

INFO 6205
Program Structures & Algorithms
Fall 2020
Assignment No.2

- **Task: InsertionSort**
- **Output : $T=(N)^2$**
- **Relationship conclusion**

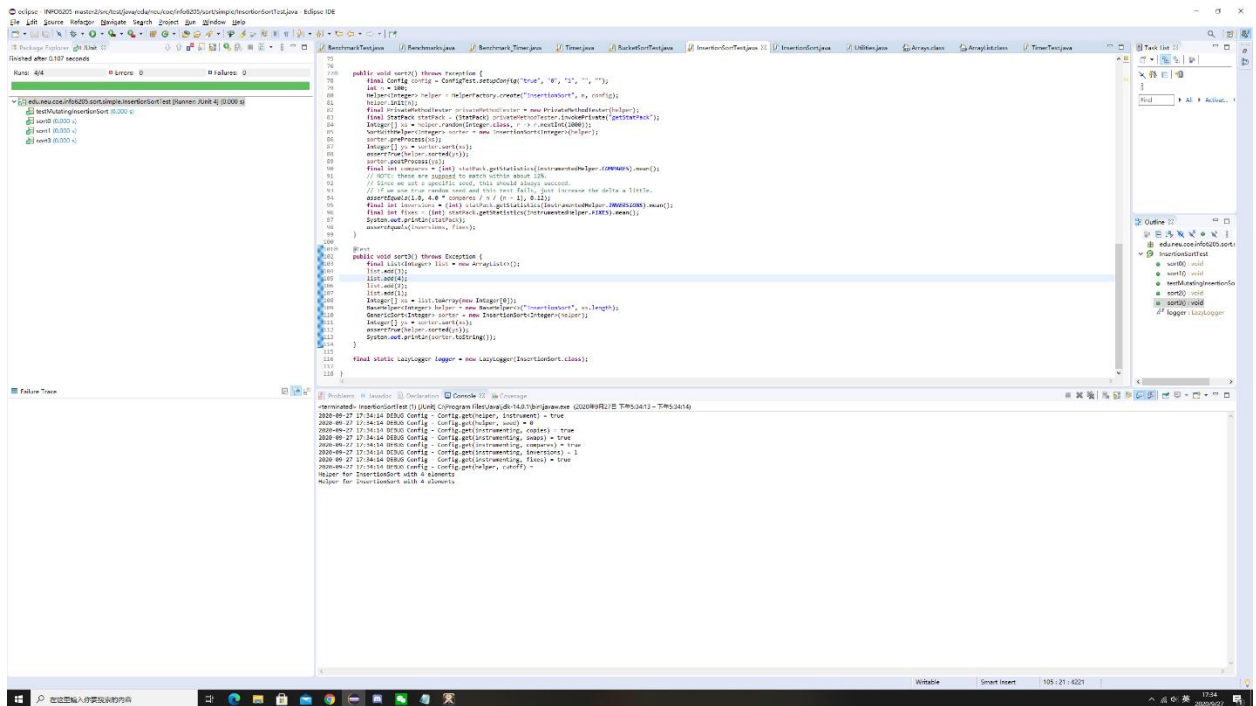
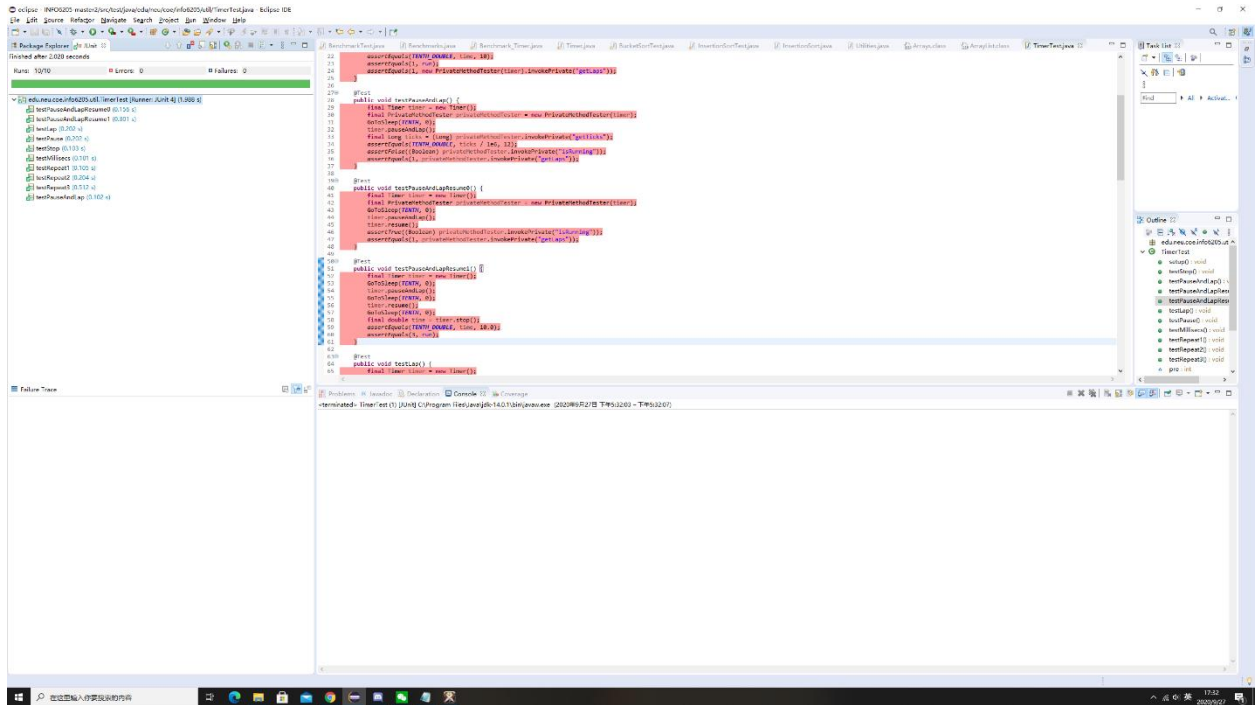
Through the experimental analysis, after 1000 repeated experiments of insertion sorting method to eliminate the systematic error of the computer, the experimental results show that in the worst case, that is, in the case of reverse order, every time n is doubled, t increases nearly 4 times, so the worst case conforms to the time complexity $O(n^2)$. In the average case, that is, in Random's experiment, every time n is doubled, t increases nearly 4 times, which means that the average time complexity $O(n^2)$ is also met. The time complexity $O(n^2)$ is also obeyed in partial order experiments. In addition, in the ideal case, that is, in the ordered experiment, every time n doubles, t doubles, which is in line with the ideal time complexity $O(n)$.

- **Evidence to support relationship**

Insert sort is executed 1000 times

N	Ordered	Partially-Ordered	Random	Reverse-Ordered
500	0.004	0.09	0.352	0.661
1000	0.006	0.646	1.343	2.629
2000	0.01	1.337	5.361	10.597
4000	0.02	5.392	21.715	42.374
8000	0.048	23.504	89.367	169.226

• Screenshot of Unit test passing



Student Name (NUID: Your Id)

The screenshot shows the Eclipse IDE with the `BenchmarkTest.java` file open. The code defines a `BenchmarkTest` class with methods for running benchmarks and getting the average time. The `main` method calls `runBenchmarks` and `getAverageTime`. The `runBenchmarks` method uses `testBenchmarks` to run the benchmarks. The `getAverageTime` method returns the average time in milliseconds.

```
package edu.nyu.cs.infoc2020;

import java.util.ArrayList;

public class BenchmarkTest {
    // ... (code for running benchmarks and getting average time) ...
}
```

The `runBenchmarks` method is called with `testBenchmarks` and `1000` as arguments. The `getAverageTime` method is called with `testBenchmarks` as an argument. The output of the program is shown in the `Console` view, indicating that the benchmarks were run successfully and the average time was calculated.

The screenshot shows the Eclipse IDE with the `Benchmark_Timer.java` file open. The code defines a `Benchmark_Timer` class that implements the `BenchmarkTest` interface. It uses a `HashMap` to store the results of the benchmarks. The `runBenchmarks` method is implemented by calling `testBenchmarks` and storing the results in the `HashMap`. The `getAverageTime` method is implemented by calculating the average time from the results in the `HashMap`.

```
package edu.nyu.cs.infoc2020;

import java.util.HashMap;
import java.util.ArrayList;

public class Benchmark_Timer {
    // ... (code for running benchmarks and getting average time) ...
}
```

The `runBenchmarks` method is called with `testBenchmarks` and `1000` as arguments. The `getAverageTime` method is called with `testBenchmarks` as an argument. The output of the program is shown in the `Console` view, indicating that the benchmarks were run successfully and the average time was calculated.

The main of the test is in the benchmark_Timer