Exploring Sorting Runtimes

Array Size	Sort Type	Bubble Sort	Selection Sort	Insertion Sort
10	Ascending	1.0	1.0	0.0
	Descending	2.0	1.0	1.0
	Random	2.6	2.3	2.7
	Nearly Sorted	9.3	1.0	0.3
100	Ascending	754.6	453.3	3.0
	Descending	202.3	95.3	127.7
	Random	431.6	98.7	65.0
	Nearly Sorted	817.3	199.3	13.0
500	Ascending	463.0	995.6	23.3
	Descending	568.3	1063.0	615.3
	Random	573.3	666.3	411.3
	Nearly Sorted	1037.3	622.0	229.6
1000	Ascending	2615.7	3036.7	29.7
	Descending	2295.3	3200.0	2906.7
	Random	2447.7	2296.3	4815.7
	Nearly Sorted	2182.7	3636.7	459.3

The sorting algorithms used in the Sorts class have various time complexities and unique properties:

Bubble sort - In the worst and average situation, bubble sort has an O(n2) time complexity, which makes it ineffective for big arrays. When two neighboring components are in the wrong sequence, it constantly compares them and switches them. Although bubble sort is straightforward to use, it is not advised for large datasets due to its low performance.

Selection sort - is ineffective for big arrays due to its time complexity, which is O(n2). The minimum element is located by iterating over the array, and it is then swapped in place of the current element. Selection sort outperforms bubble sort in terms of the quantity of swaps, but it still requires quadratic time.

Insertion sort - this outperforms bubble sort and selection sort for tiny arrays or partially sorted arrays while having an average and worst-case time complexity of O(n2). It keeps the array's sorted section intact and inserts each element in the appropriate place. For small or nearly sorted datasets, insertion sort is effective; however, it becomes slow for large datasets.

Overall, The temporal complexities and properties of the sorting algorithms in the Sorts class vary. Both the bubble sort and the selection sort are ineffective for big arrays due to their O(n2) time complexity. While selection sort finds the least element and replaces it with the current element, bubble sort compares and swaps adjacent components. The performance of insertion sort is improved for tiny or incompletely sorted arrays, but it also has an O(n2) time complexity. It keeps a segment that is sorted and places each element where it belongs. These algorithms are suited for various scenarios depending on the size and arrangement of the dataset, but they each have strengths and disadvantages.