Speed Comparison

```
Row: 100, Col: 100, ratio: 0.10
Choi  Parikh
203   306        → 50.7% more

Row: 100, Col: 100, ratio: 0.20
Choi  Parikh
261   358        → 37.2% more

Row: 200, Col: 200, ratio: 0.10
Choi  Parikh
770   987        → 28.2% more
```

Analysis

My implementation did the following:

1. Check to see if row was sorted or reverse sorted
2. If so, did nothing (reversed for reverse sorted row)
3. If not, checked to see how sorted or reverse sorted it was based on comparisons per
    a. Sweet spot was 90% sorted or reverse sorted
    b. If higher than or equal to 90%, ShellSort (Knuth sequence), which was best for partially sorted arrays
4. If completely unsorted, QuickSort, which was best for unsorted arrays
5. Merge rows into one array
6. QuickSort final array (fastest algorithm to sort array with sorted sub-sequences)

Notes

Tried to implement min-heap priority queue to quickly merge rows into final array, by queueing lowest value in each row, de-queueing the root, then adding the next lowest value from the same row as the de-queued root, and so on. However, I was stumped because I needed a max value to signify the row had no more values.

Also, it is interesting to note that my implementation gets better for larger / higher ratio arrays. This is due to the fact that the ratio of time spent sorting the rows versus the final array gets larger for larger / higher ratio arrays. More work is done on the rows before the final QuickSort.