Module 7 Homework - Magic Sort

We've introduced 5 sorting algorithms: bubble, selection, insertion, merge, and quick. All have their strengths and weaknesses. Here, you'll implement a `MagicSort` algorithm which uses a combination of three of these to achieve optimum sorting efficiency.

1) Begin with a linear scan in a function `linear_scan`; keeping track of the following to identify edge cases:

   - Is the list already sorted? If so, you can return right away
   - Are there at most 5 items out of place? If so, use insertion sort to sort the list
   - Is the list reverse sorted? If so, write a short algorithm `reverse_list()` that reverses a list in linear time

   Use a function called `linear_scan()` to do this scan, and return a value that denotes which (if any) edge cases apply.

2) If the linear scan did not detect any of the special cases above, start with quicksort. **Use the last item in a sublist as the pivot element**. This won't give optimal results, but it will make it easy for us to demonstrate how magic sort handles edge cases.

3) During quicksort, keep track of your recursive depth. The best-case maximum-depth should be `log2(n)+1`, like in mergesort. We expect the actual depth to be a little higher, since not every pivot will be exactly a median, but if our recursive depth gets too high, that's a good indication our pivot strategy isn't working. **If the recursive depth reaches twice the best-case maximum-depth, fall back to mergesort to sort the list.** It is okay to use `math.log2()` for calculating this depth.

4) mergesort and quicksort are fast on big lists, but our quadratic algorithms actually perform better on smaller lists. When the sublists of these divide-and-conquer algorithms drop to 16 or fewer items, sort those sublists using insertion sort instead. You'll need to use something like `insertion(L, left, right)` here to sort the sublist in-place.

`magic_sort(L)` should sort `L` in-place. Keep track of which algorithms are invoked during magicsort, and return them as a set:

```
>>> n = int(1E5)
>>> L = [(n-i) for i in range(n)]
>>> magic_sort(L)
{'reverse_list'}
>>> L = [(n-i) for i in range(n)]
>>> L[:6] = [-1, -2, -3, -4, -5, -6]
>>> magic_sort(L)
{'quicksort', 'insertionsort', 'mergesort'}
```

Tests

It is crucial that you solve this assignment incrementally. I would advise going in this order:

1) `linear_scan`
2) `reverse_list`
3) `insertionsort`

4) `quicksort`

5) `mergesort`

6) `magicsort`

At every step, write unittests, then implement functionality.

Submission

Students must submit **individually** by the due date (typically Tuesday at 11:59 PM EST).