Group 6

Emma Sax and Dan Stelljes

Applied Java's built-in quicksort to a numeric representation of the data, using a hash table to efficiently handle the conversion.

Correct sorting; fourth place overall.
Large Data: 454ms
Small Data: 220ms

originalData
"0.123596453"
"0.598038593"
"0.437676592"
"0.798364328"
"0.598038593"
"0.437676592"
"0.967382421"
"0.598038593"
"0.798364328"
"0.654987421"

(2)

dataAsLongs
8123596453
7598038593
9437676592
2798364328
7598038593
9437676592
4967382421
7598038593
2798364328
9654987421

(3)

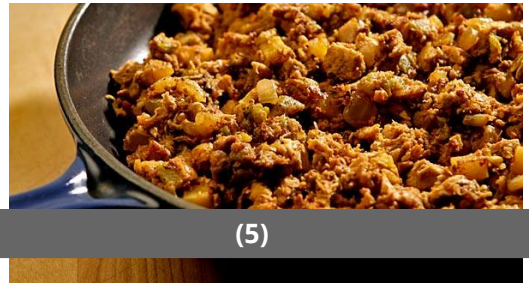
8123596453	"0.123596453"
7598038593	"0.598038593"
9437676592	"0.437676592"
2798364328	"0.798364328"
1435675867	"0.435675867"
4967382421	"0.967382421"
9654987421	"0.654987421"

1. Convert each item in `originalData` to a long. (Append `9-mod` to the nine digits of the item.)

2. Add the long value to an array analogous to `originalData`.

3. Map the long value to its string representation.

dataAsLongs
2798364328
2798364328
4967382421
7598038593
7598038593
7598038593
8123596453
9437676592
9437676592
9654987421



8123596453	"0.123596453"
7598038593	"0.598038593"
9437676592	"0.437676592"
2798364328	"0.798364328"
1435675867	"0.435675867"
4967382421	"0.967382421"
9654987421	"0.654987421"

originalData
"0.798364328"
"0.798364328"
"0.967382421"
"0.598038593"
"0.598038593"
"0.598038593"
"0.123596453"
"0.437676592"
"0.437676592"
"0.654987421"

4. Sort `dataAsLongs` using built-in quicksort.

5. Iterate over `dataAsLongs`, writing the string representation of the item to the corresponding position in `originalData`.

Running time analysis

- Steps 1-3: **Θ(n)**
  For each item in `originalData`: conversion to long, insertion to `dataAsLongs`, insertion to hash table (all constant operations).
- Step 4: **worst case $\Theta(n^2)$; expected case $\Theta(n\log_2 n)$**
  Sorting of `dataAsLongs`. (From Java documentation: Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch.)
- Step 5: **Θ(n)**
  For each item in `dataAsLongs`: checking against previous, possible lookup in hash table, insertion to `originalData` (all constant operations).

Non-constant memory

- `dataAsLongs` allocates space for `originalData.length` items.
- If there are no duplicates, the hash table also keeps track of `originalData.length` associations.

Possible optimizations

- Replacement of mod arithmetic with lookup arrays.
- Better handling of duplicate values.
- Writing a totally different algorithm.