CSCI 335 Project 2 Report

	Half Selection Sort	In Place Merge Sort	Standard Sort	Merge Sort	Half Heap Sort	Quick Select
Input 1 time	1 ms	0 ms	0 ms	0 ms	0 ms	0 ms
Input 2 time	1 ms	0 ms	0 ms.	0 ms	0 ms	0 ms
Input 3 time	2 ms	0 ms	0 ms	0 ms	0 ms	0 ms
Input 4 time	1879 ms	8 ms	4 ms	15 ms	3 ms	1 ms
Input 5 time	1858 ms	8 ms	4 ms	15 ms	3 ms	1 ms
Input 6 time	1894 ms	8 ms	4 ms	14 ms	3 ms	1 ms
Input 7 time	N/A	332 ms	189 ms	540 ms	142 ms	65 ms
Input 8 time	N/A	328 ms	190 ms	545 ms	143 ms	66 ms
Input 9 time	N/A	332 ms	190 ms	540 ms	141 ms	58 ms

WORST CASE QUICK SELECT: 1359 ms with input of 20K

	Half Selection Sort	In Place Merge Sort	Standard Sort	Merge Sort	Half Heap Sort	Quick Select
Avg. of inputs of size 1000	1.33 ms	0 ms	0 ms	0 ms	0 ms	0 ms
Avg. of inputs of size 31K	1877 ms	8 ms	4ms	14.66	3 ms	1 ms
Avg. of inputs of size 1M	N/A	330.66 ms	189.66 ms	541.66 ms	142 ms	63 ms

Algorithmic analysis

Half Selection sort: this sort tends to have a time complexity of O(n^2) in the avg. and worst case. It is the longest of all the sorts and runs way too long with large inputs. It does only go over half of the iterations performing n(n-1) / 2 comparisons in the worst case, simplifying to O(n^2). This surprised me a bit because I thought it would be able to handle large inputs being that it only goes through half.

In Place Merge Sort: Similar to merge sort. Time complexity O(N (log N)). However it works a bit faster than merge sort, still performing in logarithmic time on various inputs. Timing is not too surprising knowing that it stops when median is found and doesn't have to go through the whole thing.

Standard Sort: this sort is from the C++ library and since we don't really modify avg, time complexity will stay as O(N (log N)). Timed just as expected with logarithmic time as inputs increase.

Merge Sort: this sort is also known for being a logarithmic time complexity. Both the average and worst case are logarithmic (O(N (log N)), not surprising.

Half Heap Sort: O (N (log N)) in all cases. Only have to remove n/ 2 elements from the heap instead of n cutting on time significantly.

Quick Select: Since recursion is only on one side the avg. the time complexity is O(n). This happens when pivot is picked using a median of three. Times here are very fast even with big inputs like 1M. Even though I knew it was fast I didn't expect it to be this fast. However testing it out definitely made me aware of its speed. If the pivot is placed at the beginning or end the worst case time complexity is O(n^2).

Overall Quickselect seems to be the winner of all sorts. However in the worst case it is not as consistent.