Program Structures & Algorithms

Spring 2022

Assignment No. 2 (Benchmark)

Name: SRI VYSHNAVI KOTHA

(NUID): 002985810

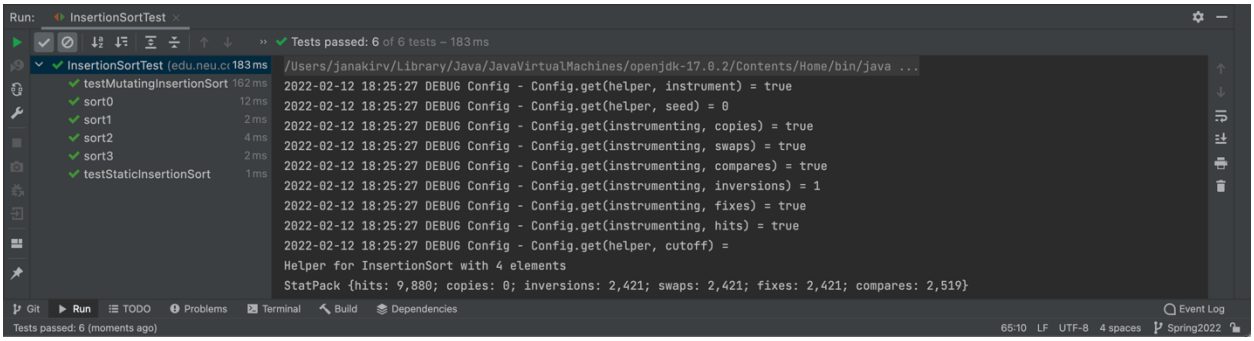
- **Task**

  (Part 1) You are to implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) of a class called *Timer*. *Timer* is invoked from a class called *Benchmark_Timer* which implements the *Benchmark* interface.

  (Part 2) Implement *InsertionSort* (in the *InsertionSort* class) by simply looking up the insertion code used by *Arrays.sort*.

  (Part 3) Implement a main program (or you could do it via your own unit tests) to actually run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered. I suggest that your arrays to be sorted are of type *Integer*.. Draw any conclusions from your observations regarding the order of growth.

- **Output screenshot**

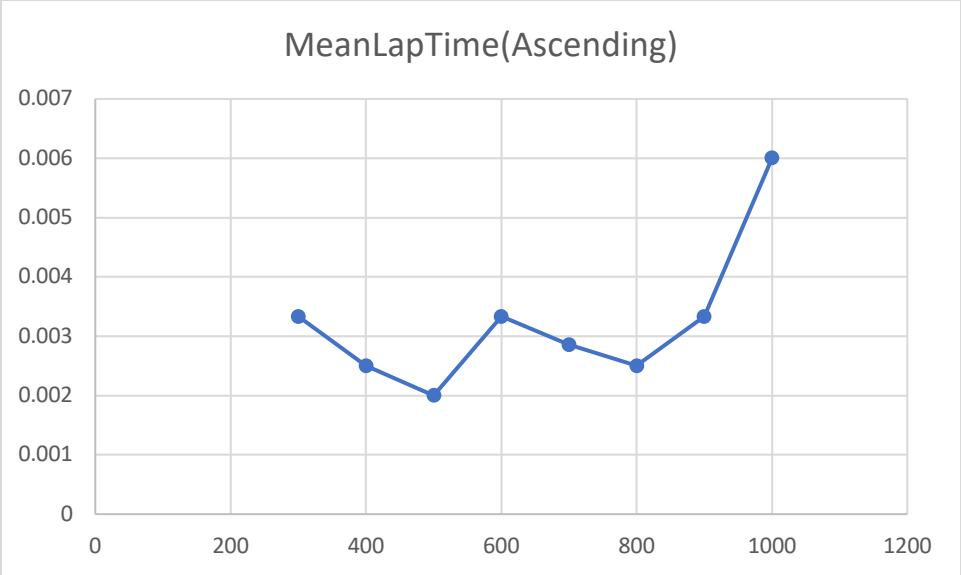


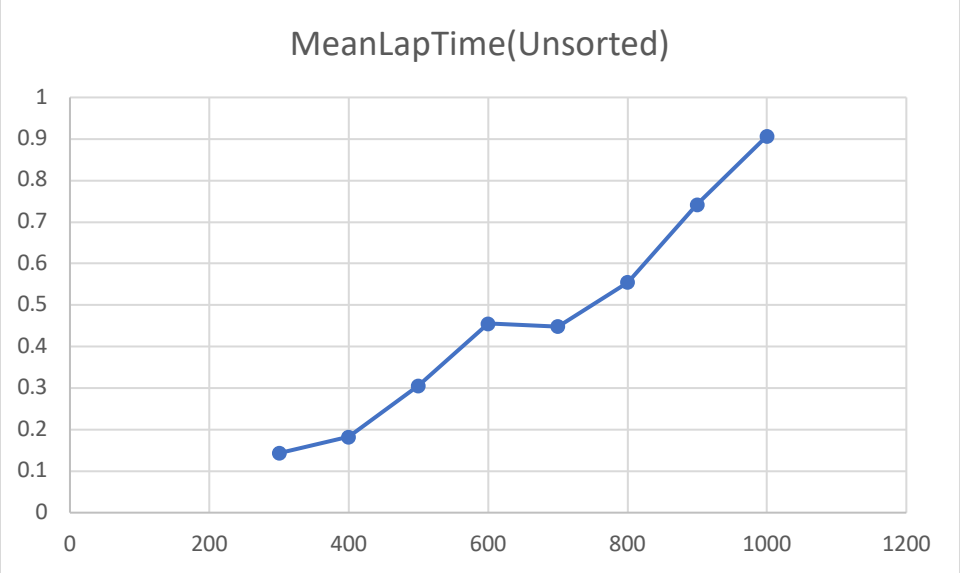
- **Relationship Conclusion**

Based on the series of experiments conducted, it is observed that MeanLapTime is approximately equal to the reciprocal of N

$$\text{meanLapTime} = 1/N$$

- **Evidence / Graph**





- **Unit tests result**

```
---------------------------------
Unsorted
N : 300
2022-02-12 22:37:42 INFO  Benchmark_Timer - Begin run: Insertion Sort with 300 runs
Run Time : 0.14333333333333334
---------------------------------
Ascending order sort
N : 300
2022-02-12 22:37:42 INFO  Benchmark_Timer - Begin run: Insertion Sort with 300 runs
Run Time : 0.0033333333333333335
---------------------------------
Decreasing order sort
N :300
2022-02-12 22:37:42 INFO  Benchmark_Timer - Begin run: Insertion Sort with 300 runs
Run Time : 0.0
---------------------------------
Partially sorted
N :300
2022-02-12 22:37:42 INFO  Benchmark_Timer - Begin run: Insertion Sort with 300 runs
Run Time : 0.0
---------------------------------
```

```
----------------------------------
Unsorted
N : 400
2022-02-12 22:37:42 INFO  Benchmark_Timer - Begin run: Insertion Sort with 400 runs
Run Time : 0.1825
----------------------------------
Ascending order sort
N : 400
2022-02-12 22:37:42 INFO  Benchmark_Timer - Begin run: Insertion Sort with 400 runs
Run Time : 0.0025
----------------------------------
Decreasing order sort
N :400
2022-02-12 22:37:42 INFO  Benchmark_Timer - Begin run: Insertion Sort with 400 runs
Run Time : 0.0025
----------------------------------
Partially sorted
N :400
2022-02-12 22:37:42 INFO  Benchmark_Timer - Begin run: Insertion Sort with 400 runs
Run Time : 0.0025
----------------------------------
```

```
----------------------------------
Unsorted
N : 1000
2022-02-12 22:37:44 INFO  Benchmark_Timer - Begin run: Insertion Sort with 1,000 runs
Run Time : 0.907
----------------------------------
Ascending order sort
N : 1000
2022-02-12 22:37:45 INFO  Benchmark_Timer - Begin run: Insertion Sort with 1,000 runs
Run Time : 0.006
----------------------------------
Decreasing order sort
N :1000
2022-02-12 22:37:45 INFO  Benchmark_Timer - Begin run: Insertion Sort with 1,000 runs
Run Time : 0.006
----------------------------------
Partially sorted
N :1000
2022-02-12 22:37:45 INFO  Benchmark_Timer - Begin run: Insertion Sort with 1,000 runs
Run Time : 0.004
```

Code

InsertionSortPart3.java

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

        int N = 300;

        for (int i = 0; i <= 5; i++) {
            while (N <= 1000) {
                Helper<Integer> helper = new BaseHelper<>("Insertion sort",
N, ConfigTest.setupConfig("true", "0", "1", "", ""));
                SortWithHelper<Integer> arraySorter = new
InsertionSort<>(helper);
                Benchmark<Integer[]> benchmarkTimer = new
Benchmark_Timer<Integer[]>("Insertion Sort", b -> arraySorter.sort(b));
                Integer[] arr = helper.random(Integer.class, r ->
r.nextInt(1000));
                System.out.println("-----------------------------------");
                System.out.println("Unsorted");
                System.out.println("N : " + arr.length);
                System.out.println("Run Time : " +
benchmarkTimer.run(arr,N));

                System.out.println("-----------------------------------");
                System.out.println("Ascending order sort");
                System.out.println("N : " + arr.length);

                Integer[] arr2 = arraySorter.sort(arr);
                System.out.println("Run Time : " +  benchmarkTimer.run(arr2,
N));

                System.out.println("-----------------------------------");
                System.out.println("Decreasing order sort");
                arraySorter.sort(arr2, arr2.length - 1, 0);
                System.out.println("N :" + arr2.length);
                System.out.println("Run Time : " + benchmarkTimer.run(arr2,
N));

                System.out.println("-----------------------------------");
                System.out.println("Partially sorted");
                arraySorter.sort(arr2, 0, (arr2.length - 1) / 2);
                System.out.println("N :" + arr2.length);
                System.out.println("Run Time : " + benchmarkTimer.run(arr2,
N));
                N = N + 100;
            }
        }
    }
```

InsertionSort.java

```
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();
    for( int i = from; i < to; i++){
        int k = i;
        while (k > 0 && helper.swapStableConditional( xs, k ) ){
            k = k- 1;
        }
    }
    // END
}
```

Timer.java

```
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U>
function, UnaryOperator<T> preFunction, Consumer<U> postFunction) {
    logger.trace("repeat: using " + n + " runs");
    pause();
    for (int i = 1; i < n+1; i++) {
        if (preFunction != null)
            preFunction.apply(supplier.get());
        resume();
        U op = function.apply(supplier.get());
        lap();
        pause();
        if (postFunction != null)
            postFunction.accept(op);
    }
    double mLapTime =  meanLapTime();
    resume();
    return mLapTime;
    // END
}
```

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


private static double toMillisecs(long ticks) {
    // Implementation
    return TimeUnit.NANOSECONDS.toMillis(ticks);
    // END
}
```