Homework 3

OPTIONAL PROBLEMS
(No due date)

1 Written Problems

1.1 Longest Increasing Subset

NOTE: this is a challenge problem
Given an array of integers, find the length of the longest increasing subsequence of these integers. (This can be done with dynamic programming in O(nlogn) time!!) For example,

1. $[0, 3, 6, 2, 10, 1, 5, 33]$ would return 5, because the longest increasing subsequence, $[0, 3, 6, 10, 33]$ is of length 5
2. $[14, 2, 15, 11]$ would return 2, because the longest increasing subsequences, $[14, 15]$, $[2, 11]$ and $[2, 15]$ are all of length 2

Solution:

```
def longest_subseq(inputArray):
    """longest_subseq: int array -> int
       Purpose: Find the length of the longest increasing subsequence in the input array
    """
    argMaxLens = [] //The indices of the final item in a subseq of len <index>
    maxLen = 0 //the current longest increasing subset

    //Loop through each index in the array, dynamically
    //building an array (argMaxLens) of the best options so far
    for i in range (0, len(inputArray)):
       // Binary search for the largest positive subsequence length j
       // such that inputArray[argMaxLens[j]] < inputArray[i]
        lo = 1
        hi = maxLen
        while lo <= hi:
            mid = ceil ((lo + hi)/2)
            if inputArray[argMaxLens[mid]] < inputArray[i]
                lo = mid + 1
            else:
                hi = mid - 1

        // After searching, lo is 1 greater than the
        // length of the longest prefix of inputArray[i], since the loop
        // condition ends only when lo > hi
```

```
        newLen = lo

        //Update the argMaxLengths array with the current index
        //i.e. "I got length newLen with my current index i"
        argMaxLengths[newLen] = i

        // If we found a subsequence longer than any we've
        // found yet, update maxLen
        if newLen > maxLen:
            maxLen = newLen

    return maxLen
```

1.2 Serena Williams' Fan Club Meeting

You know that at the club meeting, there is only one real Serena Williams. Within a group of n girls, *Serena Williams* is defined as a girl who is known by everyone, but knows no one. At the club meeting, you are only allowed to ask questions of the form "Excuse me, but do you know *that* girl over there?" Your job is to determine whether Serena Williams is actually in the group. Clearly you can do this by asking each of the n girls about each of the other $n-1$ girls, a total of $n(n-1)$ questions. But you'd like to do much better!! Note: If a group has size $n = 1$, you should assume the sole member of the group is Serena Williams (after all, she knows no one, and everyone knows her). Note 2: As creepy as it may sound in real-life, for this question you should not assume that girl A knowing girl B implies girl B knows girl A, even if both are not Serena Williams!

(a) Explain why there can be at most one *Serena Williams* in this club meeting, and describe a group of size n (for every $n > 1$) in which there's no such person.

Solution:

If there were two girls in Serena Williams's fan club, A and B, then for A to be Serena Williams, B would have to know her. But then B, knowing someone, cannot be Serena Williams.

There are a ton of answers for the no-Serena Williams part. A group of n girls where everyone knows everyone is a good one.

(b) Suppose you ask A whether they know B, and A says "Yes." What (if anything) can you conclude about the identities of A and B? What if A says "No"? What can you conclude then (if anything)?

Solution:

If A admits to knowing B, you know that A cannot be Serena Williams. If A says that she does *not* know B, then B cannot be Serena Williams, because Serena Williams is known by everyone.

(c) Describe an algorithm that, given a group of n girls, either finds Serena Williams or determines that Serena Williams isn't in the group of fans using $O(n)$ questions. A short paragraph explanation will suffice.

Solution:

(This solution is not recursive!) Denote the group of potential girls as S. Create an empty set T, to manage all the girls who cannot be Serena Williams. While the size of S is greater than one, perform the following:

1. Choose two arbitrary girls, A and B from the set S.
2. Ask A if they knows B.
    - If A does know B, we know by (b) that A is not Serena Williams. Remove A from S. Add A to T.
    - If A does not know B, we know by (b) that B is not Serena Williams. Remove B from S. Add B to T.

The last remaining girl in S may or may not be Serena Williams. To see if she is, ask if she knows everyone in T. Then ask if everyone in T knows her. If everyone in T knows her, but she knows no one, then she is Serena Williams. Otherwise, she is not.

Each iteration of the while loop requires 1 question and eliminates 1 girl from the set S. It requires $n-1$ iterations to eliminate $n-1$ girls. To ask if the last member of S knows everyone in T requires $n-1$ questions to be asked. To ask if everyone in T knows the last member of S takes $n-1$ questions to be asked. Thus the total number of questions is: $3(n-1) = 3n-3 = O(n)$ time.

(d) Prove by induction that your algorithm uses $O(n)$ questions. By this we mean use the same logical structure as an inductive proof in your written explanation (i.e. with a base case, inductive step, etc.). In order to use $O(n)$ questions, you may ask only a constant number of questions for each girl in the fan club meeting. You will lose points if you don't use induction.

Solution:

Base Case: n = 1 The size of S is 1, so we never perform the loop. T remains empty. We therefore do not need to ask any questions according to the algorithm. The algorithm will give back that the sole member is Serena Williams and take $O(1)$ time, so both the correctness and runtime meet the specification.

Inductive Step Assume the algorithm is correct and has runtime $O(k)$ for a group of size k. Consider a group K of size $k+1$. Select two arbitrary members, A and B, of K. Ask A if they knows B. If they do, eliminate A from K. If they

do not, eliminate B from K. Denote the girl you removed from K as P. Now run the algorithm on the remaining group of size k. If the algorithm reports there is not a Serena Williams in the group, we know Serena Williams is not in the original group K because neither P, nor the k remaining members meet her definition. If the algorithm reports back that it found Serena Williams R, ask R if they know P, and if P knows R. If P knows R and R does not know P, R is Serena Williams. Otherwise, R is not Serena Williams. The runtime of the algorithm is therefore $O(k)$ (assumed by the inductive step) $+O(1+1+1)$ (asking up to three additional questions for the $k+1$ member), which gives a runtime of $O(k+1)$.

---

2 Python Problems

2.1 Climbing Stairs

Given a set of stairs comprised of n steps, and that you can climb 1, 2, or 3 steps at a time, write a function that returns the number of ways to climb that set of stairs. For example,

1. climber(2) would return 2, because the sequence of steps could be $[1,1]$ or $[2]$
2. climber(3) would return 4, because the sequence of steps could be $[1,2]$, $[2,1]$, $[3]$, or $[1,1,1]$

You do not need to return the possible sequences of steps, just the number.

Solution:

```
def climb_stairs(n):
  """climb_stairs: int -> int
  Purpose: return the number of ways to climb n sets of stairs
  """
  if n <= 0:
      print "invalid input"
      return
  arr = [None]*3 #make the array at least size 3 to cover base cases
  arr[0] = 1
  arr[1] = 2
  arr[2] = 4
  if n < 3:
      print arr[n-1]
      return
  for i in range(3, n):
      arr.append(arr[i-3]+arr[i-2]+arr[i-1])
  print arr[n-1]
```

2.2 Climbing Stairs++

Modify your climber(n) function to print out the possible sequences of steps, such that climber(3) would print out "[1, 2], [2, 1], [3], [1, 1, 1]"

Solution:

Instead of storing an array of integers that keeps track of how many ways there are to get to the ith step, there could be an array of array of strings representing the ways to get to that step. Then, you would add "1" to the strings representing the ways in which you can get to the previous step, "2" for the steps two back, and "3" for the steps three back.