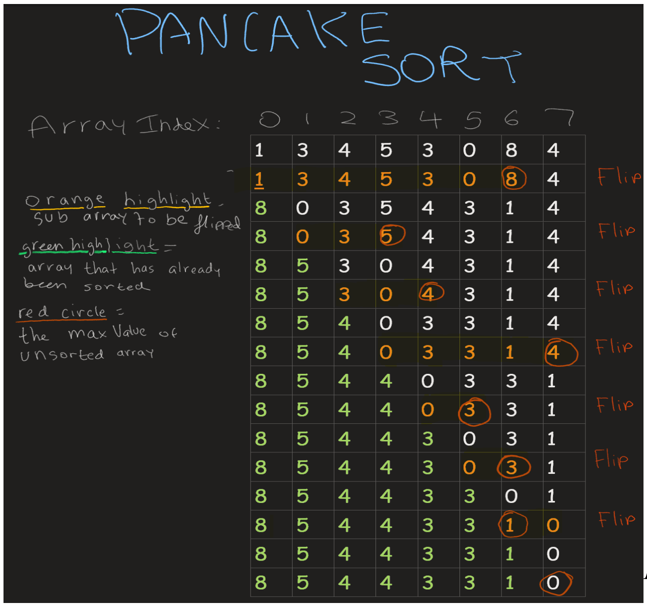
Alice Cutter May 31st, 2021

Computer Science 201: Data Structures

Pancake Sort

Pancake Sort Description

Pancake Sort is a sorting algorithm. In the algorithm, a series of comparable items is considered a stack of pancakes, with each item being a pancake. Operations that can be done on the stack are flips(reversals) and finding the maximum size. Pancake Sort has an index counter that is initialized pointing at zero and increments as the sorting method works through the array. First, the array is traversed and Pancake Sort finds the maximum value and flips everything including that index to the unsorted portion of the array. The unsorted index is then incremented and the process starts again. The data is sorted in descending order.


PANCAKE SORT
orange highlight - sub array to be flipped
green highlight = array that has already been sorted
red circle = the max Value of unsorted array

Array Index:	0	1	2	3	4	5	6	7	
	1	3	4	5	3	0	8	4	
	1	3	4	5	3	0	8	4	Flip
	8	0	3	5	4	3	1	4	
	8	0	3	5	4	3	1	4	Flip
	8	5	3	0	4	3	1	4	
	8	5	3	0	4	3	1	4	Flip
	8	5	4	0	3	3	1	4	
	8	5	4	0	3	3	1	4	Flip
	8	5	4	4	0	3	3	1	
	8	5	4	4	0	3	3	1	Flip
	8	5	4	4	3	0	3	1	
	8	5	4	4	3	0	3	1	Flip
	8	5	4	4	3	3	0	1	
	8	5	4	4	3	3	1	0	Flip
	8	5	4	4	3	3	1	0	
	8	5	4	4	3	3	1	0	

Figure 1

Pancake Sort is similar to insertion sort in that it traverses an array but is different in that it does not compare every element to each other in the array. Unlike traditional sorting algorithms that aim to sort a list with the fewest comparisons, Pancake Sort tries to do so with the fewest reversals possible. Therefore, Pancake Sort is useful when the only operation that can be performed is reversing. An example of this could be a doubly linked list with pointers for head and tail of the list. Pancake Sort is similar to insertion sort in that it traverses through an array from start to finish but different. The main difference though is that instead of swapping one item at a time, it flips a subsection of the array to put the largest item at the front of the array.

Discussion of Results and O(n) complexity:

Because the Pancake Sort algorithm consists of two parts: the find max and the flip, the O(n) complexity can be calculated as the sum of the two of them. The first part, finding the max, has its worst case run time as the sum of the series (n - 1) + (n - 2) + (n - 3) ... (n - n) , which results in n (n+1)/2 giving it an O complexity of O(n^2). The reason that n is incrementally subtracted from is that the size of the subarray being searched for the maximum value grows smaller by one each iteration of Pancake Sort's main loop. It's smaller because the search only looks from the unsorted portion onward and each iteration adds to the sorted portion.

The second part of the algorithm is the flip portion which has a run time of O(n^2). The exact runtime is calculated for every item in the array in the same way that find max is. It is the sum of the series (n - 1) + (n - 2) + (n - 3) ... (n - n). Again like before, it results in n (n +1) / 2 which comes out to be O(n^2). The reason that n is incrementally subtracted from is because the sorted portion of the array is not flipped, thus the size of the subarray getting flipped grows smaller as the sorted portion of the array grows.

Together, the runtime of the Pancake Sort is still O(n^2) because each part of the algorithm is independent of the other. Using the sum of the two part's runtime the exact runtime is O(2n^2) which results in O(n^2).

Pancake Sort is generally less efficient than Java's implemented sorting algorithms; in this case, Pancake Sort will be compared to Quick Sort because that is Java's default sort method for primitive arrays. Because of it’s runtime and general structure it is very similar to insertion sort which also traverses the unsorted portion of the array. For arrays of integers, it is less efficient than Quick Sort, which has a runtime of O(n log n). The following charts demonstrate how Java’s sort algorithm (Quick Sort) compares to Pancake Sort.

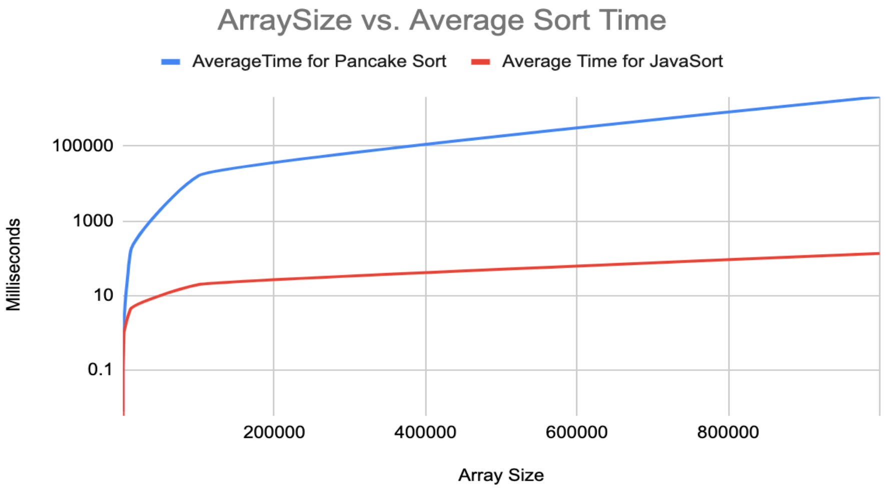
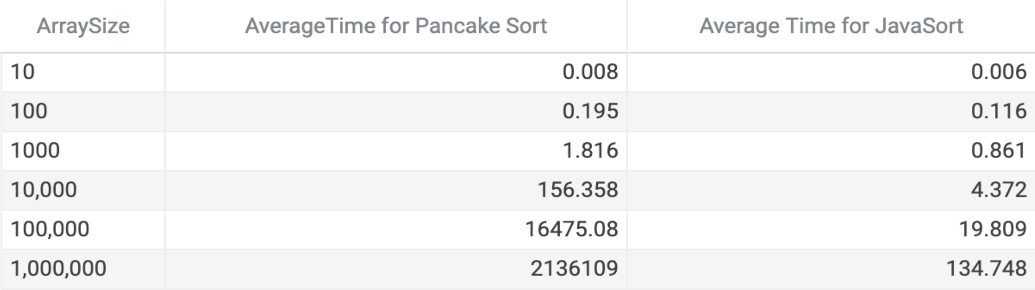


Figure 2

Data was collected through recording the time before and after each of the sorting algorithms were executed. Array sizes were calculated by exponentially increasing 10 until we reached 10^6. The exponential growth so that we could see how the array grew over time. Their content will be discussed in the next paragraphs.

The input that best suits Pancake Sort is randomly ordered arrays. It works well because when the maximum element that is found each time is closer to the already sorted portion of the array, the number of swaps in the flip method is minimized. A worst case scenario for Pancake Sort is where the elements are in reverse sorted order (ascending) because it means that when the maximum value is found(the end of the current subarray), it must sort the largest subarray possible. When comparing reversed arrays to randomly unsorted arrays, Pancake Sort took about seven times longer on a 1000 element array than it did on a random array. It took 7.732 milliseconds to sort a reversed array where a random array was 1.186 milliseconds.

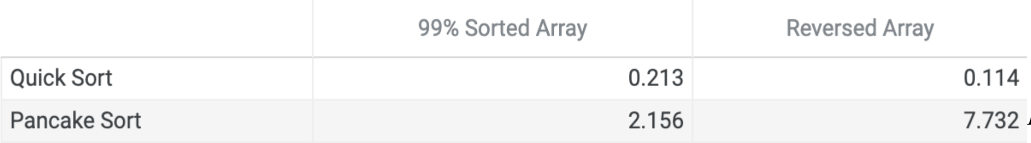
Pancake sort is generally slower than the Java implemented Quick Sort. For smaller sets of data (under 1000) the time on a computer that it takes to carry out the two algorithms are close. However as the data set gets larger, the O(n^2) complexity of Pancake sort cannot compete with the O(nlogn) complexity of Quick Sort.



ArraySize	AverageTime for Pancake Sort	Average Time for JavaSort
10	0.008	0.006
100	0.195	0.116
1000	1.816	0.861
10,000	156.358	4.372
100,000	16475.08	19.809
1,000,000	2136109	134.748

Figure3

When sorting special case arrays, such as almost sorted and reversed, Quick Sort again outperformed pancake sort. A 99% sorted array run with 1000 elements in the array produced a runtime for Quick Sort of 0.213 milliseconds whereas it took pancake sort 2.156 milliseconds. Quick Sort was about 10 times faster than pancake sort. Additionally, as mentioned earlier, when compared to a complete reversed away, Quick Sort performed the sort on an 1000 element array in an average of 0.114 milliseconds while pancake sort took 7.732 milliseconds.



	99% Sorted Array	Reversed Array
Quick Sort	0.213	0.114
Pancake Sort	2.156	7.732

Figure 4g

Pancake sort is not the best sorting algorithm. Because it needs to iterate all the way through the array, there is very little opportunity for the runtime to get cut down in the same way that it can for Quick Sort. Additionally, Pancake Sort is an in-place sorting algorithm, meaning it uses up a negligible amount of extra memory, unlike, for instance, MergeSort. In conclusion, Pancake Sort is good for the original problem that it was created for: how to sort a disordered stack or pancakes with just a spatula and the minimum number of flips (reversals).