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:

```python
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

---

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.