Program Structures and Algorithms
Spring 2023 (SEC –3)
Assignment-3: Benchmark.

NAME: Venkatesha Matam
NUID: 002740702

Task:

- Fix the Timer Class.
- Run the test cases in BenchmarkTest.java, TimerTest.java, and InsertionSortTest.java.
- Use the helper function to implement Insertion Sort.
- Create a main method that generates arrays of varying sizes using the doubling method.
- Draw conclusions from observations and evidence.

Code Change Snapshots:

1) **Timer class -**

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

    final double res = meanLapTime();
    resume();
    return res;
    // END
}
```

getClock() :

```
/**
 * 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();
    // END
}
```

toMilliSecs():

```
/**
 * 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) {
    // FIXME by replacing the following code
    return ticks / 1000000L;
    // END
}
```

2) sort() method of InsertionSort -

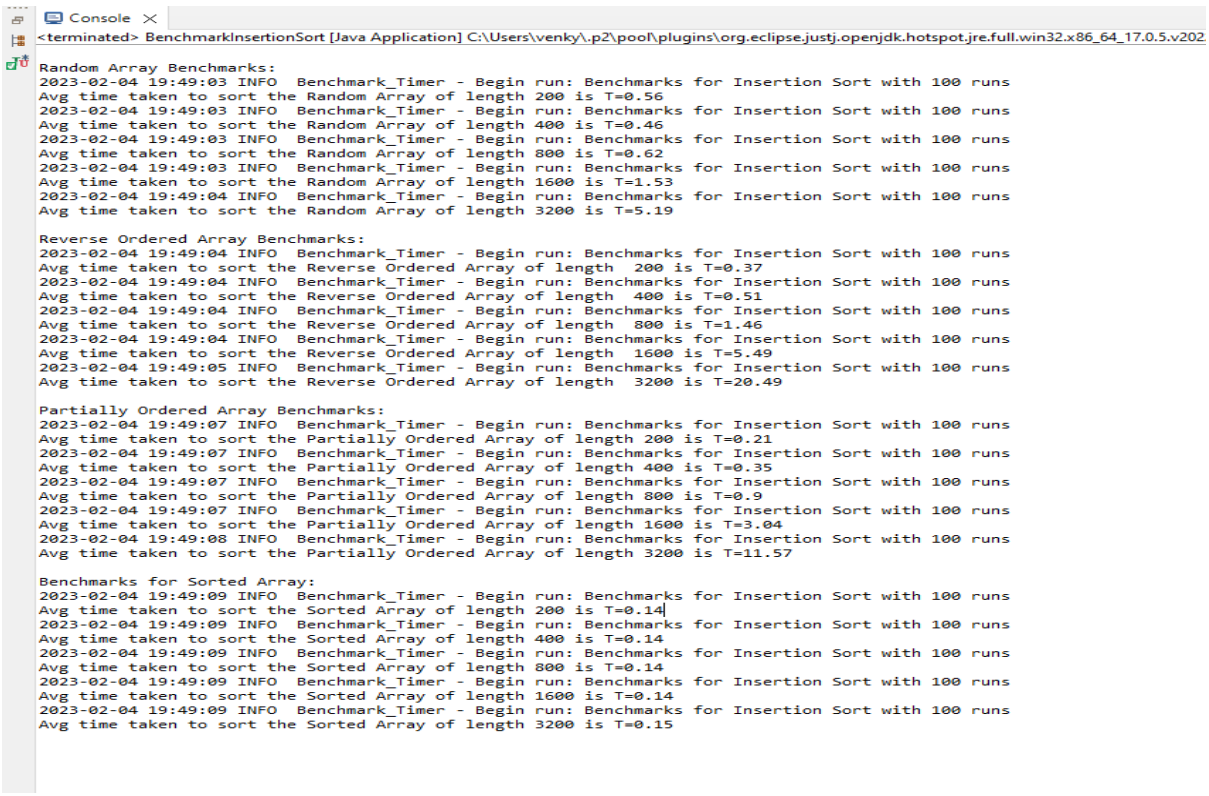
```
/**
 * Sort the sub-array xs:from:to using insertion sort.
 *
 * @param xs   sort the array xs from "from" to "to".
 * @param from the index of the first element to sort
 * @param to   the index of the first element not to sort
 */
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();
    for (int i = from + 1; i < to; i++) {
        for (int j = i - 1; j >= from; j--) {
            if (helper.compare(xs[j], xs[j + 1]) == 1) {
                helper.swap(xs, j, j + 1);
            } else {
                break;
            }
        }
    }
}
```

3) Driver Class -

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

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

import java.util.Random;

public class BenchmarkInsertionSort {


    public static void main(String[] args) {
        Benchmark_Timer benchmark_timer = new Benchmark_Timer<Integer[]>("Benchmarks for Insertion Sort",
                (Integer[] array) -> {
                    new InsertionSort<Integer>().sort(array, true);
                });
        int [] lengthOfArray={200,400,800,1600,3200};
        int n=100;

        System.out.println();
        System.out.println("Random Array Benchmarks:");
        for(int i=0;i<lengthOfArray.length;i++){
            Integer[] random=new Integer[lengthOfArray[i]];
            Random rand = new Random();
            for(int j = 0; j <random.length ; j++){
                random[j] = rand.nextInt(j+1);
            }
            double avgTime=benchmark_timer.run(random,100);
            System.out.println("Avg time taken to sort the Random Array of length "+lengthOfArray[i]+" is T="+avgTime);
        }

        System.out.println();
        System.out.println("Reverse Ordered Array Benchmarks: ");
```

```
        System.out.println();
        System.out.println("Reverse Ordered Array Benchmarks: ");
        for(int i=0;i<lengthOfArray.length;i++){
            Integer[] reverse=new Integer[lengthOfArray[i]];
            int k= 0;
            for(int j =reverse.length-1 ; j >=0; j--){
                reverse[k] = j;
                k++;
            }
            double avgTime=benchmark_timer.run(reverse,100);
            System.out.println("Avg time taken to sort the Reverse Ordered Array of length  "+lengthOfArray[i]+" is T="+avgTime);
        }


        System.out.println();
        System.out.println("Partially Ordered Array Benchmarks:");
        for(int i=0;i<lengthOfArray.length;i++){
            Random rand = new Random();
            Integer[] partial=new Integer[lengthOfArray[i]];

            for(int j = 0; j <= partial.length / 2; j++){
                partial[j] = j;
            }

            for(int j =  partial.length / 2 + 1 ; j < partial.length ; j++){
                partial[j] = rand.nextInt(partial.length - j);
            }
            double avgTime=benchmark_timer.run(partial,100);
            System.out.println("Avg time taken to sort the Partially Ordered Array of length "+lengthOfArray[i]+" is T="+avgTime);
        }


         System.out.println();
         System.out.println("Benchmarks for Sorted Array:");
         for(int i=0;i<lengthOfArray.length;i++){
             Integer[] sorted=new Integer[lengthOfArray[i]];

             for(int j = 0; j < sorted.length ; j++){
                 sorted[j] = j;
             }
             double avgTime=benchmark_timer.run(sorted,100);
             System.out.println("Avg time taken to sort the Sorted Array of length "+lengthOfArray[i]+" is T="+avgTime);
         }

    }
```

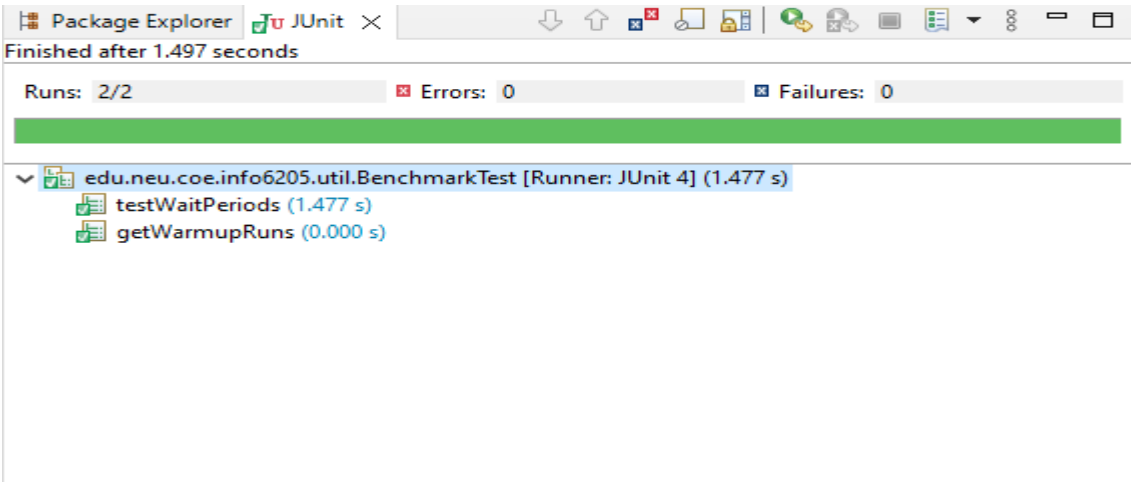
Output Snapshots:



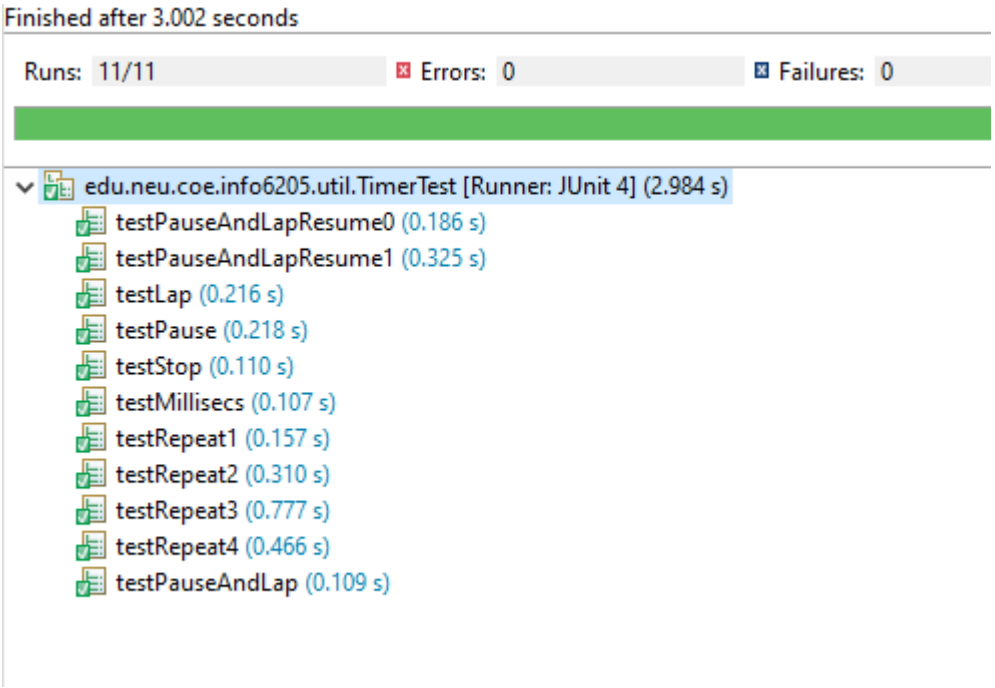
```
Console
<terminated> BenchmarkInsertionSort [Java Application] C:\Users\venky\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v202

Random Array Benchmarks:
2023-02-04 19:49:03 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Random Array of length 200 is T=0.56
2023-02-04 19:49:03 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Random Array of length 400 is T=0.46
2023-02-04 19:49:03 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Random Array of length 800 is T=0.62
2023-02-04 19:49:03 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Random Array of length 1600 is T=1.53
2023-02-04 19:49:04 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Random Array of length 3200 is T=5.19

Reverse Ordered Array Benchmarks:
2023-02-04 19:49:04 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Reverse Ordered Array of length  200 is T=0.37
2023-02-04 19:49:04 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Reverse Ordered Array of length  400 is T=0.51
2023-02-04 19:49:04 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Reverse Ordered Array of length  800 is T=1.46
2023-02-04 19:49:04 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Reverse Ordered Array of length  1600 is T=5.49
2023-02-04 19:49:05 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Reverse Ordered Array of length  3200 is T=20.49

Partially Ordered Array Benchmarks:
2023-02-04 19:49:07 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Partially Ordered Array of length 200 is T=0.21
2023-02-04 19:49:07 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Partially Ordered Array of length 400 is T=0.35
2023-02-04 19:49:07 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Partially Ordered Array of length 800 is T=0.9
2023-02-04 19:49:07 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Partially Ordered Array of length 1600 is T=3.04
2023-02-04 19:49:08 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Partially Ordered Array of length 3200 is T=11.57

Benchmarks for Sorted Array:
2023-02-04 19:49:09 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Sorted Array of length 200 is T=0.14
2023-02-04 19:49:09 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Sorted Array of length 400 is T=0.14
2023-02-04 19:49:09 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Sorted Array of length 800 is T=0.14
2023-02-04 19:49:09 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Sorted Array of length 1600 is T=0.14
2023-02-04 19:49:09 INFO  Benchmark_Timer - Begin run: Benchmarks for Insertion Sort with 100 runs
Avg time taken to sort the Sorted Array of length 3200 is T=0.15
```

Unit Test Screenshots:

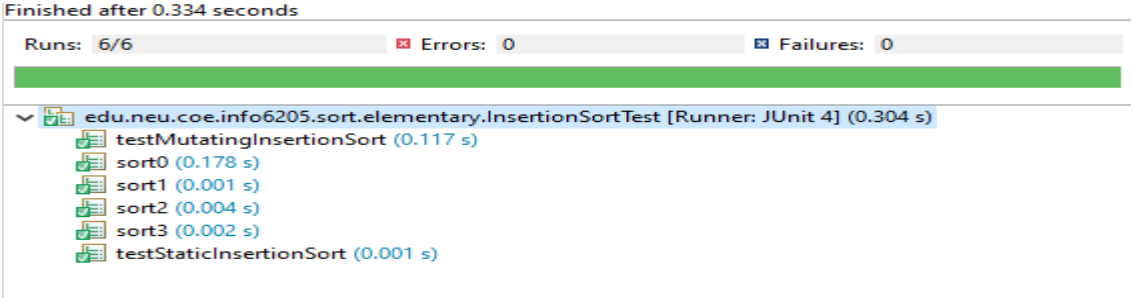
1) Benchmark Test



2) Timer Test



3) Insertion Sort Test.



Observations:

1. **Random Array:**

RANDOM ARRAY BENCHMARKS			
Array Size (N)	Time of Execution (in ms) (T)	Log(N)	Log(T)
200	0.56	2.302	-0.255
400	0.46	2.602	-0.364
800	0.62	3	-0.206
1600	1.53	3.204	0.187
3200	5.19	3.505	1.711

2. **Reverse Ordered Array Benchmarks:**

REVERSE ORDERED ARRAY BENCHMARKS			
Array Size (N)	Time of Execution (in ms) (T)	Log (N)	Log (T)
200	0.37	2.301	-0.43
400	0.51	2.602	-0.29
800	1.46	3	0.15
1600	5.49	3.204	0.74
3200	20.49	3.505	2.31

3. Partially Ordered Benchmarks:

Partially Ordered Array Benchmarks

Array Size (N)	Time of Execution (T) (in ms)	Log (N)	Log (T)
200	0.21	2.301	-0.677
400	0.35	2.602	-0.547
800	0.9	2.903	-0.954
1600	3.04	3.204	1.486
3200	11.57	3.505	2.466

4.Sorted Array Benchmarks:

SORTED ARRAY BENCHMARKS

Array Size(N)	Time of Execution (in ms) (T)	Log (N)	Log(T)
200	0.14	2.301	-1.947
400	0.14	2.602	-1.947
800	0.14	3	-1.947
1600	0.14	3.204	-1.947
3200	0.15	3.506	-1.903

Conclusion:

The speed from fastest to slowest for insertion sort of an array of the same size would be - sorted arrays, partially ordered arrays, random arrays, and reverse ordered arrays.
For reverse ordered, random, and partially ordered arrays, the order of growth of execution time is quadratic.
Insertion sort is appropriate for partially ordered arrays. The worst-case scenario is if the array is reversed.