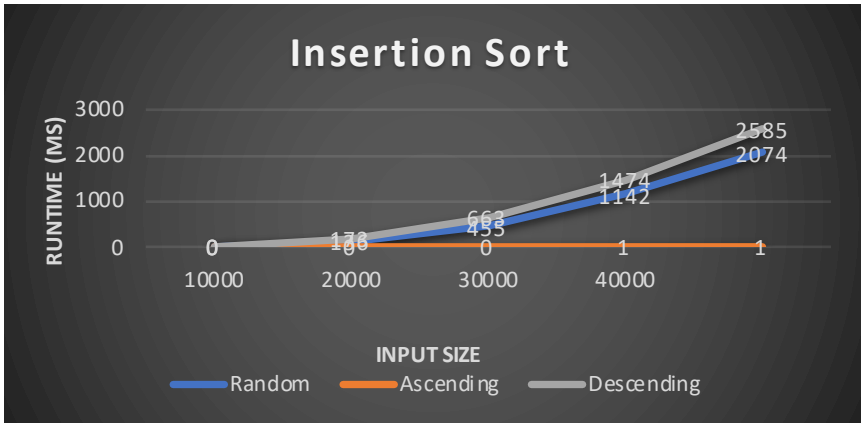
Hannah Posch HW #8

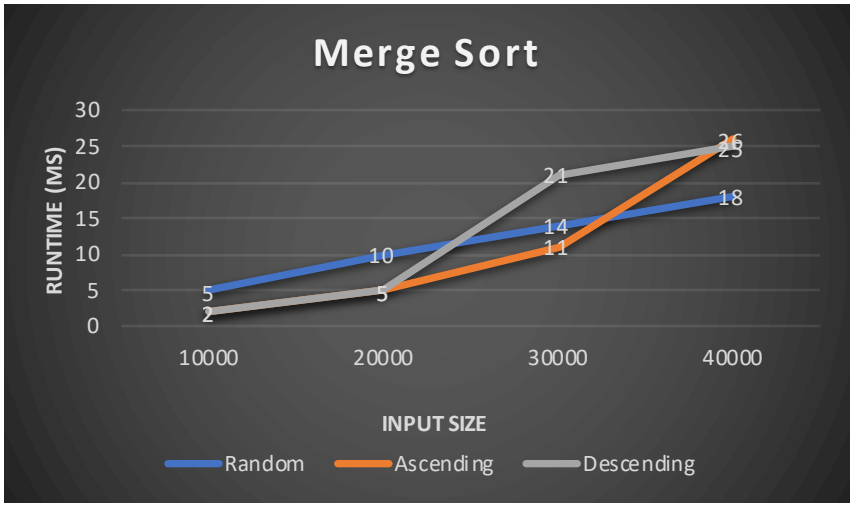
Insertion Sort

	Average runtime for three runs (ms)		
Input Size	Random	Ascending	Descending
10000	136	0	173
20000	455	0	663
30000	1142	1	1474
40000	2074	1	2585



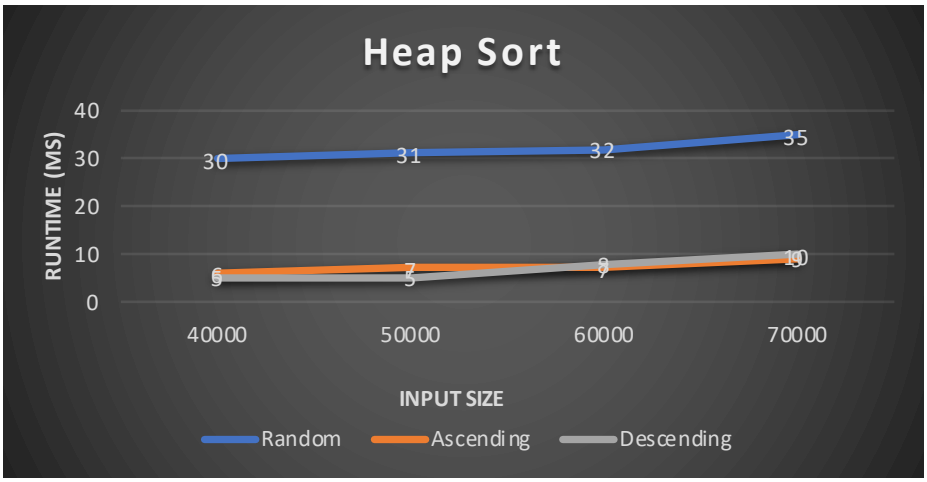
Merge Sort

	Average runtime for three runs (ms)		
Input Size	Random	Ascending	Descending
10000	5	2	2
20000	10	5	5
30000	14	11	21
40000	18	26	25



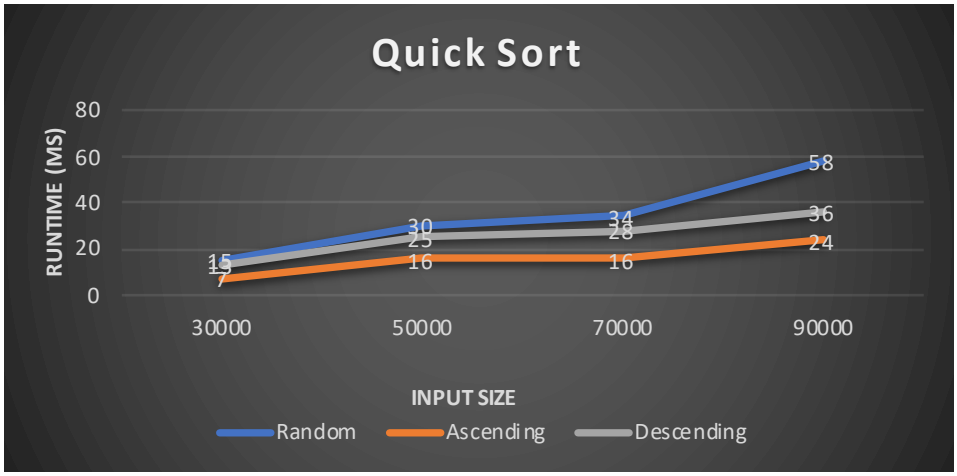
Heap Sort

	Average runtime for three runs (ms)		
Input Size	Random	Ascending	Descending
40000	30	6	5
50000	31	7	5
60000	32	7	8
70000	35	9	10



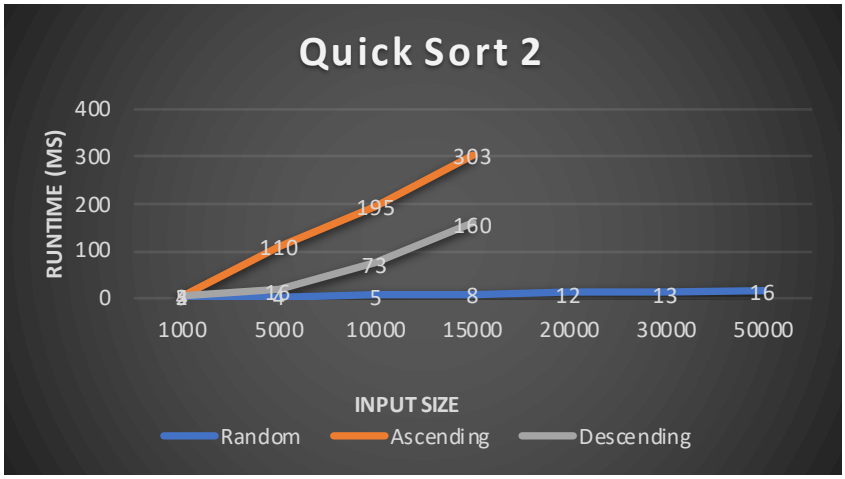
Quick Sort

	Average runtime for three runs (ms)		
Input Size	Random	Ascending	Descending
30000	15	7	13
50000	30	16	25
70000	34	16	28
90000	58	24	36



Quick Sort 2

	Average runtime for three runs (ms)		
Input Size	Random	Ascending	Descending
1000	2	4	5
5000	4	110	16
10000	5	195	73
15000	8	303	160
20000	12		
30000	13		
50000	16		



For Quick Sort 2, I was able to run the Random arrays with large N sized arrays, however after 15,000 both the Ascending and Descending arrays hit an Overflow Stack error. I believe the pivot being assigned to the right or left elements causes the algorithm to sort through already sorted material, which results in too much being pushed to the stack and causing the overflow.

Part B: I believe that when comparing the data, I can distinguish the algorithms' complexities. For example, it is obvious the insertion sort is big oh(n) for the best case, and this correlates to the times being significantly larger for the random and descending array. I don't think it would be very easy to determine from data alone which "n log n" (Quick, merge, heap) sort was running as times are similar. However, I did notice that each "n log n" did accumulate time at different rates, and responded to array type differently, so it could be possible to distinguish between them. For example, the heap sort worked well with ordered arrays, so it may help idenitfy it from the other sorts from just studying the data alone. The insertion sort and the quick sort 2 were two sorts that did become slow with

Part C: In comparing the three n log n sorts, the heap sort stood out the best if the arrays were ordered in some fashion(ascending, descending). However given a variety in the arrays, merge sort seemed the most consistent overall and the times were relatively close between each array sorted of N size. The quick sort 1 worked well, but the ascending arrays were definitely the fastest in terms of sorting. For quick sort 1, the random and descending arrays had closer times, but were not as efficient as the ascending array. For quick sort 2, the random was sorted the quickest, while the ascending and descending did not get sorted very fast due to the pivot using the last