

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 helper.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);  
    }  
}
```

```

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));
    }
}

```

```

//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

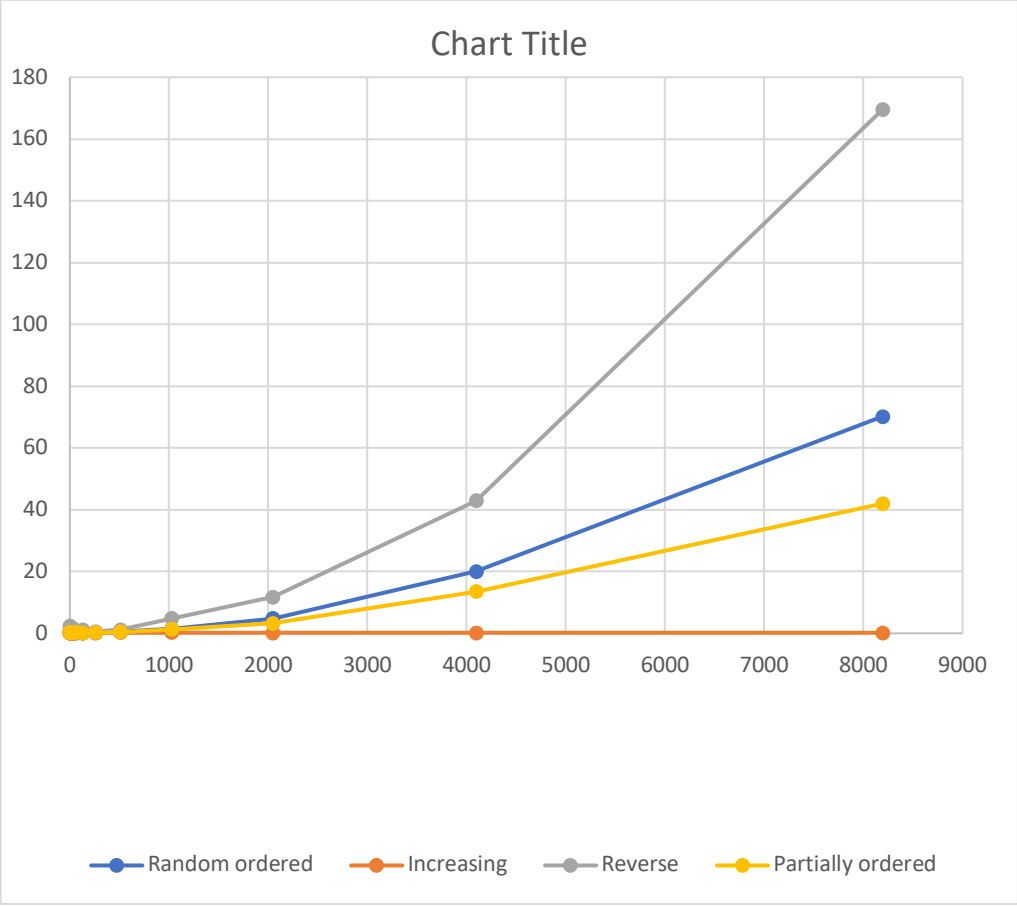
running for 100 times(runs) and then getting mean lap time in milliseconds that is taken by InsertionSort function to finish. It gave us the output time required to run the function in milliseconds for different sizes of array. Detailed evidence/report of that is present in excel file which I will be 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:

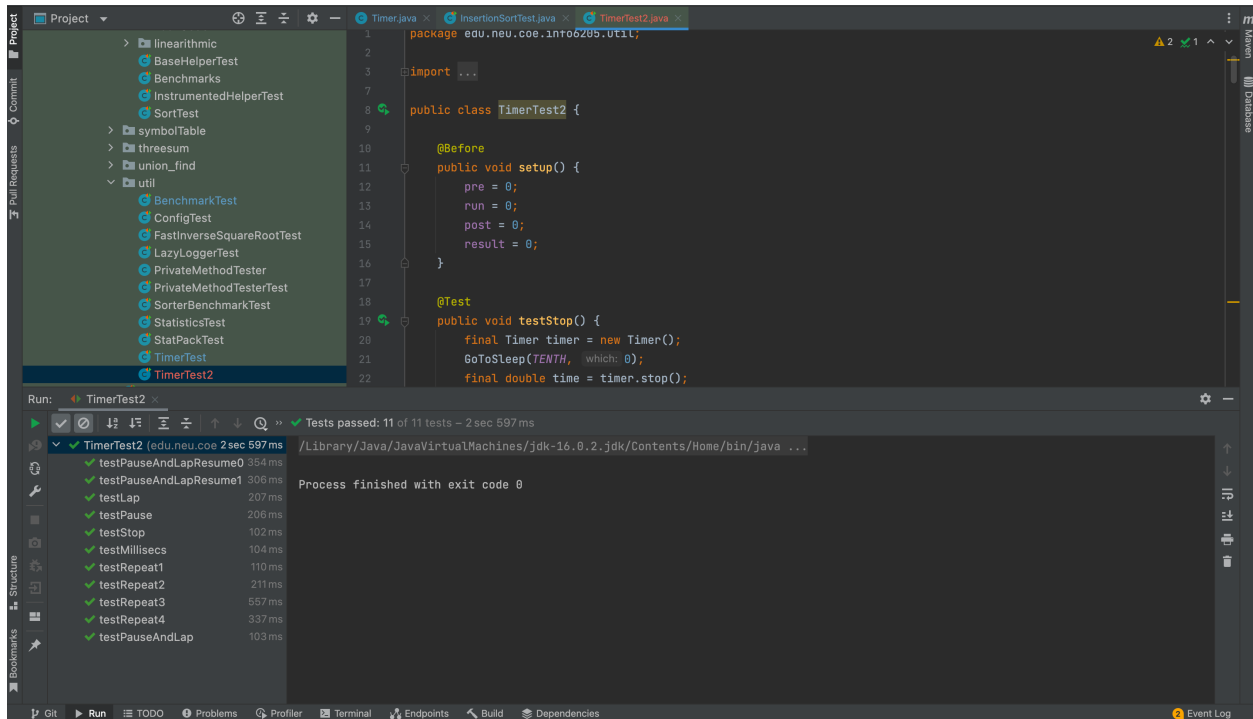
Array size	Random ordered	Increasing	Reverse	Partially ordered
1	0.26856373	0.26445039	2.27746419	0.2284912
2	0.18470413	0.14001202	1.1971479	0.20124367
4	0.15769703	0.15696503	1.01179911	0.17038665
8	0.15471125	0.14088837	0.31701792	0.21480457
16	0.16268451	0.17497703	0.24406378	0.3826704
32	0.14465045	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



- **Unit tests result**

Below screenshot of all test cases ran successfully.

Timer Test Cases:



Benchmark Test Cases:

The screenshot shows an IDE with the `BenchmarkTest.java` file open. The code defines a `BenchmarkTest` class with a `testWaitPeriods()` method. The `Run` tab shows the execution results for `TimerTest2`, which passed 11 of 11 tests in 2 seconds and 710 milliseconds.

```
package edu.neu.coe.info6205.util;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

//ALL/
public class BenchmarkTest {

    int pre = 0;
    int run = 0;
    int post = 0;

    @Test // Slow
    public void testWaitPeriods() throws Exception {
        int nRuns = 2;
        int warmups = 2;
        Benchmark<Boolean> bm = new Benchmark_Timer<>() {
            @Override
            public Boolean run() {
                // ...
            }
        };
        bm.run(nRuns, warmups);
    }
}
```

Run: **TimerTest2** (edu.neu.coe.info6205.util) 2 sec 710 ms

Test	Time
testPauseAndLapResume0	441 ms
testPauseAndLapResume1	309 ms
testLap	200 ms
testPause	205 ms
testStop	104 ms
testMillisecs	104 ms
testRepeat1	112 ms
testRepeat2	227 ms
testRepeat3	571 ms
testRepeat4	335 ms
testPauseAndLap	102 ms

Insertion Sort Test Cases:

The screenshot shows an IDE with the `InsertionSortTest.java` file open. The code defines an `InsertionSortTest` class with a `sort0()` method. The `Run` tab shows the execution results for `InsertionSortTest`, which passed 6 of 6 tests in 458 milliseconds.

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

import ...

//ALL/
public class InsertionSortTest {

    @Test
    public void sort0() throws Exception {
        final List<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        Integer[] xs = list.toArray(new Integer[0]);
        // System.out.println("Array 1 " + Arrays.toString(xs));
        final Config config = ConfigTest.saturateConfig(
            instrumentation: "true", seed: "0", inversions: "1", cutoff: "0", interInversion: "0"
        );
    }
}
```

Run: **InsertionSortTest** (edu.neu.coe.info6205.sort.elementary) 458 ms

Test	Time
testMutatingInsertionSort	416 ms
sort0	20 ms
sort1	5 ms
sort2	9 ms
sort3	4 ms
testStaticInsertionSort	4 ms

2022-02-11 06:49:32 DEBUG Config - Config.get(helper, instrument) = true
2022-02-11 06:49:32 DEBUG Config - Config.get(helper, seed) = 0
2022-02-11 06:49:32 DEBUG Config - Config.get(instrumenting, copies) = true
2022-02-11 06:49:32 DEBUG Config - Config.get(instrumenting, swaps) = true
2022-02-11 06:49:32 DEBUG Config - Config.get(instrumenting, compares) = true
2022-02-11 06:49:32 DEBUG Config - Config.get(instrumenting, inversions) = 1
2022-02-11 06:49:32 DEBUG Config - Config.get(instrumenting, fixes) = true
2022-02-11 06:49:32 DEBUG Config - Config.get(instrumenting, hits) = true
2022-02-11 06:49:32 DEBUG Config - Config.get(helper, cutoff) = 0
Helper for InsertionSort with 4 elements
StatPack {hits: 9,880; copies: 0; inversions: 2,421; swaps: 2,421; fixes: 2,421; compares: 2,519}
StatPack {hits: 19,800; copies: 0; inversions: 4,950; swaps: 4,950; fixes: 4,950; compares: 4,950}