COP 4516 Outline 1/14/2025

1. Show process 100/80/60 grading
2. Topics: Use of API built in, Greedy Algs
    - Show Outline
    - High level outline of api methods most frequently used
    - example problem solving a greedy problem

High Level Summary of Useful API calls (Mostly Built in Data Structures)

Basic Lists/Arrays try not to do O(n) operations too much
For CP time limits, we can probably get away with say about 500,000,000 simple operations, so $O(n^2)$ probably only works up to 20,000ish or so.

Key Built In Data structures that are improvements over a unsorted list we add and delete from:

Sets: unordered, ordered
Maps: unordered, ordered
Priority Queue (binary heap interally)
Custom Sorting

An unordered set allows you to add an item (no duplicates), delete and item, search for an item in O(1) expected time.

For an ordered set, the run time is O(lg n)…since it's ordered, given a number, I can find the largest number in my set that's less than my search number of smallest number greater than it, for example…[Python does NOT have ordered sets…]

Maps are like sets, but in addition to storing values, each value which is really called a key has an associated value with it:

Arup → 1234567890
Bob → 6667778880
Etc.

Orlando → 0
Tampa → 1
Miami → 2
Etc.

Dist[2][0] represents the distance from miami to orlando.

Priority Queue → allows you to add stuff, delete the min or max depending on which language run times are O(lg n). (allows for duplicates…)

Each language has a way of you defining a comparator so you can sort structs/objects in any which way you want!

Consider the current state of the top of the stacks

Q M F B A

Now, let's say we receive D, there are 3 possible pictures (not including making a new stack):

D M F B A or
Q D F B A or
Q M D B A

I argue that the stack below is equally good or BETTER than the two previous options.

This is the exchange argument. (my greedy choice is at least as good as all others that could have been made because I have the freedom to make all the future choices you do and perhaps more.)

Interval Scheduling

When scheduling events, it's best to give priority to the event that finishes first. This gives you greater freedom with future scheduling, so the exchange argument (any schedule which schedules the earliest ending item first) vs. any other schedule shows that the greedy algorithm of always scheduling the earliest ending event first of the possible remaining choices maximizes the # of events scheduled.

So, read in the event start and end times, sort by end time, and the go through the events in that order, adding an event to the schedule if it starts at or after the last scheduled event ends. (Code attached separately, coded in class in Python.)