Problem 1

I prove that MTF-every-other is $O(1)$-competitive. I use the potential function $\Phi = \sum_x N(x)I(x)$ where $N(x)$ is the number of inversions between MTF-every-other and OPT with x last and $I(x) = 2$ if x is in an odd request phase and $I(x) = 1$ if x is in an event request phase.

Suppose that item x is requested. Suppose that item x is at position k in MTF-every-other and at position j in OPT. Let v be the number of extra items before x in MTF-every-other compared to OPT. So the number of items before x in both OPT and MTF-every-other is $k - v - 1$, so $k - v - 1 \leq j - 1$ so $k - v \leq j$.

Now, consider the move of MTF-every-other.

If x is in an even request phase, then x will be moved to the front, creating $k-v-1$ new inversions (and each might increase the potential by 2) and cancelling v versions (each cancellation decreasing the potiential by 1 since $I(x) = 1$). So the amortized cost is $\leq k + 2(k - v - 1) - v = 3k - 3v - 1 \leq 3j$.

If x is in an odd request phase, then x will not be moved to the front, but $I(x)$ will go from 2 to 1. So the potiential will decrease by v. Thus the amortized cost is $= k - v = j$.

As seen in class, the move of OPT can only decrease the potential and so reduce the amortized cost.

Thus, MTF-every-other is $O(1)$-competitive.

Problem 3

Part (a)

I prove that when the online algorithm makes a mistake, the total weight of the faculty decreases by a factor of 4/3, at least. Let F denote the fraction of the weight of the faculty with the wrong answer. Since the online algorithm makes a mistake, $F \geq 1/2$. Let T be the total weight of the faculty before the correction. Then, T', the total weight of the faculty after the correction, is $T(\frac{1}{2}F + (1-F)) = T(1 - \frac{1}{2}F)$. Thus, $T/T' = \frac{1}{1-\frac{1}{2}F} \geq 4/3$ since $F \geq 1/2$.

Thus, if the algorithm does k mistakes, the total weight assigned to faculty is $\leq (\frac{3}{4})^k n$.

Part (b)

I lower-bound the weight assigned to faculty by considering the weight of the wisest faculty member in terms of m, which is $(\frac{1}{2})^m$. So the total weight assigned to faculty is $\geq (\frac{1}{2})^m$.

Part (c)

Setting the lower bound equal to the upper bound and solving for k gives us the maximum number of mistakes our algorithm can make.

$$\begin{aligned}
\frac{1}{2}^m &= \frac{3}{4}^k n \\
m \log \frac{1}{2} &= k \log \frac{3}{4} + \log n \\
-m - \log n &= k \log \frac{3}{4} \\
k &= \frac{1}{\log \frac{4}{3}}(m + \log n) \\
k &= 2.41(m + \log n)
\end{aligned}$$

Part (d)

As in part (a), the expression for the multiplicative change in the total weight as a result of reweighting for a question is $T/T' = \frac{1}{1+(\beta-1)F}$.

Part (e)

As in part (c), the total weight assigned to faculty is $\geq \beta^m$. Let F_i be the fraction of the weight of faculty with the wrong answer at the ith question. Then the total weight assigned to faculty is $\leq n \prod_i 1 + (\beta - 1)F_i$. Setting the

lower bound to the upper bound, and noticing that $\sum_i F_i = k$ (where k is the expected number of mistakes) yields,

$$\begin{aligned}
\beta^m &= n \prod_i 1 + (\beta - 1)F_i \\
m \ln \beta &= \sum_i \ln(1 + (\beta - 1)F_i) + \ln n \\
m \ln \beta &\leq \sum_i (\beta - 1)F_i + \ln n \\
m \ln \beta &\leq (\beta - 1)k + \ln n \\
m \ln \frac{1}{\beta} + \ln n &\geq (1 - \beta)k \\
k &\leq \frac{m \ln \frac{1}{\beta} + \ln n}{1 - \beta}
\end{aligned}$$

Thus, the expected number of wrong answers will be at most $\frac{m \ln \frac{1}{\beta} + \ln n}{1-\beta}$.

Part (f)

Using the inequality $\ln(1 + x) \leq x, \forall x > 0$, I upper bound k:

$$\begin{aligned}
k &\leq \frac{m \ln \frac{1}{\beta} + \ln n}{1 - \beta} \\
&\leq \frac{m \frac{1-\beta}{\beta} + \ln n}{1 - \beta} \\
&\leq \frac{m}{\beta} + \frac{\ln n}{1 - \beta}
\end{aligned}$$

To minimize the expected number of wrong answers, I take the derivative of this expression with respect to β, getting:

$$\begin{aligned}
\frac{m}{\beta^2} + \frac{\ln n}{1 - \beta} &= 0 \\
\frac{\beta}{1 - \beta} &= \sqrt{\frac{m}{\ln n}} \\
\beta &= \frac{1}{1 + \sqrt{\frac{\ln n}{m}}}
\end{aligned}$$

Substituting this expression for β back in the original expression, we get the numbers of errors to be $\ln n + m + 2\sqrt{m \ln n} = \ln n + m + O(\sqrt{m \ln n})$ as desired.

Problem 4

Part (a)

I show that **DC-TREE** is k-competitive.

I break the algorithm into phases, where in each phase the number of neighbors is fixed. I consider the changes to each of the parts of the potential function within a phase. Suppose that the number of neighbors is b at the beginning of the phase and $b-1$ at the end of the phase. Suppose that each neighbor moves a distance d towards the request.

When OPT moves by d, the change in potential is $\leq kd$. I show that when DC-TREE moves each neighbor by d, the change in potential is $\leq -bd$. So DC-TREE has only enough potential to move at most k times OPT. Thus, the algorithm is k-competitive.

Consider $\Delta_{\sum_{DC}}$. For any of the $k-b$ nodes that is not moving, there is a neighbor between it and the request. This neighbor is moving away from the fixed node. The other neighbors are not on the way, so they are moving closer to the fixed node. Thus, for every fixed node, $b-1$ neighbors are getting closer by d and 1 neighbor is getting further away by d. Also, each moving node is getting closer by $2d$ to each other moving node. So,

$$\begin{aligned}\Delta_{\sum_{DC}} &= (k-b)\cdot(-(b-1)d+d) - \frac{b(b-1)}{2}\cdot 2d \\ &= (k-b)\cdot(-bd+2d) - b(b-1)d \\ &= -kbd+2kd+b^2d-2bd-b^2d+bd \\ &= 2kd-kbd-bd\end{aligned}$$

Consider Δ_M. Without loss of generality, one of the neighbors can can be matched with the OPT server serving the request. This neighbor is getting closer to its match by d. The other neighbors might be getting away from their matches. So,

$$\Delta_M \leq -d+(b-1)d \leq bd-2d$$

Thus, the change in potential is

$$\begin{aligned}\Delta\Phi &= k\Delta_M + \Delta_{\sum_{DC}} \\ &\leq kbd-2kd+2kd-kbd-bd \\ &\leq -bd\end{aligned}$$

Part (b)

I show that any algorithm for k-server on a tree can be used to solve the paging problem by modelin paging as k-server on a particularly simple tree. Consider the tree with a root and all the branches (of length 1) off the root, where each branch represent a page that can be requested. If a server is at the leaf of a branch, it means that the page is being served and is in memory.

Part (c)

When I apply the reduction in part (b) using DC-TREE, I get Fiat's marking algorithm. An unmarked page correspond to a server at the root, and a marked page corresponds to a server at the leaf of a branch. Initially, all pages are marked, meaning that each server sits at a leaf of a branch, corresponding to a page in memory. On a fault, all the servers move to the root, corresponding to unmarking all the pages. One of the server moves to the requested leaf, corresponding to randomly evicting a random unmarked page and marking the accessed page. This will keep happening on a fault as long as some servers are at the root, i.e. as long as some pages are unmarked. Once all the pages are marked, a fault will cause them all to be unmarked, i.e. the servers will all move to the root once more.

Problem 5

Part (a)

Let S be a set of n axis-parallel rectangles in the plane. I transform the problem of reporting all rectangles in S completely contained in a query rectangle $[x : x'] \times [y : y']$ into an orthogonal range searching problem in 4 dimensions.

I transform each rectangle into a point in 4 dimensions as follow. Let (x, y) be the lower left corner of the rectangle and let (x', y') be the upper right corner. Then, the corresponding point in 4D is (x, y, x', y').

I transform the query rectangle $[x : x'] \times [y : y']$, into the simple query in 4D of searching for points in the interval from (x, y, x, y) to (x', y', x', y').

As seen in class, the orthogonal range searching problem in d dimensions uses $O(n \log^{(d-1)} n)$ storage and has $O(\log^d n + k)$ query time.

Part (b)

Let P be a set of n polzgonsin the plane. I transform the problem of reporting all polygons of P completely contained in a query rectangle into the problem of part (a).

I simply transform each polygon into the smallest rectangle containing it. I do this by tracking, across all of its vertices, the $\min x$, $\max x$, $\min y$, $\max y$. Then the smallest containing rectangle has lower left corner at $(\min x, \min y)$ and upper right corner at $(\max x, \max y)$.