YanXuan NUID:001563047

**Program Structures & Algorithms**

**Fall 2021**

**Assignment No. 2**

* **Task (List down the tasks performed in the Assignment)**

1. **Completed Timer.java and successfully passed the test.**
2. **Completed InsertionSort.java and successfully passed the test.**
3. **Completed SortTimeTestMain.java to test the InsertionSort.java.**
4. **Perform data simulation, uses four types of Integer arrays, double the number of array’s length to see how time increases, and make experimental data tables and graphs.**
5. **Analyze and summarize the experimental data, and get the conclusion.**

* **Part1**

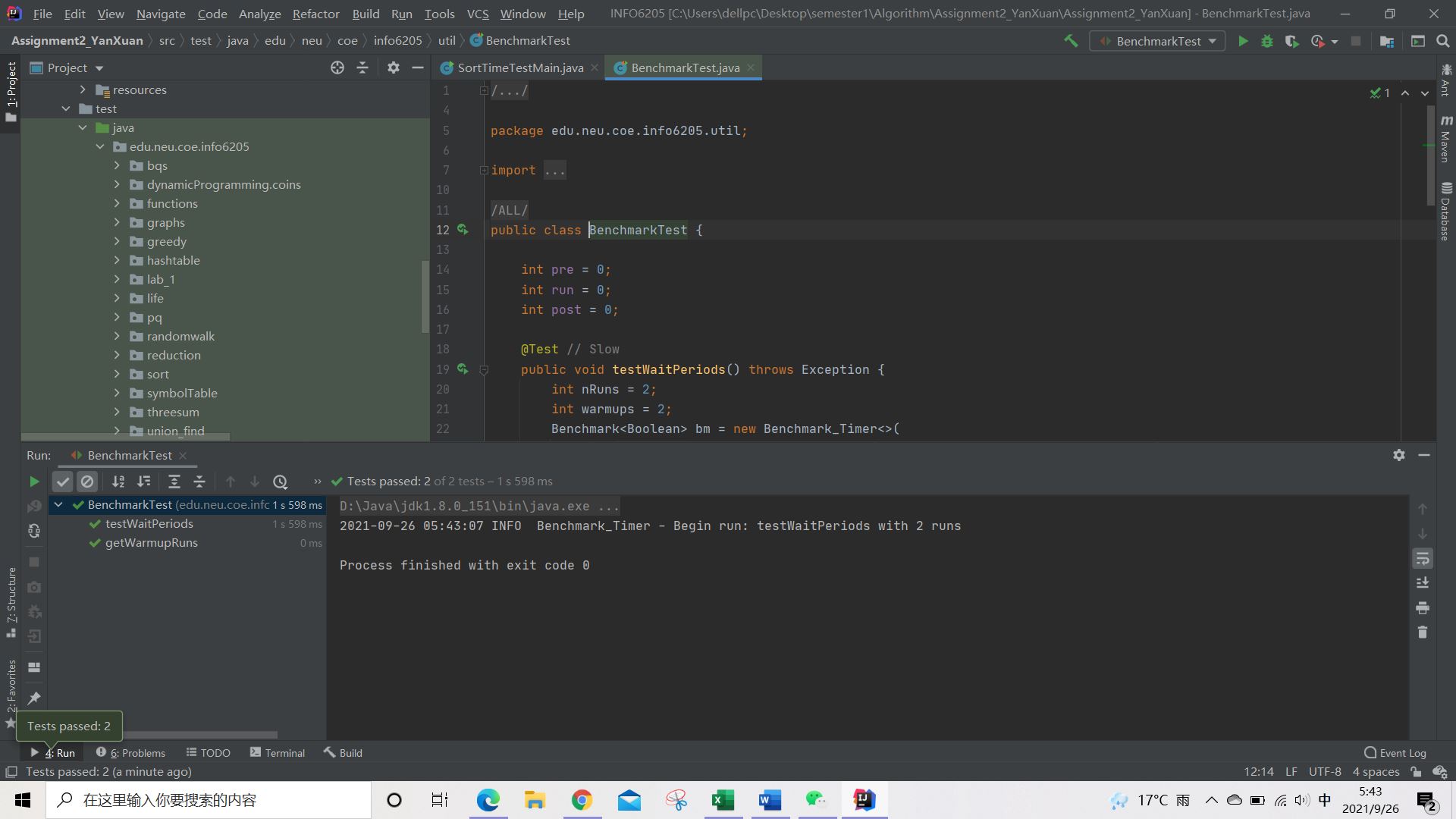
1. **Completed Timer.java**

public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> function, UnaryOperator<T> preFunction, Consumer<U> postFunction) {  
 *logger*.trace("repeat: with " + n + " runs");  
 pause();  
 T t = null;  
 U u = null;  
 // TO BE IMPLEMENTED: note that the timer is running when this method is called and should still be running when it returns.  
  
 for(int i = 0; i<n;i++){  
 if(preFunction != null){  
 t = preFunction.apply(supplier.get());  
 }  
 resume();  
 if(preFunction == null && postFunction != null){  
 u= function.apply(supplier.get());  
 lap();  
 }else if(preFunction != null && postFunction == null){  
 function.apply(t);  
 lap();  
 }else if(preFunction != null && postFunction != null){  
 u = function.apply(t);  
 lap();  
 }else if(preFunction == null && postFunction == null){  
 function.apply(supplier.get());  
 lap();  
 }  
 pause();  
 if(postFunction != null){  
 postFunction.accept(u);  
 }  
  
 }  
 double meanLapTime = meanLapTime();  
 //ticks = getClock() - start;  
 resume();  
 return meanLapTime;  
}

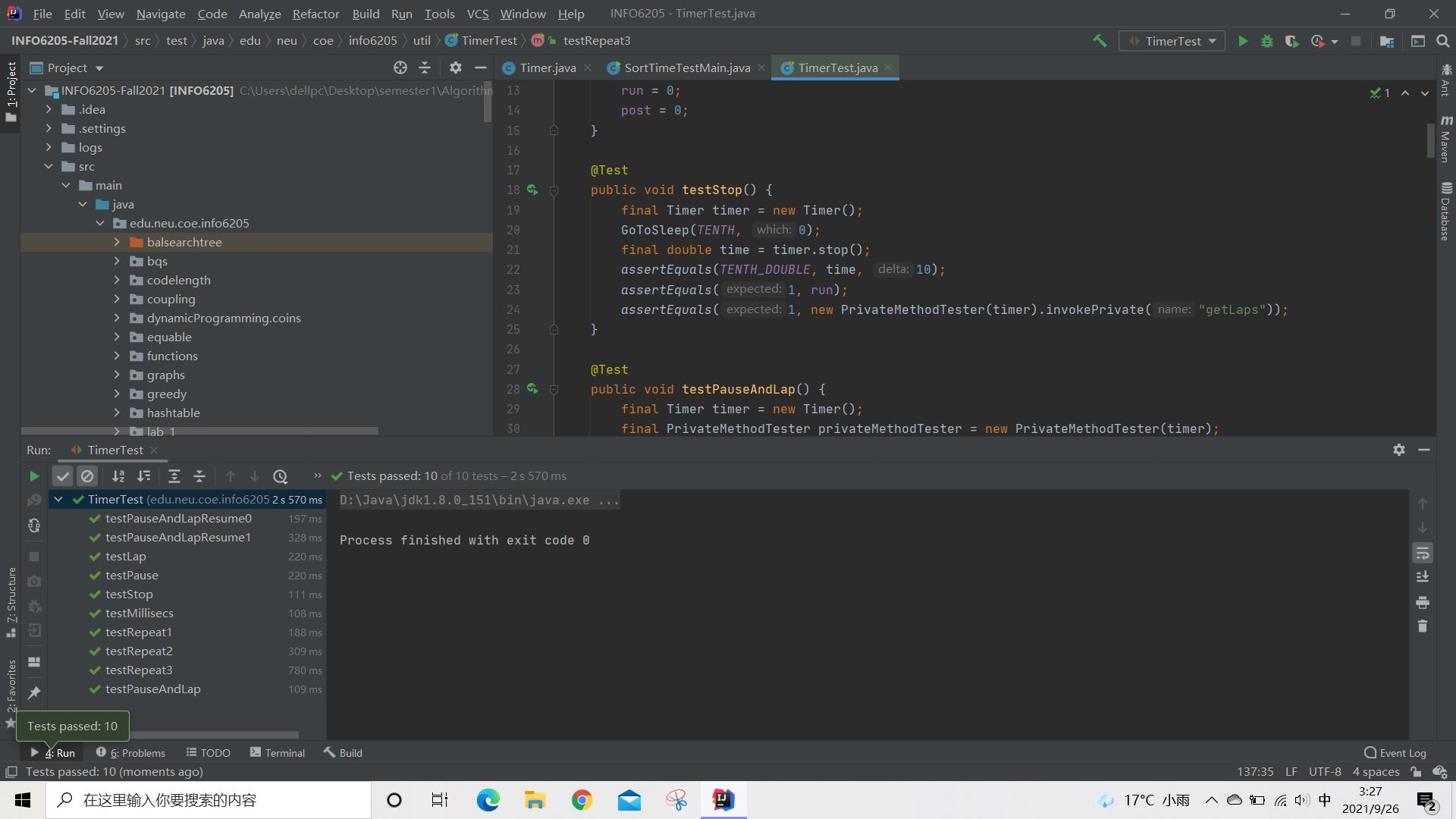
private static long getClock() {  
 // TO BE IMPLEMENTED  
 return System.*nanoTime*();  
}  
  
private static double toMillisecs(long ticks) {  
 // TO BE IMPLEMENTED  
 //long millTicks = TimeUnit.NANOSECONDS.toMillis(ticks);  
 long millTicks = TimeUnit.*MILLISECONDS*.convert(ticks, TimeUnit.*NANOSECONDS*);  
 return millTicks;  
}

1. **Unit tests result:(Snapshot of successful unit test run)**

**BenchmarkTest.java:**



**TimerTest.java:**



1. **Problems when testing**

It seems that the actual mean time is around 30, which don’t fit to the testRepeat2() and testRepeat3()’s requirements. My laptop is dell. I then changed the delta of each function and passed the test. I also used my roommate’s Mac to compile the same code with the same delta required by the professor and then succeeded. So the problem of the timer’s test maybe is the responsibility of my laptop.

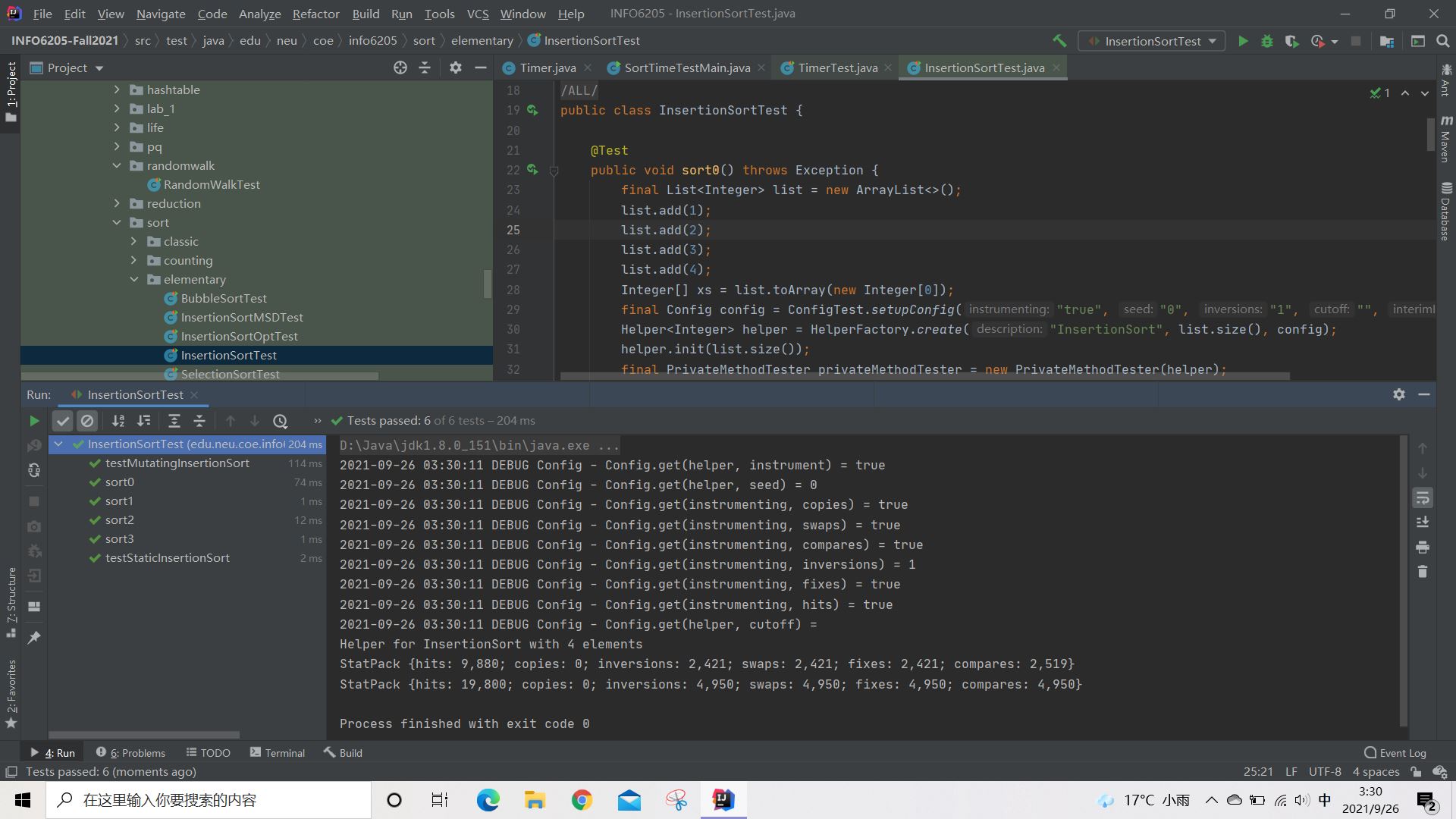
* **Part2**

1. **Completed InsertionSort.java**

public void sort(X[] xs, int from, int to) {  
 final Helper<X> helper = getHelper();  
  
 // TO BE IMPLEMENTED  
 int i = from + 1;  
 while(i < to){  
 for(int j = i;j>from;j--){  
 if(!helper.swapStableConditional(xs,j)) break;  
 }  
 i++;  
  
 }  
}

1. **Unit tests result:(Snapshot of successful unit test run)**

**InsertionSortTest.java:**



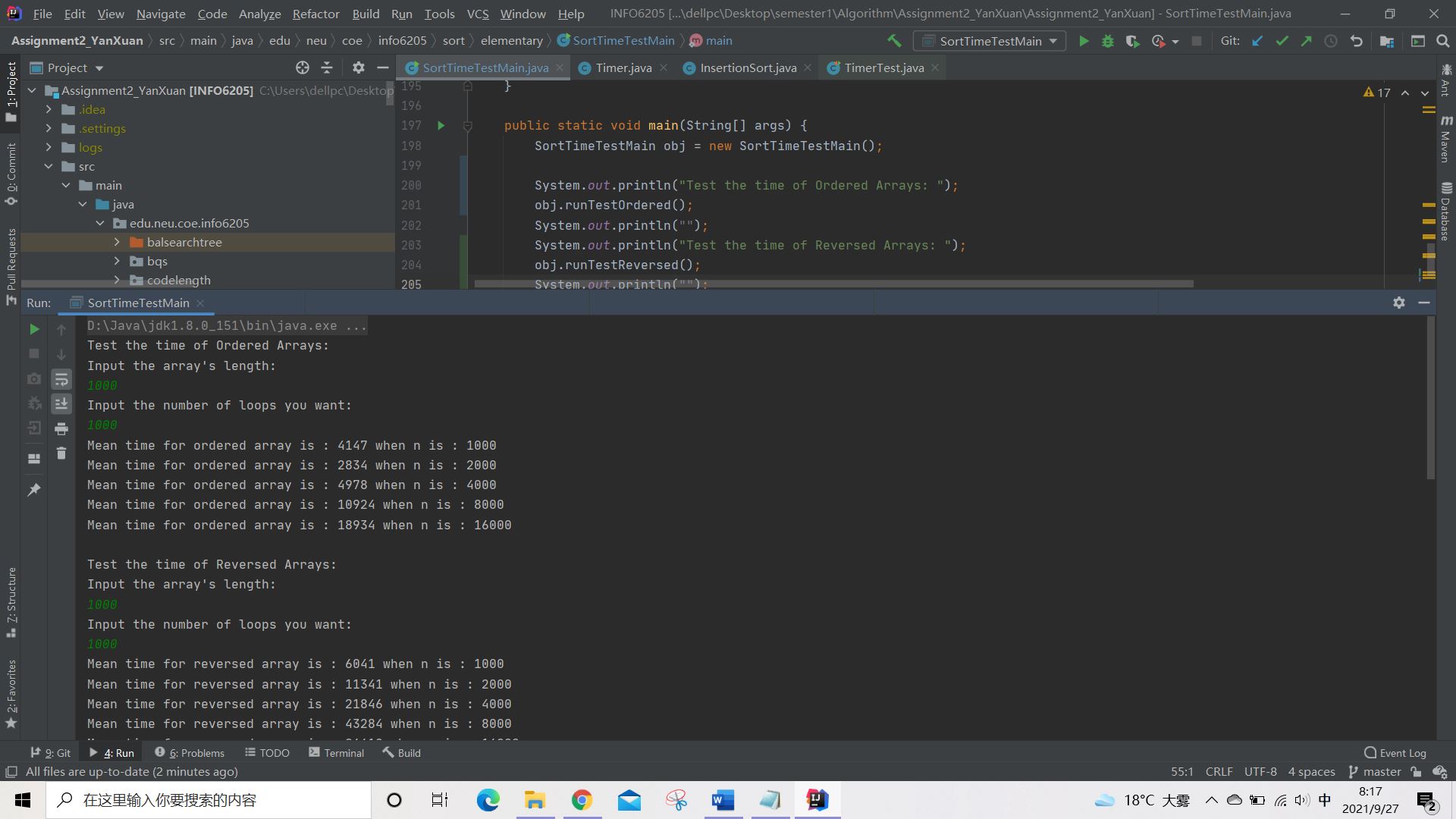
* **Part3**

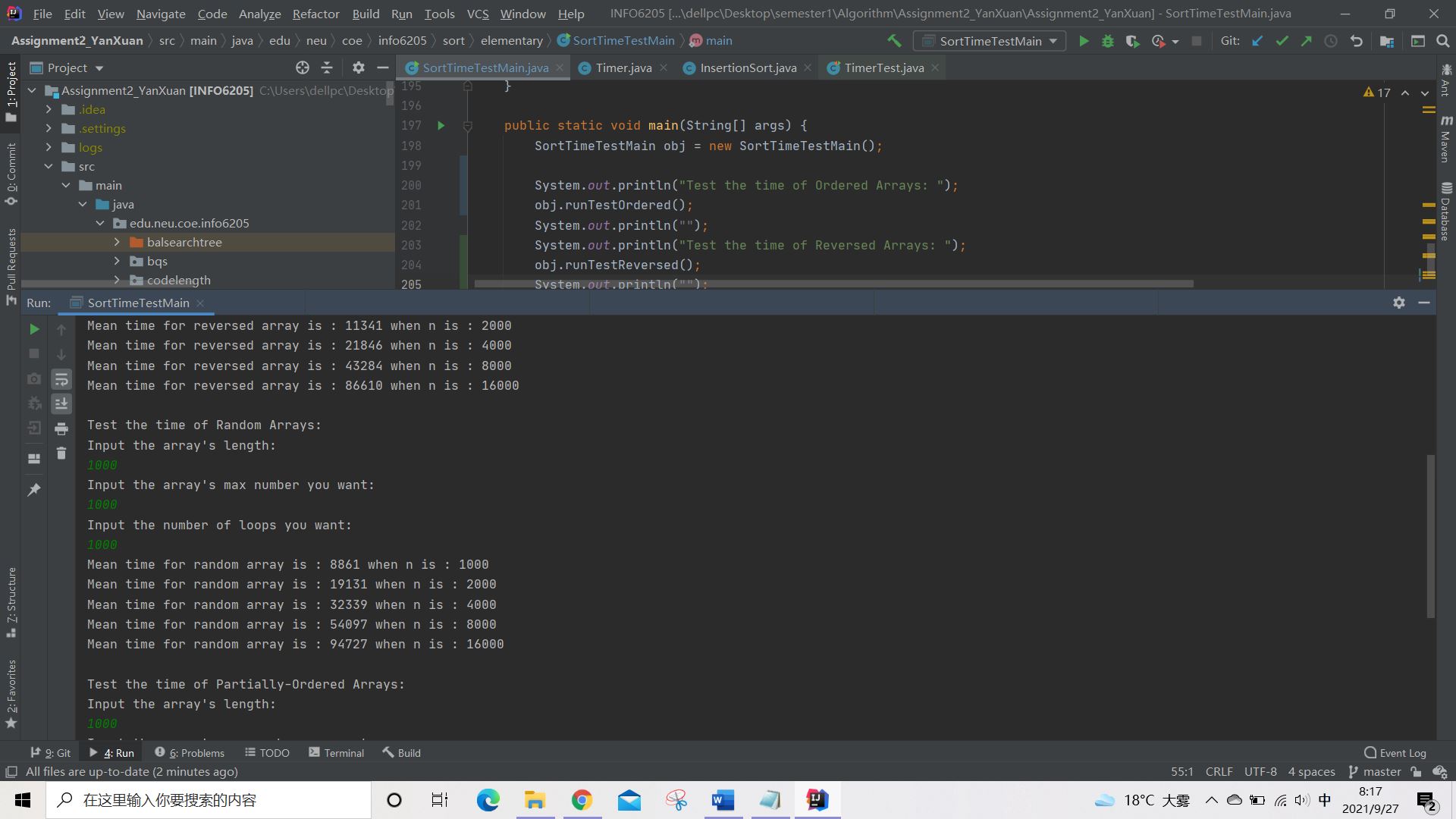
**\*The SortTimeTestMain.java and InsertionSort.java are in the same folder.**

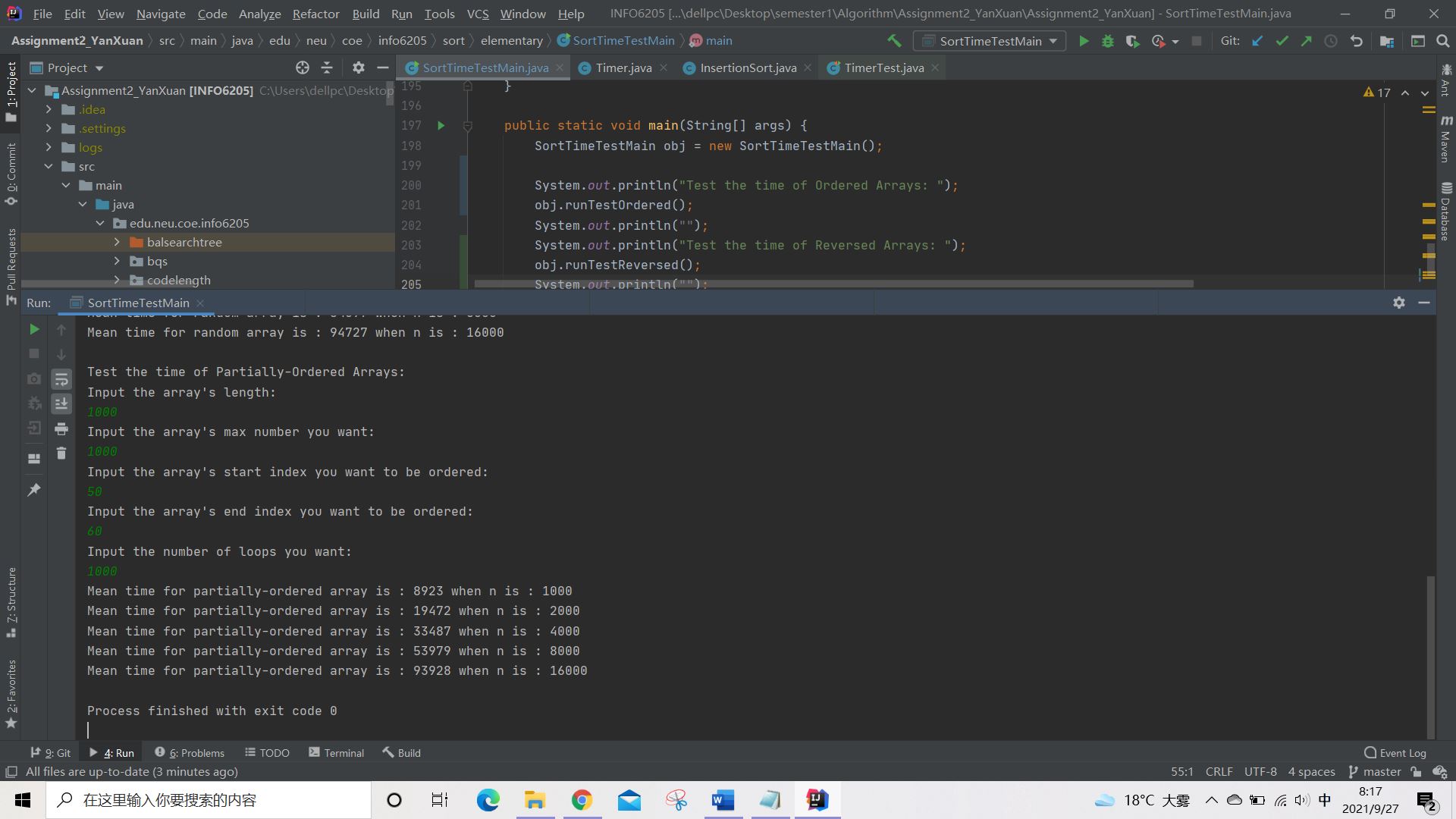
1. **Reason to create the SortTimeTestMain.java**

I firstly wrote the SortTimeTest.java in test and hoped to be able to input the array dynamically. But I then found that the IDEA’s configuration seems don’t support the dynamical input in their “Test.java”. So I created the main function instead.

1. **Unit tests result:(Snapshot of successful unit test run)**







Test the time of Ordered Arrays:

Input the array's length:

1000

Input the number of loops you want:

1000

Mean time for ordered array is : 4147 when n is : 1000

Mean time for ordered array is : 2834 when n is : 2000

Mean time for ordered array is : 4978 when n is : 4000

Mean time for ordered array is : 10924 when n is : 8000

Mean time for ordered array is : 18934 when n is : 16000

Test the time of Reversed Arrays:

Input the array's length:

1000

Input the number of loops you want:

1000

Mean time for reversed array is : 6041 when n is : 1000

Mean time for reversed array is : 11341 when n is : 2000

Mean time for reversed array is : 21846 when n is : 4000

Mean time for reversed array is : 43284 when n is : 8000

Mean time for reversed array is : 86610 when n is : 16000

Test the time of Random Arrays:

Input the array's length:

1000

Input the array's max number you want:

1000

Input the number of loops you want:

1000

Mean time for random array is : 8861 when n is : 1000

Mean time for random array is : 19131 when n is : 2000

Mean time for random array is : 32339 when n is : 4000

Mean time for random array is : 54097 when n is : 8000

Mean time for random array is : 94727 when n is : 16000

Test the time of Partially-Ordered Arrays:

Input the array's length:

1000

Input the array's max number you want:

1000

Input the array's start index you want to be ordered:

50

Input the array's end index you want to be ordered:

60

Input the number of loops you want:

1000

Mean time for partially-ordered array is : 8923 when n is : 1000

Mean time for partially-ordered array is : 19472 when n is : 2000

Mean time for partially-ordered array is : 33487 when n is : 4000

Mean time for partially-ordered array is : 53979 when n is : 8000

Mean time for partially-ordered array is : 93928 when n is : 16000

Process finished with exit code 0

1. **Test Design**

The functions: createReversed(int len), createRandom(int len, int max), createOrdered(int len) and createPartiallyOrdered(int len, int max, int start, int end) are to create required Integer arrays.

The functions: testReversed(Integer[] testReversed), testRandom(Integer[] testRandom), testOrdered(Integer[] testOrdered) and testPartiallyOrdered(Integer[] testPartiallyOrdered) are designed to accept the created array and call the InsertionSort.java to get the sort function, and count the time for the array’s each sort.

The functions: runTestReversed(), runTestRandom(), runTestOrdered() and runTestPartiallyOrdered() are designed to let us achieve the dynamical input and run several loops according to the input, increase the length of array(n) automatically, and count the mean time of sorting the array. Also, for each run, the functions run the given function ten times to get the system "warmed up".

1. **Graphical Representation(Observations from experiments should be tabulated and analyzed by plotting graphs(usually in excel) to arrive on the relationship conclusion)**

**Reversed Array:**



**Random Array:**



**Ordered Array:**



**Partially-Ordered Array:**



**Comparison:**

Note that the Random Array and the Partially-Ordered Array is overlapping because their times are too close.

1. **Conclusions**

According to the time complexity of the insertion sort algorithm , when the length of the array is doubled from n to 2n, the sorting time should be 4 times the time spent when the length of the array is n. It can be seen from the data obtained that some of the data did appear to quadruple the time (like in the Random Array when n becomes 4000 from 2000), but because of the randomness of the test, not all of them.

It can be seen that the time to sort the ordered array is still less than that of the random array. Moreover, it can be found that when the value of n is the same, the sort times of random array and partially-ordered array are very close. The reason may be that the length of the array used in this test is 1000, and the ordered length of partially-ordered array is set from index 50 to index of 60, the set of ordinal numbers is not very long.