Group 6

Blake Johnson

Scoring

Time and Correctness

(Based on updated sorting results)

- Sorted with a median of 1466 on the small set of data
- Sorted with a median of 11712 on the large set of data
- Reported no correctness errors

```
74   Group 6:
75
76   1528
77
78   1466
79
80   1461
81
82   Median: 1466.0
83   Results of diff:
```

```
74   Group 6:
75
76   11515
77
78   12455
79
80   11712
81
82   Median: 11712.0
83   Results of diff:
```

Placement

- Placed 5th with a sum of places being 11
- Had a sum of medians of 13178

```
group 3 took place  1. The sum of places is 2, the sum of medians is 2120.0

group 5 took place  2. The sum of places is 5, the sum of medians is 2262.0

group 8 took place  3. The sum of places is 5, the sum of medians is 2485.0

group 2 took place  4. The sum of places is 9, the sum of medians is 17736.0

group 6 took place  5. The sum of places is 11, the sum of medians is 13178.0

group 0 took place  6. The sum of places is 12, the sum of medians is 20613.0

group 7 took place  7. The sum of places is 12, the sum of medians is 21340.0

group 1 took place  8. The sum of places is 16, the sum of medians is 2000000.0

group 4 took place  9. The sum of places is 18, the sum of medians is 2000000.0
```

Changes from Group 0

Sorting Algorithm

- Since the Arrays.sort (Timsort) method works best on real world data (meaning partially sorted) I decided to create a partially sorted array
- Implemented a comparator for Timsort to sort by length of string to avoid converting and comparing with BigIntegers
- This partially sorts the array fairly quick and allows Timsort to work much quicker than versus the random array of BigIntegers

```
67          // YOUR SORTING METHOD GOES HERE.
68          // You may call other methods and use other classes.
69          // Note: you may change the return type of the method.
70          // You would need to provide your own function that prints your sorted array to
71          // a file in the exact same format that my program outputs
72          private static void sort(String[] arr) {
73                  SortingCompetitionComparator comp = new SortingCompetitionComparator();
74
75                  //Sorts all numbers by their lengths
76                  //When combined with Timsort, it manages to sort quicker than just Timsort on completely unsorted array
77                  //Implemented from https://www.geeksforgeeks.org/sort-array-large-numbers/
78
79                  Arrays.sort(arr, (left, right) -> {
80                          if(left.length() != right.length())
81                          /* If length of left != right, then return
82                        the diff of the length else  use compareTo
83                        function to compare values.*/
84                                  return left.length() - right.length();
85                          return comp.compare(left, right);
86                  });
87
88
89          }
```

- Source is credited in code

```
toSort = data.clone(); // clone again

Thread.sleep(10); // to let other things finish before timing; adds stability of runs

long start = System.currentTimeMillis();

sort(toSort); //Preliminary sort based on length of given number, not yet correctly sorted

Arrays.sort(toSort, comp); //Final Timsort to make array correctly and fully sorted

long end = System.currentTimeMillis();
```

- Use modified sorting algorithm to partially sort array
- Then call Arrays.sort on partially sorted array

Comparator

- Made minor changes to the compare method
- Added if statements to compare negative numbers to non-negative numbers to avoid some BigInteger comparisons
- This lead to very minor improvements, but not consistently

```java
	public static class SortingCompetitionComparator implements Comparator<String> {

		@Override
		public int compare(String o1, String o2) {

			//Checks whether or not one of the inputs is
			//negative, making an easy comparison
			if (o1.contains("-") && !o2.contains("-")){
				return -1;
			}else{
				if (!o1.contains("-") && o2.contains("-")){
					return 1;
				}
			}

			if(o1.contains("/")){
				if(o2.contains("/")){
					return compareFractions(o1, o2);
				}else{
					return compareFractionAndDecimal(o1, o2);
				}

			}else{
				if(o2.contains("/")){
					return -compareFractionAndDecimal(o2, o1);
				}else{
					return compareDecimals(o1, o2);
				}
			}

		}
```

Runtime

Algorithm Runtime

- Since this utilizes Timsort it has a worst-case (and average-case) runtime of O(nlog(n))
- Best case runtime of O(n) is reserved for already sorted arrays
- Timsort is not an inplace sorting algorithm so data is stored in small chunks that are sorted and then merged together