Random order:

By mentioning to the average time of sorting the random numbers of an array list we see, insert sort should consider as the best way to sort the batch of the random numbers. There are some reasons to confirm that why insert sorting considers as the quickest way among the other simple sorts for an array which its size either is at least 10000 or more.

1. By considering the algorithm of Insert sort we see in the each loop we get the next element of last sorted element. Then in the second loop, we get the element before that as the last sorted element to compare if the unsorted element is either less than the last sorted or might be less than the rest of sorted elements in the list. If so, we should swap the unsorted ones with last sorted element in each loop. Therefore, this execution prevent to compare each element with the all element in the compare with compilation in the Bubble sort. So, in this case, we save the timing of creating unnecessary swap in each loop.
2. In comparing with selection sort we get the first element as the current min and then compare we the whole of the list in the case that we find something less than the element we got. If we find something then we set that as the current min and we are continuing if there is nothing less then we swap the current min with the first unsorted element. Here, we see more comparison in the execution of selection sort in the compare with insert sort which proves the insertion sort as a more efficient sorting than selection sort and in this case saving more time.

In order sort (increasing):

By mentioning to the average time of sorting the in order numbers of an array list we see, Bubble sort should consider as the best way to sort the batch of the in order numbers. The following reasons confirming our statement.

1. The algorithm of Bubble sort complies in order of two loops. In the first loop, we get the first element and then in the second loop we get the element after the first one and in next we are comparing if the first element is less than the second one or not. If so, then we should swap the second one with the first element. In the case, if that element is not less than the next one then we back to the first loop and get the next element and goes for comparing in the second loop. In this case, our array is sorted in the order, so execution of Bubble sort just takes the time that read all element of the list, because there is nothing to swap. On the other hand, if we consider the execution of insert sort we see it needs to compare each element that it gets in the first loop with the last sorted element. However, we know that our array is sorted in order, so there is no swapping in this execution which is just wasting time.
2. Now in the compare with an algorithm of selection sort we see selection sort is gone take the first element as the min and then it goes to compared with the whole of the list. In the case which it might find something which is less than the current min. But we know the list is in order so the current min is minimum one and there is not any swapping and we are just gonna check the list which in the order numbers it is not necessary at all.

Reverse order (decreasing):

By mentioning to the average time of sorting the reverse numbers in order of an array list we see, selection sort considers as the best way to sort the batch of the reverse in order numbers. The following reasons confirm our claim.

1. In the compare with insertion sort, we see select sort it is gonna take the first element as the current min. By each loop, the current min goes to next one because our array is sorted in a decreasing order until it reaches the end of the list and then it swaps the last element with first unsorted position and in the total, we just have one swap in the whole of an array in each loop. However, by mentioning to the execution of insertion sort we see, we take the next element of last sorted element and then we check if that element is less than sorted one or not. If so we are gonna swap that one and next after that we need to compare with rest of the sorted ones too, in the case we find something less. So, in this case, we know that our array is sorted in the reverse order so in each loop at the first we just have one swap but as much we go forward the numbers of swapping increase until we reach the end of the list. Consequently, there are lots of more comparison and swapping than selection sort and in this case, we consider as the fastest way to sort the reverse in order numbers.
2. By considering the Bubble sort we see we have one swap in each loop because the array sorted in reverse order so in this case, the next element is always less or sometimes equal with element that we get at the first until we reach end of the list for that element and then so on for the rest of the numbers in the array. It means we have lots of more swapping than the selection sort until we reach the end of the list. That means spending more time and swapping in the compare with selection sort. Consequently, we consider the selection sort as the most efficient way to sort the reverse in order numbers among the other 2 simple sorts.