

# **Parvataneni Mohith Lalita Kumar**

## **INFO 6205 - Fall 2021**

### **Assignment - 2**

#### **Task:**

To implement Timer Class and verify BenchmarkTest and TimerTest.

Implement Insertion Sort and run Unit Texts using InsertionSort Test.

Implement a program to run benchmarks for Random, Ordered, Reverse Sorted, Partial Sorted arrays.

#### **Findings:**

2021-09-26 22:51:11 INFO Benchmark\_Timer - Begin run: Insertion sort for partially sorted array of size: 10000 with 5 runs

0.6 ms

2021-09-26 22:51:11 INFO Benchmark\_Timer - Begin run: Insertion sort for randomly sorted array of size: 10000 with 5 runs

2.0 ms

2021-09-26 22:51:11 INFO Benchmark\_Timer - Begin run: Insertion sort for a sorted array of size: 10000 with 5 runs

0.8 ms

2021-09-26 22:51:11 INFO Benchmark\_Timer - Begin run: Insertion sort for reverse sorted array of size: 10000 with 5 runs

0.6 ms

2021-09-26 23:06:49 INFO Benchmark\_Timer - Begin run: Insertion sort for partially sorted array of size: 20000 with 5 runs

0.6 ms

2021-09-26 23:06:49 INFO Benchmark\_Timer - Begin run: Insertion sort for randomly sorted array of size: 20000 with 5 runs

1.4 ms

2021-09-26 23:06:49 INFO Benchmark\_Timer - Begin run: Insertion sort for a sorted array of size: 20000 with 5 runs

1.4 ms

2021-09-26 23:06:49 INFO Benchmark\_Timer - Begin run: Insertion sort for reverse sorted array of size: 20000 with 5 runs  
0.4 ms

2021-09-26 23:07:22 INFO Benchmark\_Timer - Begin run: Insertion sort for partially sorted array of size: 30000 with 5 runs  
1.4 ms

2021-09-26 23:07:22 INFO Benchmark\_Timer - Begin run: Insertion sort for randomly sorted array of size: 30000 with 5 runs  
3.2 ms

2021-09-26 23:07:22 INFO Benchmark\_Timer - Begin run: Insertion sort for a sorted array of size: 30000 with 5 runs  
1.0 ms

2021-09-26 23:07:22 INFO Benchmark\_Timer - Begin run: Insertion sort for reverse sorted array of size: 30000 with 5 runs  
0.4 ms

2021-09-26 23:07:43 INFO Benchmark\_Timer - Begin run: Insertion sort for partially sorted array of size: 40000 with 5 runs

1.0 ms

2021-09-26 23:07:43 INFO Benchmark\_Timer - Begin run: Insertion sort for randomly sorted array of size: 40000 with 5 runs  
1.4 ms

2021-09-26 23:07:43 INFO Benchmark\_Timer - Begin run: Insertion sort for a sorted array of size: 40000 with 5 runs  
1.0 ms

2021-09-26 23:07:43 INFO Benchmark\_Timer - Begin run: Insertion sort for reverse sorted array of size: 40000 with 5 runs  
0.8 ms

2021-09-26 23:08:22 INFO Benchmark\_Timer - Begin run: Insertion sort for partially sorted array of size: 50000 with 5 runs

2.6 ms

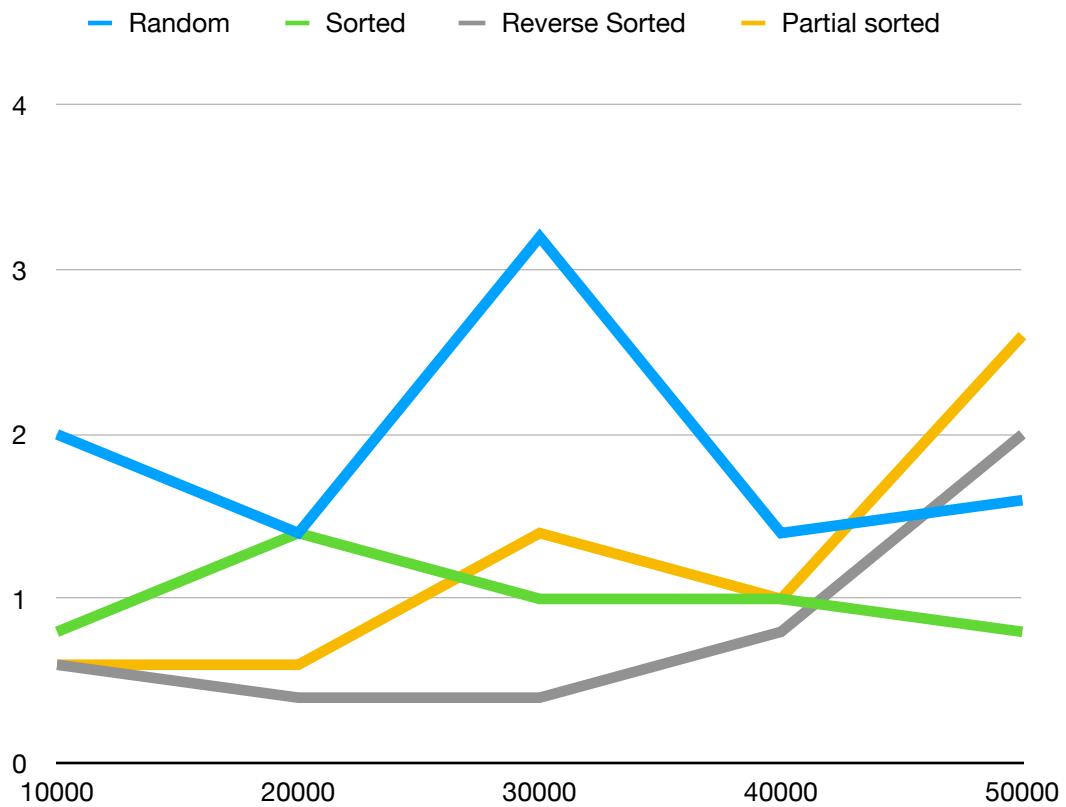
2021-09-26 23:08:22 INFO Benchmark\_Timer - Begin run: Insertion sort for randomly sorted array of size: 50000 with 5 runs  
1.6 ms

2021-09-26 23:08:22 INFO Benchmark\_Timer - Begin run: Insertion sort for a sorted array of size: 50000 with 5 runs  
0.8 ms

2021-09-26 23:08:22 INFO Benchmark\_Timer - Begin run: Insertion sort for reverse sorted array of size: 50000 with 5 runs  
2.0 ms

N	Random	Sorted	Reverse Sorted	Partial Sorted
10000	2	0.8	0.6	0.6
20000	1.4	1.4	0.4	0.6
30000	3.2	1	0.4	1.4
40000	1.4	1	0.8	1
50000	1.6	0.8	2.0	2.6

## Graph:



## **Output /Testing Screenshot :**

The screenshot shows an IDE interface with several windows open. On the left, the Git Repositories panel displays branches, tags, references, and the working tree for the INFO6205 repository. The main editor window contains the `Benchmark_Timer_InsertionSort.java` file, which imports various utility classes and defines a static method `partialArray` that generates a partially sorted array of integers. Below the editor is a terminal window showing the execution of the application, which performs insertion sort on arrays of size 10000 with 5 runs, comparing results for partially sorted, randomly sorted, and reverse sorted arrays. The bottom status bar indicates the file is writable and has 93:45 : 2429 lines.

```
Timer.java TimerTest.java Utilities.java InsertionSort.java Benchmark_Timer_InsertionSort.java

1 package edu.neu.coe.info6205.util;
2
3 import edu.neu.coe.info6205.sort.BaseHelper;
4 import edu.neu.coe.info6205.sort.GenericSort;
5 import edu.neu.coe.info6205.sort.Helper;
6 import edu.neu.coe.info6205.sort.elementary.InsertionSort;
7 import edu.neu.coe.info6205.util.Benchmark;
8 import edu.neu.coe.info6205.util.Benchmark.Timer;
9 import edu.neu.coe.info6205.util.Config;
10
11 import java.util.Arrays;
12 import java.util.Random;
13 import java.util.function.Supplier;
14
15 public class Benchmark_Timer_InsertionSort {
16
17
18     private static Config config;
19     public static Integer[] partialArray(int n){
20
21         Random rand = new Random();
22         Integer[] a = new Integer[n];
23         for (int i = 0; i<n/2; i++){
24             a[i] = i;
25         }
26
27         for (int i = n/2; i<n; i++){
28             a[i] = rand.nextInt(n) + n/2;
29         }
30
31         return a;
32     }
33
34 }
```

Synchronize Git Staging Git Reflog Properties Console Terminal JUnit

```
<terminated> Benchmark_Timer_InsertionSort [Java Application] /Library/Java/JavaVirtualMachines/jdk-16.0.2.jdk/Contents/Home/bin/java (Sep 26, 2021, 10:51:04 PM - 10:51:11 PM)
2021-09-26 22:51:11 INFO Benchmark_Timer - Begin run: Insertion sort for partially sorted array of size: 10000 with 5 runs
0.6 ms
2021-09-26 22:51:11 INFO Benchmark_Timer - Begin run: Insertion sort for randomly sorted array of size: 10000 with 5 runs
2.0 ms
2021-09-26 22:51:11 INFO Benchmark_Timer - Begin run: Insertion sort for a sorted array of size: 10000 with 5 runs
0.8 ms
2021-09-26 22:51:11 INFO Benchmark_Timer - Begin run: Insertion sort for reverse sorted array of size: 10000 with 5 runs
0.6 ms
```

The screenshot shows an IDE interface with several windows open. On the left, the project structure is displayed under 'Git Repositories'. The main workspace contains code for a 'Timer' class and its test cases.

**Code View:**

```
1 package edu.neu.coe.info6205.util;
2
3 import java.util.function.Consumer;
4
5 public class Timer {
6
7     /**
8      * Construct a new Timer and set it running.
9      */
10    public Timer() {
11        resume();
12    }
13
14    /**
15     * Run the given function n times, once per "lap" and then return the result of calling stop().
16     *
17     * @param n      the number of repetitions.
18     * @param function a function which yields a T (T may be Void).
19     * @return the average milliseconds per repetition.
20     */
21    public <T> double repeat(int n, Supplier<T> function) {
22        for (int i = 0; i < n; i++) {
23            function.get();
24            lap();
25        }
26        pause();
27        return meanLapTime();
28    }
29
30    /**
31     * Run the given functions n times, once per "lap" and then return the result of calling stop().
32     *
33     * @param n      the number of repetitions.
34     * @param supplier supplier a function which supplies a different T value for each repetition.
35     * @param function function a function T=>U and which is to be timed (U may be Void).
36     * @return the average milliseconds per repetition.
37     */
38    public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> function) {
39        return repeat(n, supplier, function, null, null);
40    }
41
42    /**
43     * Pause (without counting a lap); run the given functions n times while being timed, i.e. once per "lap", and finally return the result of
44     *
45     * @param n      the number of repetitions.
46     * @param supplier supplier a function which supplies a different T value for each repetition.
47     * @param function function a function T=>U and which is to be timed (U may be Void).
48     * @return the average milliseconds per repetition.
49     */
50    public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> function, double lapTime, double totalRunTime) {
51        for (int i = 0; i < n; i++) {
52            supplier.get();
53            function.get();
54            lap();
55        }
56        pause();
57        return (totalRunTime - lapTime * n) / (n - 1);
58    }
59
60    /**
61     * Stop the timer and return the average milliseconds per repetition.
62     *
63     * @return the average milliseconds per repetition.
64     */
65    public double meanLapTime() {
66        return (totalRunTime - lapTime * n) / (n - 1);
67    }
68
69    /**
70     * Stop the timer and return the total run time.
71     *
72     * @return the total run time.
73     */
74    public double totalRunTime() {
75        return totalRunTime;
76    }
77
78    /**
79     * Stop the timer and return the lap time.
80     *
81     * @return the lap time.
82     */
83    public double lapTime() {
84        return lapTime;
85    }
86
87    /**
88     * Resume the timer.
89     */
90    public void resume() {
91        lapTime = System.currentTimeMillis();
92    }
93
94    /**
95     * Pause the timer.
96     */
97    public void pause() {
98        totalRunTime += System.currentTimeMillis() - lapTime;
99        lapTime = System.currentTimeMillis();
100   }
101 }
```

**Test View:**

The test view shows the following results:

- Runs: 10/10
- Errors: 0
- Failures: 0

The status bar at the bottom indicates: Finished after 2.381 seconds.

The screenshot shows the IntelliJ IDEA interface with a Java project open. The left sidebar displays the project structure under 'Git Repositories'. The main area shows a code editor with a Java file named 'SortTest.java'. The code implements a sorting algorithm and includes several test methods using JUnit annotations. Below the code editor is a terminal window showing build results. At the bottom, there's a navigation bar with tabs like Synchronize, Git Staging, Git Reflog, Properties, Console, Terminal, and JUnit.

```
1 package edu.neu.coe.info6205.sort;
2
3 import edu.neu.coe.info6205.sort.linearithmic.MergeSortTest;
4
5 public class SortTest {
6
7     static class TestSorter extends SortWithHelper<Integer> {
8         public TestSorter(String description, int N, Config config) {
9             super(description, N, config);
10        }
11
12        /**
13         * Generic, mutating sort method which operates on a sub-array
14         *
15         * @param xs sort the array xs from "from" to "to".
16         * @param from the index of the first element to sort
17         * @param to the index of the first element not to sort
18         */
19        @Override
20        public void sort(Integer[] xs, int from, int to) {
21            Arrays.sort(xs, from, to);
22        }
23    }
24
25    @Test
26    public void testSort1() {
27        final TestSorter sorter = new TestSorter("test", 100, config);
28        final Helper<Integer> helper = sorter.getHelper();
29        final Integer[] xs = helper.random(Integer.class, r -> r.nextInt(1000000));
30        final Integer[] ys = sorter.sort(xs);
31        assertTrue(xs[0] < ys[1]);
32        helper.postProcess(ys); // test that ys is properly sorted.
33    }
34
35    @Test
36    public void testSort2() {
37        final int N = 100;
38        final TestSorter sorter = new TestSorter("test", N, config);
39        final Helper<Integer> helper = sorter.getHelper();
40        helper.initState();
41        final Integer[] xs = helper.random(Integer.class, r -> r.nextInt(1000000));
42        sorter.sort(xs, 0, xs.length);
43        assertTrue(xs[0] < xs[1]);
44        helper.postProcess(xs); // test that xs is properly sorted.
45    }
46
47    @Test
48    public void testSort3() {
49        final int N = 100;
50        final TestSorter sorter = new TestSorter("test", N, config);
51        final Helper<Integer> helper = sorter.getHelper();
52        helper.initState();
53        final Integer[] xs = helper.random(Integer.class, r -> r.nextInt(1000000));
54        sorter.sort(xs, 0, xs.length);
55        assertTrue(xs[0] < xs[1]);
56        helper.postProcess(xs); // test that xs is properly sorted.
57    }
58}
```

Synchronize Git Staging Git Reflog Properties Console Terminal JUnit

Finished after 0.248 seconds

Runs: 10/10 Errors: 0 Failures: 0

edu.neu.coe.info6205.sort.SortTest [Runner: JUnit 4] (0.022 s)

Failure Trace