Lab 11: Searching

Labs are graded for participation rather than correctness. Keep all your lab code in your course GitHub repo to receive credit for your work. We'll be looking to see that you have at least partially completed all problems in each lab.

If you finish the lab assignment early, you may get started on the homework.

Professional skills

If you have not yet participated in collaborative coding, consider collaborating for this lab. We'll set up some breakout rooms for you to self-select to work with other students. See lab 7 for setup instructions. You can also earn professional skills points by volunteering to walk through your lab solution toward the end of class.

Problem 1: Recursive binary search

You've already seen iterative binary search in lecture. Tonight, you will implement it **recursively**. Your solution contain a function that matches the following docstring:

```
'''
       Function -- binary_search
           Searches for the given item in the given list.
       Parameters:
           lst -- The list to search in.
           item -- The item to search for.
       Returns:
           True if item is in lst, False otherwise.
   '''
```

Binary search proceeds as follows:

- Determine the midpoint of the list
- If the value at the midpoint is equal to the search item, return true.
- Otherwise…
    - if the value at the midpoint is > *item*, discard the right half of the list and recursively call `binary_search` with the smaller list
    - if the value at the midpoint is < *item*, discard the left half of the list and recursively call `binary_search` with the smaller list
- If we end up with an empty list then we've checked all possible places that *item* could exist in the original list and didn't find it, so return false.

Binary search requires that the input list be in sorted order. Once written, test that your binary search works correctly by making sure it passes the unit tests in https://github.ccs.neu.edu/cs5001-f20-sea-pm/lecture-code/blob/master/Sample%20solutions/Lab%2011/test_binary.py

Problem 2: Finding the min in a shifted list

This is the kind of problem you might encounter at a job interview, or at work if you're a software engineer. Think it through and design a solution in pseudocode first.

Here's the problem to solve:

- Given: A sorted list of Integers that are 'shifted' some unknown number of times. For example, a sorted-shifted list might be [18, 25, 38, 1, 12, 13] or [25, 38, 1, 12, 13, 18] or [1, 12, 13, 18, 25, 38]. You don't know how many positions it's been shifted.
- Return: The smallest integer in the list
- You can assume that there are no duplicate numbers in the list and the list is not empty.

Here's what your algorithm should compute:

Input: A sorted but rotated list	**Algorithm returns:**
[18, 25, 38, 1, 12, 13]	1
[25, 38, 40, 12, 13]	12
[12, 13, 25, 38]	12
[1]	1
[2, 1]	1

Once you have an initial solution, determine its algorithmic complexity. If it's worse than *O*(lg *n*), revisit your solution and see if you can improve it. Once you have a *O*(lg *n*) solution, then implement it in Python and validate that it works as expected.