The Median of 3 QuickSort was generally the best algorithm for the random data with the Random Pivot QuickSort being a close second, and the Simple QuickSort and MergeSort being the slowest algorithms. As can be seen by the graph in Spreadsheet 1, although the Random Pivot QuickSort and Median of 3 QuickSort times were very similar the Median of 3 QuickSort continued to be slightly faster than the Random Pivot QuickSort as the array size increased. As can be seen in Spreadsheet 2, as the MIN\_SIZE increased, the Median of 3 QuickSort continued to be slightly faster than the Random Pivot QuickSort for the random data configuration. If we take a look into the algorithms, these results do make sense. In general, as the pivot value for any of the QuickSort algorithms gets closer to the true median of all of the values in the array, the closer the algorithm is to achieving its best-case runtime. In this case, when the data is initially random, we are most likely to obtain a pivot at the median of the values when using a Median of 3 QuickSort or Random QuickSort. And out of these two algorithms, Median of 3 QuickSort may be slightly more efficient because it guarantees that there is at least one item in each half after the partition for each recursive call. According to the data of spreadsheet 1, Simple QuickSort does have the fastest times for the first three array sizes in this case. However, this algorithm was not considered when comparing runtimes because it doesn’t have the runtimes for all of the array sizes. Additionally, the results of the Simple QuickSort can be different based on the random data generated. In this case, the algorithm worked well. However, the worst case can easily be reached if the second to last index in each call of the method happens to be the largest or smallest value of the data. The MergeSort had the worst runtime because there are additional comparison steps taken after the partitioning of the data where the comparisons are not done in place. Furthermore, it takes up more time to do the comparisons steps as well as transfer the data back and forth between a temporary array and the original one.

The MergeSort algorithm was generally the fastest for sorted data. As can be seen by the graph in Spreadsheet 1, the MergeSort continued to be slightly faster than the Median of 3 QuickSort and faster than the Random Pivot QuickSort and Simple QuickSort. Although at the largest array size, the Median of 3 QuickSort time was very slightly faster than the MergeSort. As can be seen in Spreadsheet 2, as the MIN\_SIZE increased, the MergeSort continued to be the fastest algorithm. The Random Pivot QuickSort and Median of 3 QuickSort was generally the best for reverse sorted data. As can be seen by Spreadsheet 1 and 2 as the array size increased and the MIN\_SIZE increased, the Random Pivot QuickSort and Median of 3 QuickSort continued to have very similar times and were the fastest times. In general, it makes sense that the Median of 3 QuickSort performed well for both reverse and sorted data. It is very likely that the middle value in an array will be the true median of all of the values because the data is in sorted or reverse sorted configuration. Additionally, Simple QuickSort has the worst runtime for sorted and reverse sorted data because the pivot chosen will always be the largest value (for sorted data) or smallest value (for revere sort). Consequently, the problem size will decrease by N-1 rather than N/2, thus reaching the worst case where runtime is O(N^2).

For all 3 configurations, the MergeSort algorithm was the fastest when the MIN\_SIZE was 3. For the Random configuration, the fastest times for the Median of 3 and Random Pivot algorithms, was when the MIN\_SIZE was 15.  For the Sorted configuration, the fastest times for the Median of 3 and Random Pivot algorithms was when the MIN\_SIZE was 50. For the Reverse Sorted configuration, the fastest times for the Median of 3 and Random Pivot algorithms, was when the MIN\_SIZE was 500. In general, the optimal MIN\_SIZE increases for the Median of 3 and Random Pivot algorithms as we change the data configuration from Random to Sorted to Reverse Sorted in that order.

I was surprised to find out that the MergeSort was the best algorithm when the data was initially sorted. I think this was the case because in the MergeSort algorithm, there is an if statement in which-if the last item in the left half is less than the first item in the right half, it skips over the additional comparison steps done with the creation of the temporary array. Instead, it deems the array as already sorted, and returns the array as it is. In this case, it makes sense that the runtime would be the fastest because if the array is already sorted, it will always go through that is statement and will never have to go through the additional comparison steps. I was also surprised that the reverse sorted data configuration achieves its optimal MIN\_SIZE at a higher MIN\_SIZE then the sorted data configuration. This surprised me because I know that the best case for InsertionSort of an array is when the data is sorted and the worst case is when it is reverse sorted. Furthermore, I was expecting the optimal MIN\_SIZE for the sorted data to be the highest because that is when it would go through the most InsertionSorts and the optimal MIN\_SIZE for the reverse sorted data to be the lowest. However, I think the main reason for reverse sorted having the highest optimal MIN\_SIZE is that the Median of 3 QuickSort and Random QuickSort algorithms are more faster for sorted data than the InsertionSort for sorted data versus how much faster the Median of 3 and Random QuickSort algorithms are for reverse sorted data then the InsertionSort for reverse sorted data.

To conclude, if I am solely basing my decision on the runtime analysis of my example, I would choose Median of 3 QuickSort. I would choose this algorithm because it has the best runtime for Random and Reverse Sorted data configurations, and the second-best runtime for Sorted data configuration. However, I would also consider the Random QuickSort as a close second because the runtimes for Random QuickSort for almost all of the data configurations were pretty good and closely followed that of the Median of 3 QuickSort. Additionally, if we take a look at the specific algorithm of Random QuickSort, the worst case is impossible when using this algorithm. Furthermore, out of all four algorithms, this is the only algorithm that doesn’t have a worst case with runtime equal to or worse than O(N^2).