Anthony Brennan ajb286

Final Report

For this experiment, I analyzed the Big-O runtime for three different sorting algorithms: Bubble Sort, Cocktail Sort, and Radix Sort. The goal for each array is to sort an array (sorted or unsorted) in ascending order (1, 2, 3, …). Bubble Sort uses the simplest algorithm of the 3. Bubble swap works by repeatedly swapping adjacent numbers if they are out of order. It goes through the array from left to right. It will continue going from left to right through the array until it completes an entire pass without swapping any numbers.

**Bubble Pass Example**

1st Pass 2nd Pass 3rd Pass:

(**5 1** 4 2) –> (**1 5** 4 2) (**1 4** 2 5) –> (**1 4** 2 5) (**1 2** 4 5) –> (**1 2** 4 5)  
(1 **5 4** 2) –> (1 **4 5** 2) 5 > 4 (1 **4 2** 5) –> (1 **2 4** 5) 4 > 2 (1 **2 4** 5) –> (1 **2 4** 5)  
(1 4 **5 2**) –> (1 4 **2 5**) 5 > 2 (1 2 **4 5**) –> (1 2 **4 5**) (1 2 **4 5**) –> (1 2 **4 5**)

Cocktail Sort shares many similarities with Bubble Sort. Like Bubble Sort, Cocktail Sort works by repeatedly swapping adjacent numbers if they are out of order. It will also continue through passes until it completes an entire pass of the array without swapping any numbers. Unlike Bubble Sort, Cocktail Sort alternates between passing the array from left to right and from right to left. It will also skip the end of the array on the next pass in the same direction.

**Cocktail Pass**

1st Forward Pass 1st Backward Pass 2nd Forward Pass

(**5 1** 4 2) –> (**1 5** 4 2) (1 4 **2 5**) –> (1 4 **2 5**) (**1 2** 4 5) –> (**1 2** 4 5)  
(1 **5 4** 2) –> (1 **4 5** 2) 5 > 4 (1 **4 2** 5) –> (1 **2 4** 5) 2 < 4 (1 **2 4** 5) –> (1 **2 4** 5)  
(1 4 **5 2**) –> (1 4 **2 5**) 5 > 2 (**1 2** 4 5) –> (**1 2** 4 5)

Radix Sort differs from both Bubble Sort and Cocktail Sort. Rather than pass through the array, Radix Sort sorts the array by the n’th digit of each number. This means that it will sort the numbers based on the 1’s place, then the 10’s place, and so on until each numbers place has been sorted. If there are two numbers ending with 5, for example, Radix Sort will keep them in the same order as the sort.

**Radix Sort Example**

Original 170, 45, 75, 90, 802, 24, 2, 66

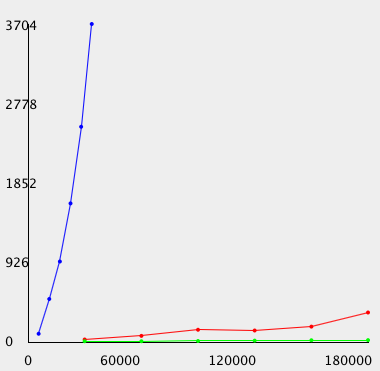
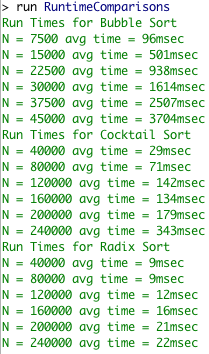
Sort by 1’s place 17**0**, 9**0**, 80**2**, **2**, 2**4**, 4**5**, 7**5**, 6**6**

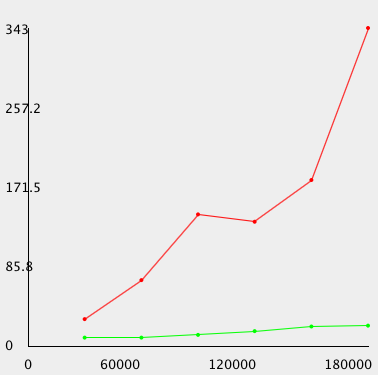
Sort by 10’s place 8**0**2, 2, **2**4, **4**5, **6**6, 1**7**0, **7**5, **9**0

Sort by 100’s place 2, 24, 45, 66, 75, 90, **1**70, **8**02

For the experiment, I created three programs, one for each of the algorithms, used the Grapher class from homework, and created a main method to run all of the algorithms. In the main method, I created 4 arrays containing random numbers between 0 and 99,999 for each of 6 array sizes. The array sizes for Bubble Sort are 7,500, 15,000, 22,500, 30,000, 37,500, and 45,000, and the array sizes for Cocktail Sort and Radix Sort are 40,000, 80,000, 120,000, 160,000, 200,000, and 240,000. The arrays were randomly generated as it would be extremely time consuming and difficult to create such large arrays. The 4 arrays created for each size were averaged out. This was done to reduce the chance for outliers affecting the results. The array size differences for the algorithms was done because it became clear Bubble Sort would take minutes to run large arrays, while small arrays for the other two algorithms had runtimes that were too small to make any conclusions. The start and stop times are recorded from the when each iteration of the algorithms begins sorting to when they are finished sorting.

After running the experiment, we get a fairly clear picture of the Big-O runtimes for all 3 algorithms. Based on the results below, Bubble Sort (blue in the graph) increases upwards exponentially at a very fast rate, indicating that it has a Big-O runtime of O(N2). Cocktail Sort (red) also appears to have an upward curve, although gentler curve than Bubble Sort. This indicates a Big-O runtime of O(N2). Radix Sort (green) shows a linear trend, indicating a Big-O runtime of O(N).



*Blue: Bubble, Red: Cocktail, Green: Radix*

All of these results are consistent with the theoretical Big-O runtimes. Both Bubble Sort and Cocktail Sort are expected to have Big-O runtimes of O(N2) at worst for unsorted arrays and O(N) for sorted arrays. This is because both use nested loops. But if an array is pre-sorted, both algorithms will only do one pass and the runtime will be the total numbers, N, in the array. Radix sort is expected to have a Big-O runtime of O(N), since although it has many loops, none of them are nested. Between the 3 algorithms, Radix Sort would be the most efficient to use. It has smaller runtimes, plus it is O(N) meaning that runtimes will increase at a constant rate as arrays get bigger, rather than at an exponential rate like with Bubble Sort and Cocktail Sort. If one had to choose between the two swapping algorithms, Cocktail Sort is way more efficient.

**Appendix**

* The main method file is called RuntimeComparisons.java, the entire experiment is run through this java file. The top part is for generating the arrays, the middle is for running the algorithms, and the bottom is for creating the graph.
* BubbleSort.java, CocktailSort.java, and RadixSort.java are the 3 sorting algorithms.
* Grapher.java is used to create the graphs for the runtimes.
* In the graph, Bubble Sort is blue, Cocktail Sort is red, and Radix Sort is green.