Section 6: Midterm Review

In this section, we review over topics from previous sections to help prepare for the midterm exam.

1. Practice A Reduction

You have a set of r riders and h horses, but unfortunately, $2h < r < 3h$, i.e. there are many more riders than horses. You wish to setup a set of 3 rides which will give each rider exactly one chance to ride a horse. To keep things fair among the horses, you wish for each to have exactly 2 or 3 rides.

Because it's winter, by the time the third ride starts it will be very dark, so every rider would prefer *any* horse on the first two rides over being on the third ride. Between the first two rides, each rider doesn't have a preference over time of day, and has the same fixed preference over horses. If a rider must be on the third ride, it has the same preference list for that ride as well.

Each horse has a single list over riders, which doesn't change by ride. Since horses love their jobs, they prefer to be one of the horses on the third ride instead of not being on a ride.

Design an algorithm which calls the following library **exactly once** and ensures there are no pairs r, h which would both prefer to change the matching and get a better result for themselves.

`BasicStableMatching`
Input: A set of k horses and k riders. Each horse has a preference list of all k riders, and each rider has a preference list of all k horses.
Output: A stable matching among the k horses and k riders.

(a) Give a 1-2 sentence summary of your idea.

(b) Give the algorithm you're going to run.

(c) Give a 1-2 sentence summary of the idea of your proof.

(d) Write a proof of correctness.

(e) Give the running time of your algorithm; briefly justify (1-3 sentences). You can assume `BasicStableMatching` has a runtime of $\Theta(k^2)$.

2. Graph Modeling: Running Out of Rooms

In Madison, Wisconsin[1] essentially every apartment lease ends on August 14 with new leases beginning on August 15th. Dissatisfied with the current system (which leaves some people sleeping with their belongings in U-hauls overnight while waiting to move into their new apartment[2]), the City of Madison has given you the power to fix the problem. You will receive a list of all n people moving in the city, their current apartment (or `null` if they are newly moving to Madison from far away), and their new apartment (or `null` if they are moving far away from Madison).

Your goal is to find an ordering of people that will allow for *easy movement*. By that we mean, for every person, by the time you reach them in the list, the apartment they are moving to will have already been vacated by its current occupant (if any), ensuring no one ends up taking a nap in a U-haul. It may be that some apartments are currently vacant (or will become vacant after the move is completed); if an apartment appears only as someone's "new apartment" you may assume that it is currently vacant.

[1] home of *the other* UW.
[2] Yes, really: https://onwisconsin.uwalumni.com/traditions/moving-day/.

For simplicity, you may assume moving in or out of an apartment is instantaneous for each individual and that each apartment has only one tenant.

Given a list. you will return either

- `Easy Movement` and an ordering of all n people, allowing for easy movement.
- `No Easy Movement` and list of people who prevent easy movement.

For example,

Sample Input I
Swati [123 Fake St., 200 State St.]
Ewin [200 State St., 1000 Main St.]
Xin [`null`, 123 Fake St.]
Victor [567 Broadway, `null`]

You might return `Easy Movement` and [Victor, Ewin, Swati, Xin] (there are other valid lists to return here, you only need to give one).

Sample Input II
Swati [123 Fake St., 200 State St.]
Ewin [200 State St., 1000 Main St.]
Xin [1000 Main St, 123 Fake St.]
Victor [567 Broadway, `null`]

You would return `No Easy Movement` and [Swati, Ewin, Xin].
There is no easy movement in this example, because none of Swati, Ewin, and Xin can move first – they each need one of the others to go first.

(a) Describe an algorithm to solve this problem.

(b) Give some intution for why your algorithm is correct. (Don't write a full proof of correctness).

(c) If your list has n people, what is the worst-case running time. Briefly (1-2 sentences) explain.

3. Greedy Algorithms: Fractional Knapsack

You are given a weight capacity $W \in \mathbb{Z}_{>0}$ and n items. Each item i has a weight $w_i > 0$, and a value $v_i > 0$. You are allowed to take any fraction of an item. That is, for each item i, you may take any amount between 0 and 1 of the item, receiving proportional weight and value. Your goal is to choose a subset of items whose total weight is at most W and whose total value is maximized.

Design a greedy algorithm that always produces an optimal solution to the fractional knapsack problem. Describe the algorithm clearly and show that it is optimal.

4. Divide and Conquer: Binary Search Variant

Let $A[1..n]$ be an array of `ints`. Call an array a **mountain** if there exists an index i called "the peak", such that:
$\forall 1 \leq j < i (A[j] < A[j+1])$
$\forall i \leq j < n (A[j] > A[j+1])$

Intuitively, the array increases to the "peak" index i, and then decreases. Note that either of these conditions could be vacuous if the peak is index 1 or n (e.g., a decreasing array is still a mountain).

(a) Given an array `A[1..n]` that you are promised is a mountain, find the index peak index.

(b) Can you design an algorithm with the same running time that also **determines** whether a given array is a mountain (and if it is, finds the peak)?

5. Dynamic Programming: Longest Palindromic Subsequence

Given an input string s, return the length of the longest palindromic subsequence in s. A subsequence is a non-contiguous substring.

For example, the input "abcda" has a longest palindromic subsequence of length 3 ("aba", "aca", and "ada" are all different equally-long palindromic subsequences). The input "mnopqqrom" has a longest palindromic subsequence of length 6 ("moqqom" has 6 characters).

5.1. Write the Dynamic Program

(a) Formulate the problem recursively – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation?

(b) Write a recurrence for solving the problem you defined in the last part (the recurrence is for the answer, **not** the running time).

(c) What is your final answer (e.g. what parameters for the recurrence do you need? Is it a single value or the max/min of a set of values?)?

(d) Give a brief justification for why your recurrence is correct. You do not need a formal inductive proof, but your intuition will likely resemble one.

5.2. Analyze the Dynamic Program

(a) Describe a memoization structure for your algorithm.

(b) Describe a filling order for your memoization structure.

(c) State and justify the running time of an iterative solution.