Problem 1

(a) We choose a set of faculty members, if their total weight is at least 1/2. If they make a mistake, and their weights are all decreased by 1/2, thus the total weights are decreased by at least 1/4. This reaches the desired factor of 4/3.

Now we use this decreasing factor to upper bound the total weights of n faculty member

$$\text{upper bound} = n\left(\frac{3}{4}\right)^q,$$

where q is total number of queries.

(b) Every time the wisest faculty made a mistake, his/her weight decreases by 1/2, at most every faculty is the wisest, therefore

$$\text{lower bound} = \left(\frac{1}{2}\right)^m.$$

(c) From (a) and (b) we have

$$\begin{aligned}
& \text{upper bound} \geq \text{lower bound} \\
\implies \quad & n\left(\frac{3}{4}\right)^q \geq \left(\frac{1}{2}\right)^m \\
\implies \quad & \log n + q \log\left(\frac{3}{4}\right) \geq m \log\left(\frac{1}{2}\right) \\
\implies \quad & -\lg n - q \lg\left(\frac{3}{4}\right) \geq m \\
\implies \quad & q \leq \frac{m + \lg n}{-\lg\left(\frac{3}{4}\right)},
\end{aligned}$$

and $1/-\lg\left(\frac{3}{4}\right) \sim 2.41$. Therefore in the asymptotic sense, this scheme has 2.41-competitive against best faculty advised life choice.

(d) The multiplicative change can be given by applying β reduction on fraction F faculties,

$$\beta F + (1 - F) = 1 - (1 - \beta)F$$

(e) Similarly in (b), the lower bound can be written as β^m. Similarly in (a), the upper bound can be written as the product of the multiplicative change in (d), denoting F_i as fraction of error at each query,

$$n \prod_i 1 - (1-\beta)F_i.$$

Notice that $1 - x \le e^{-x}$, we will have

$$n \prod_i 1 - (1-\beta)F_i \le n\, e^{\sum_i -(1-\beta)F_i},$$

where $\sum_i F_i$ is the expected number of wrong answers, i.e., q in (a), therefore

$$n \prod_i 1 - (1-\beta)F_i \le n\, e^{\sum_i -(1-\beta)F_i} = ne^{-(1-\beta)q}.$$

Similarly in (c), we let upper bound $\ge$ lower bound, and we have[1]

$$\begin{aligned} & ne^{-(1-\beta)q} \ge \beta^m \\ \implies \quad & \log n - (1-\beta)q \ge m \log \beta \\ \implies \quad & q \le \frac{\log n - m \log \beta}{1-\beta}, \end{aligned}$$

as desired.

(f) The upper bound in (e) can be equivalently written as

$$\frac{\log n + m \log 1/\beta}{1-\beta}$$

Notice that $\log x \le x - 1$, we will have the upper bound further bounded by

$$\frac{\log n + m(1/\beta - 1)}{1-\beta} = \frac{\log n}{1-\beta} + \frac{m}{\beta}$$

So now we need to prove there exists β yields

$$\frac{a}{1-\beta} + \frac{b}{\beta} \le a + b + \gamma\sqrt{ab},$$

where γ is some constant (and a, b correspond to $\log n, m$).

Since the abstracted expression is generic, WLOG[2], we assume $a > b$, then we can set $\beta = \sqrt{b/(a+b)}$. Now we show this choice of β satisfies our need.

[1] In my convention, *log* uses base e, which is the *ln* used in the question.

[2] Notice that $\beta \in (0,1)$, we can switch a, b by writing $\beta' = 1 - \beta$.

On the one hand, consider

$$\frac{b}{\beta} = \sqrt{a+b}\sqrt{b} \leq \sqrt{ab} + b$$

On the other hand, also consider

$$\frac{a}{1-\beta} = \frac{a}{1-\sqrt{\frac{b}{a+b}}},$$

since $a > b$, we know $\sqrt{\frac{b}{a+b}} < \sqrt{\frac{1}{2}} < 1$. Taylor expands $\frac{1}{1-x}$, we will have $1+x+o(x^2)$. Then we know for sufficiently large *constant* γ, we will have $\frac{1}{1-x} \leq 1+\gamma x$, therefore we know we can find some constant γ such that

$$\begin{aligned}
\frac{a}{1-\sqrt{\frac{b}{a+b}}} &\leq a\left(1+\gamma\sqrt{\frac{b}{a+b}}\right) \\
&\leq a + \gamma\sqrt{\frac{ab}{1+b/a}} \\
&\leq a + \gamma\sqrt{\frac{ab}{1+1}} \\
&= a + O(\sqrt{ab}).
\end{aligned}$$

Therefore, we reach that by setting $\beta = \sqrt{b/(a+b)}$ we will have

$$\frac{a}{1-\beta} + \frac{b}{\beta} \leq a + b + O(\sqrt{ab}).$$

By plugging in $\log n$ and m, we can get the desired β to achieve 1-competitive algorithm.

Problem 2

(a) Notice that the dating and deciding problem has no knowledge of future arriving subjects, therefore the adversary can construct a sequence of dating subjects that arrives worse and worse, until the algorithm picks a subject. At that moment on, the adversary can immediately provide the best subject. In this case, the algorithm would be forced to pick the worst subject in this adversary sequence, and the competitive ratio can be made as miserable as it can be depending on the difference between the best and the worst.

(b) For the randomized strategy, we can randomly divide subjects into 2 groups[3]. Because of the randomness, any subject has equal probability falling into the 2 groups. Now the strategy is to date all subjects in one group, and record the best subject as a reference. Then start dating the second group, and immediately decide to stay forever if the subject is better than the reference.

Because we decide the group division randomly, the best subject being in the second group has probability $1/2$ and the second best subject being in the first group also has probability $1/2$. Therefore, choosing the best subject has probability $1/2 \times 1/2 = 1/4$ in this strategy.

(c) We want to bound the probability of bad choice to a small value. This can be achieved by picking the reference in (b) some logarithmic value of the best. Specifically, consider the same random division of two groups as in (b), then we date all subjects in one group and sort the rank of them. Next, we date another group and decide to stay forever if the rank is higher than the $\log_2 k^2 = \lg k^2$, where k is the total number of subjects[4].

The bad thing happens when all best $\lg k$ subjects appear in the first group, then we end up with someone arbitrarily worse. Since each subject has equal probability to fall in any of the two groups, the probability for this to happen is

$$\left(\frac{1}{2}\right)^{\lg k^2} = \frac{1}{k^2}$$

Then the expectation of subject rank we end up staying with is bounded by (suppose in the bad case we end up with the worst one), for sufficiently large k

$$\frac{1}{k^2} \times k + \left(1 - \frac{1}{k^2}\right) \times \lg k^2 = \frac{1}{k} + 2\lg k - \frac{2\lg k}{k^2} = O(\log k)$$

[3]This is essentially trying to make the problem similar to online apartment rental problem, where best apartment can appear anywhere in a sequence. We allow ourselves to pick any subject we want in the pool, not necessarily follow a given sequence.

[4]This analysis is for large enough k, such that $\lg k^2 < k$.

Problem 3

(a) Notice that we know we can answer the query of d dimension range query in $O(\log^d n)$ time ($O(\log^d n + k)$ for k being the number of reported answers). And the data structure for doing so requires $O(n \log^{d-1} n)$ space. Specifically, we can use dimension augmentation to generalize the range searching using BST to higher dimensions.

For this problem, notice that a rectangle is fully contained if and only if the two endpoints on one diagonal are both inside the range in both x and y dimensions. Formally, we want to query if *both* (x_1, y_1) and (x_2, y_2) belong to search region $[x : x']$ and $[y : y']$. In other words, we query if $(x_1, y_1, x_2, y_2) \in [x : x'] \times [y : y'] \times [x : x'] \times [y : y']$. This is essentially range query in 4-dimensional space.

We can use the similar idea of augmenting 1-dimensional range searching to 2-dimensional, to augment the range searching in higher dimension using induction. Specifically, in the class we have the data structure consisting of a BST storing x searching ranges on the leaves and y searching range on each *internal* nodes, which supports $O(\log^2)$ query time and $O(n \log n)$ space. Now suppose we already have a data structure that supports $O(\log^{d-1})$ query time for $d-1$ dimension with $O(n \log^{d-2} n)$ space, we extend this data structure to support d dimension. We first create a balanced binary search tree for all the points w.r.t. the d's dimension. Then on each of the internal node of the tree, we create the data structure that we assumed we had for $d-1$ dimensions, with all nodes that are in the sub-tree interval of the internal node.

Then the range lookup involves a query in the primary d's dimension BST search, and it takes $O(\log n)$ to determine at most $O(\log n)$ nodes constituting the search interval in d's dimension, then continuous searching the rest $d-1$ dimensions in these nodes. Therefore the total query time for d dimension is $O(\log^d n)$.

For storage, notice that the depth of the primary tree for d's dimension is $O(\log n)$. On each level of depth, each node takes $O(s_i \log^{d-2} s_i)$ space for the rest $d-1$ dimensions, where s_i is the interval size on each node and in total the interval sums up to n, therefore we know $\sum_i O(s_i \log^{d-2} s_i) \leq O(n \log^{d-2} n)$. Since the depth is at most $O(\log n)$, we know the total space is bounded by $O(n \log^{d-2} n) \times O(\log n) = O(n \log^{d-1} n)$.

Therefore, in the case of 4-dimensional space, we will have the desired $O(\log^4 n + k)$ query time for k number of reported answers, with $O(n \log^3 n)$ data structure space.

(b) Notice that the query box is still the rectangle, then it suffices to query is the smallest rectangle that contains the polygon is contained in the query rectangle. Because if the smallest containing rectangle is fully contained, then since polygon is contained in the smallest rectangle, it is also contained in the query box. If the smallest rectangle is not fully contained, since the query box is in parallel to the

sides, one of the sides must be 'sliced-off' by the query box. Since it is the *smallest* rectangle containing the polygon, one of the corner of the polygon must touch the side being cut, thus part of the corner will also be cut, therefore the polygon is also not contained in the query box.

The preprocessing time of surrounding polygon with smallest rectangle is $O(n)$ for n polygon (suppose the number of corners is bounded), as we need to find the left-most, up-most, right-most and down-most coordinate of the polygon corner.

Then we reduce the problem in (a), which can take $O(\log^4 n + k)$ query time for k number of reported answers and take $O(n \log^3 n)$ space.

Problem 4

We observe that the line segments do not intersect, therefore within the set of line segments of sweeping (no other line segments present and no current segments depart), the order of segment distances are preserved.

We first sort line segment end points based on the angle to the player. Then, we maintain a balanced binary search tree, with each node presenting a line segment (actually with its *current* distance encoded, the reason will be clear in the following constructions). The sweeping event can occur by examining end points in the order of the angle (this simulates counter-clock wise sweeping). Upon hitting an endpoint, the line segment is either entering the BST or leaving BST, depending on if the endpoint is the first or second point hit for the line segment.

If a line segment is entering the BST, meaning the endpoint gets swept is the first in this segment, it will be inserted based on its current distance to the player. Notice that the line segments maintain their distance in the tree between two sweeping event, so they preserve their positions in the BST. Thereofre we can do the insertion in $O(\log n)$ time using binary search, where we compute the current distance of the line segments this insertion is comparing with. If the insertion of the line segment is with the smallest distance, we mark the segment 'visible'. This is the case when the segment is directly visible.

A line segment is leaving the BST when we sweep the second endpoint. If the segment to delete is at the smallest distance, we mark the segment 'visible'. This is the case that the segment is directly visible. Also, after the deletion, we mark the segment in the BST with smallest distance also 'visible', because the removal of the current wall makes the segment right behind it visible.

The original sorting of endpoint angle takes $O(n \log n)$ time. On each endpoint sweeping event, the insertion takes $O(\log n)$, deletion takes $O(1)$. Therefore, in total the running time is $O(n \log n)$.