1. Bubble Sort:
2. 10 Numbers:

Start: 15:21:44.161

End: 15:21:44.162

Time taken: 0.001 sec

1. 100 Numbers:

Start: 21:29:24.159

End: 21:29:24.165

Time taken: 0.006 sec

1. Million Numbers (Almost Sorted):

Start: 16:27:33.750

End: 16:48:45.816

Time take: 21 min 12.066 sec

1. Million Numbers (Random):

Start: 16:52:15.657

End: 17:34:48.76

Time taken: 42min 33.103sec

Algorithm Description: Bubble sort is the simplest sort that repeatedly swaps two numbers that are next to each other in the array if the one before is greater value then the one that comes after. Bubble sort always runs O(n^2) time even if the array is sorted. You can optimize this by putting an ‘if’ statement to stop the algorithm if the inner loop didn’t doesn’t need to swap.

1. Selection Sort:
2. 10 Numbers:

Start: 15:36:09.676

End: 15:36:09.677

Time taken: 0.001 sec

1. 100 Numbers:

Start: 15:36:43.429

End: 15:36:43.430

Time taken: 0.001 sec

1. Million Numbers (Almost Sorted):

Start: 17:38:08.694

End: 17:55:24.458

Time taken: 17min 15.764 sec

1. Million Numbers (Random):

Start: 18:01:07.719

End: 18:18:26.928

Time taken: 17 min 19.209 sec

Algorithm Description: This sorting algorithm simply runs through the array and compares the values in the left most side in the array and when it finds a value larger then what is in the left most side of the array it swaps the number and runs through the array again. If the number in the left most side of the array is the smallest, it moves to the next position in the array and repeats the process until the array is sorted.

This algorithm is not good for large data sets as its best, average and worst-case complexities are all of Ο(n^2), where n is the number of items which means that regardless of whether the array is sorted, partially sorted or all random it will still have about the same run time regardless. It is still faster than bubble sort.

1. Insertion Sort:
2. 10 Numbers:

Start: 15:35:14.134

End: 15:35:14.135

Time taken: Time taken: 0.001 sec

1. 100 Numbers:

Start: 15:37:10.898

End: 15:37:10.899

Time taken: 0.001 sec

1. Million Numbers (Almost Sorted):

Start: 15:39:11.410

End: 15:39:40.792

Time taken: 29.382 sec

1. Million Numbers (Random):

Start: 15:58:02.668

End: 16:05:45.207

Time taken: 7 min 42.539

Algorithm Description: Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part. Insertion sort best case is a sorted or an almost sorted list as the complexity is Ο(n) and O(n^2) when the list is unsorted. The worst case for insertion sort is an array sorted in reverse.

1. Shell Sort:
2. 10 Numbers:

Start: 15:37:33.953

End: 15:37:33.955

Time taken: 0.002 sec

1. 100 Numbers:

Start: 15:38:04.595

End: 15:38:04.596

Time taken: 0.001 sec

1. Million Numbers (Almost Sorted):

Start: 12:21:49.171

End: 12:21:49.527

Time taken: 0.356 sec

1. Million Numbers (Random):

Start: 15:57:29.742

End: 15:57:30.589

Time taken: 0.847 sec

Algorithm Description: Shell sort is kind of like a variation of insertion sort. It splits the array into many segments or shells and organizes parts of the data with insertion sort running on each shell, to be more organized so that it can finally run an insertion sort through the entire list to be completely sorted. The time complexity is usually O(n^3/2) but I have read that it can be O(n^4/3) and even O(n\*log2(n)) depending on how the array is sorted.