GROUP 3: DEEPBLUE
CSCI 3501 - SORTING COMPETITION

Ken Broden
Ethan Graybar
Andrew Lam

Sorting Competition And Medians:

Prelim 1:

- Correctly sorted all rounds ✔
- Failed to beat Group0
- group 3 took place 4. The sum of places is 8, the sum of medians is 1063.0

Prelim 2:

- Correctly sorted all rounds ✔
- Beat Group0 all rounds
- group 3 took place 1. The sum of places is 2, the sum of medians is 574.0

Changes from group 0:

- We use combination of **Merge Sort** and **Insertion Sort** with new method such as MergeSort method, InsertionSort method, Merge method which is different from group0 where they use Arrays.sort.

Sorting Algorithm Description:

- The main method initiates the sorting process by calling the `sort` function, which in turn calls the `mergeSort` function on the entire array.
- In the `mergeSort` function, the array is recursively divided into smaller subarrays.
- If the size of a subarray is less than or equal to a predefined threshold (`INSERTION_SORT_THRESHOLD`), then insertion sort is applied to the subarray for efficiency.
- The `insertionSort` method performs insertion sort by iterating through a subarray and shifting elements to the right until the correct position for each element is found.
- If the size of a subarray is larger than the predefined threshold, then `mergeSort` is used. The method splits the subarray into two halves, recursively sorts each half, and then merges them using the `merge` function.
- The `merge` function combines two sorted subarrays into a single sorted array by comparing elements using a custom comparator, `SortingCompetitionComparator`, and inserting the smaller element into the correct position.

Worst Case and Expected Case Running Time:

- Worst Case run-time for Merge Sort is O(nlogn), Worst Case run-time for Insertion Sort is O(n^2). Combined Worst-Case running time is **O(nlogn)**.
- Expected Case run-time for Merge Sort is O(nlogn), expected case run-time for Insertion Sort is O(n). Combined Expected-Case running time is **O(nlogn)**.

Data Stored In Sorting Method

- The data is stored in a 2D (two-dimensional) integer array **int[][]** with each subarray representing a sequence of integers that needs to be sorted.
- The data will be read from a file using readData method then stored it in the 2D integer array. Then clone the integer array into a main 2D (two-dimensional) integer array using clone().

Additional Data Structures:

- Temporary array in the Merge Method, where **leftArray** and **rightArray** used to hold left and right halves of array being merged. The creation of temporary arrays, copying elements where clone elements from original array to temporary arrays, merging elements involves comparing and copying elements from temporary arrays back to original array are part of the total time complexity of the sorting algorithm.

Correctness Discussion:

“I see no issues”

Outcome of correctness discussion:

No change.

✅

Algo Version 1 NO INSERTION SORT		
Array Size	**Time to sort**	**Time per array**
100,000	179 ms	0.00179 ms/array
100,000 (sorted)	33 ms	0.00033 ms/array
500,000	504 ms	0.001008 ms/array
500,000 (sorted)	129 ms	0.000258 ms/array

Algo Version 2 INSERTION SORT THRESHOLD: 10		
Array Size	**Time to sort**	**Time per array**
100,000	34 ms	0.00034 ms/array
100,000 (sorted)	6 ms	0.00006 ms/array
500,000	141 ms	0.000282 ms/array
500,000 (sorted)	35 ms	0.00007 ms/array

Algo Version 2 INSERTION SORT THRESHOLD: 5		
Array Size	**Time to sort**	**Time per array**
100,000	85 ms	0.00085 ms/array
100,000 (sorted)	8 ms	0.00008 ms/array
500,000	231 ms	0.000462 ms/array
500,000 (sorted)	72 ms	0.000144 ms/array

Algo Version 2 INSERTION SORT THRESHOLD: 20		
Array Size	**Time to sort**	**Time per array**
100,000	60 ms	0.0006 ms/array
100,000 (sorted)	7 ms	0.00007 ms/array
500,000	237 ms	0.000474 ms/array
500,000 (sorted)	63 ms	0.000126 ms/array

Unimplemented Version 3 Algorithm:

- Used a single auxiliary array that is passed through mergeSort and merge, opposed to leftArray and rightArray.
- Reduced memory overhead by reusing a single auxiliary array throughout the sorting process.
- For 500,000 arrays, typically saw improvement of 10 ms
- Not fully tested, deemed not worth marginal improvement.

Algo Version 3 INSERTION SORT THRESHOLD: 10		
Array Size	**Time to sort**	**Time per array**
100,000	75 ms	0.00075 ms/array
100,000 (sorted)	7 ms	0.00007 ms/array
500,000	127 ms	0.000254 ms/array
500,000 (sorted)	41 ms	0.00008 ms/array
700,000	196 ms	0.000280 ms/array
700,000 (sorted)	58 ms	0.00008 ms/array

Final Sorting Competition And Medians:

Final :

- Correctly sorted all rounds ✔
- Beat Group0 !!!
- group 3 took place 4. The sum of places is 8, the sum of medians is 529.0

Testing data and different versions of the algorithm:

https://github.com/KenBroden/sorting-comp_deepblue/blob/main/TestingNotes.md