Sort Algorithm	Trial 1: size 1k	Trial 2: size 1k	Trial 3: size 1k	Trial 1: size 31k	Trial 2: size 31k	Trial 3: size 31k	Trial 1: size 1M	Trial 2: size 1M	Trial 3: size 1M	time complexity
StandardSort	0ms	0ms	0ms	2ms	2ms	2ms	162ms	169ms	125ms	O(nlogn)
MergeSort	0ms	0ms	0ms	2ms	1ms	3ms	215ms	108ms	195ms	O(nlogn/2)
InPlaceMergeSort	0ms	0ms	0ms	2ms	3ms	2ms	121ms	105ms	181ms	O(nlogn)
QuickSelect	0ms	0ms	0ms	0ms	0ms	0ms	0ms	0ms	0ms	O(n)
WorstCaseQuickSelect	0ms	0ms	0ms	0ms	0ms	0ms	28ms	10ms	4ms	O(n)
HalfHeapSort	0ms	0ms	0ms	0ms	1ms	1ms	77ms	96ms	27ms	O(nlogn/2)
HalfSelectionSort	0ms	0ms	0ms	160ms	106ms	122ms	140943	137917 ms	113337 ms	O(n^2/2)

The duration (in milliseconds) when executing each sorting algorithm depending on the input size was very interesting because when the input size was only 1000 integers, all sorting algorithms resulted to 0ms which represents how it seems there's no difference between them due to low input size. All three trials with input size 1000 took 0ms to find the median. However, after increasing the input size to 31k, there's a noticeable difference between the different sorting algorithms and HalfSelectionSort seems to be the most inefficient when it comes to finding the median. HalfSelectionSort gets even worse when the input size is 1 million. However, StandardSort, MergeSort, InPlaceMergeSort all generally take the same amount of time. QuickSelect is the fastest, second is WorstCaseQuickSelect and third is HalfHeapSort.

Even though HalfSelectionSort is recognized to be a simple algorithm, it is exponentially inefficient with larger datasets. Reason being it involves swapping elements in a nested loop which leads to higher numbers of comparisons and swaps. Unlike other algorithms, there's no early termination if conditions are met such as finding the median. Also, it ignores the order of elements in the array and always searches for the minimum element regardless if it is sorted or unsorted.

QuickSelect remains to be the most efficient/fastest when it comes to finding the median. This is because of its divide and conquer approach and dividing into subproblems by selecting pivot and partitioning. There's also early termination when the element is found in the array which saves a lot of time.