Zigeng Fu (NUID: 001448458)

INFO 6205
Program Structures & Algorithms
Spring 2019
Assignment 2

The purpose of this assignment is to implement the *doRun()* method and *main()* method in Benchmark.java, and then deduct the order of growth of the sorting algorithm *Arrays.sort()* by observing the running time of the algorithm when arrays with different sizes and orders are input.

After implementing the program, testing the cases and running out the data, I figured out the general running time when a random array is input. The detailed analyzing processes are as follows:

I. Implementation and Testing

In the implementation of this benchmark case, I used Functional interfaces of Java 8, and *Arrays.sort()* method for sorting an array. I calculated the estimated time by starting a timer at the beginning and stopping it after algorithm running to obtain the time difference. The key method used in this step is *System.nanoTime()*.

In the *main()* method, I generated 4 kinds of arrays with different orders, which were **sequential order**, **reversed order**, **partially sequential order**, and **random order**. For each kind of order, 5 arrays with different sizes **between 1000 and 16000** were input in turn, and the running time was calculated. The running time data is shown as Table 1.

	Run Time (ms)				
Array Length	Sequential Order	Reverse Order	Partial Order	Random Order	Average Case
1000	0.0113	0.0155	0.0331	0.2773	0.0843
2000	0.0185	0.0236	0.1317	1.1385	0.3281
4000	0.0434	0.0502	0.2596	1.0483	0.3504
8000	0.0426	0.0948	0.4994	1.1169	0.4384
16000	0.0239	0.0627	1.1039	2.2997	0.8726

Table 1 Running Time of Different Cases

II. Observation and Conclusion

Zigeng Fu (NUID: 001448458)

According to the running time data shown in Table 1, I figured out the order of growth of the algorithm *Arrays.sort()* as the following Chart 1 shows:

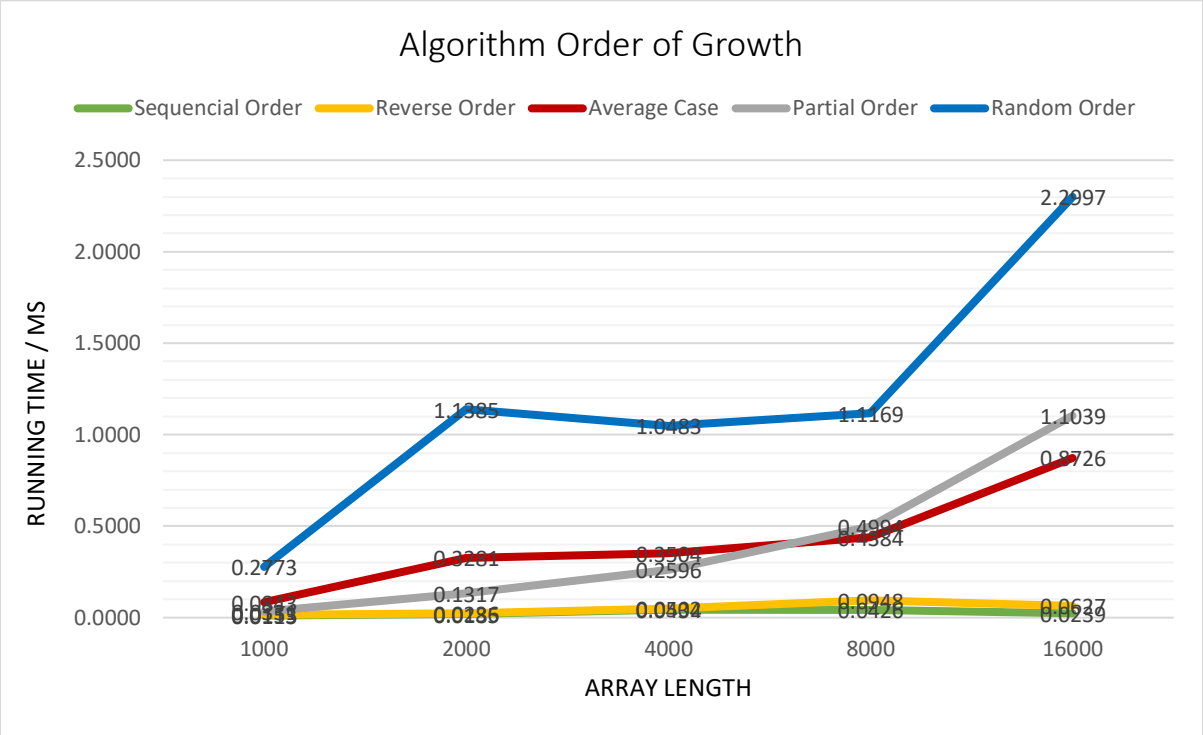

Chart 1 Algorithm Order of Growth

The Chart 1 shows the running time of the algorithm with different input arrays. According to the chart, I observed that among the 4 kinds of input arrays, the **sequentially ordered one is the best case**, which has the lower bound on time cost. And the **randomly ordered one is the worst case**, which has the upper bound on time cost. **For the average case**, I calculated the average time cost of 4 kinds of arrays, and the order of growth is shown as the red line.

Thus, according to the data and chart, I conclude that when a **random array** is input into the algorithm, the average time cost will have a **linearithmic growth** as the size of the array growing.