Mod 6 Homework - Quadratic Sorts

We will use "half sorted" to describe a list consisting of a series of negative integers, followed by a 0, followed by a series of positive integers:

```
[-2, -3, -1, 0, 3,  1,  2]
 <--neg--->     <--pos-->

^value          *
|                      *
|                  *
+-----------*-----------> idx
|        *
|*
v    *
```

We have provided an algorithm `sort_halfsorted()` that efficiently sorts such a list:

```python
def sort_halfsorted(L, sort):
    idx_zero = find_zero(L)      # find the 0 index
    sort(L, 0, idx_zero)         # sort left half
    sort(L, idx_zero+1, len(L)) # sort right half
```

It is up to you to implement the following algorithms such that `sort_halfsorted` works as expected:

- `find_zero(L)` - return the index of the `0` in such a list in $\mathcal{O}(log(n))$
- `bubble(L, left, right)`, `selection(...)`, and `insertion(...)`
    - sort the sub-list `L[left:right]` using the appropriate sorting algorithm
    - sort the list in-place (do not return anything)
    - `bubble` and `insertion` should be adaptive ($\mathcal{O}(n)$ in the best case)
    - Follow Python convention - `L[left:right]` includes `L[left]` but not `L[right]`

Tests

- Test each of your sorting algorithms *thoroughly*:
    - Test a range of lengths and patterns, e.g. n=1, 2, 3, 4, … 50
        - **do not just randomly test a few lengths** - you're looking to see if you have an off-by-one error that only appears at certain lengths, so you need to test every length in a wide range
        - Test a full range of the possible 0 indices for each length
        - see the provided tests for `find_zero()` for an example of generating and testing a range of length + zero indices
    - Make sure your final list has the same items as the original list. Pseudocode:

    ```
    1) Generate a half-sorted list
    2) Make a deep-copy of that list using slicing
    3) Sort the original list using e.g. `sort_halfsorted(L, bubble)`
    4) Test that the original list is now sorted
    5) Test that the original list and the deep copy have the same elements
    ```

`unittest.TestCase` provides a method `assertCountEqual(L1, L2)` that will help with step 5.

- tests for `find_zero` are included
- `generate_halfsorted()` and `is_sorted()` methods are provided to help with testing

Submitting

At a minimum, submit `hw6.py` and `TestHw6.py` containing the requested algorithms and unittests.

Students must submit **individually** by the due date (typically Tuesday at 11:59 pm EST) to receive credit.