4. A very good sorting algorithm that uses recursion is named *merge sort*. It works as follows:

   1. To start, consider each element of the **ArrayList** as a "section" of size 1 (numbered 0 through $n-1$).
   2. Merge neighbouring *sections* together so that the resulting section (of double the size of the original section) is sorted.
   3. Repeat the previous step until there is only one section left.

   The above looks well and good in concept but there are a couple of snags. The first is how to handle weirdly-sized sections (*leftovers*). Another is how to store the sections while we are working with them.).

   Here are some example runs of merge sort:

```
Input: arr =        { 1 8 4 13 99 23 17 7 25 }
Initial sections =  {1} {8} {4} {13} {99} {23} {17} {7} {25}
Merge #1 =          {1 8} {4 13} {23 99} {7 17} {25}
Merge #2 =          {1 4 8 13} {7 17 23 99} {25}
Merge #3 =          {1 4 7 8 13 17 23 99} {25}
Cleanup =           {1 4 7 8 13 17 23 25 99}

Input: arr =        { 13 99 47 0 23 13 86 }
Merge #1 =          { 13 99 } { 0 47 } { 13 23 } { 86 }
Merge #2 =          { 0 13 47 99 } {13 23 86 } //Note odd sized section
Merge #3 =          { 0 13 13 23 47 86 99 }
```

   You are required to:

   (a) Perform a *merge sort* on the following **ArrayList** (showing all steps as above):

   `{ 13 47 200 53 0 100 33 8 31 75 123 47 99 }`

   (b) Write a recursive method, `mergeSort(ArrayList<Integer> arr)` that sorts the input `ArrayList` using merge sort.
   Hint: You'll need a *helper function*, `merge`, that merges two sections together.