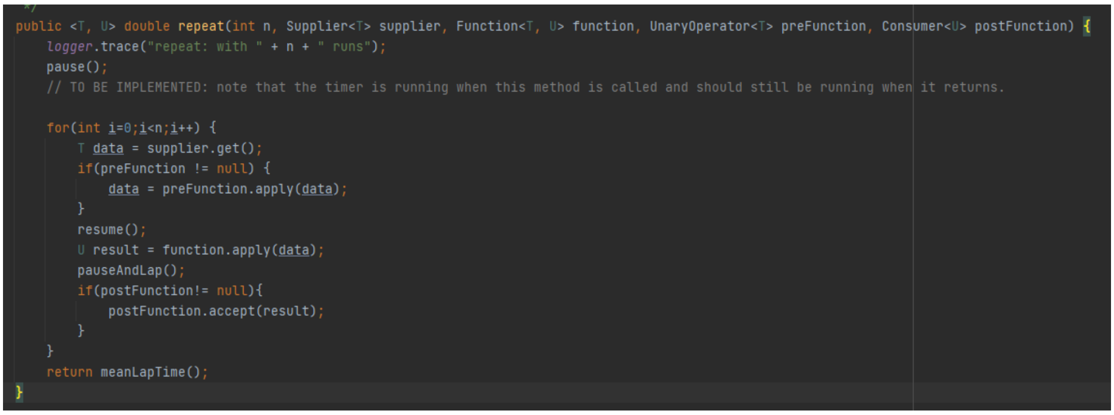
Student Name: **Jiachen Luo (001061582)**

INFO 6205
Program Structures & Algorithms
Spring 2021
Assignment No.2

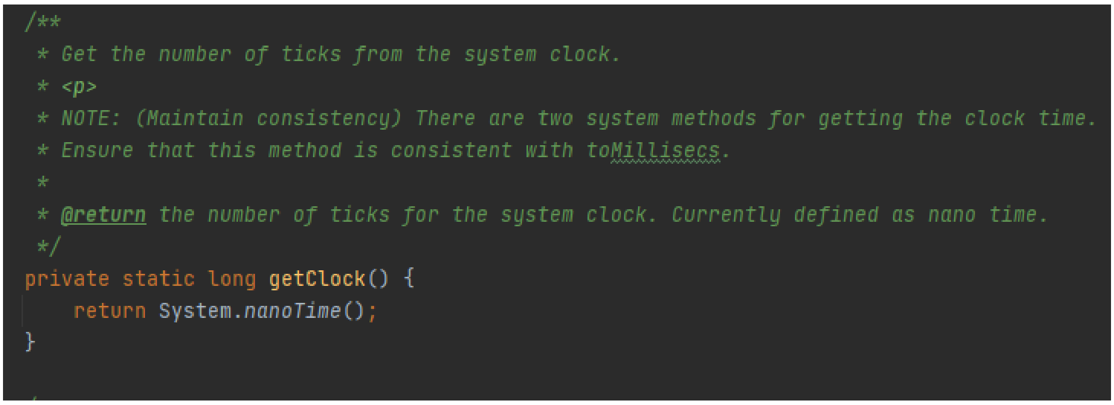
1. **Task:** Implement three methods of a class called Timer. Implement InsertionSort to run unit tests in InsertionSortTest. Implement a main program to actually run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered. Draw any conclusions from observations regarding the order of growth

2. **Output:**

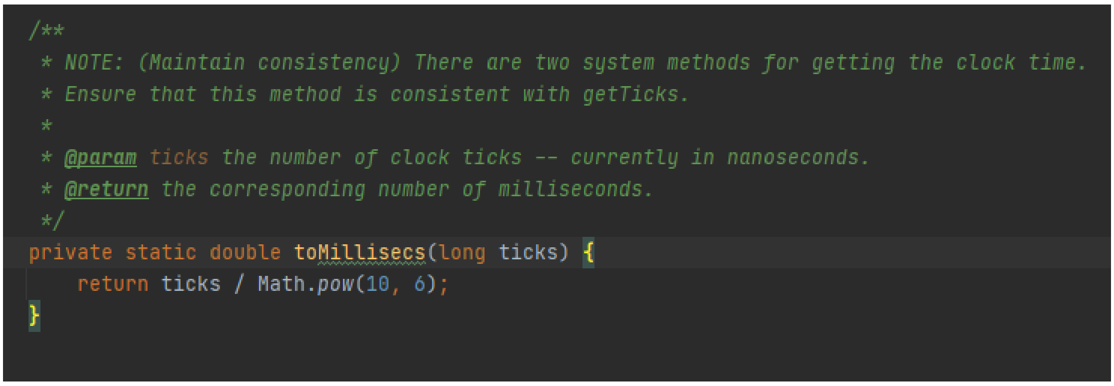
   Part1: The three methods



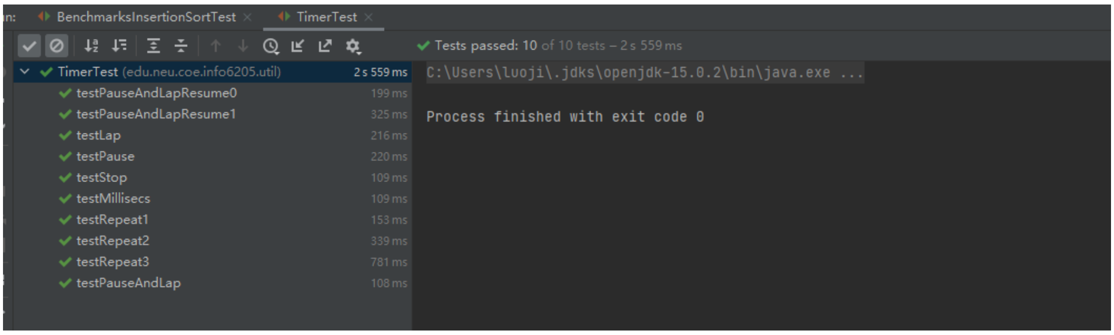
```java
*/
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> function, UnaryOperator<T> preFunction, Consumer<U> postFunction) {
    logger.trace("repeat: with " + n + " runs");
    pause();
    // TO BE IMPLEMENTED: note that the timer is running when this method is called and should still be running when it returns.

    for(int i=0;i<n;i++) {
        T data = supplier.get();
        if(preFunction != null) {
            data = preFunction.apply(data);
        }
        resume();
        U result = function.apply(data);
        pauseAndLap();
        if(postFunction!= null){
            postFunction.accept(result);
        }
    }
    return meanLapTime();
}
```



```java
/**
 * Get the number of ticks from the system clock.
 * <p>
 * NOTE: (Maintain consistency) There are two system methods for getting the clock time.
 * Ensure that this method is consistent with toMillisecs.
 *
 * @return the number of ticks for the system clock. Currently defined as nano time.
 */
private static long getClock() {
    return System.nanoTime();
}
```



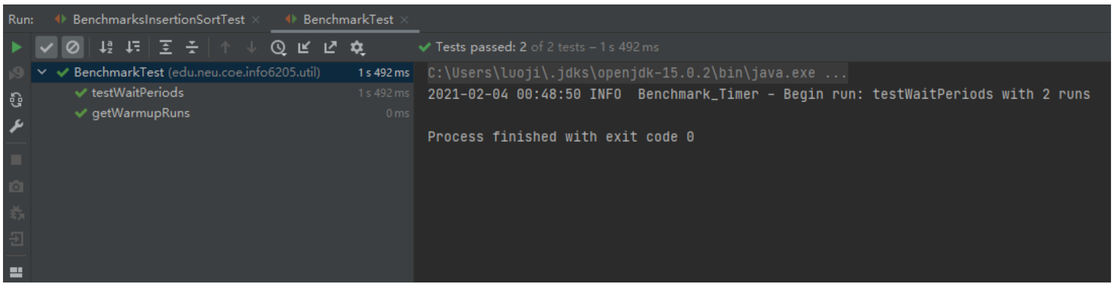
```java
/**
 * NOTE: (Maintain consistency) There are two system methods for getting the clock time.
 * Ensure that this method is consistent with getTicks.
 *
 * @param ticks the number of clock ticks -- currently in nanoseconds.
 * @return the corresponding number of milliseconds.
 */
private static double toMillisecs(long ticks) {
    return ticks / Math.pow(10, 6);
}
```



✓ Tests passed: 10 of 10 tests – 2 s 559 ms

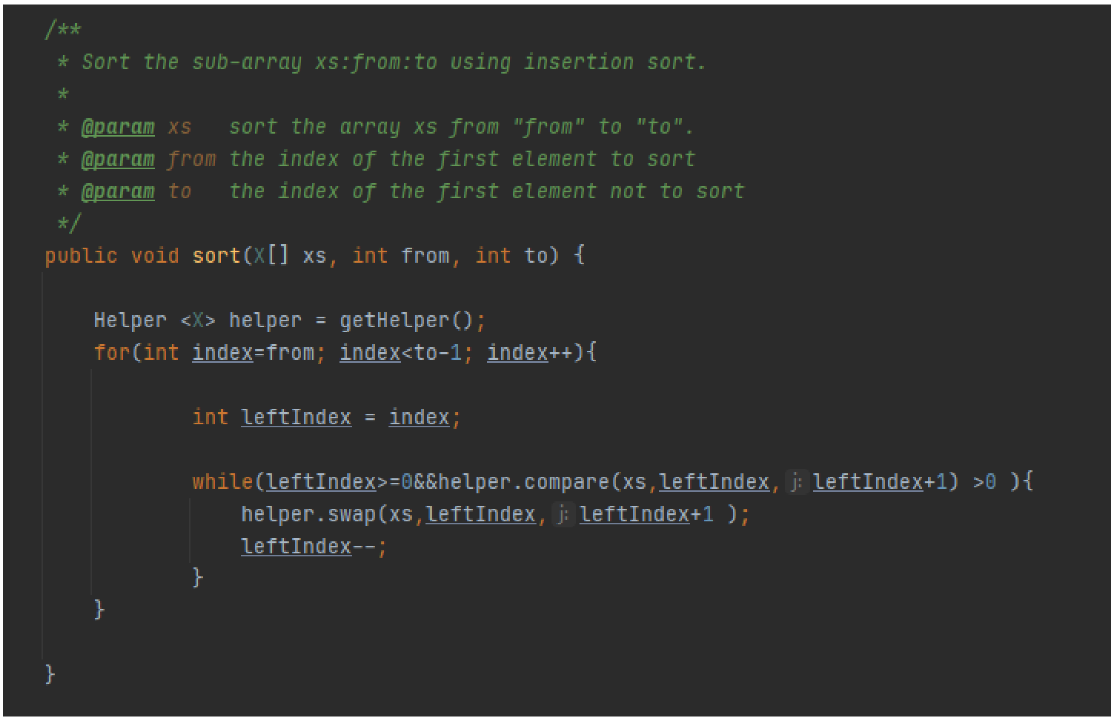
```
✓ TimerTest (edu.neu.coe.info6205.util)   2 s 559 ms
    ✓ testPauseAndLapResume0                199 ms
    ✓ testPauseAndLapResume1                325 ms
    ✓ testLap                               216 ms
    ✓ testPause                             220 ms
    ✓ testStop                              109 ms
    ✓ testMillisecs                         109 ms
    ✓ testRepeat1                           153 ms
    ✓ testRepeat2                           339 ms
    ✓ testRepeat3                           781 ms
    ✓ testPauseAndLap                       108 ms
```

```
C:\Users\luoji\.jdks\openjdk-15.0.2\bin\java.exe ...

Process finished with exit code 0
```

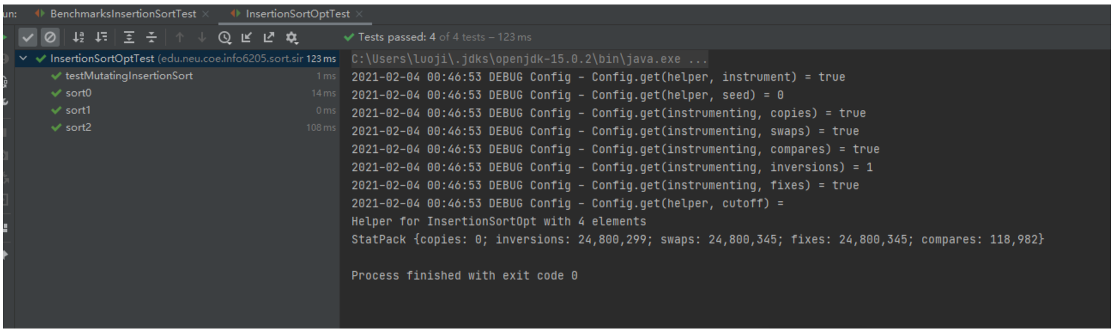

Run:
BenchmarksInsertionSortTest
BenchmarkTest
Tests passed: 2 of 2 tests – 1 s 492 ms
BenchmarkTest (edu.neu.coe.info6205.util) 1 s 492 ms
testWaitPeriods 1 s 492 ms
getWarmupRuns 0 ms

```
C:\Users\luoji\.jdks\openjdk-15.0.2\bin\java.exe ...
2021-02-04 00:48:50 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0
```

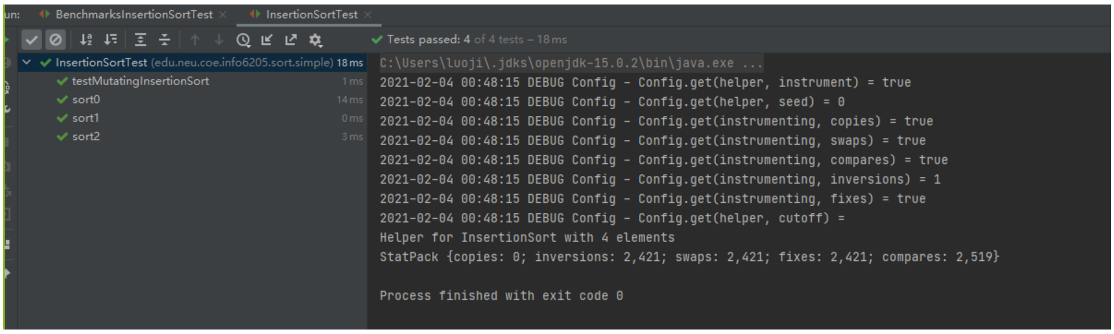
Part2: Insertion sort class



```
/**
 * Sort the sub-array xs:from:to using insertion sort.
 *
 * @param xs   sort the array xs from "from" to "to".
 * @param from the index of the first element to sort
 * @param to   the index of the first element not to sort
 */
public void sort(X[] xs, int from, int to) {

    Helper <X> helper = getHelper();
    for(int index=from; index<to-1; index++){

            int leftIndex = index;

            while(leftIndex>=0&&helper.compare(xs,leftIndex, j: leftIndex+1) >0 ){
                helper.swap(xs,leftIndex, j: leftIndex+1 );
                leftIndex--;
            }
    }

}
```

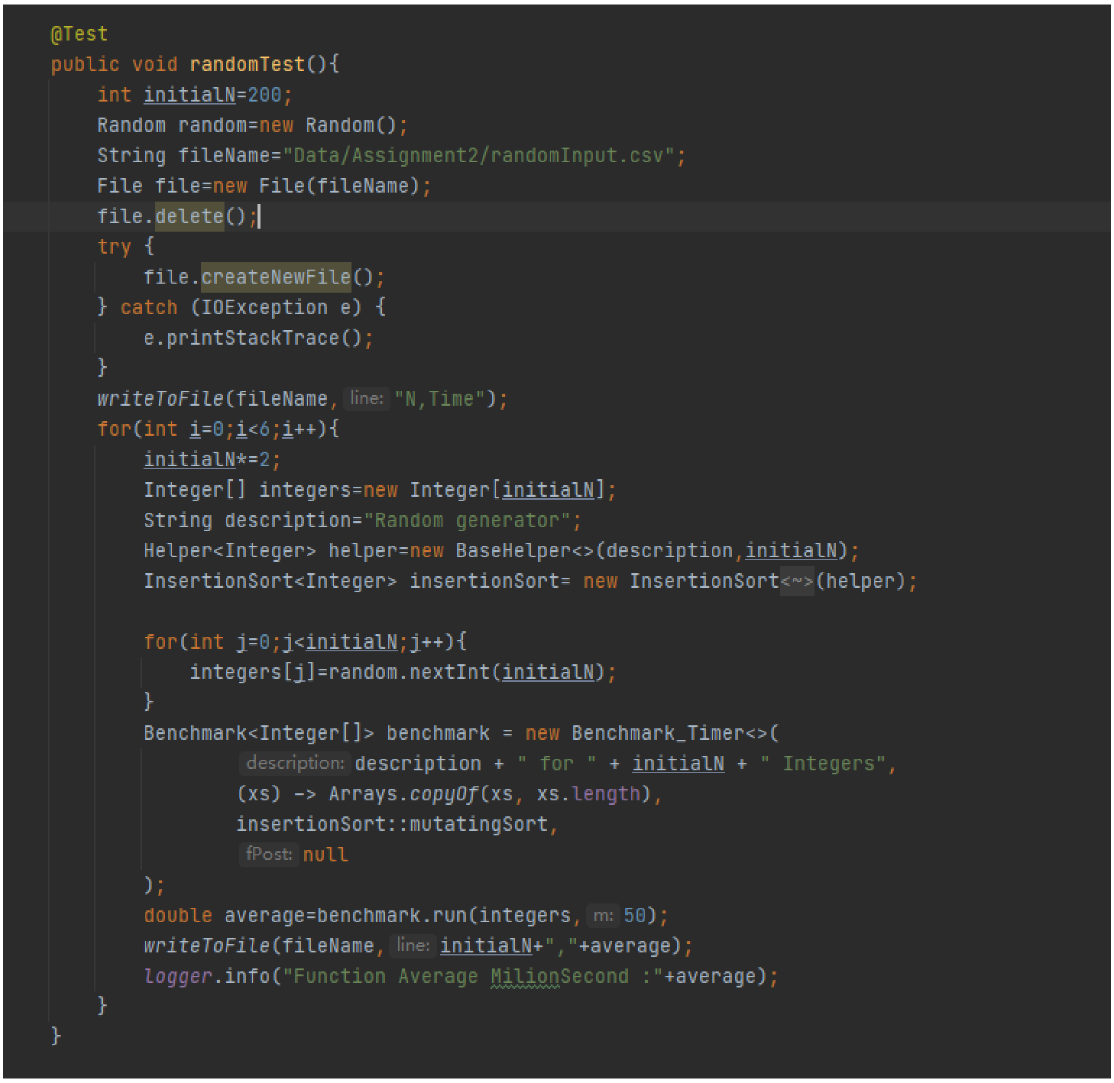

un:
BenchmarksInsertionSortTest
InsertionSortOptTest
Tests passed: 4 of 4 tests – 123 ms
InsertionSortOptTest (edu.neu.coe.info6205.sort.sir 123 ms
testMutatingInsertionSort 1 ms
sort0 14 ms
sort1 0 ms
sort2 108 ms

```
C:\Users\luoji\.jdks\openjdk-15.0.2\bin\java.exe ...
2021-02-04 00:46:53 DEBUG Config - Config.get(helper, instrument) = true
2021-02-04 00:46:53 DEBUG Config - Config.get(helper, seed) = 0
2021-02-04 00:46:53 DEBUG Config - Config.get(instrumenting, copies) = true
2021-02-04 00:46:53 DEBUG Config - Config.get(instrumenting, swaps) = true
2021-02-04 00:46:53 DEBUG Config - Config.get(instrumenting, compares) = true
2021-02-04 00:46:53 DEBUG Config - Config.get(instrumenting, inversions) = 1
2021-02-04 00:46:53 DEBUG Config - Config.get(instrumenting, fixes) = true
2021-02-04 00:46:53 DEBUG Config - Config.get(helper, cutoff) = 
Helper for InsertionSortOpt with 4 elements
StatPack {copies: 0; inversions: 24,800,299; swaps: 24,800,345; fixes: 24,800,345; compares: 118,982}

Process finished with exit code 0
```

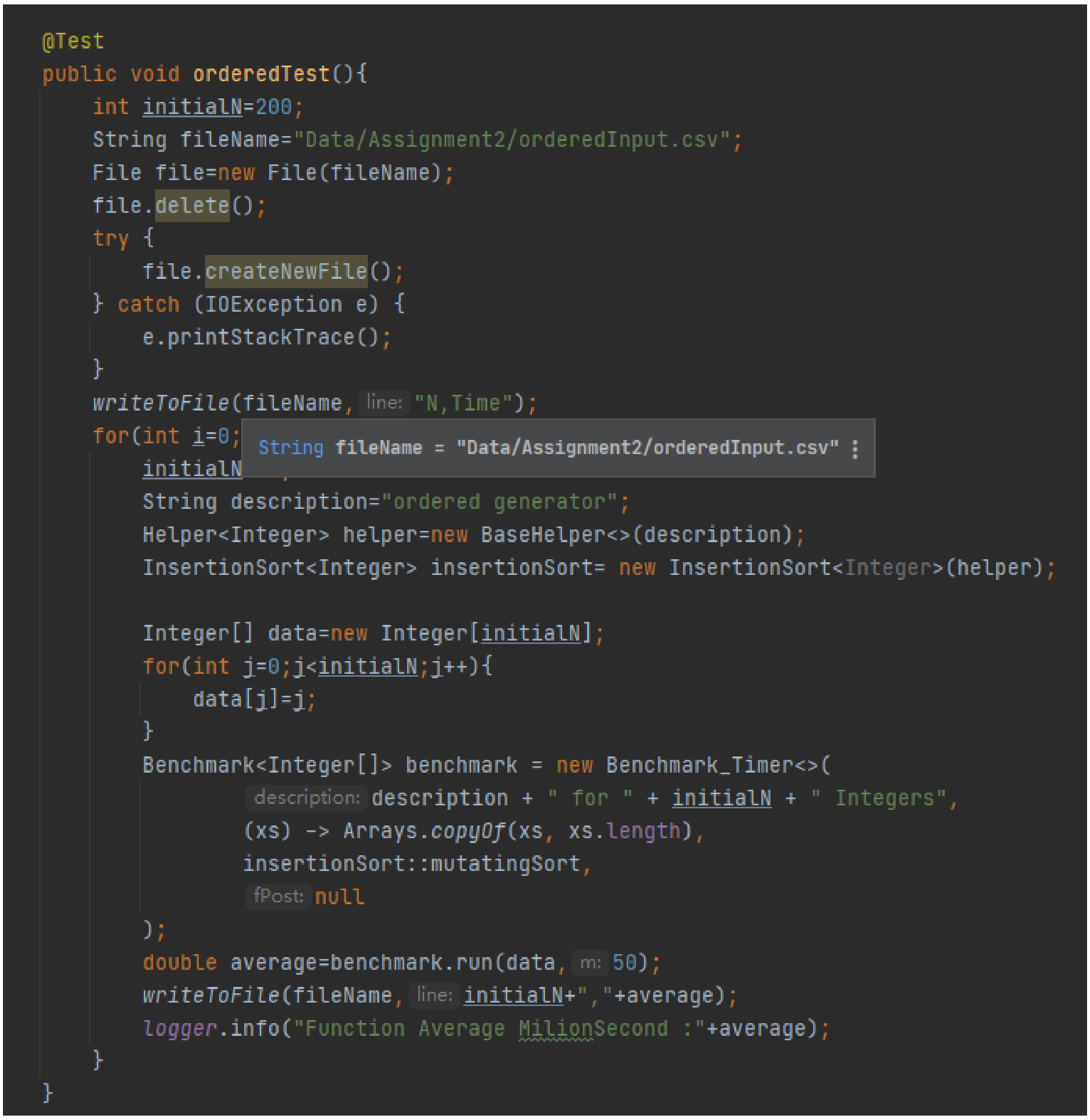

un:
BenchmarksInsertionSortTest
InsertionSortTest
Tests passed: 4 of 4 tests – 18 ms
InsertionSortTest (edu.neu.coe.info6205.sort.simple) 18 ms
testMutatingInsertionSort 1 ms
sort0 14 ms
sort1 0 ms
sort2 3 ms

```
C:\Users\luoji\.jdks\openjdk-15.0.2\bin\java.exe ...
2021-02-04 00:48:15 DEBUG Config - Config.get(helper, instrument) = true
2021-02-04 00:48:15 DEBUG Config - Config.get(helper, seed) = 0
2021-02-04 00:48:15 DEBUG Config - Config.get(instrumenting, copies) = true
2021-02-04 00:48:15 DEBUG Config - Config.get(instrumenting, swaps) = true
2021-02-04 00:48:15 DEBUG Config - Config.get(instrumenting, compares) = true
2021-02-04 00:48:15 DEBUG Config - Config.get(instrumenting, inversions) = 1
2021-02-04 00:48:15 DEBUG Config - Config.get(instrumenting, fixes) = true
2021-02-04 00:48:15 DEBUG Config - Config.get(helper, cutoff) =
Helper for InsertionSort with 4 elements
StatPack {copies: 0; inversions: 2,421; swaps: 2,421; fixes: 2,421; compares: 2,519}

Process finished with exit code 0
```

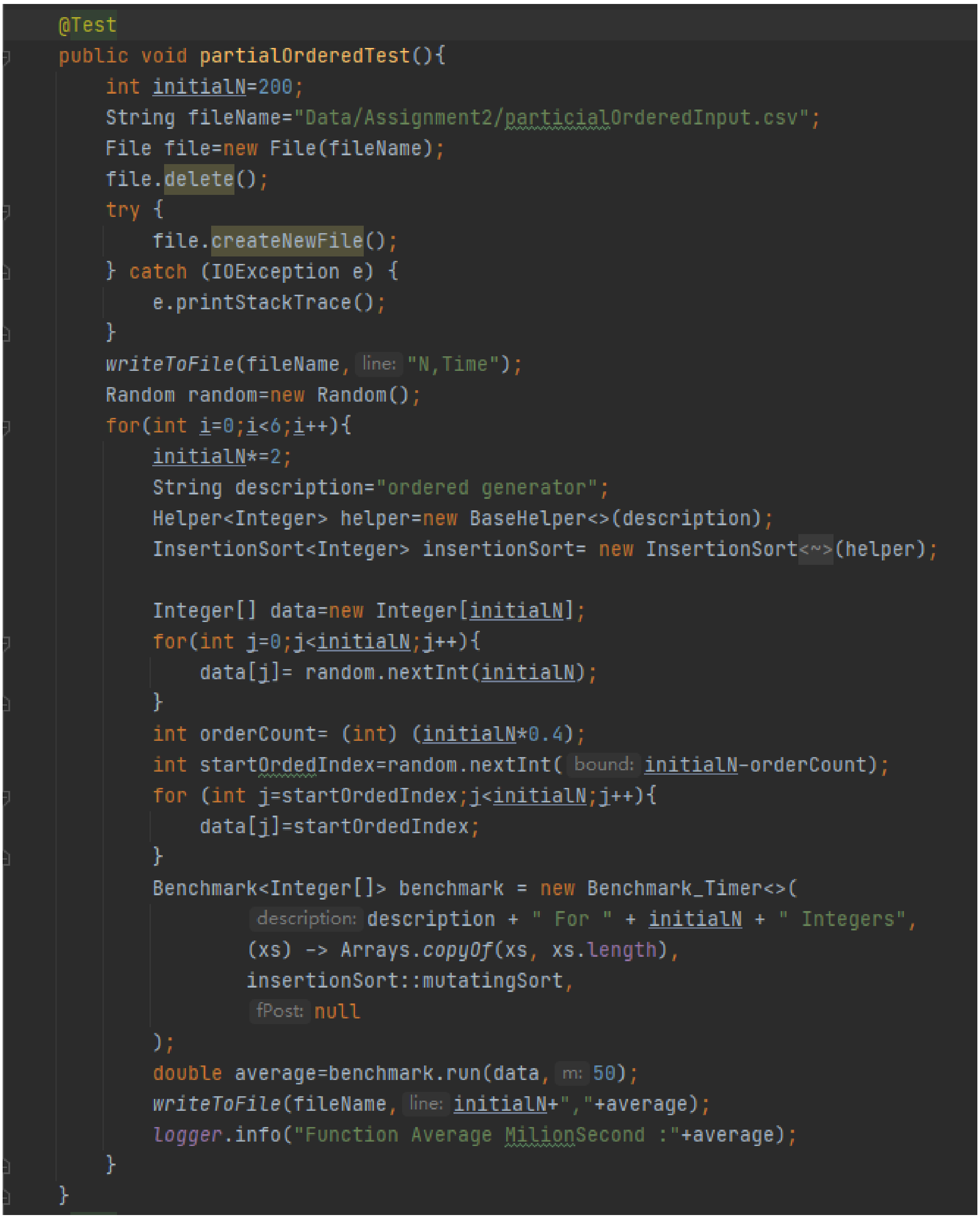
Part3: Implementation of unit test for insert sorting benchmark test on different input data



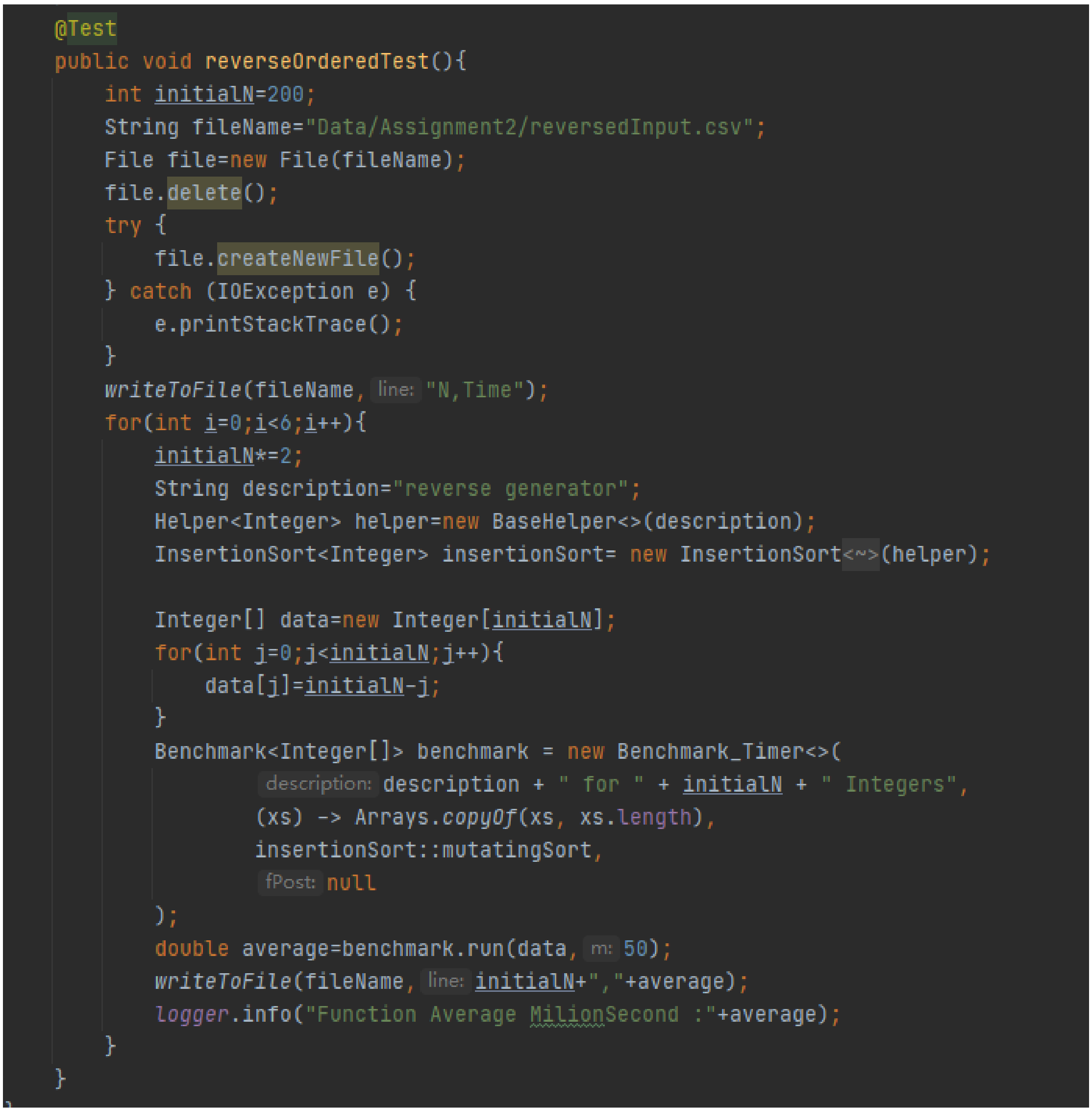
```
@Test
public void randomTest(){
    int initialN=200;
    Random random=new Random();
    String fileName="Data/Assignment2/randomInput.csv";
    File file=new File(fileName);
    file.delete();
    try {
        file.createNewFile();
    } catch (IOException e) {
        e.printStackTrace();
    }
    writeToFile(fileName, line: "N,Time");
    for(int i=0;i<6;i++){
        initialN*=2;
        Integer[] integers=new Integer[initialN];
        String description="Random generator";
        Helper<Integer> helper=new BaseHelper<>(description,initialN);
        InsertionSort<Integer> insertionSort= new InsertionSort<~>(helper);

        for(int j=0;j<initialN;j++){
            integers[j]=random.nextInt(initialN);
        }
        Benchmark<Integer[]> benchmark = new Benchmark_Timer<>(
                description: description + " for " + initialN + " Integers",
                (xs) -> Arrays.copyOf(xs, xs.length),
                insertionSort::mutatingSort,
                fPost: null
        );
        double average=benchmark.run(integers, m: 50);
        writeToFile(fileName, line: initialN+","+average);
        logger.info("Function Average MilionSecond :"+average);
    }
}
```



```java
@Test
public void orderedTest(){
    int initialN=200;
    String fileName="Data/Assignment2/orderedInput.csv";
    File file=new File(fileName);
    file.delete();
    try {
        file.createNewFile();
    } catch (IOException e) {
        e.printStackTrace();
    }
    writeToFile(fileName, line: "N,Time");
    for(int i=0;  String fileName = "Data/Assignment2/orderedInput.csv" ⋮
        initialN
        String description="ordered generator";
        Helper<Integer> helper=new BaseHelper<>(description);
        InsertionSort<Integer> insertionSort= new InsertionSort<Integer>(helper);

        Integer[] data=new Integer[initialN];
        for(int j=0;j<initialN;j++){
            data[j]=j;
        }
        Benchmark<Integer[]> benchmark = new Benchmark_Timer<>(
                description: description + " for " + initialN + " Integers",
                (xs) -> Arrays.copyOf(xs, xs.length),
                insertionSort::mutatingSort,
                fPost: null
        );
        double average=benchmark.run(data, m: 50);
        writeToFile(fileName, line: initialN+","+average);
        logger.info("Function Average MilionSecond :"+average);
    }
}
```

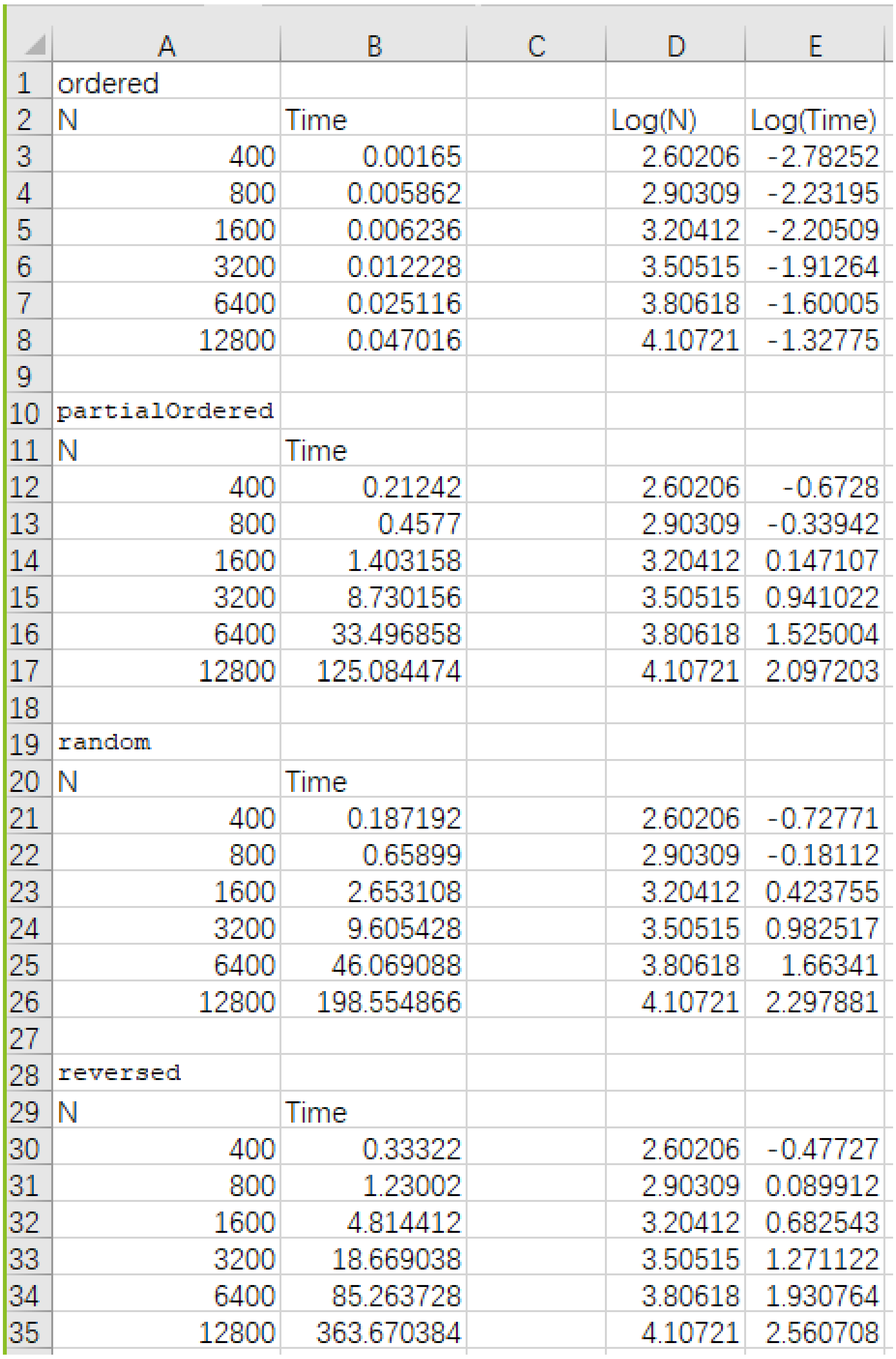


```
@Test
public void partialOrderedTest(){
    int initialN=200;
    String fileName="Data/Assignment2/particialOrderedInput.csv";
    File file=new File(fileName);
    file.delete();
    try {
        file.createNewFile();
    } catch (IOException e) {
        e.printStackTrace();
    }
    writeToFile(fileName, line: "N,Time");
    Random random=new Random();
    for(int i=0;i<6;i++){
        initialN*=2;
        String description="ordered generator";
        Helper<Integer> helper=new BaseHelper<>(description);
        InsertionSort<Integer> insertionSort= new InsertionSort<~>(helper);

        Integer[] data=new Integer[initialN];
        for(int j=0;j<initialN;j++){
            data[j]= random.nextInt(initialN);
        }
        int orderCount= (int) (initialN*0.4);
        int startOrdedIndex=random.nextInt( bound: initialN-orderCount);
        for (int j=startOrdedIndex;j<initialN;j++){
            data[j]=startOrdedIndex;
        }
        Benchmark<Integer[]> benchmark = new Benchmark_Timer<>(
                description: description + " For " + initialN + " Integers",
                (xs) -> Arrays.copyOf(xs, xs.length),
                insertionSort::mutatingSort,
                fPost: null
        );
        double average=benchmark.run(data, m: 50);
        writeToFile(fileName, line: initialN+","+average);
        logger.info("Function Average MilionSecond :"+average);
    }
}
```

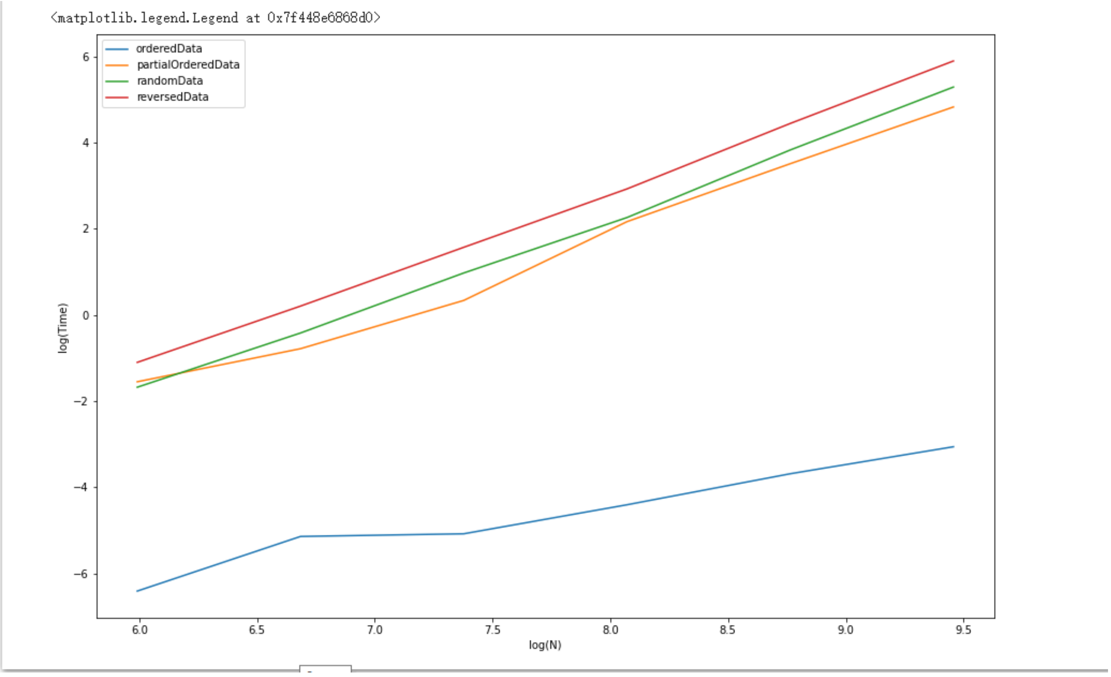


```
    @Test
    public void reverseOrderedTest(){
        int initialN=200;
        String fileName="Data/Assignment2/reversedInput.csv";
        File file=new File(fileName);
        file.delete();
        try {
            file.createNewFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
        writeToFile(fileName, line: "N,Time");
        for(int i=0;i<6;i++){
            initialN*=2;
            String description="reverse generator";
            Helper<Integer> helper=new BaseHelper<>(description);
            InsertionSort<Integer> insertionSort= new InsertionSort<~>(helper);

            Integer[] data=new Integer[initialN];
            for(int j=0;j<initialN;j++){
                data[j]=initialN-j;
            }
            Benchmark<Integer[]> benchmark = new Benchmark_Timer<>(
                    description: description + " for " + initialN + " Integers",
                    (xs) -> Arrays.copyOf(xs, xs.length),
                    insertionSort::mutatingSort,
                    fPost: null
            );
            double average=benchmark.run(data, m: 50);
            writeToFile(fileName, line: initialN+","+average);
            logger.info("Function Average MilionSecond :"+average);
        }
    }
```

Conclusion and Evidance:



	A	B	C	D	E
1	ordered				
2	N	Time		Log(N)	Log(Time)
3	400	0.00165		2.60206	-2.78252
4	800	0.005862		2.90309	-2.23195
5	1600	0.006236		3.20412	-2.20509
6	3200	0.012228		3.50515	-1.91264
7	6400	0.025116		3.80618	-1.60005
8	12800	0.047016		4.10721	-1.32775
9					
10	partialOrdered				
11	N	Time			
12	400	0.21242		2.60206	-0.6728
13	800	0.4577		2.90309	-0.33942
14	1600	1.403158		3.20412	0.147107
15	3200	8.730156		3.50515	0.941022
16	6400	33.496858		3.80618	1.525004
17	12800	125.084474		4.10721	2.097203
18					
19	random				
20	N	Time			
21	400	0.187192		2.60206	-0.72771
22	800	0.65899		2.90309	-0.18112
23	1600	2.653108		3.20412	0.423755
24	3200	9.605428		3.50515	0.982517
25	6400	46.069088		3.80618	1.66341
26	12800	198.554866		4.10721	2.297881
27					
28	reversed				
29	N	Time			
30	400	0.33322		2.60206	-0.47727
31	800	1.23002		2.90309	0.089912
32	1600	4.814412		3.20412	0.682543
33	3200	18.669038		3.50515	1.271122
34	6400	85.263728		3.80618	1.930764
35	12800	363.670384		4.10721	2.560708



```
<matplotlib.legend.Legend at 0x7f448e6868d0>
```

Conclusion: Different type of input data always lead to different time consumption. According to the graph, the degree of time consumption is reserved_data, random_data, partial_ordered_data and ordered_data. At the same time, it can be seen that when using Ordered_data, the cost of time is significantly smaller than the other three types. In addition, in this log-logplot, the size of the input data has a linear relationship with the logarithm of the running time cost.