost of randomized algorithm}] &\leq \frac{M+1}{2} + \frac{4p^2}{M} \\
&= p\sqrt{2} + \sqrt{2}p \\
&\leq 3p
\end{aligned}$$

□

TODO.