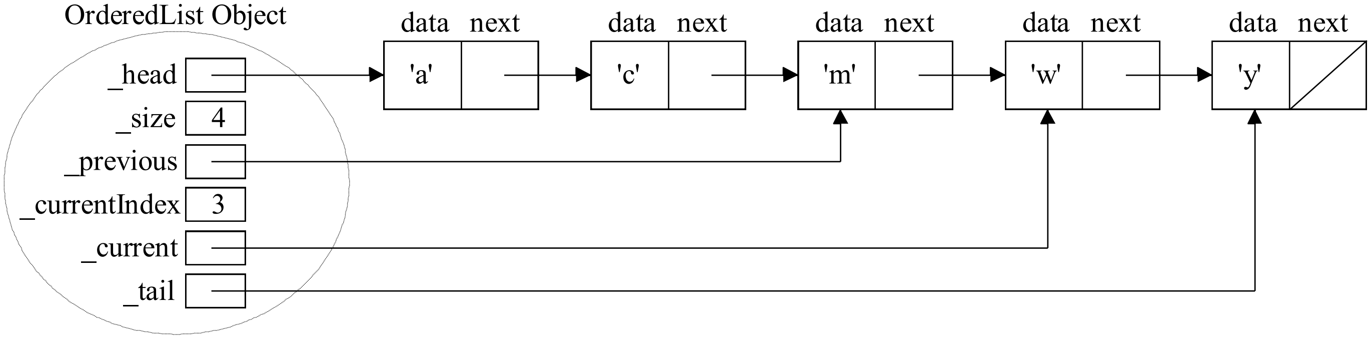
Data Structures (CS 1520) Lab 5 Name:_________________

Objective: To understand recursion by writing simple recursive solutions.

To start the lab: Download and unzip the lab5.zip file from eLearning.

Part A: Recall: We modified the textbook's ordered list ADT that uses a singly-linked list implementation by adding the `_size`, `_tail`, `_current`, `_previous`, and `_currentIndex` attributes:



```
##    NON-RECURSIVE CODE WE ARE REPLACING

 def search(self, targetItem):
    if self._current != None and \
       self._current.getData() == targetItem:
         return True

    self._previous = None
    self._current = self._head
    self._currentIndex = 0
    while self._current != None:
        if self._current.getData() == targetItem:
            return True
        elif self._current.getData() > targetItem:
            return False
        else: #inch-worm down list
            self._previous = self._current
            self._current = self._current.getNext()
            self._currentIndex += 1
    return False
```

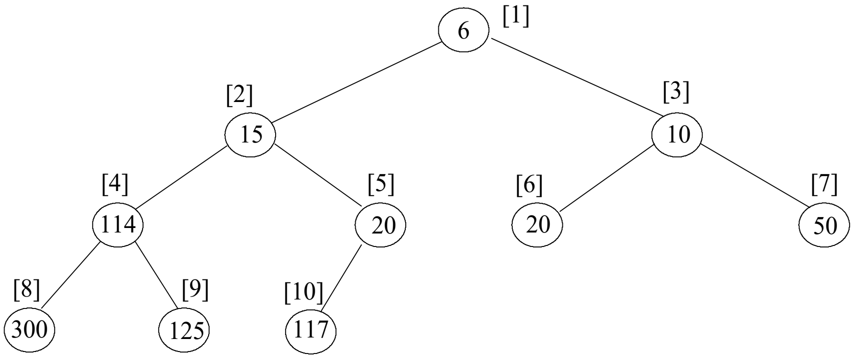
```
def search(self, targetItem):
  def searchHelper():
     """ Recursive helper function that moves down the linked list.
      It has no parameters, but uses self._current, self._previous,
      and self._currentIndex."""
    # ADD CODE HERE




   # START OF SEARCH - DO NOT MODIFY BELOW CODE
   if self._current != None and \
      self._current.getData() == targetItem:
       return True

   self._previous = None
   self._current = self._head
   self._currentIndex = 0
   return searchHelper() # Return the result of searchHelper
```

a) What are the base case(s) for the `searchHelper` that **halt** the while-loop of the non-recursive search codc?

b) What are the recursive case(s) for the `searchHelper` that replaces the while-loop of the non-recursive search codc?

c) Complete the recursive `searchHelper` function in the `search` method of our `OrderedList` class in `ordered_linked_list.py`. Run the `ordered_linked_list.py` test code at the bottom of the class, or test it with the `listTester.py` program.

Part B: Recall that Lecture 7 and Section 6.6 discussed a very "non-intuitive", but powerful list/array-based approach to implement a priority queue, call a binary heap. The list/array is used to store a *complete binary tree* (a full tree with any additional leaves as far left as possible) with the items being arranges by *heap-order property*, i.e., each node is ≤ either of its children. An example of a *min* heap "viewed" an a complete binary tree would be:



	0	1	2	3	4	5	6	7	8	9	10
Python List actually used to store heap items	not used	6	15	10	114	20	20	50	300	125	117

Recall the General Idea of `insert(newItem)`:

- append newItem to the end of the list (easy to do, but violates heap-order property)
- restore the heap-order property by repeatedly swapping the newItem with its parent until it *percolates up* to the correct spot

Recall the General Idea of `delMin()`:

- remember the minimum value so it can be returned later (easy to find - at index 1)
- copy the last item in the list to the root, delete it from the right end, decrement size
- restore the heap-order property by repeatedly swapping this item with its smallest child until it *percolates down* to the correct spot
- return the minimum value

Originally, we used iteration (i.e., a loop) to percolate up (see `percUp`) and percolate down (see `percDown`) the tree. (textbook code below)

```
##    NON-RECURSIVE CODE WE ARE REPLACING

def percUp(self,i):
    while i // 2 > 0:
        if self.heapList[i] < self.heapList[i//2]:
           tmp = self.heapList[i // 2]
           self.heapList[i // 2] = self.heapList[i]
           self.heapList[i] = tmp
        i = i // 2
```

```
def percDown(self,i):
    while (i * 2) <= self.currentSize:
        mc = self.minChild(i)
        if self.heapList[i] > self.heapList[mc]:
            tmp = self.heapList[i]
            self.heapList[i] = self.heapList[mc]
            self.heapList[mc] = tmp
        i = mc
```

For part B, I want you to complete the recursive `percUpRec` and recursive `percDownRec` methods in `binHeap.py`. Run the `binHeap.py` file which has test code at the bottom to test **both** methods.

After you have correct code for both parts of the lab, submit a lab5.zip containing your code on eLearning.

If you do not get done today, then submit it by next week's lab period.

(If you have extra time, work on previous labs or homeworks!)