Timothy Ha
junkwan
junkwan@uw.edu
1367917
CSE 373 HW 6

1.

	Alpha			Beta		
n = 16	Runtime	# Comparisons	# Movements	Runtime	# Comparisons	# Movements
InOrder	0	46	36	0	85	136
ReverseOrder	0	55	80	0	72	120
AlmostOrder	0	54	41	0	85	137
Random	0	54	72	0	81	131

	Delta			Epsilon		
n = 16	Runtime	# Comparisons	# Movements	Runtime	# Comparisons	# Movements
InOrder	0	150	15	0	120	0
ReverseOrder	0	158	39	0	120	24
AlmostOrder	0	139	18	0	120	3
Random	0	108	66	0	120	45

	Gamma			Zeta		
n = 16	Runtime	# Comparisons	# Movements	Runtime	# Comparisons	# Movements
InOrder	0	15	30	0	32	128
ReverseOrder	0	120	150	0	32	128
AlmostOrder	0	24	39	0	38	128
Random	0	83	101	0	46	128

2.

Alpha	Runtime (ms)			
	n = 128000	n = 256000	n = 512000	n = 1024000
InOrder	4	8	15	27
ReverseOrder	7	13	30	50
AlmostOrder	6	11	22	48
Random	17	38	77	148

Beta	Runtime (ms)			
	n = 128000	n = 256000	n = 512000	n = 1024000
InOrder	12	26	54	112
ReverseOrder	14	28	58	118
AlmostOrder	12	30	55	123
Random	19	41	91	200

Gamma	Runtime (ms)			
	n = 6400	n = 12800	n = 25600	n = 51200
InOrder	0	0	0	0
ReverseOrder	30	112	440	1775
AlmostOrder	2	8	27	112
Random	14	58	224	878

Delta	Runtime (ms)			
	n = 1500	n = 3000	n = 6000	n = 12000
InOrder	2	7	30	111
ReverseOrder	2	7	28	111
AlmostOrder	0	0	1	1
Random	1	1	1	1

Epsilon	Runtime (ms)			
	n = 12500	n = 25000	n = 50000	n = 100000
InOrder	112	450	1740	7240
ReverseOrder	101	396	1564	6203
AlmostOrder	84	338	1329	5730
Random	87	334	1342	5792

Zeta	Runtime (ms)			
	n = 128000	n = 256000	n = 512000	n = 1024000
InOrder	15	31	50	104
ReverseOrder	12	26	48	100
AlmostOrder	14	30	62	126
Random	25	51	99	222

3.
Alpha: O(N logN)
Beta: O(N logN)
Gamma: O(N^2)
Delta: O(N^2)
Epsilon: O(N^2)
Zeta: O(N logN)

4.
Alpha is quick sort (optimized). This is true because Alpha is by far the quickest algorithm and the optimized version of quick sort allows the best case to be O(N) and almost completely eliminates the usual worst case complexity of O(N^2) that a simple quick sort has.

Beta heap sort. Beta and heap sort both share the worst case O(N logN) complexity. Beta and heap sort also share the quality of many movements. It's easy to see in #1 that Beta had many movements due to initially having to build the heap. Also, Beta performs the best with InOrder and ReverseOrder lists just like heap sort would.

Gamma is insertion sort. An insertion sort on an InOrder list will have exactly n-1 comparisons because you compare the first two, then the second and third, then the third and fourth, and so on until the last element without ever switching values. Problem 1 showed that for a n=16 array, there were 15 comparisons. Also, a reverseOrder list is the worst scenario for an insertion sort and that was shown. The table for Gamma for #2 really shows how important being INORDER is for insertion sort. AlmostOrder is very fast and reverseOrder is very slow.

Delta is quick sort (simple). Although quick sort can be pretty quick, it has a pretty bad worst case complexity of O(N^2) and there is evidence of this in #2. Also, for $N < 20$, insertion sort is actually faster than quick sort (simple). For $n = 16$, although both times were below 1 millisecond, we can see that the quick sort has done many more comparisons than insertion sort. Also looking at the #2 table, we can see that InOrder and ReverseOrder are very slow, this is because the first value of the list is picked as the pivot for quick sort (simple) which gives it the worst case scenario, as opposed to picking the median or a random value.

Epsilon is selection sort. This one is very evident with the worst case complexities from #2 and #2. We can see in #1 that there are always 120 comparisons, this is because selection sort goes through every single value looking for the lowest value each time which is why it's best, average, and worst cases are O(N^2).

Zeta is merge sort. Merge sort's worst case complexity is O(N logN), the same as Zeta. But that's not very convincing because Zeta could be heap sort or the optimized quick sort. However, Zeta and merge sort both share the quality of having the minimum amount of comparisons for InOrder and ReverseOrder lists AND having the same number of movements regardless of the kind of ordered list. This is true because the merge sort recursively splits the list and moves values regardless of the order.