**Observations**

‘

Random File

2000 → 1.69, 1.962, 1.687 average = 1.7796

4000 → 6.279, 5.79, 7.271 average = 6.4466

8000 → 29.464, 22.545,21.663 average = 24.5573

16000 → 89.373, 87.334,91.079 average = 89.262

Reversed File

2000 → 1.575, 1.556, 1.555 average = 1.562

4000 → 5.721, 6.323 ,5.811 average = 5.951

8000 → 21.867, 22.04,20.996 average= 21.634

16000 → 86.006, 86.732, 84.383 average = 85.707

Sorted File

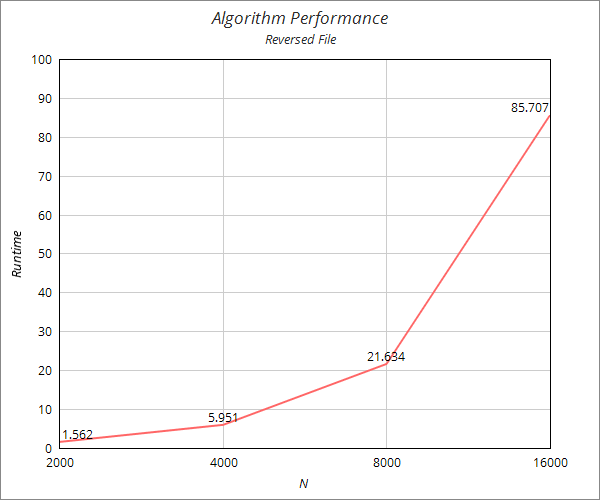
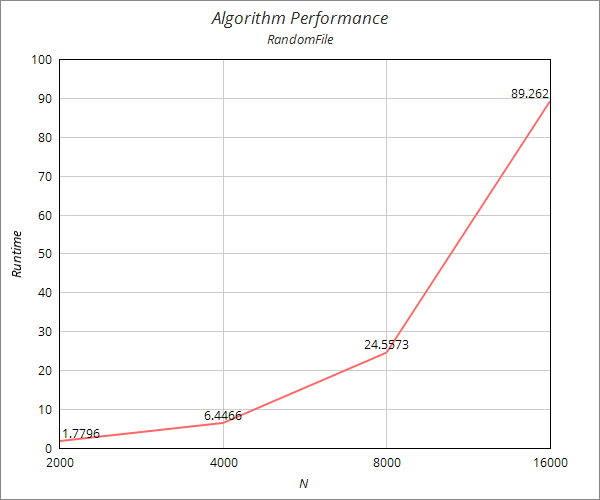
2000 → 1.708, 1.795, 1.792 average = 1.765

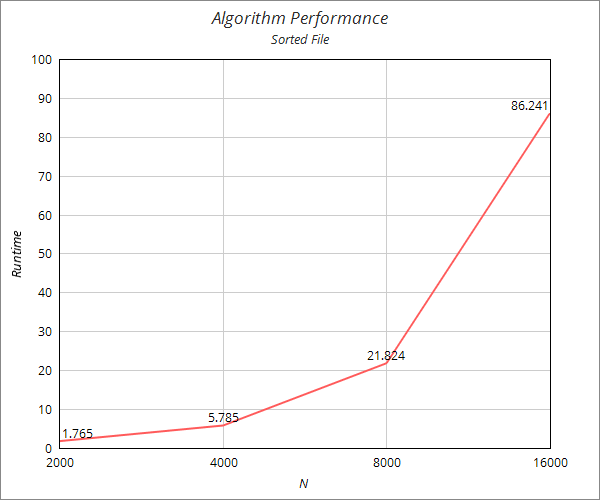
4000 → 5.673, 5.958,5.725 average = 5.785

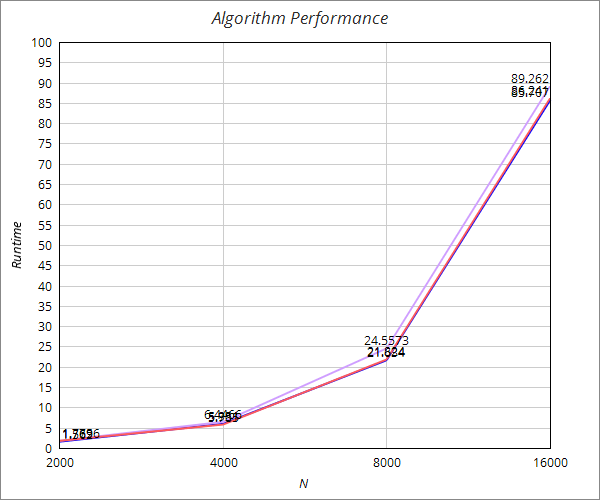
8000→ 21.185, 20.917, 23.37 average = 21.824

16000→ 86.721, 87.665, 84.339 average = 86.241

**Graphs**







**3)** Analyze the rate of growth of the selection sort algorithm using the doubling rule.

Doubling hypotheses proposes that, T(n) = an^b log n. Obviously, log n could be neglected for small input sizes like the ones we are working on. When the code is analyzed, one can observe that selection sort algorithm has a double nested loop. Which means in our case b =2.

Then,

T(n) = an^2. So, T(2n) = a(2n)^2 = 4T(n).

Supporting this equation, when I divided the average results of various two consequent steps, I observed that when the input doubles, the runtime gets raised to the power of 2 of the last time.

In conclusion, we can argue that Selection.sort() function demonstrates a quadratic growth model.

**4 )** Order of growth.

The key point in analyzing the running time of a program is this: for a great many programs, the running time satisfies the relationship T(n)∼cf(n) where c is a constant and f(n) a function known as the order of growth of the running time. In the case of Selection.sort(), f(n) is n^2 because of the double nested loop.

In the code, array access occurs in the nested loop. If we assume that an array access costs the constant of ∼n then the order of growth of this function would be represented as ∼n^2 in tilde notation.

**5)** Selection sort is a relatively easy algorithm to analyze since it demonstrates the qualities of a brute-force algorithm. None of the loops depend on the content of the date. To find the lowest value in n elements, algorithm scans all the n-1 elements and swaps the lowest one with the first one. For the remaining n-1 unsorted elements, process repeats in the same way as in the first step.

In other words, as can be observed in the code, exch() and less() functions are called in each iteration of the loop regardless of the content of the data.

Furthermore, the last graphic above demonstrates the significant parallelism between the results of 3 different files which supports the idea that the order of the input has no effect on the selection sort process.