

PROGRAM STRUCTURES AND ALGORITHMS

FALL 2021

Assignment - 2

Shashwat Shrey -- 002128122

Tasks Performed :

1. Implemented the 3 functions in timer.java
2. Implemented Insertion Sort
3. Plotted a graphical relation for growth in random ordered , partially ordered , ordered and reverse ordered arrays.

1. Timer.java

First 3 images are the executed functions, last image is the unit test

```
private static double toMillisecs(long ticks) {  
    // TO BE IMPLEMENTED  
    return ticks / Math.pow(10,6);  
}
```

```
*/  
private static long getClock() {  
    // TO BE IMPLEMENTED  
    return System.nanoTime();  
}
```

```

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");
    // TO BE IMPLEMENTED: note that the timer is running when this method is called and should still be running when it returns
    pause();
    T input = supplier.get();
    for(int i = 0; i < n; i++){
        if(preFunction != null){
            preFunction.apply(input);
        }
        resume();
        U output = function.apply(input);
        pauseAndLap();
        if(postFunction != null){
            postFunction.accept(output);
        }
    }

    //meanLapTime();

    return meanLapTime();
}

```

The screenshot shows an IDE with the following components:

- Project Explorer:** A tree view on the left showing the project structure. The 'util' package is expanded, showing files like BenchmarkTest, ConfigTest, FastInverseSquareR, LazyLoggerTest, PrivateMethodTeste, and TimerTest.
- Editor:** The main window displays the `TimerTest` class. It includes a package declaration, imports, and two methods: `setup()` (annotated with `@Before`) and `testStop()` (annotated with `@Test`). The `testStop()` method contains logic to create a `Timer` object, sleep for a tenth of a second, and then call `stop()` to get the time.
- Run Console:** At the bottom, the 'Run' tab shows the test results. It indicates that 10 tests passed in 2 seconds and 335 milliseconds. The tests listed are:
 - `testPauseAndLapResume0`: 285 ms
 - `testPauseAndLapResume1`: 317 ms
 - `testLap`: 210 ms
 - `testPause`: 211 ms
 - `testStop`: 105 ms
 - `testMilliseccs`: 103 ms
 - `testRepeat1`: 126 ms
 - `testRepeat2`: 246 ms
 - `testRepeat3`: 626 ms
 - `testPauseAndLap`: 106 ms

2.InsertionSort(First image is the code and second image is of the unit tests)

```
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();
    int start = from;
    int items = from + to;
    for(int i = start; i < items; i++){
        for(int j = i; j > from && helper.compare(xs, i: j - 1, j) > 0; j-- ){
            helper.swap(xs, i: j - 1, j);
        }
        start++;
    }
}
```

The screenshot shows an IDE with the following components:

- Project Explorer:** A tree view on the left showing the project structure. The 'elementary' package is expanded, showing files like BubbleSortTest, InsertionSortMST, InsertionSortOpt, InsertionSortTest, SelectionSortTest, and ShellSortTest.
- Editor:** The main window displays the source code for 'InsertionSortTest.java'. The code defines a public class 'InsertionSortTest' with a '@Test' method 'sort0()' that creates a list, adds elements, and calls the 'sort()' method. It also includes configuration for instrumentation and a helper object.
- Run Console:** The bottom panel shows the execution results of the test. It indicates that 6 out of 6 tests passed in 287ms. The output includes debug messages for configuration values and performance statistics for the 'SortPack'.

Run Console Output:

```
Tests passed: 6 of 6 tests - 287ms
InsertionSortTest (edu.neu.coe.info6205.sort.elementary)
  testMutatingInsertionSort 236 ms
  sort0 24 ms
  sort1 2 ms
  sort2 14 ms
  sort3 7 ms
  testStaticInsertionSort 4 ms
2021-09-25 17:15:05 DEBUG Config - Config.get(helper, instrument) = true
2021-09-25 17:15:05 DEBUG Config - Config.get(helper, seed) = 0
2021-09-25 17:15:05 DEBUG Config - Config.get(instrumenting, copies) = true
2021-09-25 17:15:05 DEBUG Config - Config.get(instrumenting, swaps) = true
2021-09-25 17:15:05 DEBUG Config - Config.get(instrumenting, compares) = true
2021-09-25 17:15:05 DEBUG Config - Config.get(instrumenting, inversions) = 1
2021-09-25 17:15:05 DEBUG Config - Config.get(instrumenting, fixes) = true
2021-09-25 17:15:05 DEBUG Config - Config.get(instrumenting, hits) = true
2021-09-25 17:15:05 DEBUG Config - Config.get(helper, cutoff) = true
Helper for InsertionSort with 4 elements
StatPack {hits: 9,684; 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}
Process finished with exit code 0
```

3. Analysis of random , ordered, partially ordered and reverse ordered array.

=====Reverse=====

```
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Reverse -> 100 with 100 runs
4.896E-4
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Reverse -> 200 with 100 runs
8.4871E-4
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Reverse -> 400 with 100 runs
0.00312921
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Reverse -> 800 with 100 runs
0.00274078
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Reverse -> 1600 with 100 runs
0.00540955
```

=====Random=====

```
2021-09-25 16:52:23 INFO Benchmark_Timer - Begin run: Random -> 10000 with 100 runs
0.0941350000000001
2021-09-25 16:52:23 INFO Benchmark_Timer - Begin run: Random -> 20000 with 100 runs
0.07234498
2021-09-25 16:52:25 INFO Benchmark_Timer - Begin run: Random -> 40000 with 100 runs
0.14900833
2021-09-25 16:52:34 INFO Benchmark_Timer - Begin run: Random -> 80000 with 100 runs
0.2878596200000004
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Random -> 160000 with 100 runs
0.5908650200000001
```

=====Ordered=====

```
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Ordered -> 100 with 100 runs
5.89010000000001E-4
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Ordered -> 200 with 100 runs
9.174799999999999E-4
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Ordered -> 400 with 100 runs
0.001514610000000002
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Ordered -> 800 with 100 runs
0.00276623
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: Ordered -> 1600 with 100 runs
0.00529169
```

=====Partially ordered=====

```
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: PartiallyOrdered ->100 with 100 runs
4.4336E-4
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: PartiallyOrdered ->200 with 100 runs
8.32070000000001E-4
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: PartiallyOrdered ->400 with 100 runs
0.002812089999999997
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: PartiallyOrdered ->800 with 100 runs
0.00273625
2021-09-25 16:54:04 INFO Benchmark_Timer - Begin run: PartiallyOrdered ->1600 with 100 runs
0.00549628
```

RANDOM									
N	TIME	T(N)	T(N) = (2 ^N -33.2103) * (N ² 999)		log(N)	log(T(N))	log(T(N)) / log(N)		
100	0.00454086	0.0001001621262			-7.782818727	-13.28537529	1.70713327	Mean ->	0.9638174009
200	0.00245126	0.0008007417853			-8.672260769	-10.28637529	1.186123845		
400	0.00340501	0.006401495566			-8.198125249	-7.287375287	0.8889075325		
800	0.01140583	0.0511764794			-6.454084755	-4.288375287	0.6844435966		
1600	0.05852463	0.4091281508			-4.094812281	-1.289375287	0.3148801943		
					Average ->			0.952273699	
ORDERED									
N	T	T(N)	T(N) and N		log(N)	log(T(N))	log(T(N)) / log(N)		
100	0.0039479	0.0001001621262			-7.984698838	-13.28537529	1.663854274		
200	0.00747792	0.0008007417853			-7.063147248	-10.28637529	1.456344449		
400	0.02043042	0.006401495566			-5.613137327	-7.287375287	1.298271334		
800	0.02912919	0.0511764794			-5.101390605	-4.288375287	0.8406286871		
1600	0.05802542	0.4091281508			-4.10717113	-1.289375287	0.3139326914		
					Average ->			1.114606287	
PARTIALLY ORDERED									
N	T	T(N)	T(N) and N		log(N)	log(T(N))	log(T(N)) / log(N)		
100	0.00384331	0.0001001621262			-8.023434936	-13.28537529	1.658821402		
200	0.00743536	0.0008007417853			-7.071381689	-10.28637529	1.454648573		
400	0.01910835	0.006401495566			-5.709652982	-7.287375287	1.276325428		
800	0.01976669	0.0511764794			-5.660784883	-4.288375287	0.7575584262		
1600	0.05058419	0.4091281508			-7.619766339	-1.289375287	0.1692145441		
					Average ->			1.062713675	
REVERSE									
N	T	T(N)	T(N) vs. N		log(N)	log(T(N))	log(T(N)) / log(N)		
100	4.34E-04	0.0001001621262			-11.16941911	-13.28537529	1.189441918		
200	7.91E-04	0.0008007417853			-10.30346939	-10.28637529	0.9983409371		
400	0.00141625	0.006401495566			-9.463708329	-7.287375287	0.7700338001		
800	0.00267488	0.0511764794			-8.546310114	-4.288375287	0.5017809124		
1600	0.00501586	0.4091281508			-7.639287204	-1.289375287	0.1687821458		
					Average ->			0.7256759427	