MSD Radix Sort

Here is a video walkthrough of the solutions.

Recursively implement the method `msd` below, which runs MSD radix sort on a `List` of `Strings` and returns a sorted `List` of Strings. For simplicity, assume that each string is of the same length. You may not need all of the lines below.

In lecture, recall that we used counting sort as the subroutine for MSD radix sort, but any sort works! For the subroutine here, you may use the `stableSort` method, which sorts the given list of strings in place, comparing two strings by the given index. Finally, you may find following methods of the `List` class helpful:

1. `List<E> subList(int fromIndex, int toIndex)`. Returns the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.
2. `addAll(Collection<? extends E> c)`. Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.

```
1  public static List<String> msd(List<String> items) {
2
3      return ____________________________________________________________;
4  }
5
6  private static List<String> msd(List<String> items, int index) {
7
8      if (____________________________________________________________) {
9          return items;
10     }
11     List<String> answer = new ArrayList<>();
12     int start = 0;
13
14     ____________________________________________________________;
15     for (int end = 1; end <= items.size(); end += 1) {
16
17         if (____________________________________________________________) {
18
19             ____________________________________________________________;
20
21             ____________________________________________________________;
22
23             ____________________________________________________________;
24         }
25     }
26     return answer;
27 }
28 /* You don't need to understand the implementation of this method to use it! */
29 private static void stableSort(List<String> items, int index) {
30     items.sort(Comparator.comparingInt(o -> o.charAt(index)));
```

```
}
```

Solution:

```
public static List<String> msd(List<String> items) {
    return msd(items, 0);
}

private static List<String> msd(List<String> items, int index) {
    if (items.size() <= 1 || index >= items.get(0).length()) {
        return items;
    }
    List<String> answer = new ArrayList<>();
    stableSort(items, index);
    int start = 0;
    for (int end = 1; end <= items.size(); end += 1) {
        if (end == items.size() || items.get(start).charAt(index) != items.get(end).charAt(index)) {
            List<String> subList = items.subList(start, end);
            answer.addAll(msd(subList, index + 1));
            start = end;
        }
    }
    return answer;
}

/* You don't need to understand the implementation of this method to use it! */
private static void stableSort(List<String> items, int index) {
    items.sort(Comparator.comparingInt(o -> o.charAt(index)));
}
```

Explanation: MSD sort starts with the leftmost (most significant) digit, grouping and sorting all elements by that digit. It then proceeds recursively on each group. The helper function `msd(items, index)` tells us which index we're currently sorting items by, which is initialized to `0` by the original `msd` function. The base case is if there is 1 item or less (the list is already sorted), or if we've sorted every possible index.

Otherwise, we use `stablesort` to sort by the current index. Note that the subroutine to sort by index *must* be stable; otherwise we lose the ordering imposed by the previous indices we've already sorted.

Inside the loop, `start` and `end` track the start and end indices of our curent group (items that share the same value at `index`). If our current `end` differs from `start`, we must have reached an item with a different value at `index`, so we take everything from `start: end` (exclusive) to get the current group, recursively sorting that group on the next `index`.