Statistics on Timing

Sort Function	Input of size 1000 (Inputs 1-3)	Inputs of size 31K (Inputs 4-6)	Inputs of size 1M (Input 7-9)
HalfSelection Sort	3 milliseconds	3461 milliseconds	Size Too Big
Standard Sort	0 milliseconds	8 milliseconds	335 milliseconds
MergeSort	0 milliseconds	29 milliseconds	1060 milliseconds
MergeSort(In-Place)	0 milliseconds	15 milliseconds	597 milliseconds
HeapSort	0 milliseconds	5 milliseconds	262 milliseconds
Quickselect	0 milliseconds	0 milliseconds	46 milliseconds
Quickselect (Worst Case)	1371 milliseconds with an input size of 20,000.		

HalfSelection Sort

File	Duration (milliseconds)
input1	3
input2	3
input3	3
input4	3489
input5	3444
input6	3451
input7	Size Too Big
input8	Size Too Big
input9	Size Too Big

In general, the time complexity of the selection sort is $O(N^2)$ on average and worst. HalfSelectionSort reduces the number of iterations through the list by half, which saves us time since we don't have to go through the whole input. So, the time complexity would look more like this $O\left(\left(\frac{N}{2}\right)^2\right)$, but this simplifies to $O(N^2)$ in big-O notation.

StandardSort

File	Duration (milliseconds)

input1	0
input2	0
input3	0
input4	7
input5	8
input6	8
input7	328
input8	345
input9	332

StandardSort from the C++ library takes O(NlogN) on average and worst case. Since we are not modifying it, this sort will remain at O(NlogN) on average and worst case.

MergeSort

File	Duration (milliseconds)
input1	0
input2	0
input3	0
input4	30
input5	27
input6	31
input7	1047
input8	1071
input9	1064

MergeSorts time complexity is O(Nlog(N)) average and worst case because of the divide and conquer algorithm. Since we cannot interrupt mergesort, it will remain O(Nlog(N)).

InPlaceMergeSort

File	Duration (milliseconds)

input1	0
input2	0
input3	0
input4	15
input5	14
input6	17
input7	609
input8	589
input9	593

InPlaceMergeSort time complexity is also O(Nlog(N)) average and worst case because of the divide and conquer algorithm. Since we cannot interrupt mergesort, it will remain as O(Nlog(N)). However, InPlaceMergeSort performs better than MergeSort. This is because MergeSort had to dynamically allocate new memory during the merging process. While InPlaceMergeSort does not have this problem, it saves more time, resulting in a lower duration for all inputs.

HalfHeapSort

File	Duration (milliseconds)
input1	0
input2	0
input3	0
input4	5
input5	6
input6	5
input7	257
input8	267
input9	264

In general, heap sort's time complexity is O(Nlog(N)) average and worst case. For heapsort, building the heap will take O(N). Then, we delete the minimum n times, which is O(NlogN) time. Altogether, the time for heapsort would be O(N) + O(NlogN) = O(NlogN). So, for HalfHeapSort, we save time by only deleting the elements less than the median or $\frac{N}{2}$ elements. However, we still need to build the heap. Together, the time complexity would be O(N) + O($\frac{N}{2}\log\frac{N}{2}$). In big-O notation, this will simplify to O(NlogN).

Surprisingly, HalfHeapSort took significantly less time than MergeSort and InPlaceMergeSort. I was expecting HalfHeapSort to perform worse because of the time it takes to build the heap and percolate down after deleting the min every time.

Quickselect

File	Duration (milliseconds)
input1	0
input2	0
input3	0
input4	0
input5	0
input6	1
input7	44
input8	45
input9	50

Quickselect time complexity is O(n) for the average case. This is because Quickselect only recurses on one side, unlike Quicksort. As a result, the time complexity is reduced from O(NlogN) to O(N).

WorstCaseQuickselect
The time for WorstCaseQuickselect is about 1371 milliseconds. This is because we are intentionally picking the worst pivot possible. As a result, the partitioning is only removing one element at a time, so the time complexity would be O(N^2).