Program Structures & Algorithms
Spring 2022
Assignment No. 2 (Benchmark)

Name: Aditya Mulik
(NUID): 002127694

- **Task (use of System.nanoTime() used across the application)**

Part I: Implement 3 Methods (repeat, getClock & toMillisecs) of class Timer.

Part II: Implementation of insertion sort using the helper function swapStableConditional to check if the array is out of order.

Part III: To implement insertion sort using the benchmark class to identify the trends generated and time complexity involved.

- **Output screenshot**

Part I:

repeat() – return Double

```
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 t = supplier.get();
        if (preFunction != null){
            preFunction.apply(t);
        }
        resume();
        U u = function.apply(t);
        pauseAndLap();
        if (postFunction != null) {
            postFunction.accept(u);
        }
    }

    return meanLapTime();
}
```

getClock() - return Long

```
private static long getClock() {
    return System.nanoTime();
}
```

toMillisecs() – return Double

```
private static double toMillisecs(long ticks) {
    return ticks / 1000000 ;
}
```

Part II:

Insertion Sort (sort code with help of helpder.swapStableConditional() method)

```
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();
    for (int i=from; i<to; i++) {
        int j = i;
        while (j > 0 && helper.swapStableConditional(xs, j--));
    }
}
```

Part III:

Insertion Sort uses Benchmark Class to analyze the time complexity with various array data which is arranged in several orders, such as Sorted Ordered, Reversed, partially ordered and Random Ordered).

**** Note: no of runs (n): 10**

```
public class InsertionSortTest {

    private static final int UPPER_BOUND = (int)Math.pow(2,14);
    private static final int n = 10;

    public static double benchmark(String description, final Integer[] arr) {

        Benchmark<Integer[]> benchmarkTimer = new Benchmark_Timer<>(description, x ->
            x.clone(),
            input_arr -> new InsertionSort<Integer>().sort(input_arr, from: 0,input_arr.length), fPost: null
        );

        return benchmarkTimer.run(arr, n);
    }
```

```java
public static void main(String[] args) {

    List<Double> timings = new ArrayList<>();

    //Input array as Reverse Ordered
    for (int i = 1; i < UPPER_BOUND; i *= 2) {
        ArrayFactory af = new ArrayFactory(i);
        Integer[] arr = af.getDecreasing(); timings.add(benchmark( description: "Reverse-Ordered", arr.clone()));
    }

    for (int i = 1, j = 0; i < UPPER_BOUND; i *= 2, j++) {
        System.out.println("Size of Array: " + i + " -> " + timings.get(j));
    }

    //Input array as Partially Ordered
    timings = new ArrayList<>();

    for (int i = 1; i < UPPER_BOUND; i *= 2) {
        ArrayFactory af = new ArrayFactory(i);
        Integer[] arr = af.getPartial(); timings.add(benchmark( description: "Partially-Ordered", arr.clone()));
    }

    for (int i = 1, j = 0; i < UPPER_BOUND; i *= 2, j++) {
        System.out.println("Size of Array: " + i + " -> " + timings.get(j));
    }
```

```java
//Input array as Sorted Array
timings = new ArrayList<>();

for (int i = 1; i < UPPER_BOUND; i *= 2) {
    ArrayFactory af = new ArrayFactory(i);
    Integer[] arr = af.getIncreasing(); timings.add(benchmark( description: "Increasing-Ordered", arr.clone()));
}

for (int i = 1, j = 0; i < UPPER_BOUND; i *= 2, j++) {
    System.out.println("Size of Array: " + i + " -> " + timings.get(j));
}

//Input array as Random Numbers

timings = new ArrayList<>();
for (int i = 1; i < UPPER_BOUND; i *= 2) {
    ArrayFactory af = new ArrayFactory(i);

    Integer[] arr = af.getRandom(); timings.add(benchmark( description: "Random-Ordered", arr.clone()));
}

for (int i = 1, j = 0; i < UPPER_BOUND; i *= 2, j++) {
    System.out.println("Size of Array: " + i + " -> " + timings.get(j));
}
```

- **Relationship Conclusion**

To conclude, performing sorting on array with different order of arrangement using Insertion Sort, time complexity differs as per below order.

Ordered Array < Partially Ordered < Randomly Ordered < Reversed Ordered

- **Evidence / Graph**

To analyze the InsertionSort function we ran the function with array of different size(n) each

running for 100 times(runs) and then getting mean lap time in milliseconds that is taken by InsertionSort function to finish. It gave us the output time required to run the function in milliseconds for different sizes of array. Detailed evidence/report of that is present in excel file which I will be uploading with this file. Please check that. I am mentioning my final observation below.

Final Observation:

When we observed the function for a high size of array, we can see following variation based on the types of arrays passed in function.

Data:

Array size	Random ordered	Increasing	Reverse	Partially ordered
1	0.26856373	0.26445039	2.27746419	0.2284912
2	0.18470413	0.14001202	1.1971479	0.20124367
4	0.15769703	0.15696503	1.01179911	0.17038665
8	0.15471125	0.14088837	0.31701792	0.21480457
16	0.16268451	0.17497703	0.24406378	0.3826704
32	0.14465045	0.18226953	1.45296747	0.19465292
64	0.12763288	0.29718451	0.56606751	0.20596702
128	0.13172332	0.12649916	1.08743463	0.15537873
256	0.25554045	0.12394372	0.53036997	0.28544625
512	0.6469049	0.20168422	1.16428706	0.41503755
1024	1.40173251	0.25448382	4.79229905	1.23335914
2048	4.85150753	0.18174461	11.7550388	3.2451021
4096	20.01241543	0.16105416	42.9187	13.5113888
8192	70.18831667	0.1443296	169.505525	41.98263873

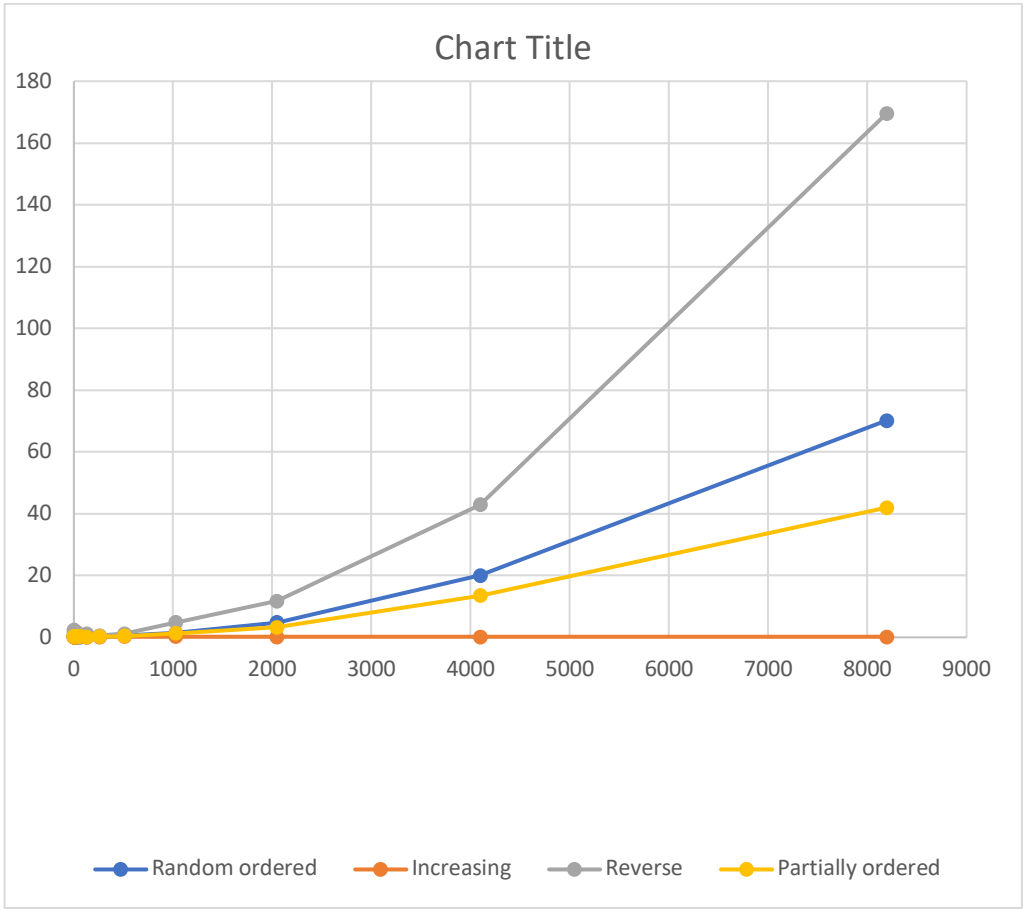
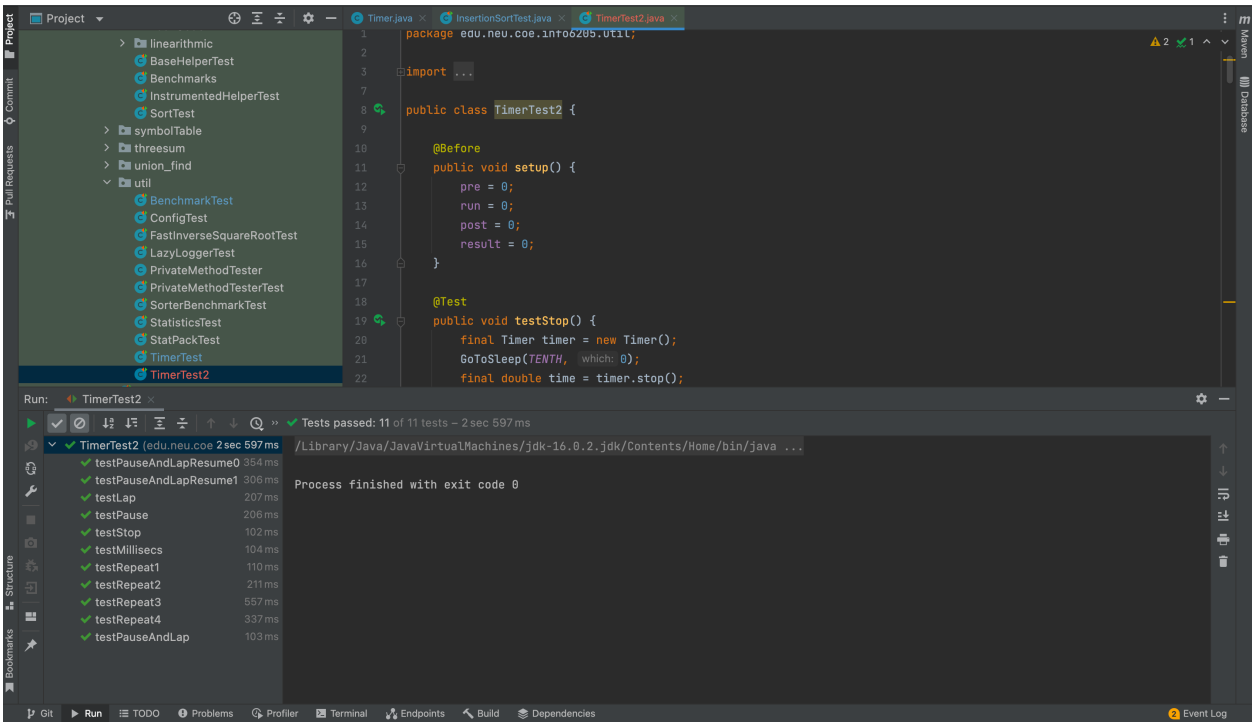


Chart Title
180
160
140
120
100
80
60
40
20
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
Random ordered
Increasing
Reverse
Partially ordered

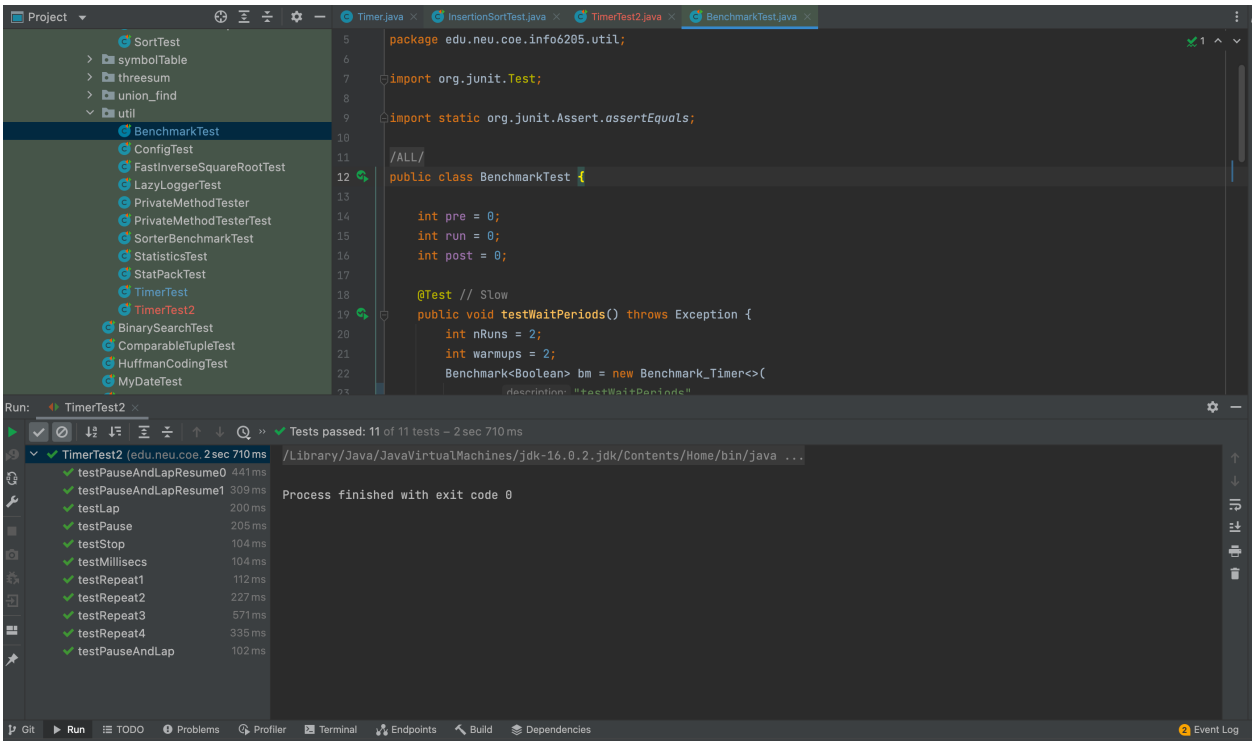
- **Unit tests result**

Below screenshot of all test cases ran successfully.

Timer Test Cases:



Benchmark Test Cases:



Insertion Sort Test Cases: