

Timing Statistics:

Input Size	Algorithm	Average Time (ms)
1k	Standard Sort	0.4
	HalfSelectionSort	3
	MergeSort	1
	InPlaceMerge	1
	HalfHeapSort	0.2
	QuickSelect	0.8
	QS w/ MedianofMedians	0.6
31k	Standard Sort	3
	HalfSelectionSort	1188
	MergeSort	44
	InPlaceMerge	27
	HalfHeapSort	7
	QuickSelect	2
	QS w/ MedianofMedians	10
1M	Standard Sort	52
	HalfSelectionSort	Input too big
	MergeSort	1094
	InPlaceMergeSort	641
	HalfHeapSort	157
	QuickSelect	37
	QS w/ MedianofMedians	131

Worst Case Input quickSelect:

Median of 3: ~167 ms average

Median of Medians: ~3 ms average

Algorithmic Analysis:

HalfSelectionSort:

$O(n^2)$ but more precisely we are only swapping and comparing about half as much as a full selection sort, so the constant factor will be a lot smaller, and in practice it will be faster than full selection sort.

std::sort

is $O(n^2)$ although this is a rare case. Its average and best case is $O(n \log n)$ and in practice it's generally faster than any other algorithm used in this project apart from quickSelect.

mergeSort and inplaceMergeSort

Both mergeSort and inplaceMergeSort are $O(n \log n)$ but inplaceMerge has a smaller space complexity. std::merge has a space complexity $O(n+m)$ and std::inplace_merge has space complexity $O(1)$ since it uses a constant amount of additional memory

halfHeapSort

Is still $O(n \log n)$ like full Heap Sort. This is because you still build a full heap which is $O(n)$ and the deleteMax phase will be $O(\log n)/2$ which is still $O(\log n)$. In practice it'll be faster but the worst case time complexity is the same.

quickSelect

Has average time complexity $O(n)$ and worst case time complexity $O(n^2)$ although this is quite rare and depends on the pivot selection method used and the input. Median of 3 generally avoids worst case scenarios and helps quickSelect average $O(n)$ time. My worstCasePivot generator function causes quickSelect with the median of 3 pivot selection method to approach $O(n^2)$.

Using the **median of medians** method for pivot selection guarantees $O(n)$ complexity although this comes with a large constant factor. In practice, median of medians was a lot slower for larger inputs. However it really improved performance for quickselect on the worst case input (~3ms instead of ~167ms average).

What surprised me: That inplaceMerge generally performs better than merge. Also how much better medianOfMedians performs for worst case input. 167ms vs 3ms is a huge difference!!