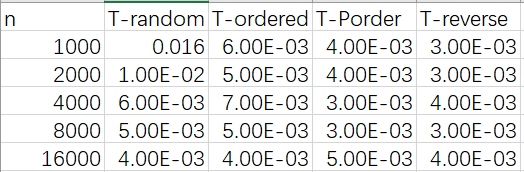
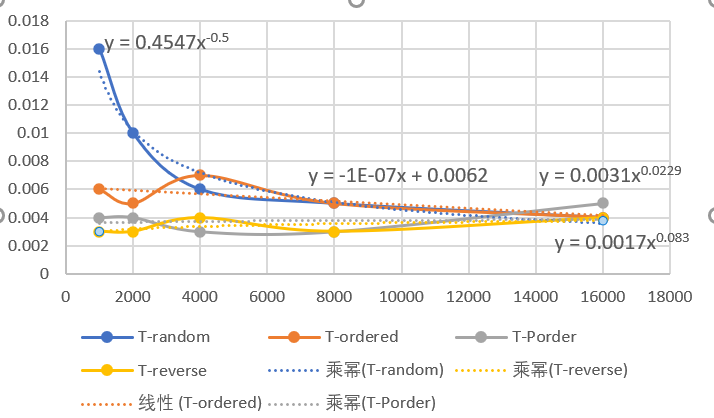
I measure the running times of insertion sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered. And I used the doubling method choosing n = 1000, 2000, 4000, 8000, 16000. For each type of array, I found the relationship of repetition number n and running time T. In insertion sort, the time complexity is O(n+f(n)) where n is the inversion count.

This screen shot of spreadsheet shows the data of my experiment. From these data, I drew a line graph and a trend line with a fitted function expression, which is the relationship between time and n. In the order array, which is the best case, there is a linear relationship between N and time t, and the expression is as follows. Thus the time complexity of implement insertion sort for an ordered array is O(n), f(n)=0. In the reverse-ordered array, which is the worst case, the relation between N and time t is power. According to mathematical derivation, f(n) = n\*(n-1)/2 = O(n^2), thus the time complexity of implement insertion sort for a reverse-ordered array is O(n^2). For random array and partially-ordered array, f(n)!=0, so the time complexity is still O(n^2). The relation between N and time t from the graph is also power.

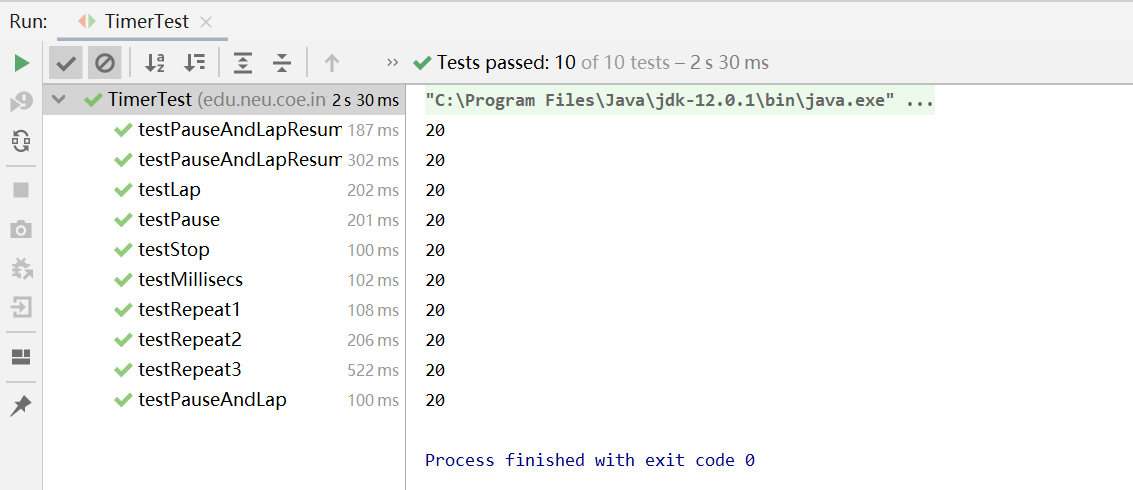
Because the number of experiments is not enough, and the array may not be large enough, the experimental results may deviate from the theory.





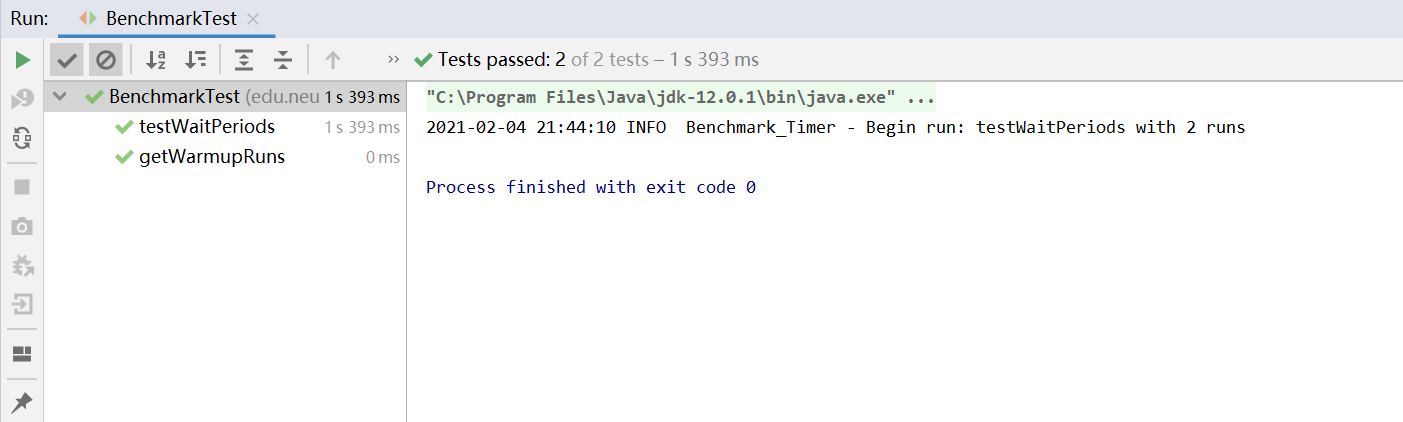
* TimerTest.java

Result:



* BenchmarkTest.java

Result:



* InsertionSortTest.java

Result:

