Question 1:

Full statistics for input size 16: Prepare a table (for each sort) showing the runtime, number of comparisons, and number of movements for an array of size 16 and with separate results for each choice of how the elements are initially ordered. The full answer for this question is just one table per sorting implementation, no explanation required.

Sort Type	Greek	Data Type	Comparisons	Movements	Runtime
Quick sort	Alpha	In order	150	15	0
		Reverse order	158	39	0
		Almost order	141	18	0
		Random order	108	78	0
Selection sort	Beta	In order	120	0	0
		Reverse order	120	24	0
		Almost order	120	3	0
		Random order	120	45	0
Insertion sort	Gamma	In order	15	30	0
		Reverse order	120	150	0
		Almost order	28	43	0
		Random order	68	85	0
Shell sort	Delta	In order	30	60	0
		Reverse order	45	82	0
		Almost order	42	73	0
		Random order	61	97	0
Merge sort	Epsilon	In order	32	128	0
		Reverse order	32	128	0
		Almost order	37	128	0
		Random order	45	128	0
Heap sort	Zeta	In order	85	136	0
		Reverse order	72	120	0
		Almost order	84	135	0
		Random order	81	133	0

Question 2:

Runtimes for at least four nontrivial input sizes: Prepare another table (for each sort) showing (just) the runtime of the sort for at least four different non-trivial array sizes. A non-trivial array size is one where the runtime is more than just a few milliseconds. When you can, increase the input size until the runtime takes at least one second. If input sizes are chosen well, they will allow you to see enough variation in the statistics to give you strong evidence to support your answers to questions (3) and (4) below. You do not need to use the same input sizes for every algorithm, since this might not produce the most useful data. The full answer for this question is just one table per sorting implementation, no explanation required.

Sort Type	**Greek**	**Input Size**	**Data Type**	**Run time**
Quick Sort	Alpha	2222	In order	2
			Reverse order	2
			Almost order	0
			Random	0
		3333	In order	6
			Reverse order	6
			Almost order	0
			Random	1
		5000	In order	12
			Reverse order	19
			Almost order	0
			Random	0
		100000	In order	52
			Reverse order	60
			Almost order	1
			Random	1
Selection Sort	Beta	2222	In order	3
			Reverse order	2
			Almost order	5
			Random	7
		6666	In order	26
			Reverse order	21
			Almost order	18
			Random	19
		13332	In order	100
			Reverse order	92
			Almost order	75
			Random	189
		66666	In order	2809
			Reverse order	2789
			Almost order	1798
			Random	1749

Insertion Sort	Gamma	2222	In order	0
			Reverse order	3
			Almost order	0
			Random	4
		6666	In order	0
			Reverse order	20
			Almost order	2
			Random	10
		13332	In order	0
			Reverse order	80
			Almost order	6
			Random	44
		133320	In order	0
			Reverse order	8054
			Almost order	624
			Random	4694
Shell Sort	Delta	2222	In order	1
			Reverse order	1
			Almost order	1
			Random	0
		6666	In order	1
			Reverse order	1
			Almost order	1
			Random	1
		133320	In order	2
			Reverse order	4
			Almost order	13
			Random	21
		1048576	In order	26
			Reverse order	33
			Almost order	137
			Random	168
Merge Sort	Epsilon	50000	In order	3
			Reverse order	3
			Almost order	5
			Random	6
		100000	In order	6
			Reverse order	5
			Almost order	9
			Random	13
		500000	In order	37
			Reverse order	40
			Almost order	50
			Random	77

		1048576	In order	79
			Reverse order	101
			Almost order	100
			Random	181
Heap Sort	Zeta	50000	In order	3
			Reverse order	3
			Almost order	3
			Random	5
		100000	In order	8
			Reverse order	88
			Almost order	8
			Random	10
		500000	In order	45
			Reverse order	41
			Almost order	43
			Random	61
		1048576	In order	85
			Reverse order	89
			Almost order	97
			Random	187

Question 3:

Worst-case asymptotic run-time: Your estimation of the worst case asymptotic ("big-O") runtime for each sorting implementation. You must gather appropriate data that clearly shows the relevant patterns of the algorithm's runtime. This likely means measuring different algorithms at different input sizes, since some algorithms are much faster than others. No explanation is required here. However, if the data you collected in question (2) is in conflict or in no way suggesting the runtimes you give for question (3), then you should consider collecting more runtime data. We do not expect curves that show a perfect n log n or anything like that though, just reasonable data.

Sort type	**Greek**	**Worst-case runtime**
Quick Sort	Alpha	O(n^2)
Selection Sort	Beta	O(n^2)
Insertion Sort	Gamma	O(n^2)
Shell Sort	Delta	O(n)
Merge Sort	Epsilon	O(nlog(n))
Heap Sort	Zeta	O(nlog(n))

Question 4:

Identifying the algorithms: State which sorting implementation uses which algorithm, along with your reasoning and supporting evidence. You should base your reasoning on the following:

- growth rate of the algorithm's runtime as input size grows
- speed or number of comparisons and data movements of the algorithm compared to the other algorithms
- changes in behavior of the algorithm, if any, for different input orderings

If errors caused by the algorithm caused you to deduce that algorithm, explain why you suspect that algorithm would cause the error.

Alpha → Quick Sort

1. The worst case runtime is O(n^2), which can be distinguished by comparing growth rate of reverse order as input size grows.
2. The average case runtime is O(nlog(n)), which can be distinguished by comparing growth rate of random order as input size grows.
3. The best case runtime is O(nlog(n)), which can be distinguished by comparing growth rate of in-order order as input size grows.
4. Error: Stack over flow happens when it runs in order or reverse order at certain size, as when sorting in order or reverse order, quick sort would keep a lot of recursion calls in the stack. So not huge input size could generate stack overflow error, which fits my hypothesis that Alpha is quick sort.

Beta → Selection Sort

1. The worst-case runtime is O(n^2), which can be distinguished by comparing growth rate of reverse order as input size grows.
2. The average case runtime is O(n^2), which can be distinguished by comparing growth rate of random order as input size grows.
3. The best case runtime is O(n^2), which can be distinguished by comparing growth rate of in-order order as input size grows.
4. For in order, random order, the runtime of these are close to the runtime of reverse order. That is the best-case runtime and average case runtime are all O(n^2). So it is selection sort.

Gamma → Insertion Sort

1. The worst case runtime is O(n^2), which can be distinguished by comparing growth rate of reverse order as input size grows.
2. The average case runtime is O(n^2), which can be distinguished by comparing growth rate of random order as input size grows.
3. And the best case runtime is O(n), which can be distinguished by comparing growth rate of in-order order as input size grows., and the runtime of almost is very small comparing to the runtime of random case.
4. Runtime of random order is almost half of runtime of reverse order. It is because the for each step in the outer loop of insertion sort, the chosen element will be swapped to the middle of sorted part on average, however, in reverse order, the element will be swapped to the left end.

Delta → Shell Sort

1. The worst-case runtime is O(n), which can be distinguished by comparing growth rate of reverse order as input size grows.
2. The best-case runtime is O(nlog(n)), which can be distinguished by comparing growth rate of in-order order as input size grows.
3. It has O(n) worst-case and O(nlog(n)) best-case, so it is shell sort.
4. Additionally, there is no regular pattern for average case, as demonstrated by random order, which makes sense, because Shell sort is very unstable, dependent on gap sequence.

Epsilon → Merge Sort

1. The worst-case runtime is O(nlog(n)), which can be distinguished by comparing growth rate of reverse order as input size grows.
2. The average case runtime is O(nlog(n)), which can be distinguished by comparing growth rate of random order as input size grows.
3. The best case runtime is O(nlog(n)), which can be distinguished by comparing growth rate of in-order order as input size grows.
4. For in order of merge sort, the comparisons would be exactly N/2log(N) (Here N is 2^n). That is for input size N=16, the comparisons would be exactly 32. (For N=32, it is 80), so delta is merge sort.

Zeta → Heap Sort

1. The worst-case runtime is O(nlog(n)), which can be distinguished by comparing growth rate of reverse order as input size grows.
2. The average case runtime is O(nlog(n)), which can be distinguished by comparing growth rate of random order as input size grows.
3. The best case runtime is O(nlog(n)), which can be distinguished by comparing growth rate of in-order order as input size grows.
4. The comparisons and runtime of in order are pretty close to these of reverse order. This is because in heap sort, when building the heap using the binary tree, reverse order and in order would take almost the same steps.
5. So Zeta could be heap sort or merge sort, but for in order of merge sort, the comparisons would be exactly N/2log(N) (Here N is 2^n). Zeta does not satisfy this, so it is heap sort.