Assignment Title: Sorting Algorithms Performance Evaluation

Team Members:
- Karen Livni
- Mariia Shaposhnikova

1. Source Code:
   - Sort.java
   - Main.java

2. Environment Specifications:
   - CPU Model: MacBook Pro   Model Identifier:Mac 14,7   Chip:Apple M2
   - OS Version: 7459.121.3
   - RAM Size: 8 GB
   - Cache Size: 192 KB instruction cache, 128 KB data cache, and 12 MB shared L2 cache
   - Java Version: JDK 19

3. Radix Sort Results:
   - Random Inputs (Size: 500000)

	- Range [0, 2^10]:	- Range [0, 2^20]:	- Range [0, 2^30]:
Base 2	avg: 213.41 std deviation: 38.20421311845069	avg: 465.29 std deviation: 28.78238871254434	avg: 712.03 std deviation: 37.71563468907822
Base 2^5	avg: 45.94 std deviation: 0.8101851640211634	avg: 96.16 std deviation: 4.406177481672749	avg: 142.65 std deviation: 5.455960043841954
Base 2^10	avg: 39.58 std deviation: 0.5325410782277735	avg: 46.25 std deviation: 0.9630680142129111	avg: 70.55 std deviation: 0.8874119674649417
Base 2^15	avg: 39.57 std deviation: 0.6205642593640077	avg: 45.91 std deviation: 1.3423486879346975	avg: 46.15 std deviation: 0.5545268253204716
Base 2^20	avg: 39.6	avg: 40.41	avg: 52.05

	std deviation: 0.6164414002968976	std deviation: 0.8135723692456615	std deviation: 0.4974937185533104
Base 2^25	avg: 39.54 std deviation: 0.537028863283902	avg: 40.5 std deviation: 0.8774964387392122	avg: 142.72 std deviation: 23.25858121210317
Base 2^30	avg: 39.53 std deviation: 0.5908468498688981	avg: 40.64 std deviation: 0.9329523031752475	Java heap space

Analysis:
In order to determine the optimal base for an array of unknown range, we calculated the average time of all the three ranges for each base. Thus we determined that overall, the optimal base is 2^15 as its average running time overall was 43.87, which was lower than any other base. If the numbers in the array range up to 2^10, the base should be >=2^10, as the time is higher for smaller bases and after that point it does not make much of a difference. If the numbers in the array range up to 2^20, the base should be >= 2^20, as again, the times for smaller bases is higher and after this point there is little difference. Radix sort with arrays ranging up to 2^30 are slightly problematic to run on our laptops due to the amount of space needed for counting sort to create an array the size of all possible ‘keys,’ but from the information we were able to gather, the base should be 2^15, as this took the least amount of time. Overall, it would seem that for most input ranges *x*<2^30, one might expect the optimal base to be some number >=*x*.

Base choice: 2^15

4. Sorting Algorithms Comparison Results:

- Random Inputs:

Size	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
[redacted]	quickSortRecitation	quickSortClass	mergeSortRecursive	mergeSortIterative	radixSort	Java Array.sort
10000	1.55	1.49	3.52	2.23	19.77	2.51
std	1.2990381056766591	1.1873921003611247	14.725814069177929	2.2398883900766164	0.6611353870426245	1.6748432762500518
50000	7.85	7.35	11.13	11.09	98.41	10.13

std	2.1834605560898046	0.668954408012983	1.3539202339872163	2.474247360309797	2.319892238876625	1.9217439996003638
100000	17.52	17.08	26.08	25.96	208.25	24.04
std	3.5397175028524543	3.170741238259596	4.237168866118036	4.520884869137899	22.706992315143808	6.56036584345721
500000	108.98	106.38	179.41	206.15	1042.89	166.41
std	23.421349235259694	28.559684872211044	38.59328827659027	35.03066513784745	104.45026519832301	49.19066883058209
1000000	360.63	356.53	284.77	302.36	160.9	212.29
std	28.234839402046405839	30.93402848039265839	13.466889024566882	22.662974209048556	8.73097932651315	15.686487815951667

- Sorted Inputs in Increasing Order:

Size	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	Algorithm 6
Size	ckSortRecitation	quickSortClass	rgeSortRecursive	ergeSortIterative	radixSort	va Array.sort
10000	23.65	45.44	3.61	1.47	19.7	0.02
	11.628735958822011	9.97085252122405	12.816750758285005	0.7860661549767913	1.3228756555322976	0.13999999999999987
50000	-	-	6.24	6.16	96.66	0.08
	-	-	6.356288225057144	4.544711211947361	1.5888360519575324	0.30594117081556643

100000	-	-	11.52	11.3	193.76	0.16
	-	-	0.7807688518377264	0.6708203932499376	2.720735194758943	0.3666060555964676
500000	-	-	103.55	102.13	260.73	1.11
	-	-	17.391017796552337	8.129766294303913	3.4755718953864267	0.421781934179263
1000000	-	-	234.13	267.32	315.86	3.48
	-	-	20.278389975537994	29.553301000057512	6.374723523416527	1.6277591959500646

- Sorted Inputs in Decreasing Order:

Size	gorithm 1	gorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	Algorithm 6
	rtRecitation	uickSortClass	mergeSortRecursive	mergeSortIterative	radixSort	Java Array.sort
10000	67.68	68.14	3.14	1.07	19.57	0.05
	1.85946278671117	7.87149269843688	18.551560581255632	0.7649182962905249	1.064847406908615	0.21794494717703397
50000	-	-	7.0	6.69	96.52	0.44
	-	-	11.908820260630353	5.224356419694193	0.9327379053088815	0.5351635264103862
100000	-	-	11.94	11.84	194.46	0.57
	-	-	0.9779570542718125	0.8333066662399855	2.9442146660866975	0.4950757517794622
500000	-	-	99.03	101.52	250.52	7.53

	-	-	13.85096025552019	16.950209438234094	7.136413665140216	4.678578844050831
1000000	-	-	223.74	261.47	295.69	24.26
	-	-	25.274342721424034	37.5047876943731	8.309849577459271	13.5982498873936

5. Results Analysis:

Random array:

When sorting a random array, from the developed sorting algorithms it seems that radixSort is the least time-efficient. Along all input sizes it manages to sort the array the slowest, whilst also mainataining a low standard deviation.

On most input sizes Arrays.sort seems to be the most efficient.

The differences in time between the recursive and iterative methods of mergeSort seem to be insignificant according to our data (where the differences vary only about 1-2 units). The same situation arises when comparing the two quickSort algorithms.

Sorted in increasing order array:

For sorted inputs, the quicksort algorithms do not run with array sizes >= 50000 as their running time increases from O(nlogn) to O(n^2). The radix sort algorithm seems to be the least time efficient. Both merge algorithms have similar running times. However, as expected Arrays.sort remains as the quickest from all.

Sorted in decreasing order array:

For sorted in reverse arrays, the same situation arises with the quicksort algorithms for the same reasons. Both merge algorithms have similar time durations. Again we notice radix sort is the slowest algorithm of them all. As in previous results, Arrays.sort is the quickest due to the use of double pivots.

6. Cases with Failures:

Radix sort failed to complete using the base 2^30 with ranges 0-2^30 returning the error java heap space. This error occurred along all required ranges used. This issue is most likely due to the limited space capacity in the laptop used for testing. This error was consistent even after running it 3 times (where each time each algorithm and each input size would be run NUMITER = 100 times)

When comparing all the algortihms quicksort failed to run after incrweasing the array size to 10000. This was already previosulky discussed above.