Pratyusha Thundena

Laboratory 10

FA2017 CS 103L-F4 Introduction to Computation Lab

November 17, 2017

Source Code

Figure 1:

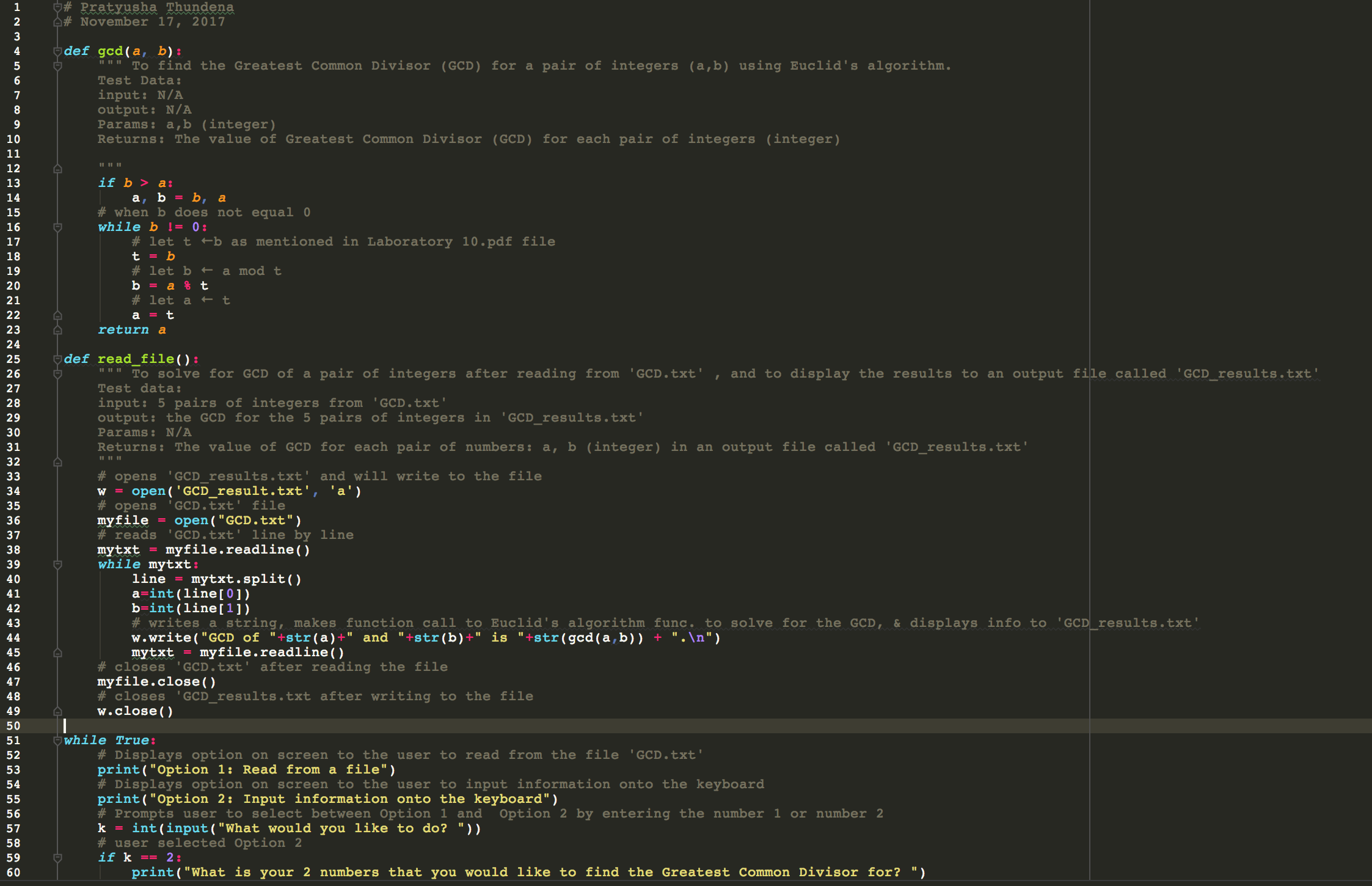


Figure 1 Cont. 1:

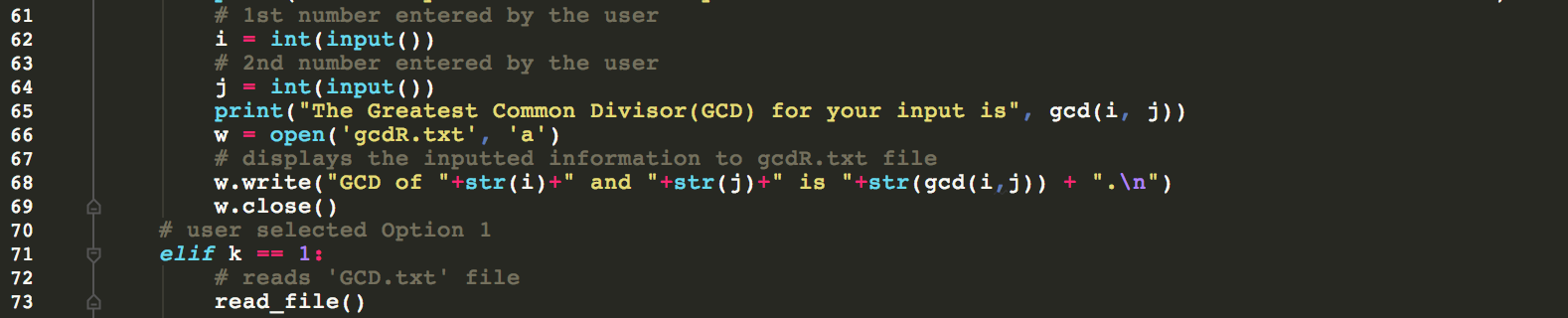


Figure 1a:

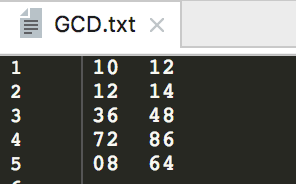


Figure 1b:

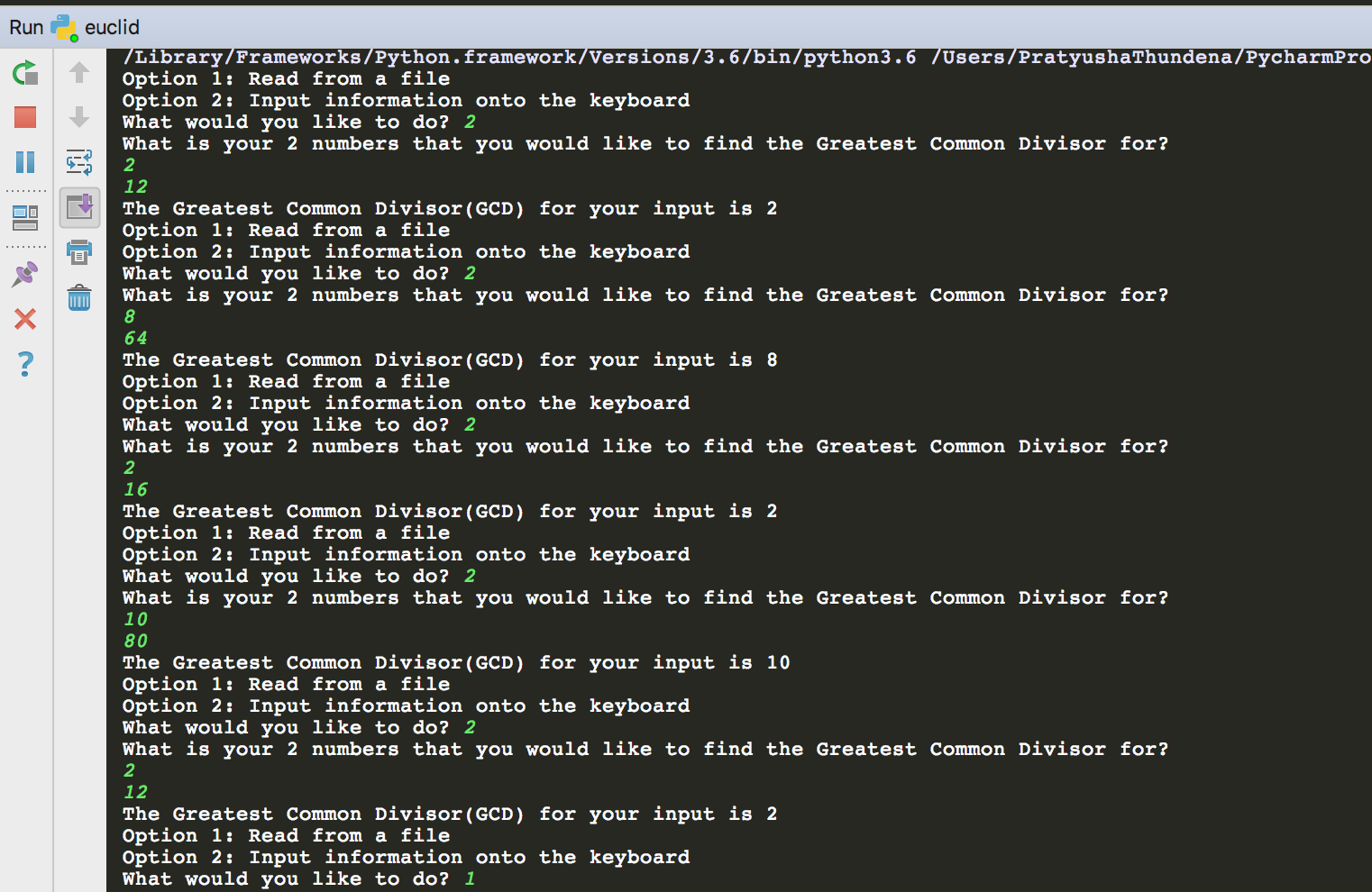


Figure 2:



Figure 2 Cont. 1:

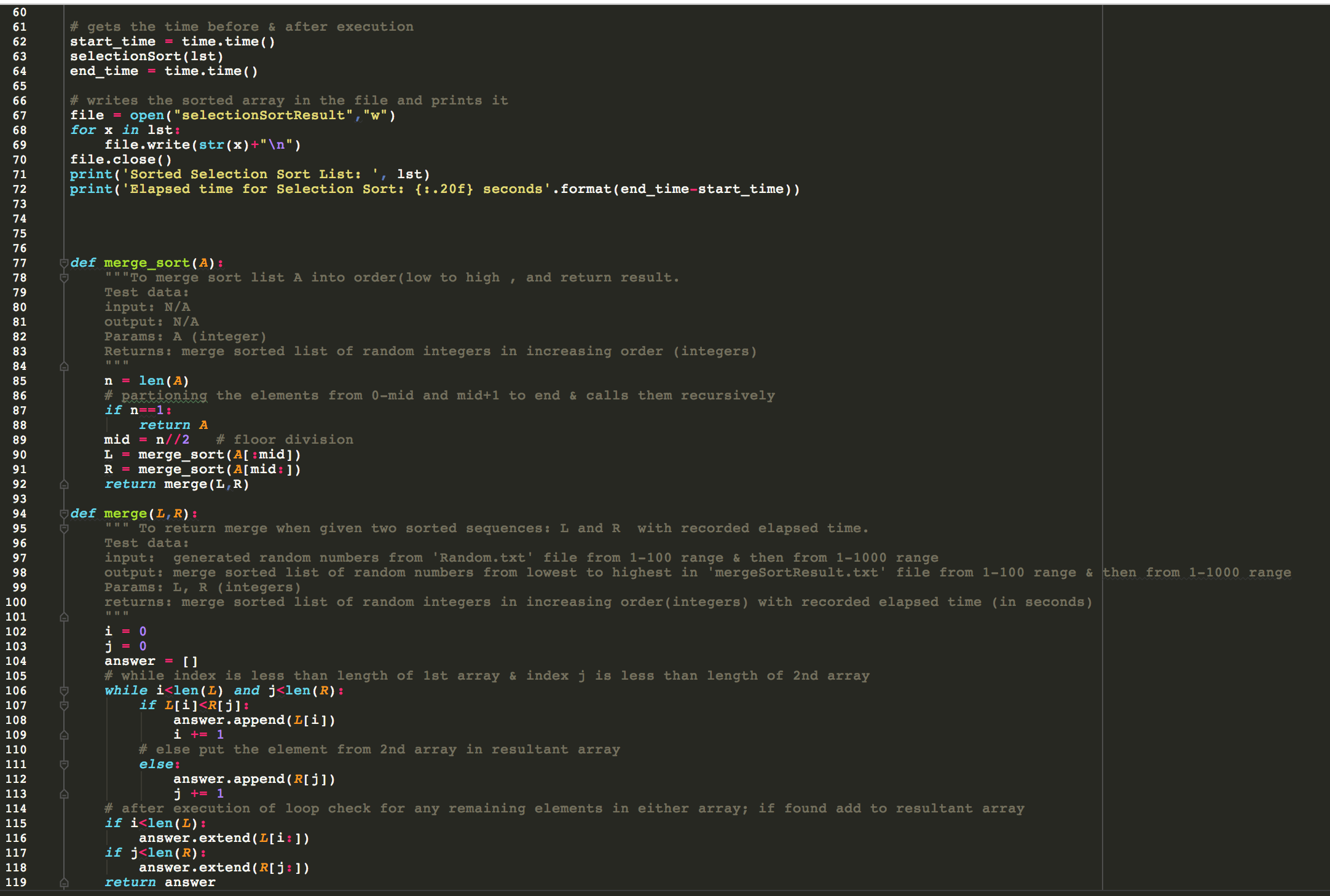


Figure 2 Cont. 2:

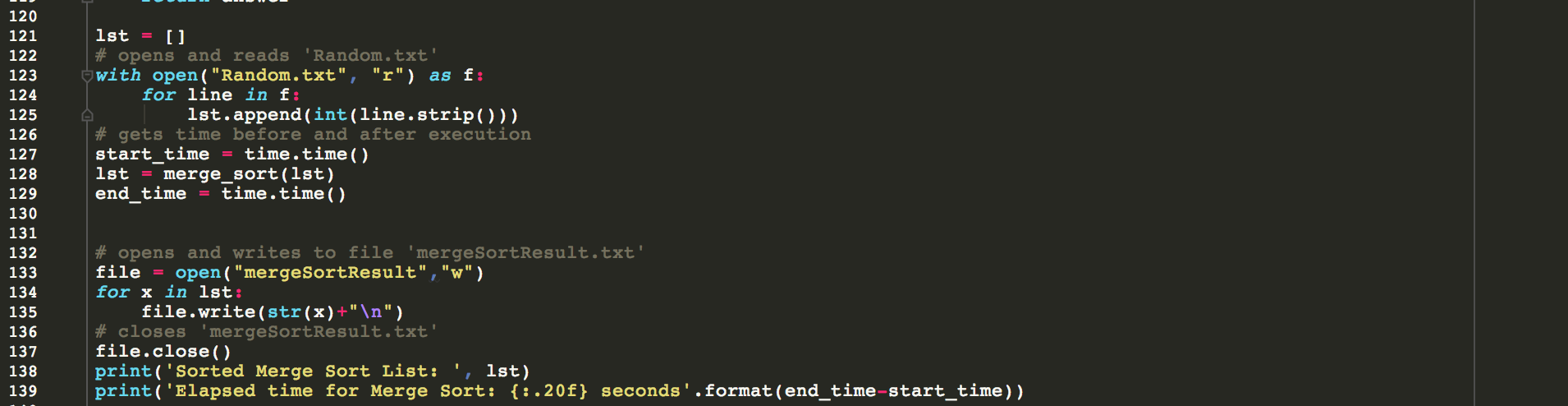


Figure 2a:

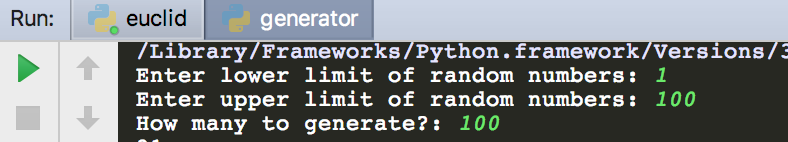
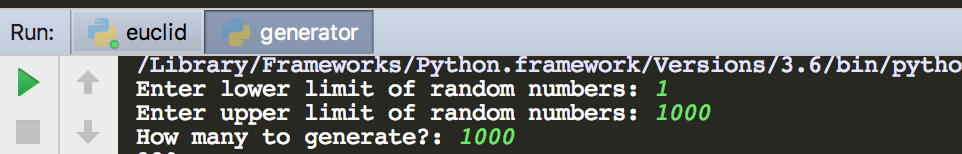


Figure 2b:



Demonstration to TA

Source code will be demonstrated on 11/27/2017 at 1:20 pm to BreAunna.

Program Results

Figure 1 Output Results 1:

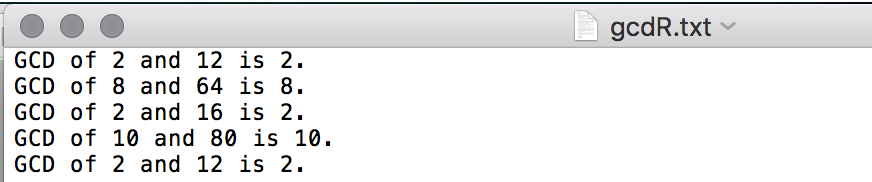


Figure 1 Output Results 2:

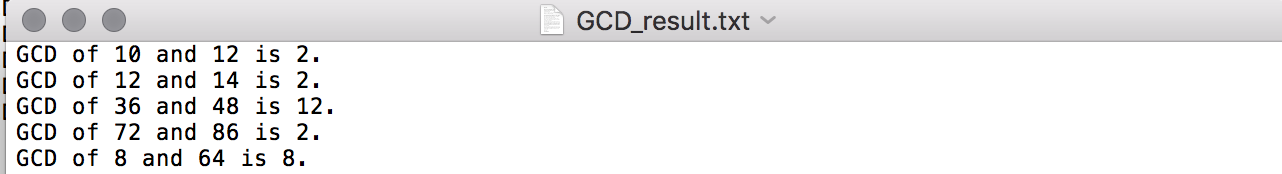


Figure 2 Output Results 1:

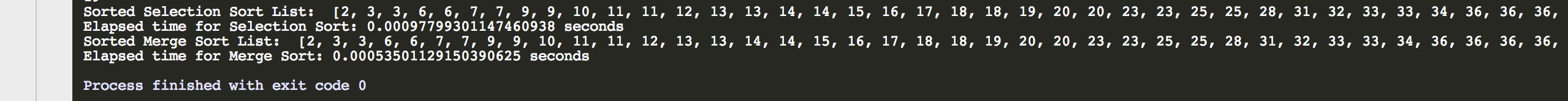


Figure 2 Output Results 2:

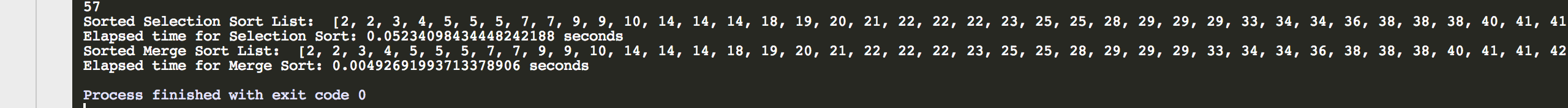


Figure 2 Output Results 3:

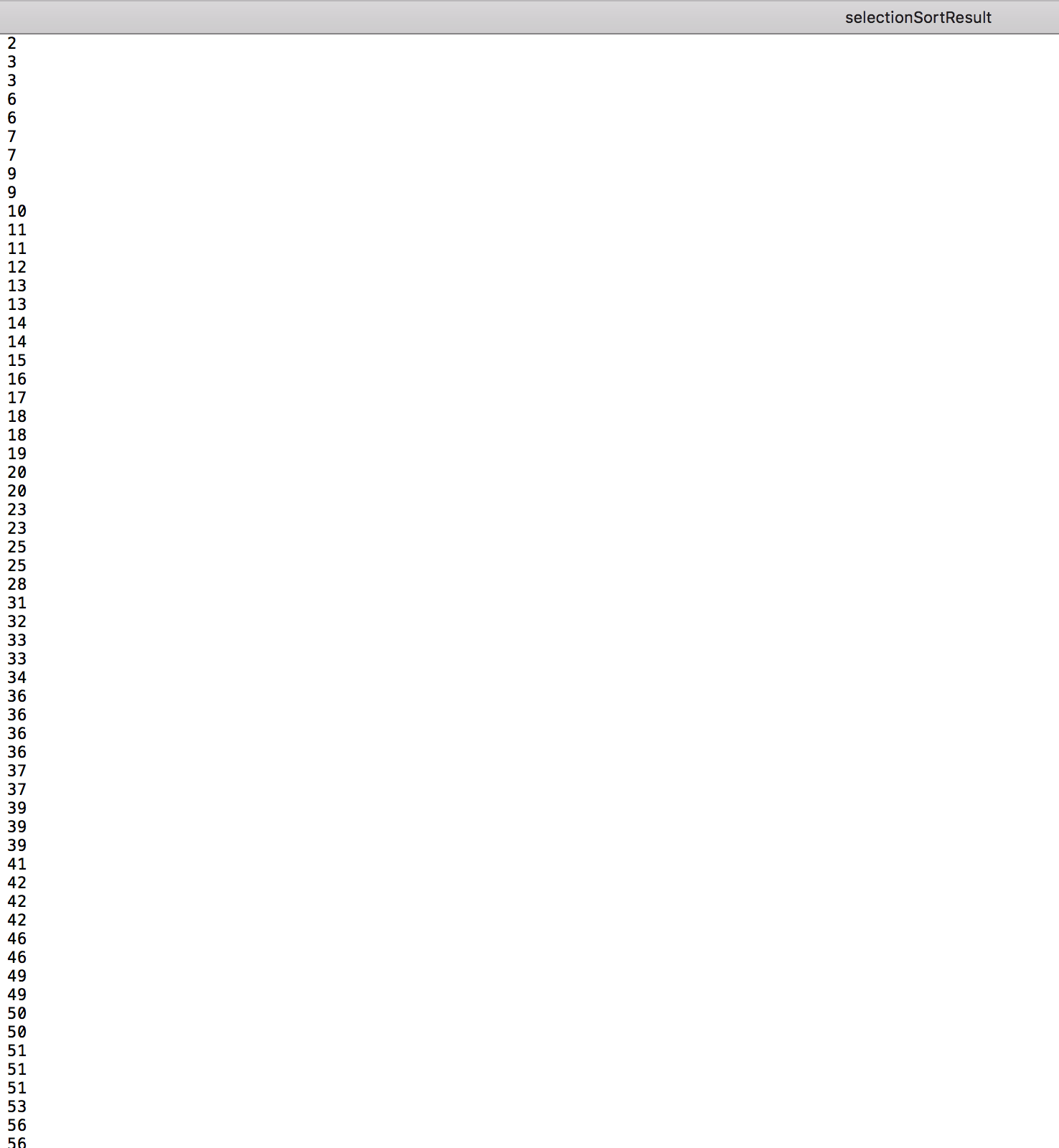


Figure 2 Output Results 3a:

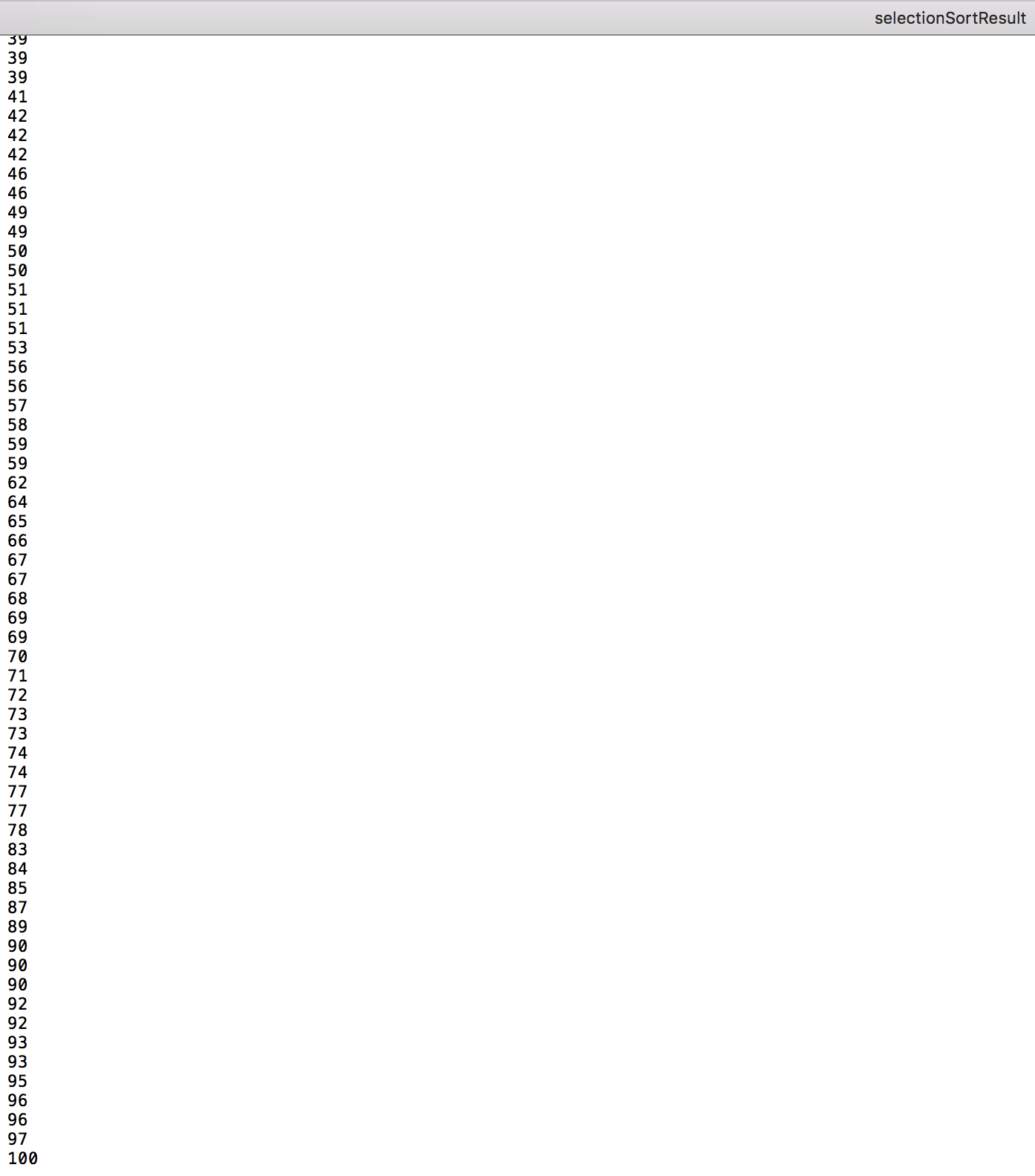


Figure 2 Output Results 4:

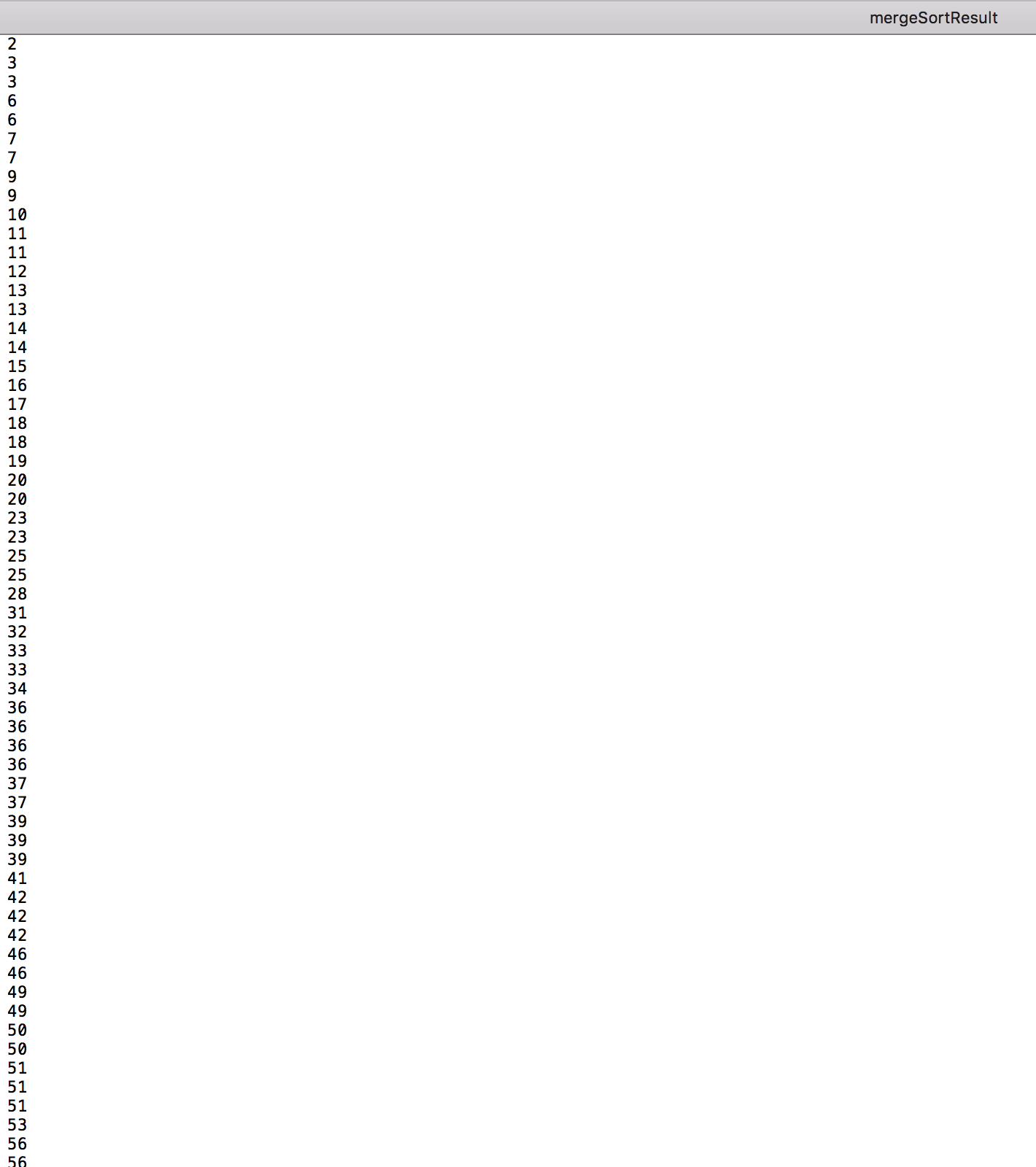


Figure 2 Output Results 4a:

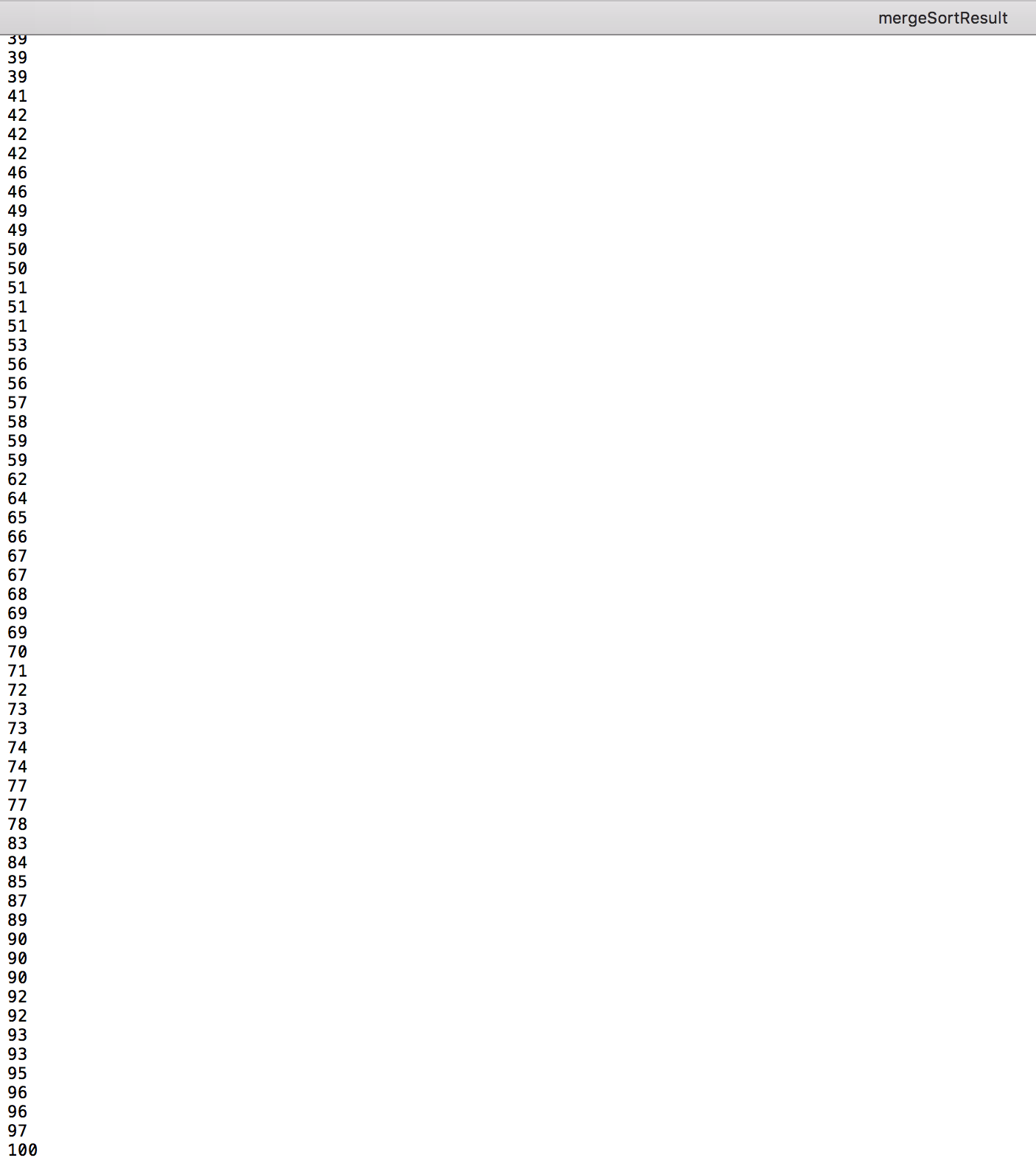


Figure 2 Output Results 5:

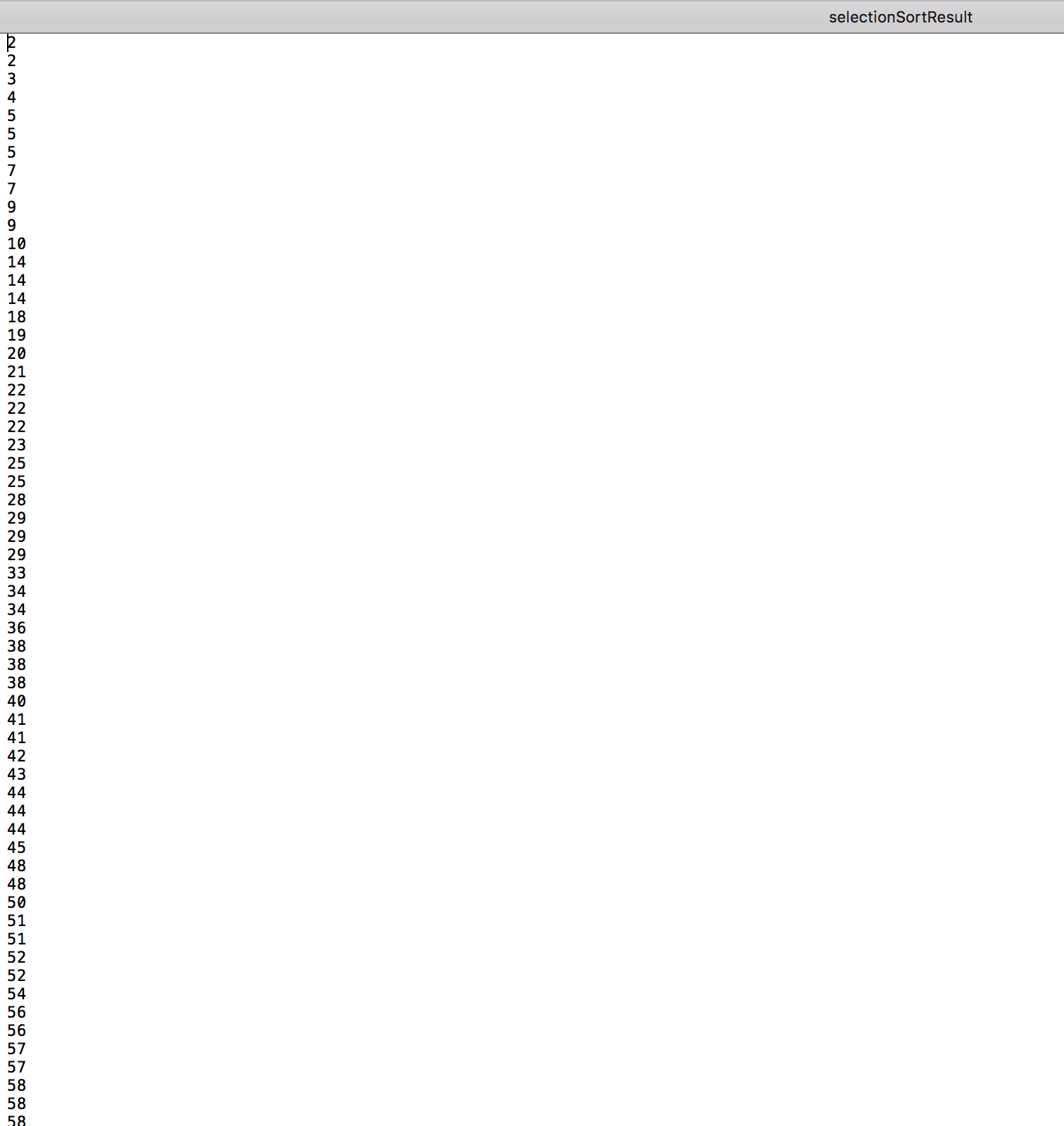


Figure 2 Output Results 5a:

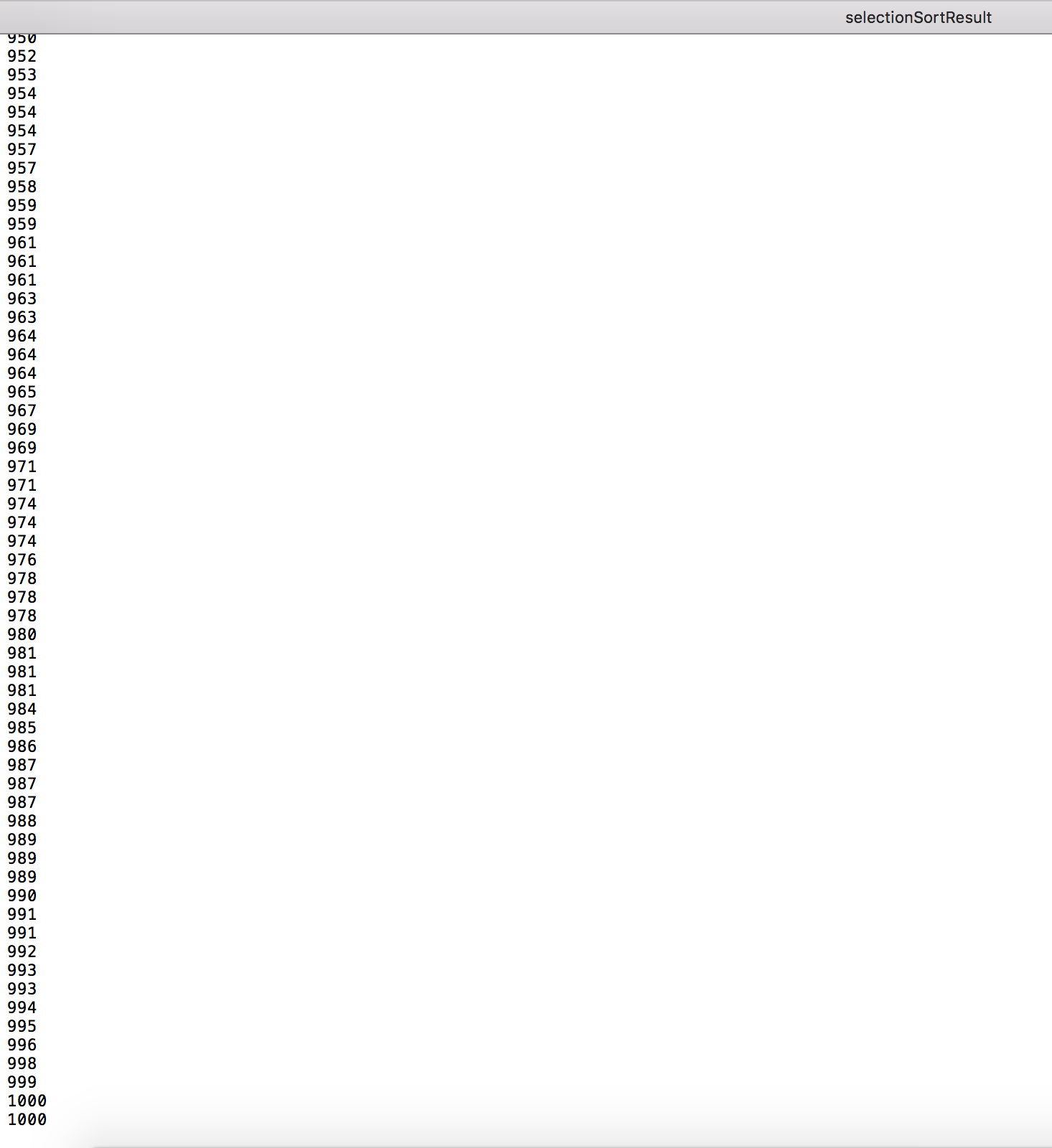


Figure 2 Output Results 6:

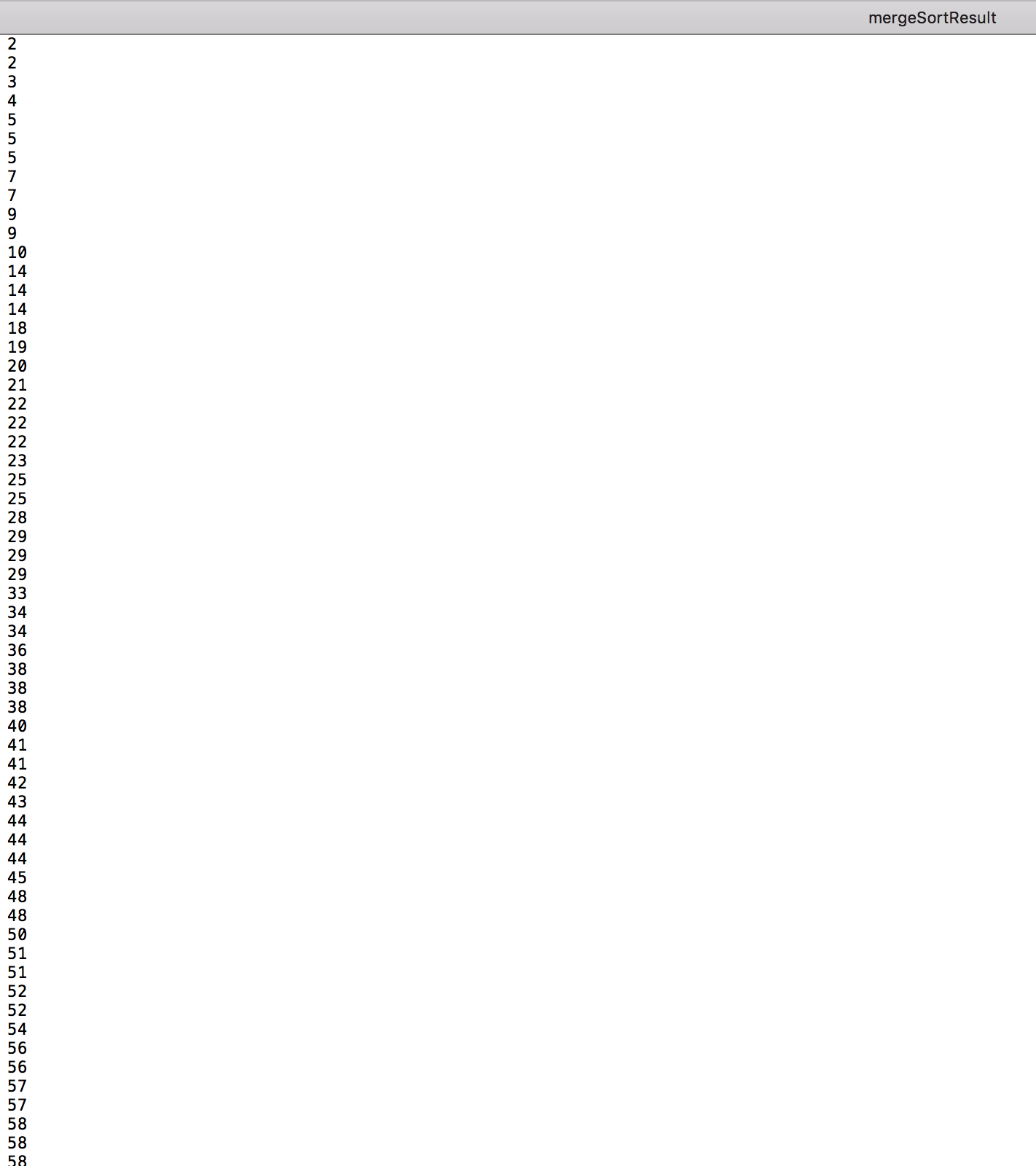


Figure 2 Output Results 6a:

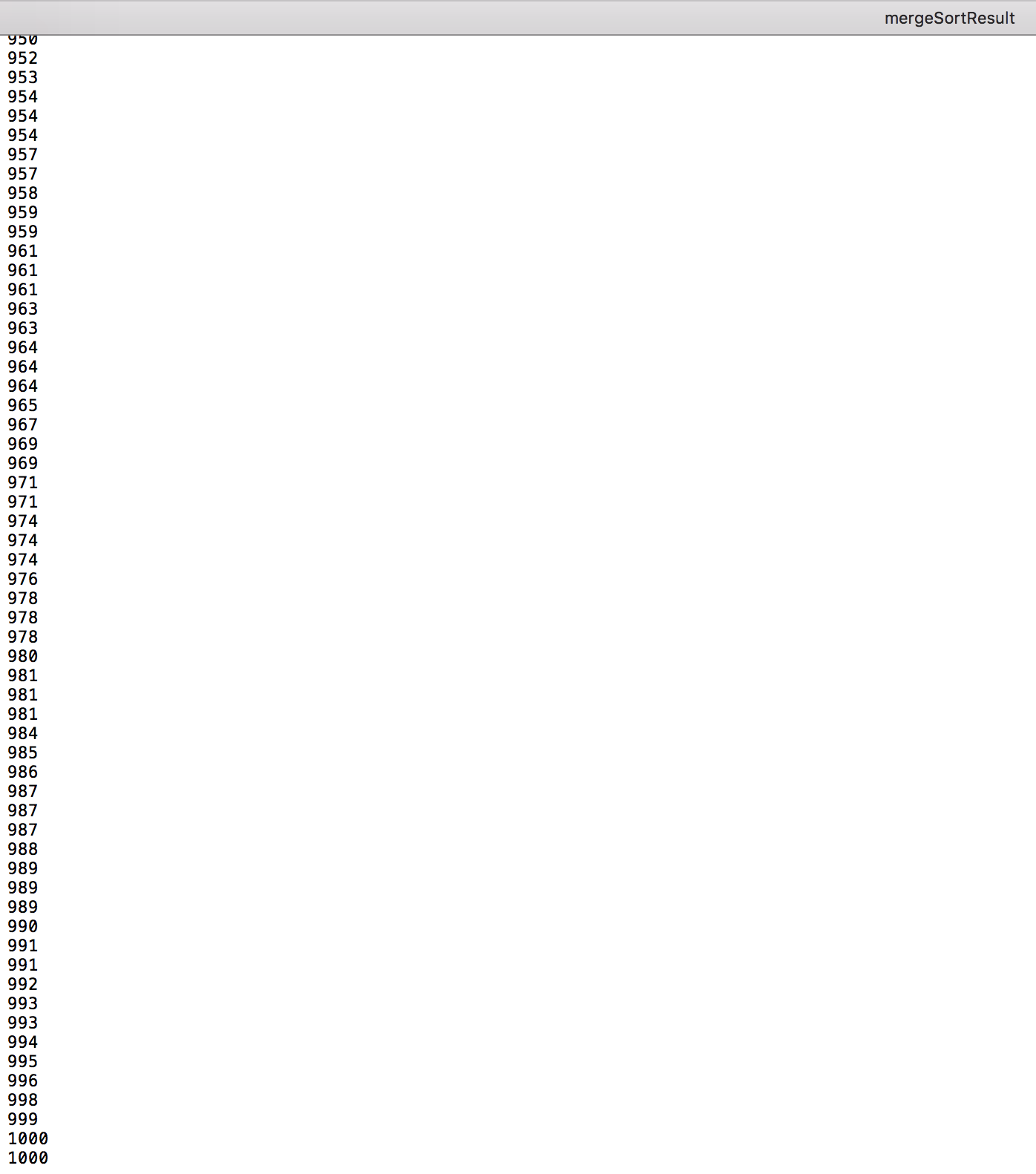
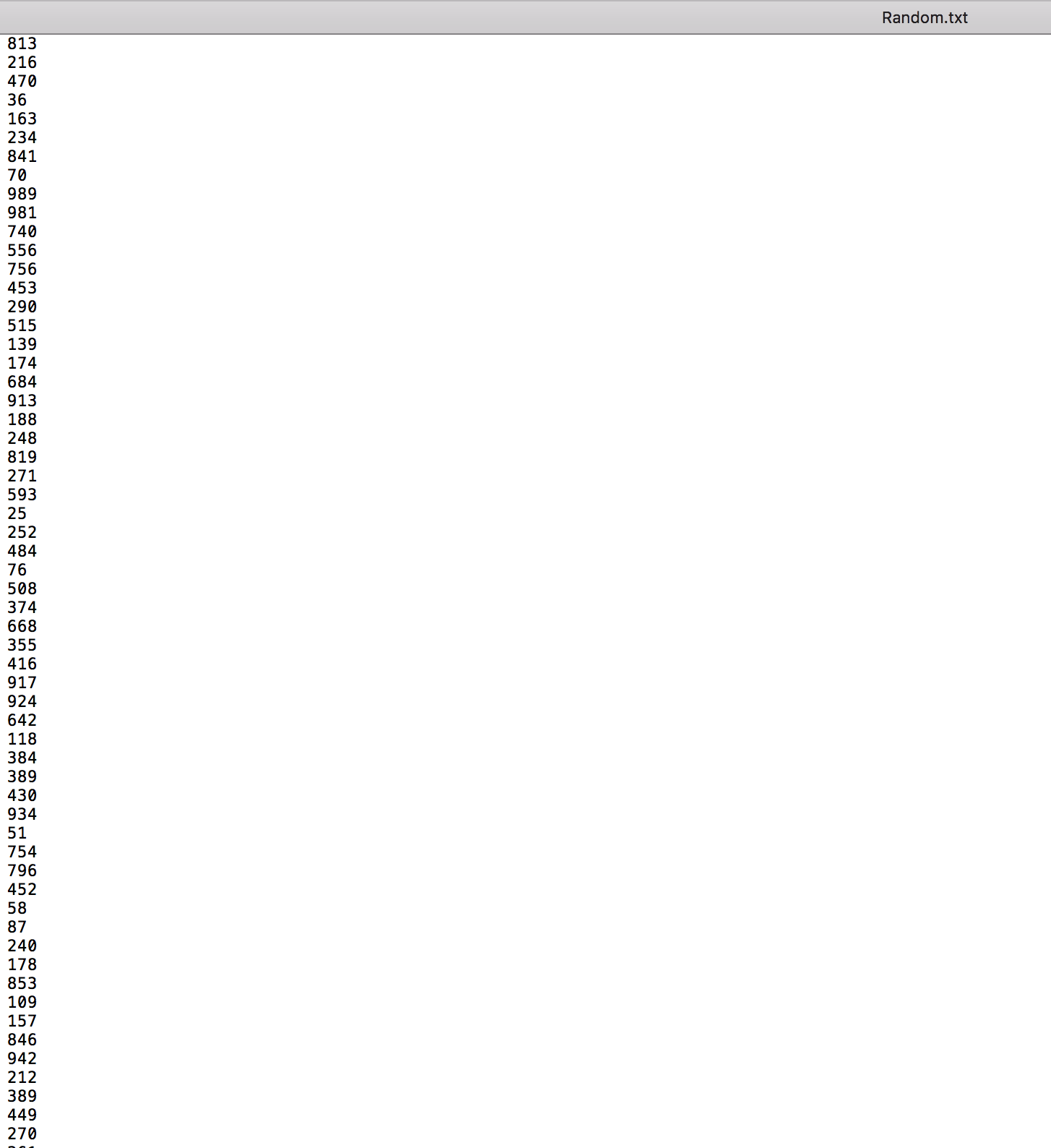


Figure 2 Output Results 7



Conclusion and Results

The first portion of the lab involved implementing Euclid’s algorithm in order to find the Greatest Common Divisor (GCD). The program had to ask the user whether input would be entered from the keyboard or read from a file (Figure 1, Figure 1 Cont.). If the input was entered from the keyboard, then the user is queried for two numbers and GDD is determined both on the screen and by writing to an output file called ‘gcdR.txt’ (Figure 1b, Figure 1 Output Results 1). If input was read from a file, then the file had to be opened and read. The input file called ‘GCD.txt’ had two numbers on each line for five test cases (Figure 1a). The output file called ‘GCD\_result.txt’ contained the solved GCD for the five test cases from the previously mentioned input file (Figure 1 Output Results 2). This lab was challenging because I had a difficult time writing files. After this lab, I have learned how to properly open, read, and write to files.

The second portion of the lab involved generating random numbers and writing them to a file called ‘Random.txt’ (Figure 2 Output Results 7). The user is asked to input the range over which the random numbers will be generated and how many numbers are desired (Figure 2a, Figure 2b, Figure 2, Figure 2 Cont. 1, Figure 2 Cont. 2). The Python pseudorandom number generator was used to create random numbers. The ‘Random.txt’ file was opened and sorted, and the sorted output was written to two new files; specifically, ‘selectionSortResult’ file and ‘mergeSortResult’ file (Figure 2 Output Results 3 – Figure 2 Output Results 6a). A timer was used to compare the elapsed time between the results from selection sort and the results from merge sort. Test case one involved sorting random numbers up to hundred and test case 2 involved sorting random numbers up to a thousand. Merge sort of the random numbers was faster than selection sort of the random numbers in the two test cases (Figure 2 Output Results 1 and Figure 2 Output Results 2). I collaborated with Matt Campbell and a site called <http://cs.umw.edu/-finlayson/class/falll2/cpsc110/notes/12-sorting.html>, which was immensely helpful in figuring out merge sort and selection sort algorithms.