Group 16 - LemmonSort2015

Algorithm: Aaron Lemmon
Presentation: Emma Sax and Dan Stelljes

The Idea Behind LemmonSort2015

- Make a new modData object array
  - fullString
  - modValue
  - integerValue
- In a *for* loop, make new modData objects of each string and put into the modData array
- Sort the modData array (using a handwritten comparator)
- In a *for* loop, copy the fullString component of each object in the sorted modData array back into the original array

The ModData Comparator

- Given two modData objects, the comparator should return:
  - a negative number if the first object should go before the second object
  - a positive number if the second object should go before the first object
  - zero if the two objects are equal
- If the two objects do not have the same mod, then return:

```
object2.modValue - object1.modValue
```

- Else, if the two objects have the same mod, then return:

```
object1.integerValue - object2.integerValue
```

Big-O and Effectiveness of LemmonSort2015

- *Arrays.sort* (TimSort)…… $\Theta(n\log_2 n)$
- Two *for* loops that parse *n* data…… additional 2n
- Extra memory:
  - *n* objects
  - objects of array of length *n*

- Making new objects for each component:
  - avoided time consuming parsing back and forth between integer and string
  - each piece of information is sorted inside each object; nothing needs to be recalculated multiple times for each comparison

Looking Back...

- DUPLICATE CASES
  - add a countDuplicates field to the modData objects
  - increment the countDuplicates field of what we’ve seen before instead of making an entirely new object
  - when copying back at the end, add enough strings back to fulfill however many duplicates there were