1.c.

sorted array to be sent again(BEST CASE)

the running time for a random array to be sorted : **7799 nanoseconds**

* Since array is already sorted will be easy to sort.

1.d.

reverse sorted array to be sent again(WORST CASE)

the running time for a random array to be sorted : **18788800** **nanoseconds**

* Since array is already sorted will be hard to sort.

1.e.

For this question, I only changed the array size.

Array size = 100000

1.e.c.

sorted array to be sent again(BEST CASE)

the running time for a random array to be sorted : **813200** **nanoseconds**

* Since array is already sorted will be easy to sort.

1.e.d.

reverse sorted array to be sent again(WORST CASE)

the running time for a random array to be sorted : **1896541400** **nanoseconds**

* Since array is already sorted will be hard to sort.

2.b.

[98, 83, 15, 61, 82, 9, 12, 16, 71, 14, 56, 60, 64, 48, 80, 14] => original array

[9, 12, 14, 14, 15, 16, 48, 56, 60, 61, 64, 71, 80, 82, 83, 98] => sorted array

3.a.

insertion: 67099293400 nanoseconds

merge: 121490700 nanoseconds

3.b.

Merge sort quite faster than insertion sort. To show this, I used their algorithms. If the n value is bigger than 43 merge sort will be faster than insertion sort. That’s why when n is large enough merge sort will be faster. All values are listed in the answer until 43.