Venkata Sai Akhil Vemula (002981289)

Program Structures & Algorithms
Fall 2021
Assignment No. 2

A) Tasks Performed:

1) Implemented Methods in Timer.java

```
/**
 * Pause (without counting a lap); run the given functions n times while b
 *
 * @param n            the number of repetitions.
 * @param supplier     a function which supplies a T value.
 * @param function     a function T=>U and which is to be timed.
 * @param preFunction  a function which pre-processes a T value and which
 * @param postFunction a function which consumes a U and which succeeds th
 * @return the average milliseconds per repetition.
 */
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> fu
    logger.trace("repeat: with " + n + " runs");
    // TO BE IMPLEMENTED: note that the timer is running when this method
    return 0;
}

/**
 * Stop this Timer and return the mean lap time in milliseconds.
 *
 * @return the average milliseconds used by each lap.
 * @throws TimerException if this Timer is not running.
 */
public double stop() {
    pauseAndLap();
    return meanLapTime();
}

/**
 * Return the mean lap time in milliseconds for this paused timer.
 *
 * @return the average milliseconds used by each lap.
```

```
/**
 * Pause (without counting a lap); run the given functions n times while bei
 *
 * @param n            the number of repetitions.
 * @param supplier     a function which supplies a T value.
 * @param function     a function T=>U and which is to be timed.
 * @param preFunction  a function which pre-processes a T value and which p
 * @param postFunction a function which consumes a U and which succeeds the
 * @return the average milliseconds per repetition.
 */
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> func
    logger.trace("repeat: with " + n + " runs");
    pause();
    for (int i = 0; i < n; i++) {
        if (preFunction!=null)preFunction.apply(supplier.get());
        resume();
        U t = function.apply(supplier.get());
        pauseAndLap();
        if (postFunction!=null) postFunction.accept(t);
    }
    return meanLapTime();
}

/**
 * Stop this Timer and return the mean lap time in milliseconds.
 *
 * @return the average milliseconds used by each lap.
 * @throws TimerException if this Timer is not running.
 */
public double stop() {
    pauseAndLap();
```

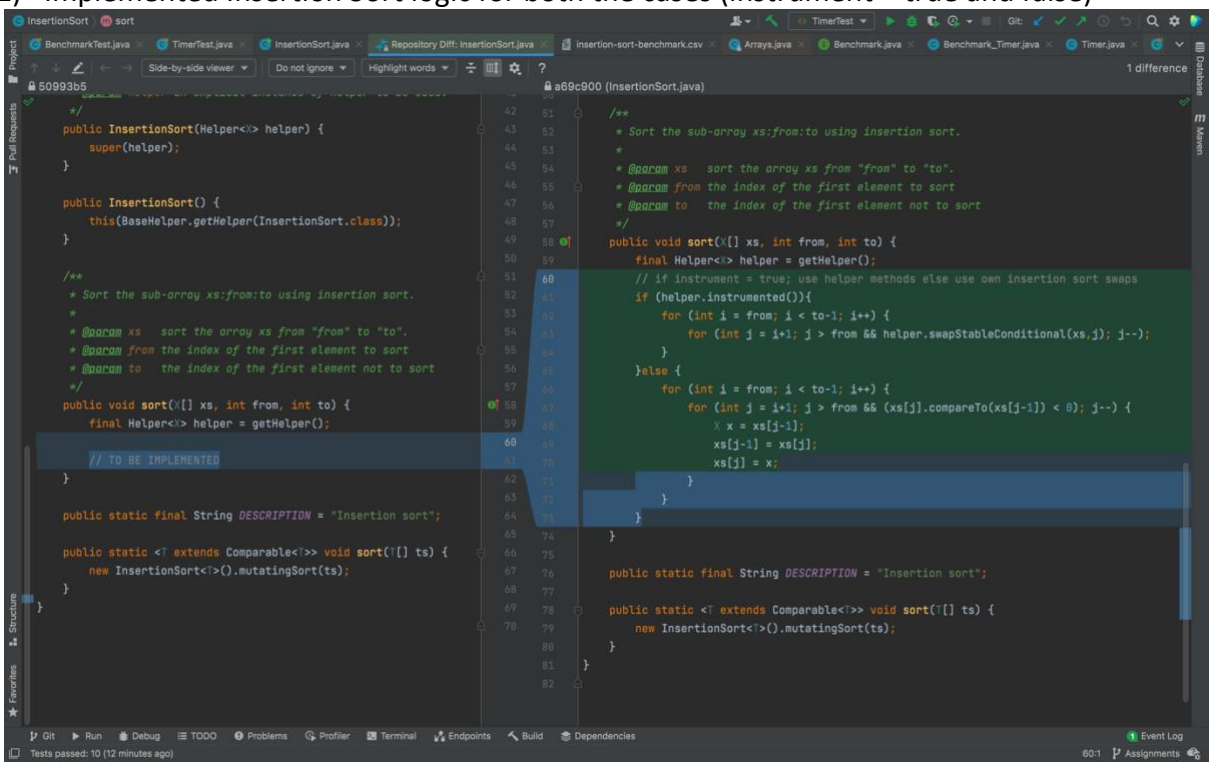
c25ea2c

```
/**
 * Get the number of ticks from the system clock.
 * <p>
 * NOTE: (Maintain consistency) There are two system methods for getting t
 * Ensure that this method is consistent with toMillisecs.
 *
 * @return the number of ticks for the system clock. Currently defined as
 */
private static long getClock() {
    // TO BE IMPLEMENTED
    return 0;
}

/**
 * NOTE: (Maintain consistency) There are two system methods for getting t
 * 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) {
    // TO BE IMPLEMENTED
    return 0;
}

final static LazyLogger logger = new LazyLogger(Timer.class);

static class TimerException extends RuntimeException {
    public TimerException() {
    }

    public TimerException(String message) {
        super(message);
    }
```

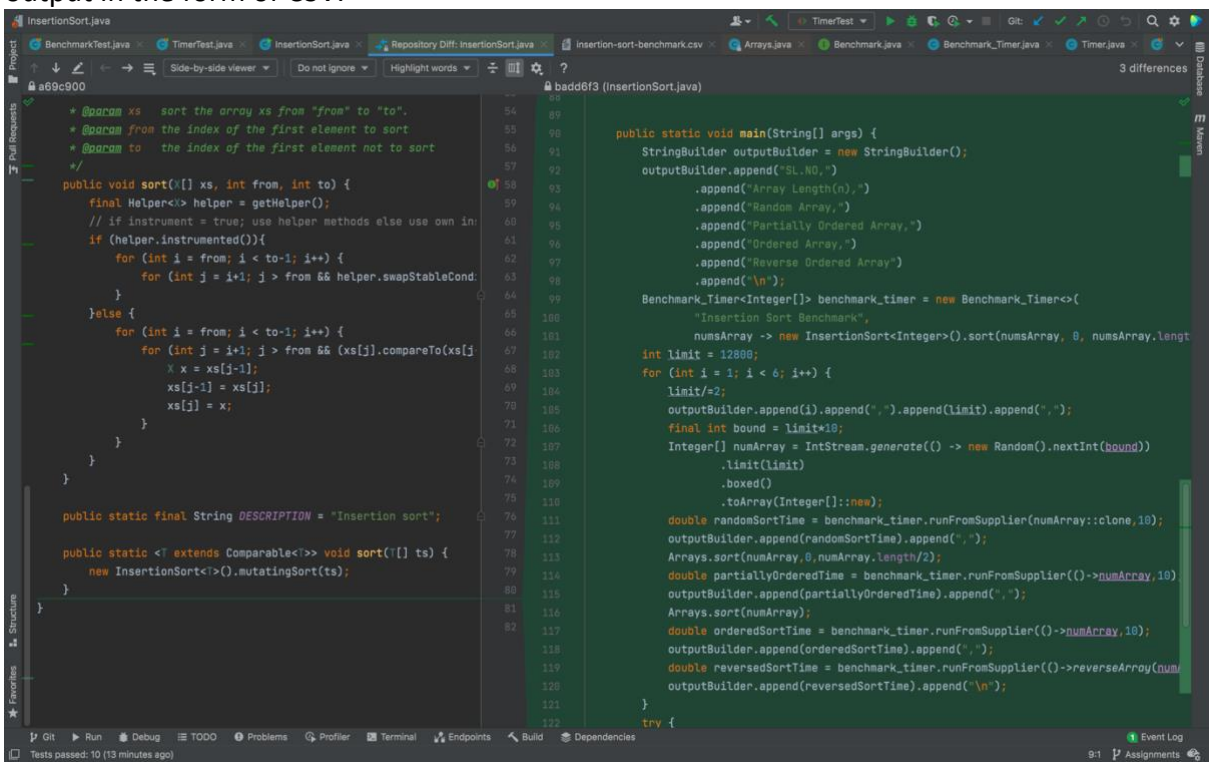
50993b5 (Timer.java)

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

/**
 * NOTE: (Maintain consistency) There are two system methods for getting the
 * 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 TimeUnit.NANOSECONDS.toMillis(ticks);
}

final static LazyLogger logger = new LazyLogger(Timer.class);

static class TimerException extends RuntimeException {
    public TimerException() {
    }

    public TimerException(String message) {
        super(message);
    }

    public TimerException(String message, Throwable cause) {
```

2) Implemented Insertion Sort logic for both the cases (instrument = true and false)

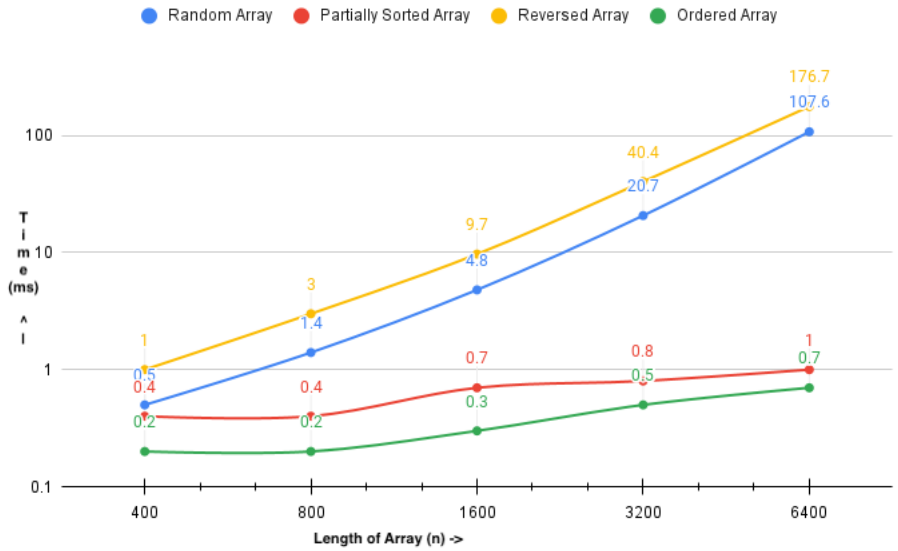


3) Added Main Method for running Tests on Insertion Sort. Also added logic to get output in the form of CSV.



B) Analysis

Graph 1: time (ms) vs length of array



- From the values obtained, it can be observed that it takes longer time to sort reversed array compared to others.
- It takes longer time to sort random array compared to Partially ordered and ordered arrays. Next comes partially ordered, it lies between random and ordered as shown in graph.
- It takes very less time for processing Ordered Array.

Google Sheet URL : https://docs.google.com/spreadsheets/d/1zli1ynr85myV2n-GUD-QXEA8twzSWIgNFFzbWE6uAF8/edit?usp=sharing

C) Unit Tests & Main Code Output Screenshots

Image C1) BenchmarkTests

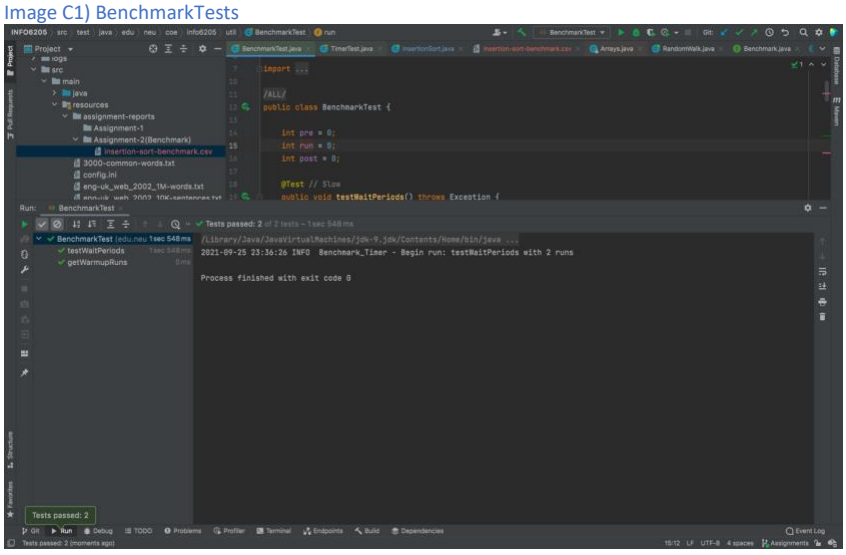


Image C2) Timer Tests

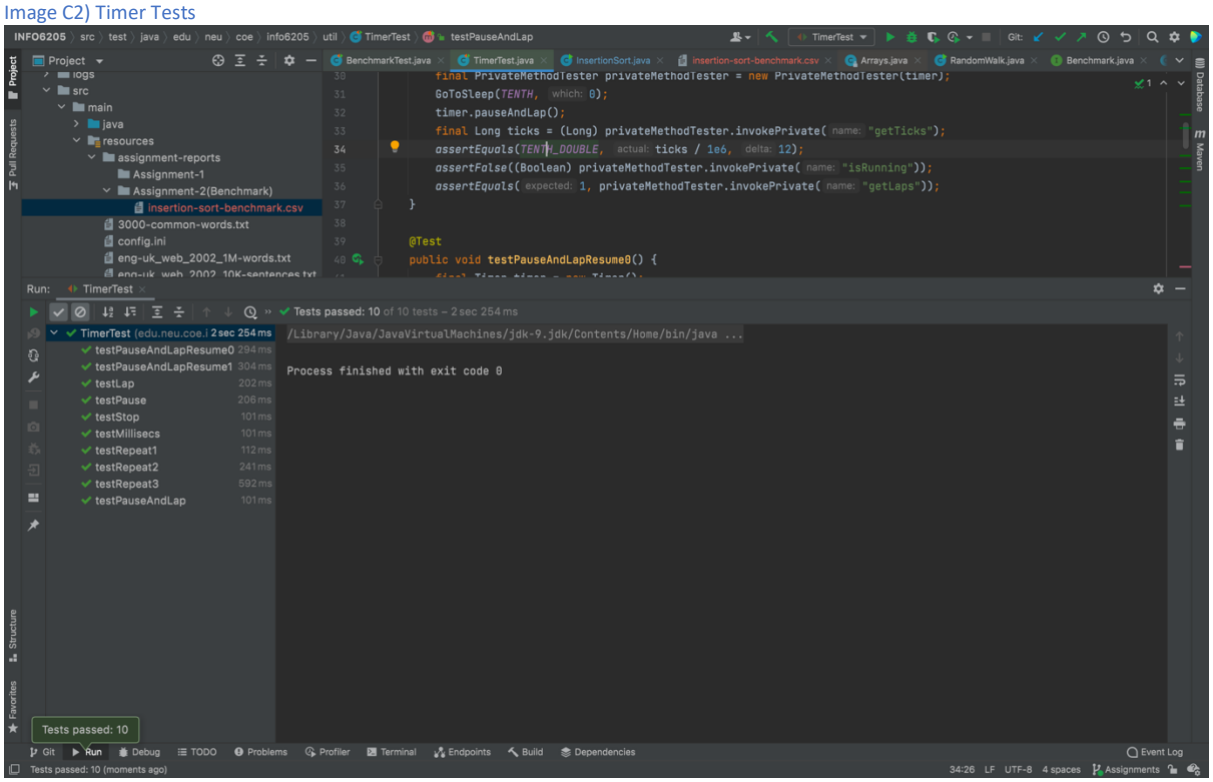


Image C3) InsertionSortTests

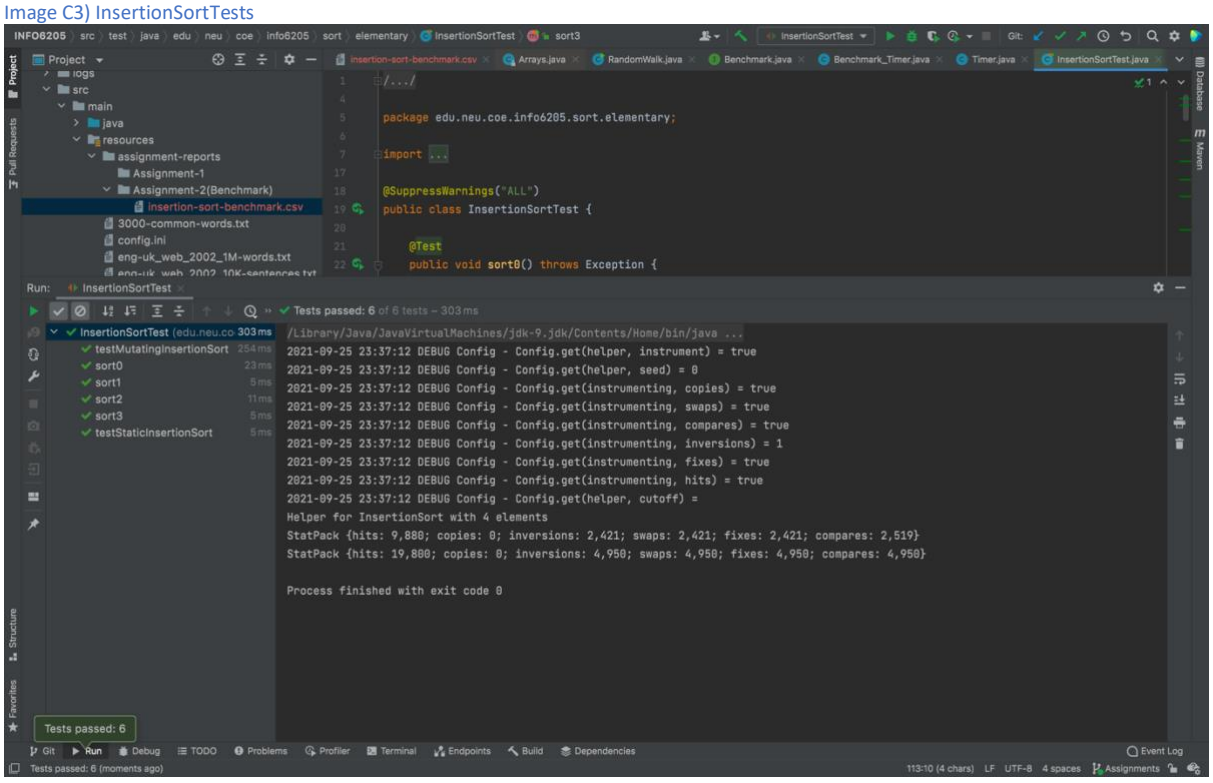


Image C4) Main Method Output