Program Structures and Algorithms
Spring 2023(SEC –8)

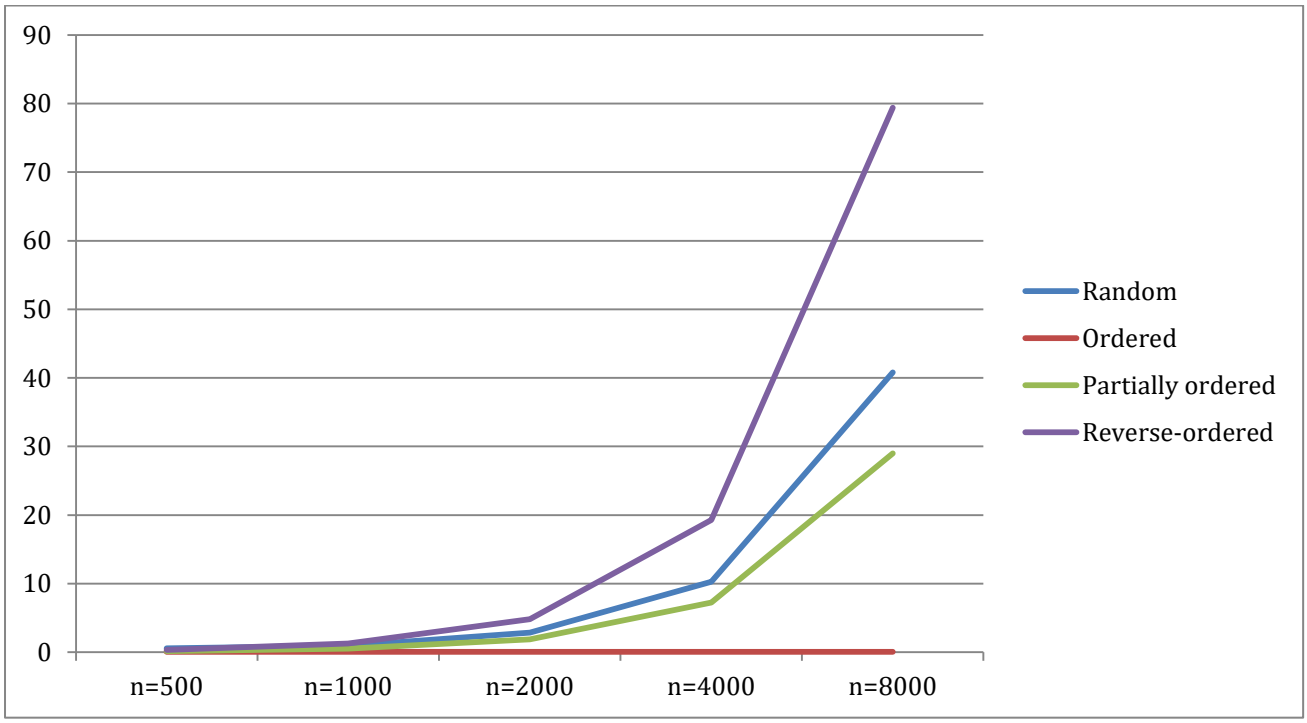
NAME: Junlong Qiao
NUID: 002784609

Task: Assignment 3(Benchmark)

Implement three methods of a class called Timer.
Implement InsertionSort
Implement a class to test insertion sort in different case.

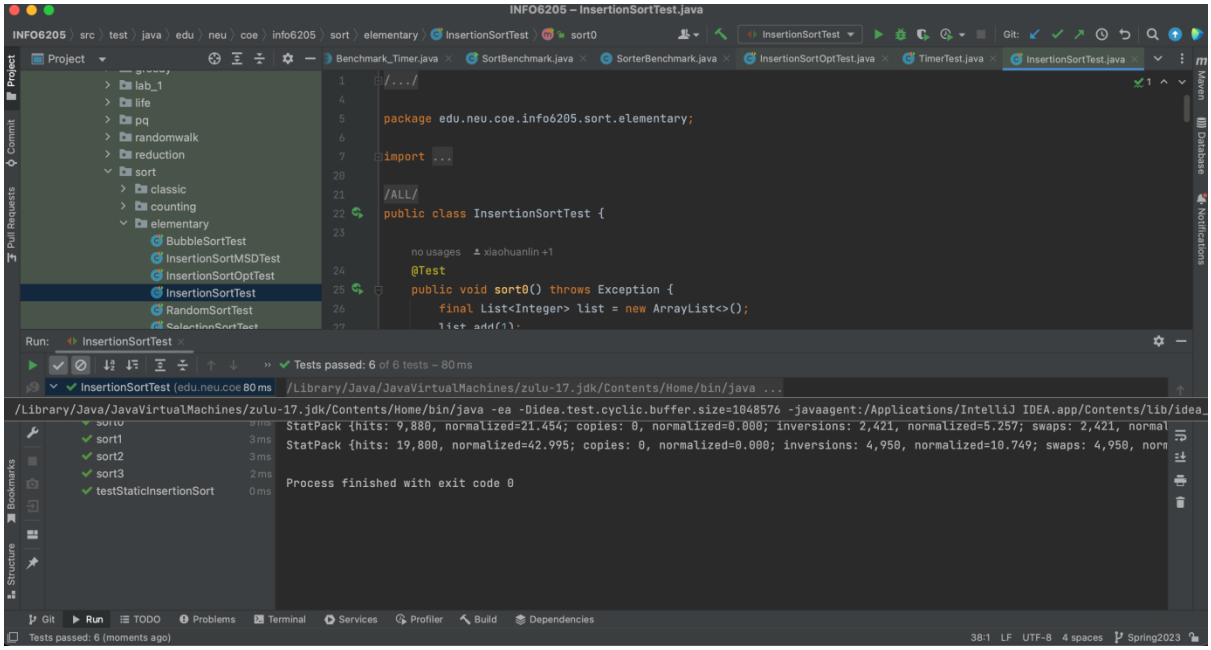
Runtie Relationship Conclusion:

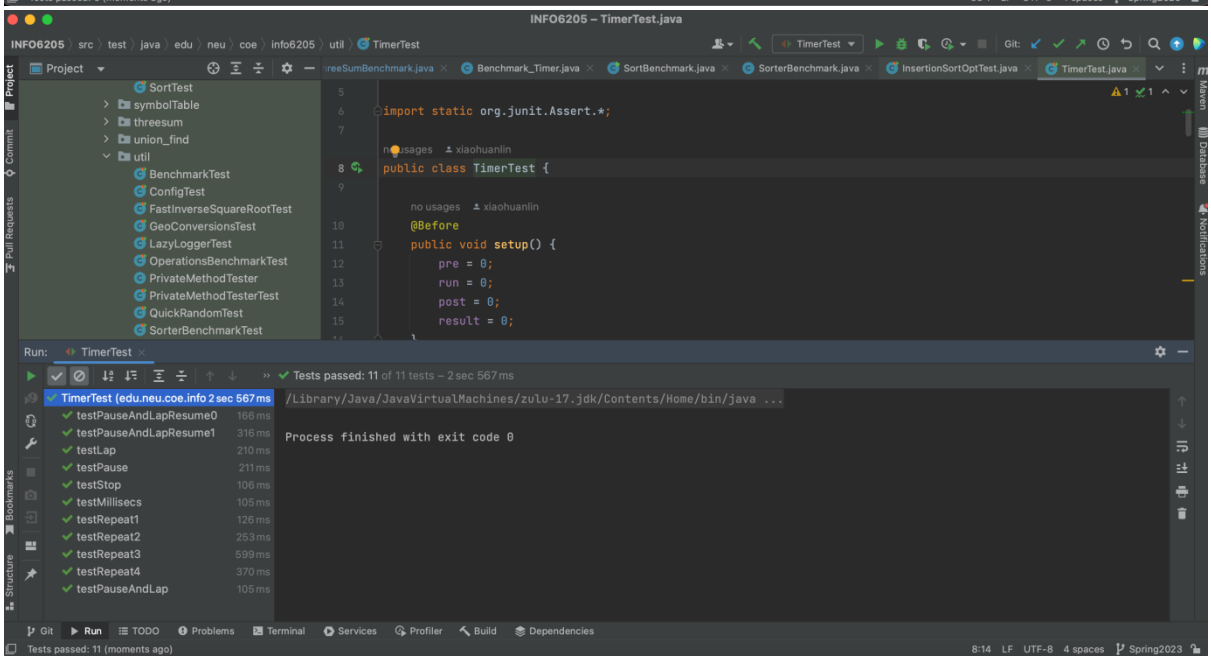
Time\Array	Random	Ordered	Partially ordered	Reverse-ordered
n=500	0.590	0.054	0.171	0.355
n=1000	0.965	0.056	0.518	1.280
n=2000	2.851	0.057	1.888	4.825
n=4000	10.288	0.079	7.270	19.292
n=8000	40.793	0.073	29.004	79.372



The more reverse-ordered the array, the longer the sorting time, and when the array is an ordered array, the sorting time grows close to linear

Unit Test Screenshots:





Code of 3-Sum:

ThreeSumQuadratic.java

```
package edu.neu.coe.info6205.sort.elementary;


import edu.neu.coe.info6205.util.Benchmark_Timer;

import java.util.Arrays;

import java.util.Collections;
import java.util.Random;
import java.util.function.Consumer;
```

```
import java.util.function.Supplier;

public class InsertionSortBenchMark {
    private final int runs;
    private final int n;
    private final Supplier<Integer[]> supplier;
    private final Consumer<Integer[]> consumer;


    public InsertionSortBenchMark(int runs, int n, int type) {
        this.runs = runs;
        this.n = n;
        if(type==0) {
            this.supplier = new Supplier<Integer[]>() {
                @Override
                public Integer[] get() {
                    return randomArrayGenerator(n);
                }

                private Integer[] randomArrayGenerator(int n) {
                    Random random = new Random();
                    Integer[] array = new Integer[n];
                    for (int i = 0; i < array.length; i++) {
                        array[i] = random.nextInt();
                    }
                    return array;
                }
            };
        }else if(type==1){
            this.supplier = new Supplier<Integer[]>() {
                @Override
                public Integer[] get() {
                    return randomArrayGenerator(n);
                }

                private Integer[] randomArrayGenerator(int n) {
                    Random random = new Random();
                    Integer[] array = new Integer[n];
                    for (int i = 0; i < array.length; i++) {
                        array[i] = random.nextInt();
                    }
                    Arrays.sort(array);
                    return array;
                }
            };
        }else if(type==2){
            this.supplier = new Supplier<Integer[]>() {
                @Override
```

```
            public Integer[] get() {
                return randomArrayGenerator(n);
            }

            private Integer[] randomArrayGenerator(int n) {
                Random random = new Random();
                Integer[] array = new Integer[n];
                for (int i = 0; i < array.length; i++) {
                    array[i] = random.nextInt();
                }
                InsertionSort insertionSort = new InsertionSort<>();
                insertionSort.sort(array, 0, array.length/2);
                return array;
            }
        };
    }else{
        this.supplier = new Supplier<Integer[]>() {
            @Override
            public Integer[] get() {
                return randomArrayGenerator(n);
            }

            private Integer[] randomArrayGenerator(int n) {
                Random random = new Random();
                Integer[] array = new Integer[n];
                for (int i = 0; i < array.length; i++) {
                    array[i] = random.nextInt();
                }
                Arrays.sort(array, Collections.reverseOrder());
                return array;
            }
        };
    }
    this.consumer = new Consumer<Integer[]>() {
        @Override
        public void accept(Integer[] integers) {
            InsertionSort insertionSort = new InsertionSort<>();
            insertionSort.sort(integers, 0, integers.length);

        }
    };
}

public void runBenchMarks(int type) {
    switch (type){
        case 0:
            System.out.println("Random");
            break;
```

```
            case 1:
                System.out.println("Ordered");
                break;
            case 2:
                System.out.println("Partially-ordered");
                break;
            case 3:
                System.out.println("Reverse-ordered");
                break;
        }
        System.out.println("InsertionSort: N = " + n);
        Benchmark_Timer timer = new Benchmark_Timer<>("InsertionSortBenchmark",
consumer);
        System.out.println("Average time : " + timer.runFromSupplier(supplier, runs));
    }

    public static void main(String[] args) {
        new InsertionSortBenchMark(100, 500,0).runBenchMarks(0);
        new InsertionSortBenchMark(100, 1000,0).runBenchMarks(0);
        new InsertionSortBenchMark(100, 2000,0).runBenchMarks(0);
        new InsertionSortBenchMark(100, 4000,0).runBenchMarks(0);
        new InsertionSortBenchMark(100, 8000,0).runBenchMarks(0);
        new InsertionSortBenchMark(100, 500,1).runBenchMarks(1);
        new InsertionSortBenchMark(100, 1000,1).runBenchMarks(1);
        new InsertionSortBenchMark(100, 2000,1).runBenchMarks(1);
        new InsertionSortBenchMark(100, 4000,1).runBenchMarks(1);
        new InsertionSortBenchMark(100, 8000,1).runBenchMarks(1);
        new InsertionSortBenchMark(100, 500,2).runBenchMarks(2);
        new InsertionSortBenchMark(100, 1000,2).runBenchMarks(2);
        new InsertionSortBenchMark(100, 2000,2).runBenchMarks(2);
        new InsertionSortBenchMark(100, 4000,2).runBenchMarks(2);
        new InsertionSortBenchMark(100, 8000,2).runBenchMarks(2);
        new InsertionSortBenchMark(100, 500,3).runBenchMarks(3);
        new InsertionSortBenchMark(100, 1000,3).runBenchMarks(3);
        new InsertionSortBenchMark(100, 2000,3).runBenchMarks(3);
        new InsertionSortBenchMark(100, 4000,3).runBenchMarks(3);
        new InsertionSortBenchMark(100, 8000,3).runBenchMarks(3);
    }
}
```