# Empirical study of the insertion sort.

Aim : Plan and run an empirical study of insertion sort in order to deduce the complexity for the best worst and average cases.

# Experiment Design

For this experiment I will decide on some pre-set values of n, the size of the array of numbers, and I will produce code to efficiently analyse the complexities for best, worst and average cases.

For the particular values of n, I have decided to begin with a base number of 5 as it is low enough to establish a minimum but also high enough to allow for multiple different random cases. I then multiplied this by 4 four times to give the n values 5, 20, 80, 320 and 1280. I decided this as when it comes to making the graphs it will create a much wider scale which will better represent the worst, best and average complexities.

When it comes to generating the random values of n I will generate 5 random cases in order to give a fair average of each of the 5 values of n and to generate these ill simply enter all the numbers into an array and use a random shuffle method which will completely change the order of the array in an unpredictable manner in order to produce fair results for the average complexity. When it comes to generating the best- and worst-case complexities however I’ll simply fill an array from 0 to n in increments of 1 for the best case and then for the worst case I’ll do the same but from n to 0.

The code produced will include a variable “compCount” and a print statement in the sort method where count will increment when a comparison between two numbers in the given array is carried out in the sort so that the time complexity to be calculated before its printed off to allow me to record it for each value of n. The values will be recorded firstly into a table before they are converted into various graphs in order to compare complexities between best, worst and average as well as to individually represent each. The shapes of the curves of the graphs will allow me to distinguish their asymptotic complexity.

# Producing Code

Firstly the code for an insertion sort was created which was found from reference [1] which was tested with a few test arrays to make sure it was producing the expected outputs and after being confident it worked as expected a variable “compCount” was added to count the number of comparisons in the insertion sort. The method was then altered to print the amount of comparisons before the sorted array was printed. The code for the insertion sort can be seen in figure 1.

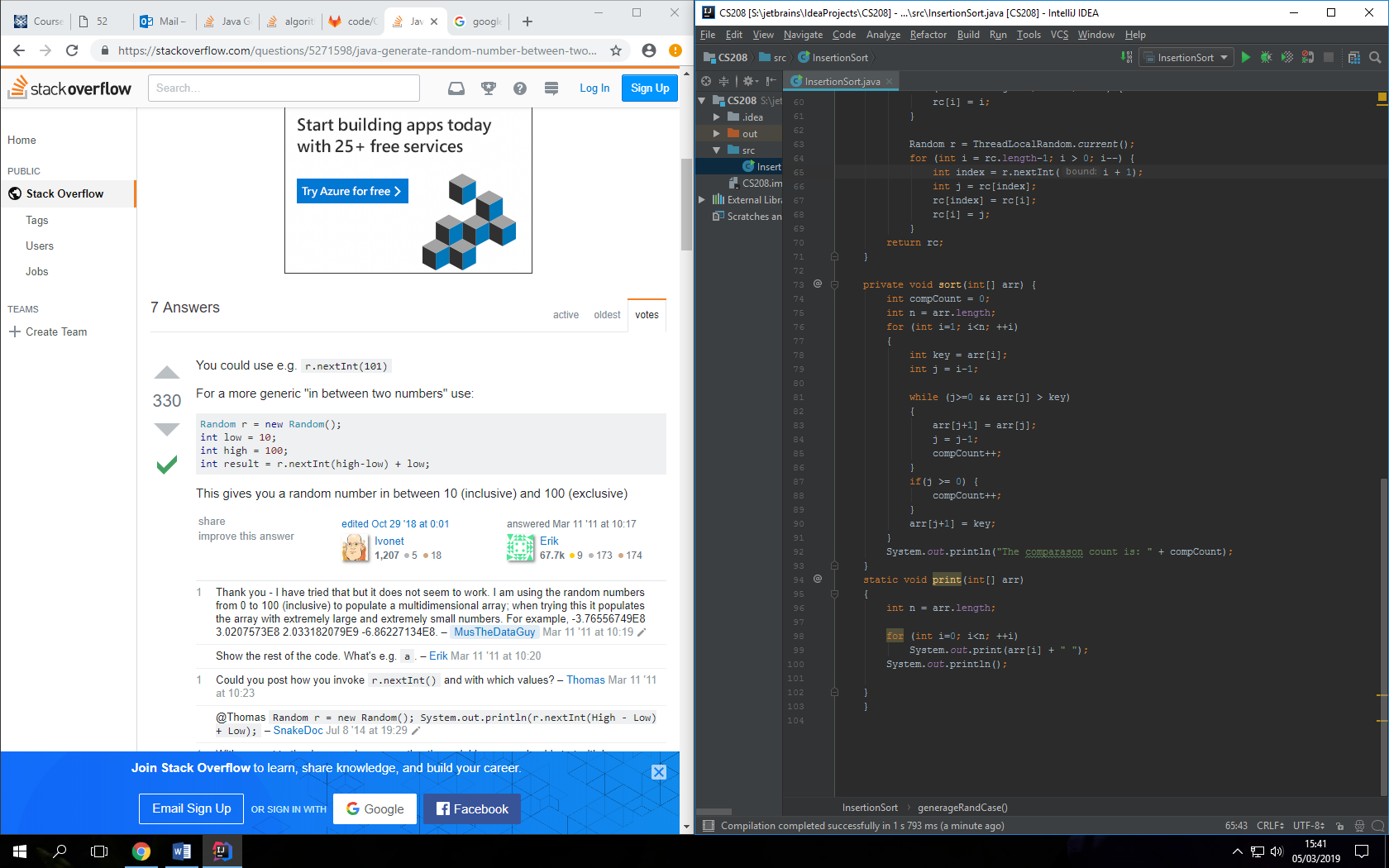


Figure 1: Insertion sort code followed by the print method.

After being able to visibly see that the code was producing the expected results the main method was filled along with 3 new methods to produce best, worst and random cases. The 3 new methods can be seen in figure 2. The best and worst case simply worked by created an array of size “length” and filling them with the numbers from 1 to length for best case and from length to 1 for worst case. As for the random cases, a new array was made and filled with numbers before random pairs of random numbers get shuffled around producing a fully randomised array.

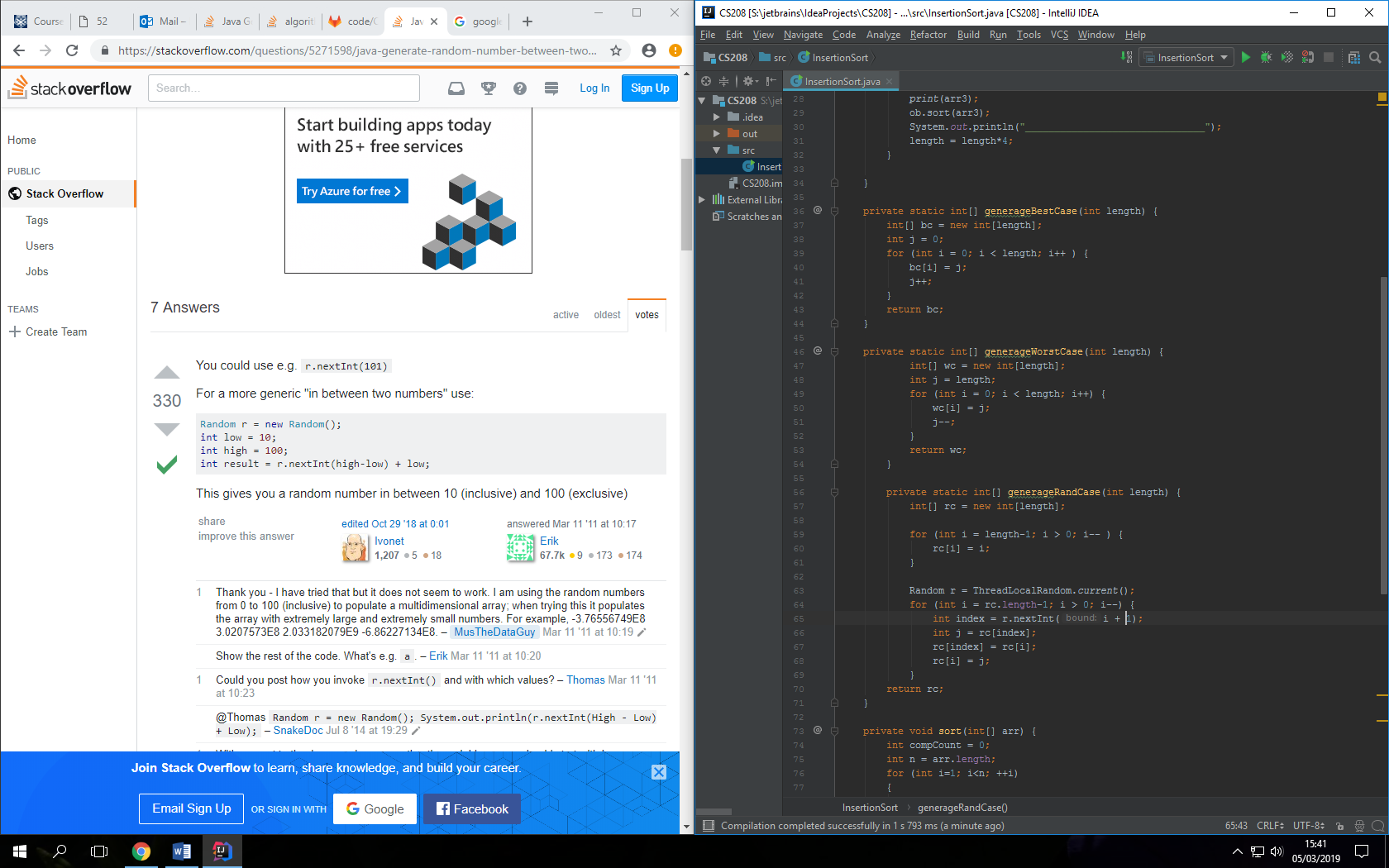


Figure 2: Generate methods for best, worst and random cases.

The main method contained the code shown in figure 3 which produced both the best and worst cases for the array of size “length” as well as 5 randomised cases for each “length”. It produces 5 sets of these results and each time length was multiplied by 4.

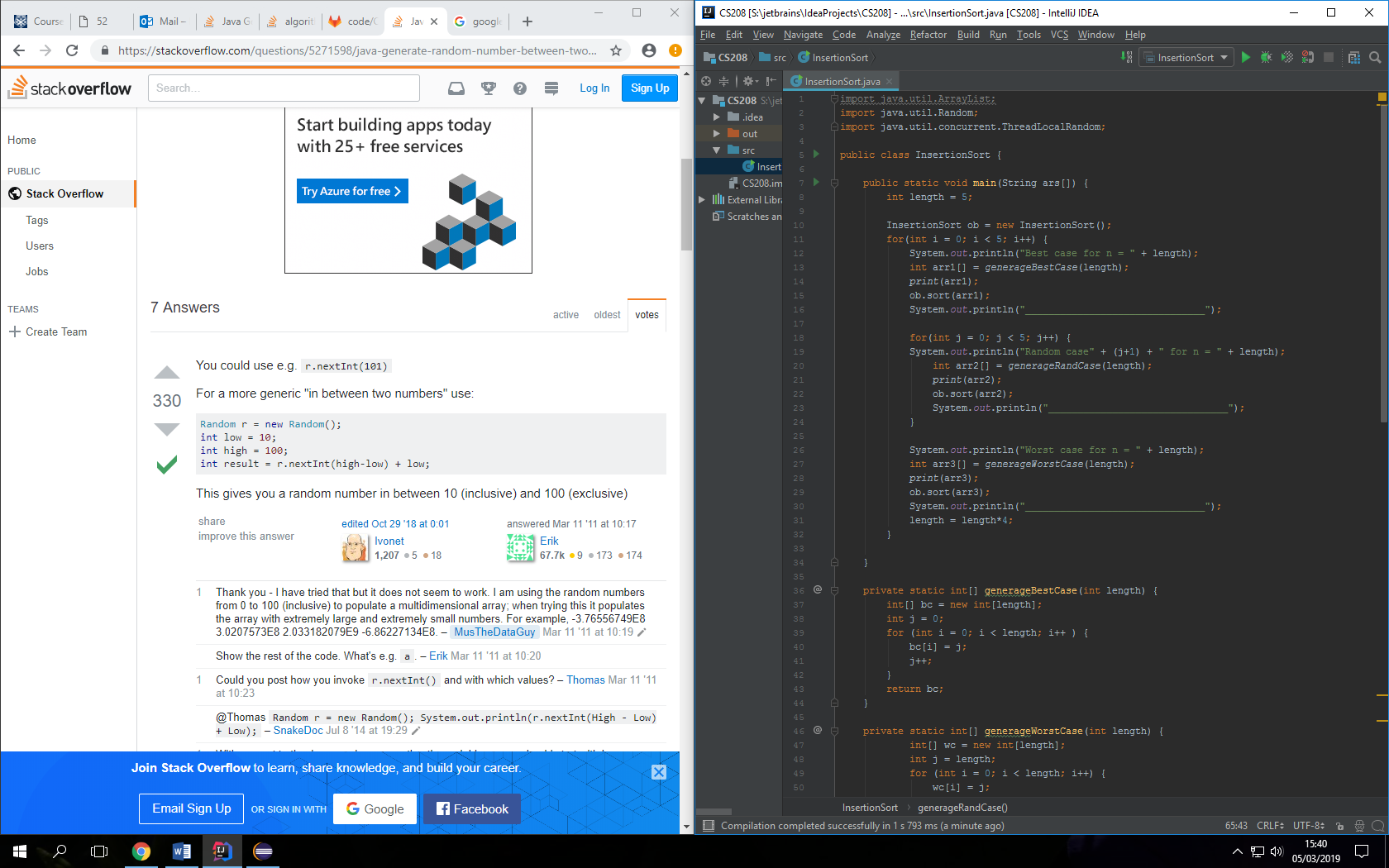


Figure 3: Main method for generating the testing.

The code shown above produced the outputs as follows for the cases when n=5, n=20, n=80, n=320 and n=1280 shown in figures 4,5,6,7 and 8 respectively.

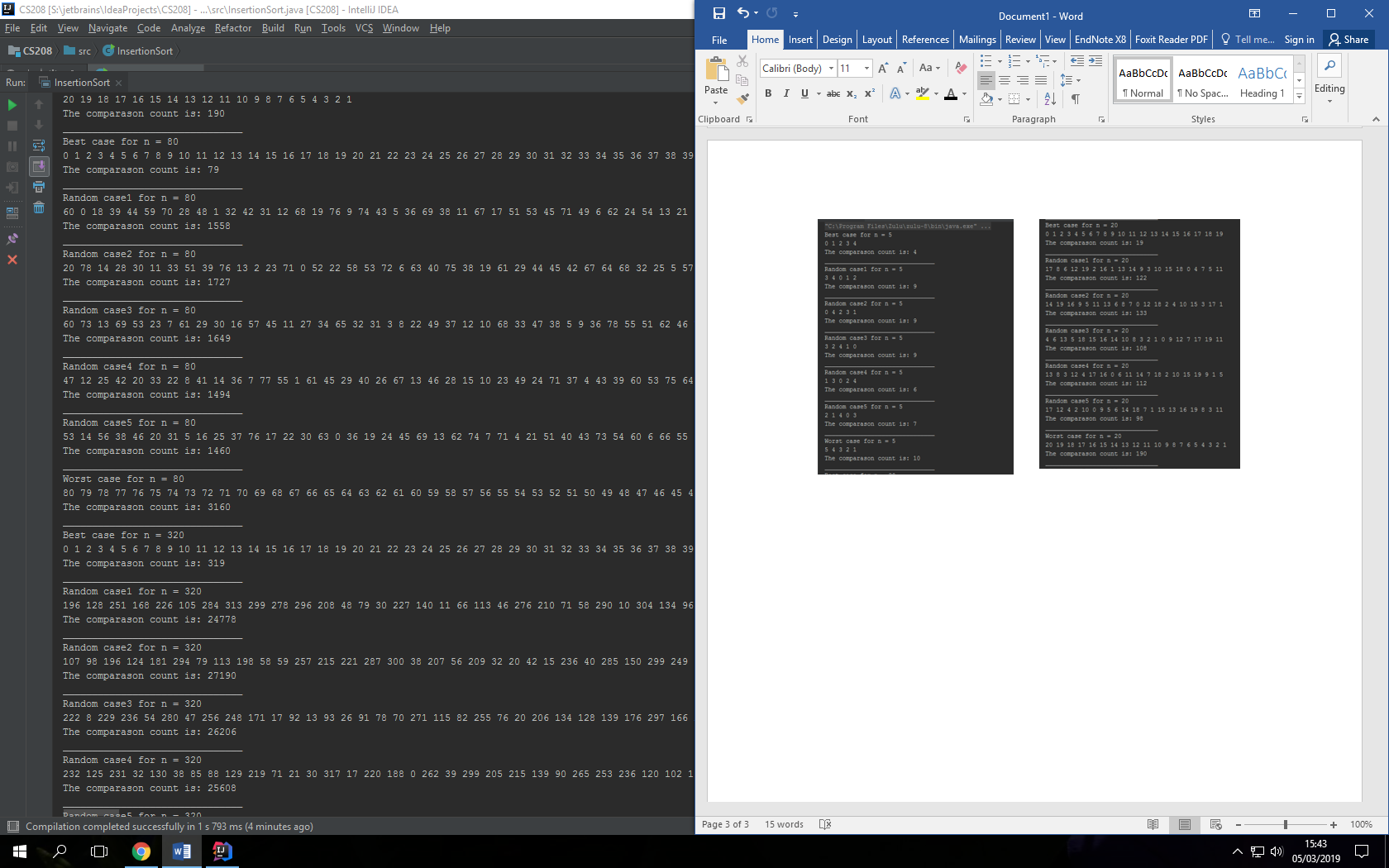
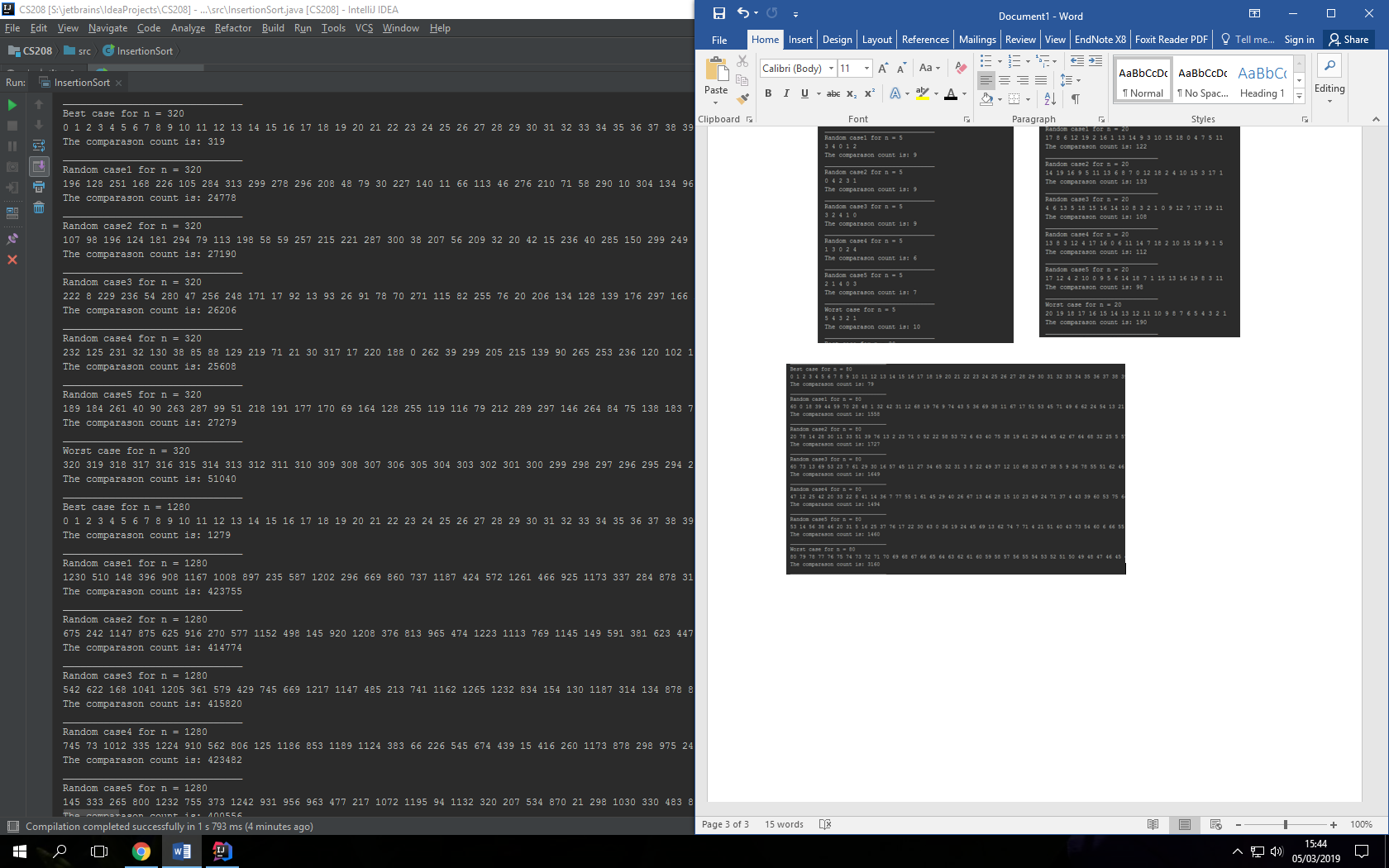
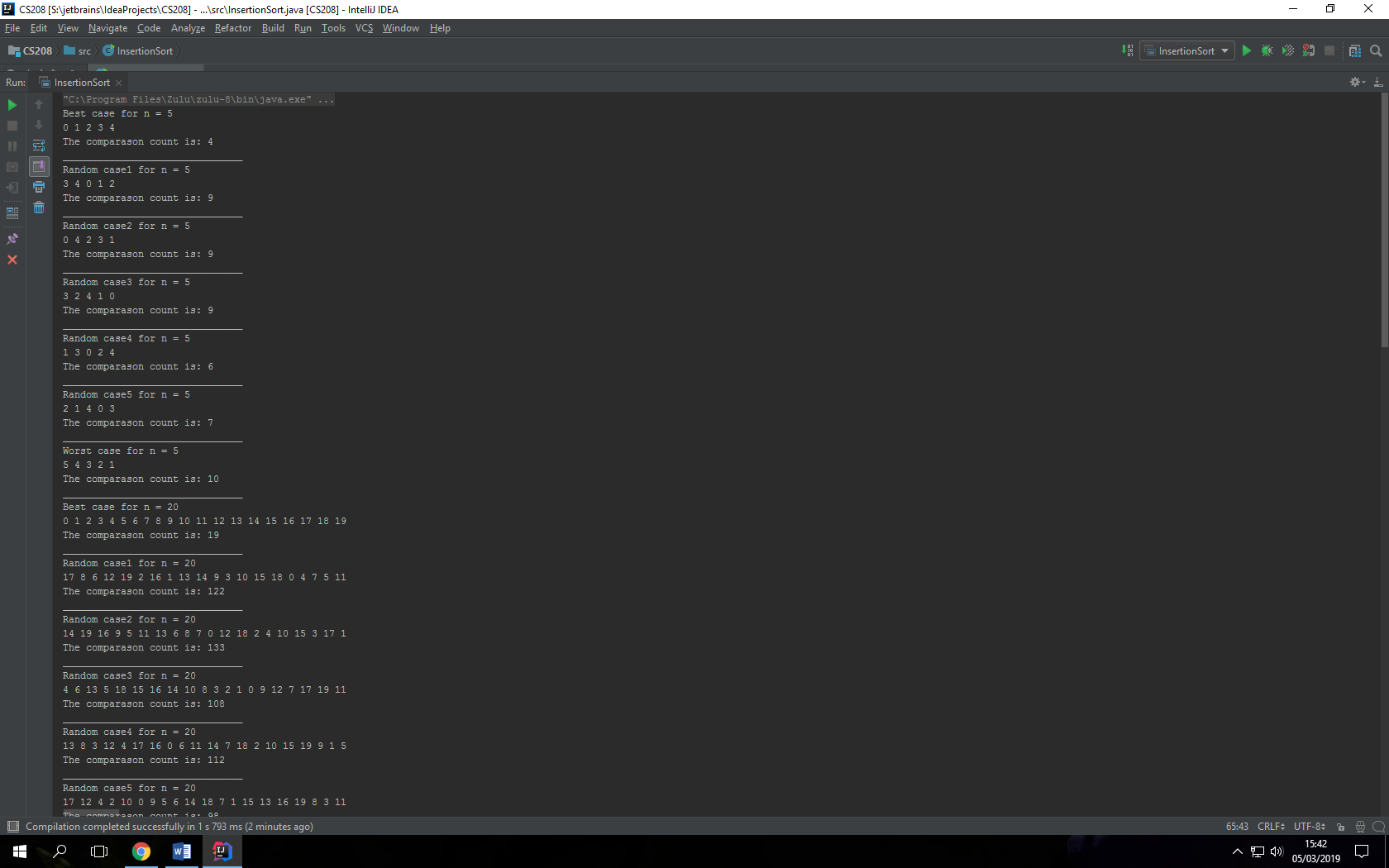
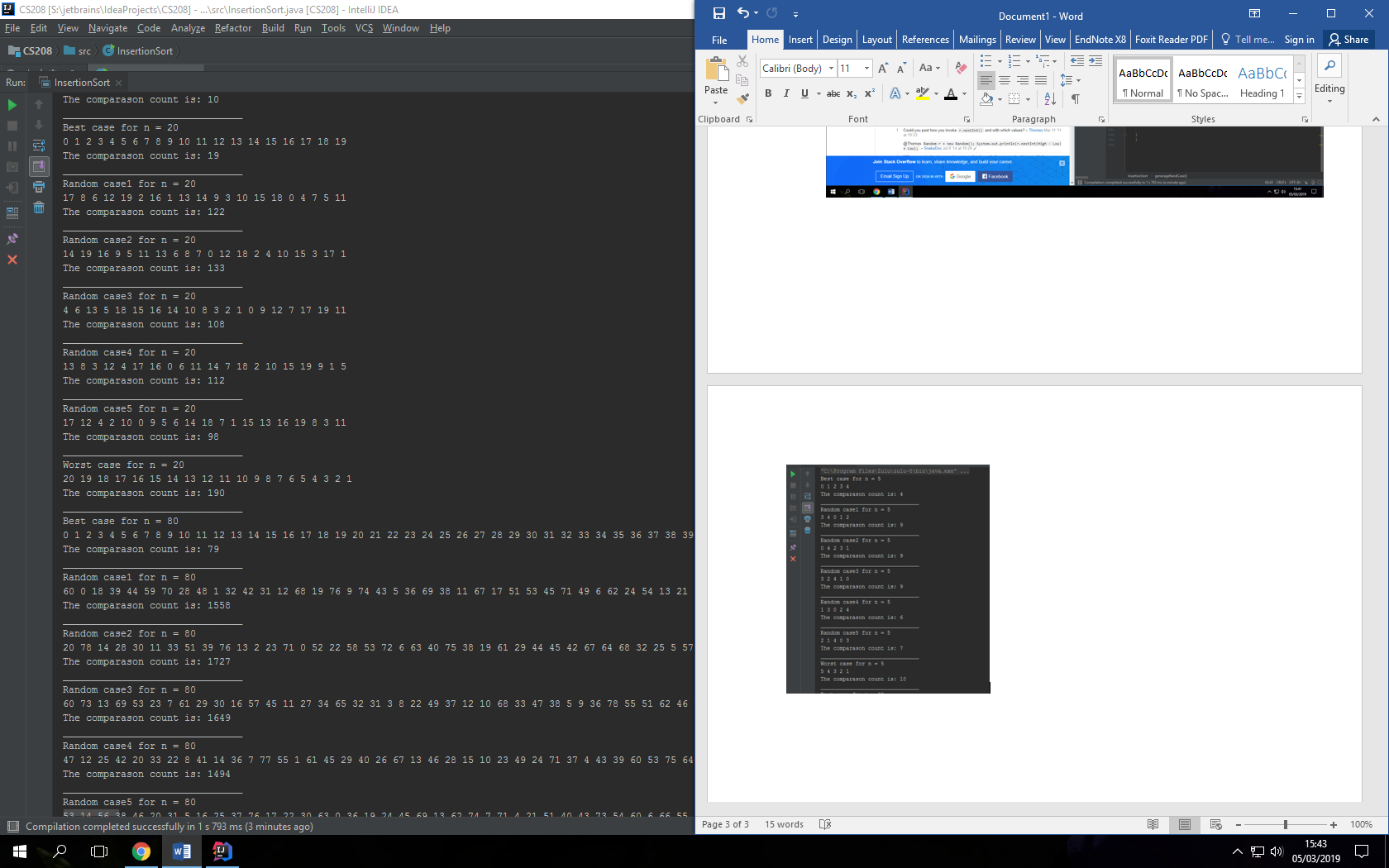
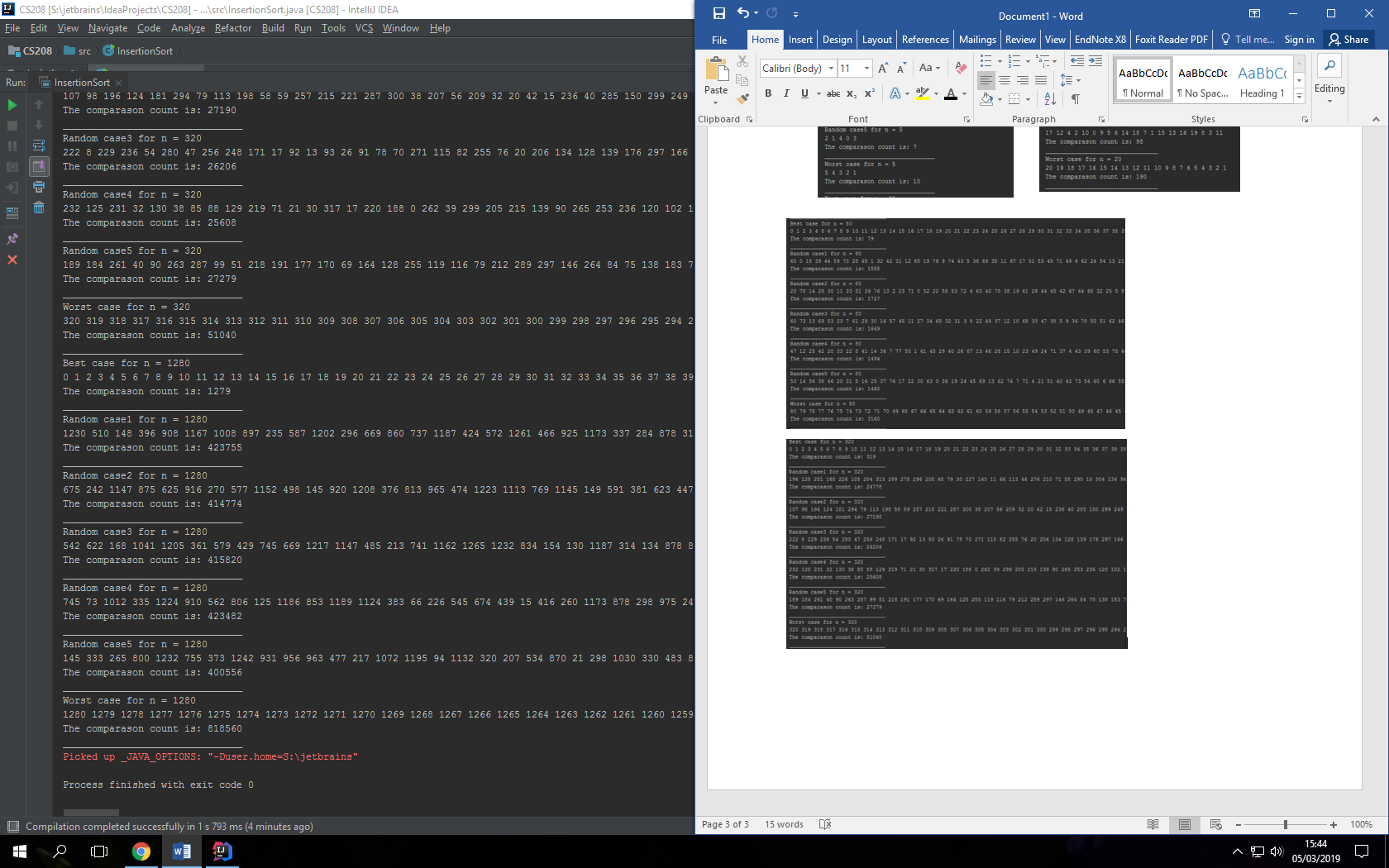
   


Figure 7 : Output when length = 320.

Figure 5 : Output when length = 20.

Figure 4 : Output when length = 5.

Figure 6 : Output when length = 80.

Figure 8: Output when length = 1280.

# Combining Results

The results were taken down in an excel spreadsheet and the results produced the following graphs. Figure 9 shows the table comparing best, worst and average cases, Figure 10 shows the Complexity graph of the best cases, Figure 11 shows the worst cases and Figure 12 shows the random cases. Figure 13 shows the 3 compared on the same graph using an average of the random cases instead.

|  |  |  |  |
| --- | --- | --- | --- |
| n | best case | worst case | AVG random |
| 5 | 4 | 10 | 8 |
| 20 | 19 | 190 | 114.6 |
| 80 | 79 | 3160 | 1577.6 |
| 320 | 319 | 51040 | 26212.2 |
| 1280 | 1279 | 818560 | 415677.4 |

Figure 9: Table of best, worst and average random cases against

Figure 10: Graph for best case complexity

Figure 11: Worst case complexity graph

Figure 12: Random case complexity graph

Figure 13 : Complexity comparisons between Best, Worst and Average

# Results

By looking at the table shown in figure 9 it could clearly be seen that the complexity of the best case for O(n-1) as every value was 1 less than n. By simply looking at the graph shown in figure 10 it could easily be seen that the best case has linear time asymptotic complexity, i.e. O(n), which matches us with the O(n-1) which would also have the asymptotic complexity of O(n).

Looking at the table in figure 9 again worst case was harder to distinguish however it could be seen from the graph in figure 11 that the asymptotic complexity was quadratic, i.e. O(n2). By looking at theory it could be seen that the actual worst case complexity should be O((n2 – n)/2) and by putting in the values in the table in figure 9 it could be seen to match up with this. Furthermore, this matches up with the asymptotic complex as this would also give us O(n2).

When looking for the average case it could be seen in the graph of figure 12 that the asymptotic complexity was again quadratic, O(n2), however by looking at the following graph in figure 13 the graph could be seen to be about half way between the best and worst case so it was clear that it had a lower complexity than worst case. When comparing this to theory it was given that the average case complexity was O((n2-n)/4) which not only matches up visually with the graph but when values were put in it came out to match almost exactly for the larger values. Furthermore, it could be seen by comparing figures 11 and 12 that the maximums of the scales were exactly half which matches with the given complexities. When looking at the lower values it was slightly off but this is simply due to how small the range of the values are so in order to gain a more confident result the larger values would be chosen anyway.

In conclusion, this study has given that the best case is O(n-1), the worst case is O((n2 – n)/2) and the average case is O((n2 – n)/4) with the asymptotic complexity of best case being O(n), worst case being O(n2) and the average case also being O(n2) with all the found values match up with theory.

# References

<https://www.geeksforgeeks.org/insertion-sort/> [1]