Mod 6 Homework - Quadratic Sorts

Modify the bubble and insertion sort algorithms for (situationally) better performance.

Part 1 - `champaign_sort`

`bubble_sort` works well on rabbits (large items near the beginning) but poorly on turtles (small items near the end). Consider the following list, with 5 as a rabbit and 1 as a turtle:

```
[5, 2, 2, 2, 2, 2, 1] # initial list
[2, 2, 2, 2, 2, 1, 5] # first pass
[2, 2, 2, 2, 1, 2, 5] # second pass
[2, 2, 2, 1, 2, 2, 5] # third pass
[2, 2, 1, 2, 2, 2, 5] # fourth pass
[2, 1, 2, 2, 2, 2, 5] # fifth pass
[1, 2, 2, 2, 2, 2, 5] # sixth pass
```

We can reduce the impact of turtles by alternating between (1) bubbling large items from left to right and (2) bubbling small items from right to left: this is called `champaign_sort`. Consider the same list as above, sorted with `champaign_sort`:

```
[5, 2, 2, 2, 2, 2, 1] # initial list
[2, 2, 2, 2, 2, 1, 5] # first pass: left-to-right
[1, 2, 2, 2, 2, 2, 5] # second pass: right-to-left
```

Implement `champaign_sort`. It should be able to sort lists with a constant number of rabbits and turtles in `O(n)`.

(assignment continued on next page)

Part 2 - `opt_insertion_sort`

In a first pass at `insertion_sort`, you might come up with something like the following:

```
01 def insertion_sort(L):
02     n = len(L)
03     for j in range(n): # go through every item
04         for i in range(n - 1 - j, n - 1): # bubble it into a sorted sublist
05             if L[i] > L[i+1]:                 # 1 comparison
06                 L[i], L[i+1] = L[i+1], L[i]   # 2 writes
07             else: break
```

The code above has some iefficiencies. Assuming the sorted sub-list contains `m` items, the worst case for bubbling an item into its correct position involves ~`3m` operations:

- `m` comparisons
- `m` writes as items in the sorted sublist are shifted
- `m` writes as the item being added to the sorted sublist is written on line 6

We can do better:

- Reduce the number of comparisons to find where a new item should go to `O(logm)` by using a binary search on the sorted sublist
- Reduce the number of times the new item (`L[i]` on line 6 above) is written to `O(1)` - it does not need to be written every time another item is shifted. You may need to use a temporary variable to store this item.

Your optimized insertion sort should be 2~3x faster than the insertion sort algorithm given above:

```
===Random Distribution====
        insert  opt_insert
     n  t(s)    t(s)
--------------------------
   100  0.001   0
   200  0.001   0
   500  0.013   0.005
  1000  0.054   0.018
  2000  0.19    0.084
  5000  1.2     0.42
 10000  5.1     1.7
--------------------------
```

Submitting

At a minimum, submit the following files:

- `hw6.py` - contains
  - `champaign_sort()`
  - `opt_insertion_sort()`

Students must submit to Mimir **individually** by the due date (typically, the second Wednesday after this module opens at 11:59 pm EST) to receive credit.

Grading

This assignment will be manually graded. You should write your own tests to verify your sorting algorithms sort lists correctly.

- 30 - `champaign_sort`
  - 5 - sorts input correctly
  - 25 - champaign_sort logic is correct (manual)
- 70 - `opt_insertion_sort`
  - 5 - sorts input correctly
  - 35 - binary search implemented correctly
  - 30 - new item is written O(1) time per loop

Feedback

If you have any feedback on this assignment, please leave it here.

We check this feedback regularly. It has resulted in:

- A simplified, clear **Submitting** section on all assignments
- A simplified, clear **Grading** section on all assignments
- Clearer instructions on several assignments (particularly in the recursion module)