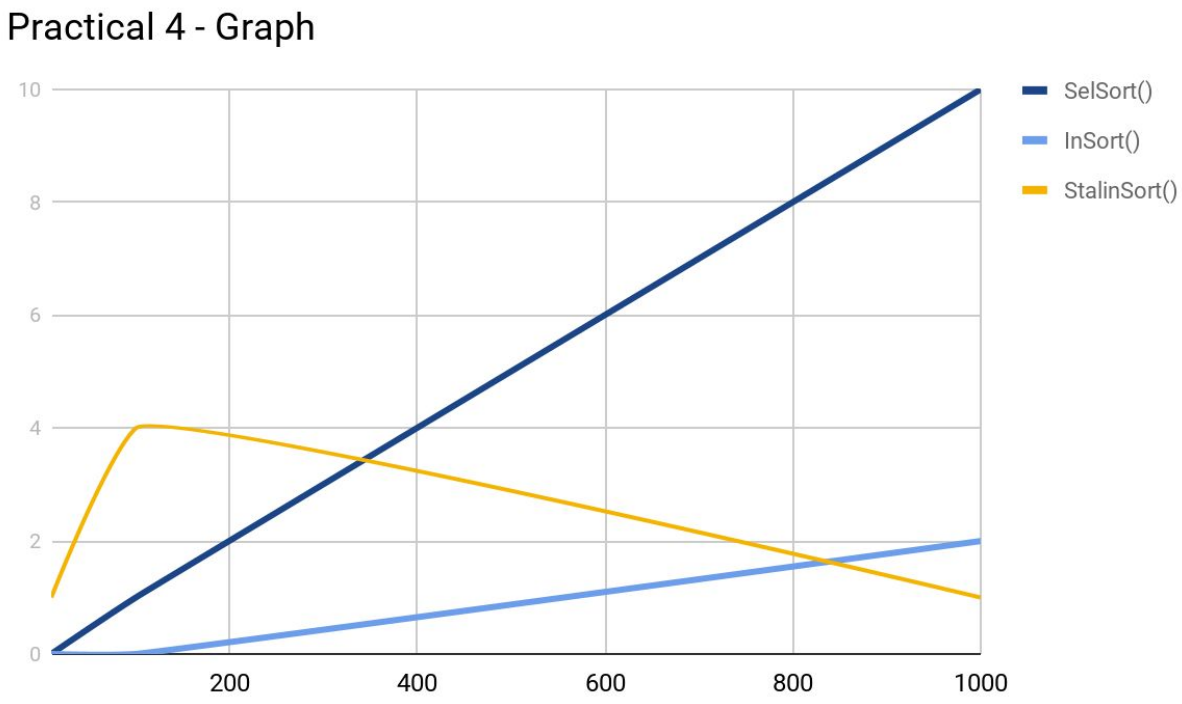
Practical 4

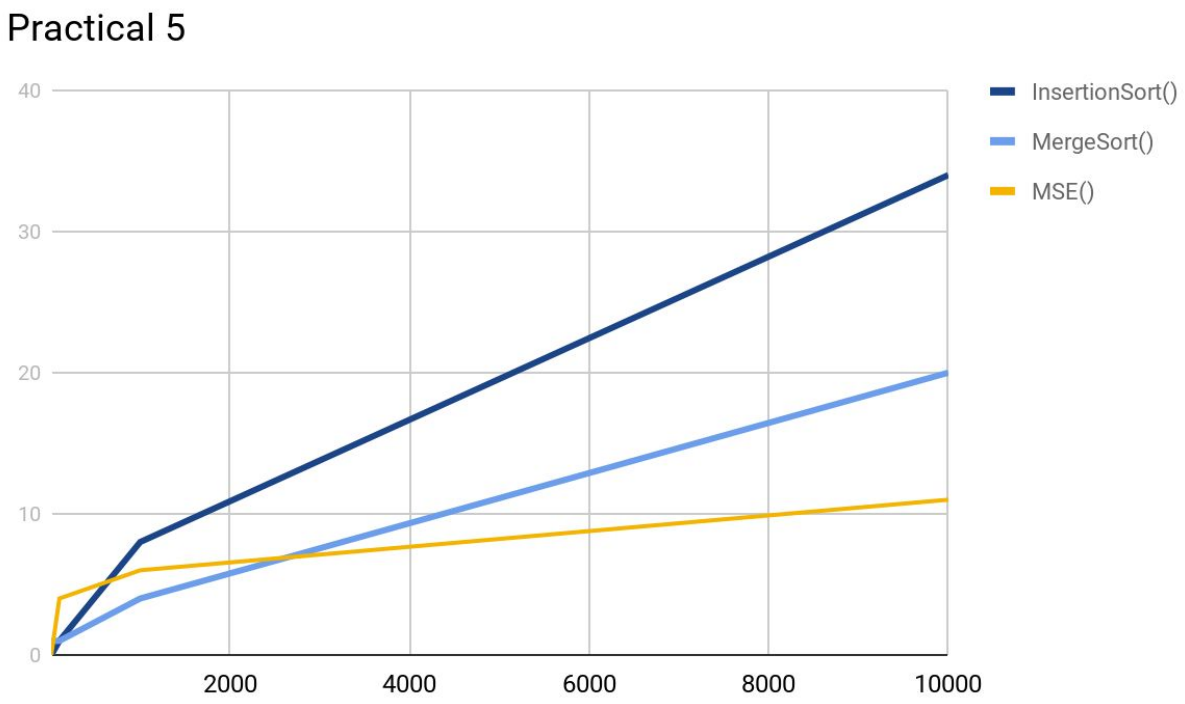
| Size | SelSort() | InSort() | StalinSort() |
|---|---|---|---|
| 10 | 0 | 0 | 1 |
| 100 | 1 | 0 | 4 |
| 1000 | 8 | 7 | 1 |




Reasoning:
SelSort() → its complexity is O(n^2). This makes sense for the milliseconds needed to sort arrays of different lengths. As you can see on the graph, the time taken increases at an exponential rate.
InSort() → its complexity is O(n^2). Although this graph does not depict this, this complexity makes sense as well. In the worst case scenario and every value in the array was in the worst location, it would take this much time because the second loop would have to iterate until hitting the first spot.
StalinSort() → its complexity is O(n). However, this is not depicted in the graph. This is because StalinSort is a very unreliable algorithm, since it literally removes values that do not comply with the order. Worst scenario, it would have to remove every element which would be dependent on the number of elements in the array.

Practical 5

| Size | InsertionSort() | MergeSort() | MSE() |
|---|---|---|---|
| 10 | 0 | 1 | 0 |
| 100 | 1 | 1 | 4 |
| 1000 | 8 | 4 | 6 |
| 10000 | 34 | 20 | 11 |




Reasoning:
InsertionSort() → As mentioned earlier, InsertionSort has a time complexity of O(n^2). As the input size gets larger, it takes even longer to sort through the array.
MergeSort() → The time complexity of mergeSort is O(nlogn). This is because every time the method is called, it halves the array size. This allows for the data to be sorted at a quicker pace, but it is also dependent on the size of the array.
MergeSortEnhanced() → This has the best time complexity, because it optimizes both of these sorting algorithms. Since insertionSort works well with smaller input sizes, by creating a “cutoff” point it uses insertionSort only for smaller arrays. Since mergeSort works well with larger arrays, it allows MSE to optimize on this as well.

Practical 6

| Size | MergeSort() | QuickSort() | QSE() |
|---|---|---|---|
| 10 | 1 | 0 | 0 |
| 100 | 1 | 0 | 6 |
| 1000 | 9 | 1 | 33 |
| 10000 | 21 | 2 | 664 |