Sneha Ravichandran (001096251)

Program Structures & Algorithms INFO 6205

Fall 2021

Assignment 2

Task:

1. Implement repeat(), get clock() and toMillisecs() functions and run the benchmark timer and timer test
2. Implement Insertion sort code using helper functions and pass test cases
3. Implement insertion sort for 4 types of arrays and draw conclusion for the same(random, sorted, partially sorted and reverse sorted)

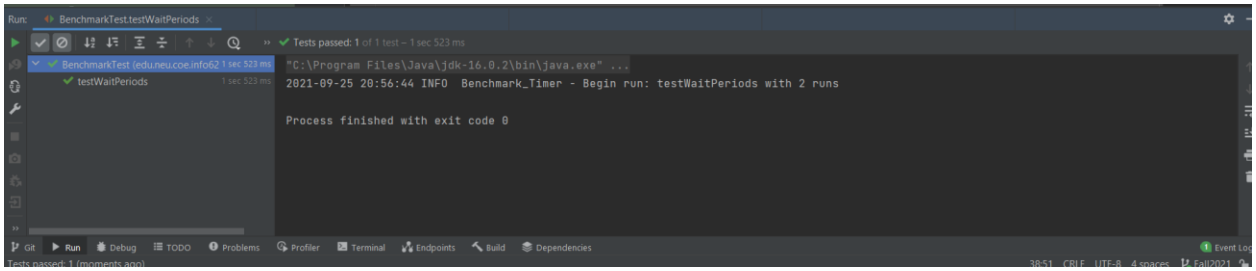
Part 1:

```
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();
    for(int i=0;i<n;i++)
    {
        T t = supplier.get();
        if(preFunction!= null) {
            t = preFunction.apply(t);
        }
        resume();
        U u=function.apply(t);
        pauseAndLap();
        if(postFunction!=null) {
            postFunction.accept(u);
        }
    }
    return meanLapTime();
}
```

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

```java
private static double toMillisecs(long ticks) {
    // TO BE IMPLEMENTED
    //double durationInMs =(double) ticks/1000000;
    //double durationInMs= (double)TimeUnit.NANOSECONDS.toMillis(ticks);
    double durationInMs =(double) TimeUnit.MILLISECONDS.convert(ticks,TimeUnit.NANOSECONDS);
    return durationInMs;
}
```

Test Cases for Benchmark Testing



Test Cases for Timer Test

Part2:

Insertion Sort code using helper functions:

```
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();

    // TO BE IMPLEMENTED
    for(int i=from;i<to;i++)
    {
        for(int j=i;j>from;j--)

        {
            if(helper.compare(xs, i: j-1,j)>0)
            {
                helper.swap(xs, i: j-1,j);
            }
            else
                break;

        }
    }
}
```

Test cases for Insertion sort:

Part 3:

Code:

```
InsertionSort ins=new InsertionSort();
Benchmark_Timer<Integer[]> timer_r=new Benchmark_Timer<>( description: "Benchmarking", fPre: null,(x)->ins.sort(x, from: 0,x.length), fPost: null);
Benchmark_Timer<Integer[]> timer_r2=new Benchmark_Timer<>( description: "Benchmarking", fPre: null,(x)->ins.sort(x, from: 0,x.length), fPost: null);
Benchmark_Timer<Integer[]> timer_r3=new Benchmark_Timer<>( description: "Benchmarking", fPre: null,(x)->ins.sort(x, from: 0,x.length), fPost: null);
Benchmark_Timer<Integer[]> timer_r4=new Benchmark_Timer<>( description: "Benchmarking", fPre: null,(x)->ins.sort(x, from: 0,x.length), fPost: null);
Supplier sup_reverse=() -> array_reverse;
Supplier sup_random=() -> array_random;

Supplier sup_sorted=() -> array_sorted;
Supplier sup_partial=() -> array_partial;

double time_reverse=timer_r.runFromSupplier(sup_reverse,runs);
System.out.println("When n is "+n+" mean time is "+time_reverse+" for a reverse array");
System.out.println();
double time_random=timer_r.runFromSupplier(sup_random,runs);

System.out.println("When n is "+n+" mean time is "+time_random+" for a random array");
System.out.println();

double time_sorted=timer_r.runFromSupplier(sup_sorted,runs);
System.out.println("When n is "+n+" mean time is "+time_sorted+" for a sorted array");
System.out.println();
double time_partial=timer_r.runFromSupplier(sup_partial,runs);
System.out.println("When n is "+n+" mean time is "+time_partial+" for a partial sorted array");
System.out.println();
```

Test for n=1250,2500,5000,10000,20000

Outputs:

```
2021-09-26 22:59:09 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 1250 mean time is 0.02 for a reverse array

2021-09-26 22:59:09 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 1250 mean time is 0.008 for a random array

2021-09-26 22:59:09 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 1250 mean time is 0.006 for a sorted array

2021-09-26 22:59:09 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 1250 mean time is 0.008 for a partial sorted array
```

```
2021-09-26 22:46:41 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 2500 mean time is 0.036 for a reverse array

2021-09-26 22:46:41 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 2500 mean time is 0.02 for a random array

2021-09-26 22:46:41 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 2500 mean time is 0.016 for a sorted array

2021-09-26 22:46:41 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 2500 mean time is 0.018 for a partial sorted array
```

```
2021-09-26 22:19:14 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 5000 mean time is 0.048 for a reverse array

2021-09-26 22:19:14 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 5000 mean time is 0.036 for a random array

2021-09-26 22:19:14 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 5000 mean time is 0.03 for a sorted array

2021-09-26 22:19:15 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 5000 mean time is 0.034 for a partial sorted array
```

```
2021-09-26 22:22:50 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 10000 mean time is 0.11 for a reverse array

2021-09-26 22:22:51 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 10000 mean time is 0.086 for a random array

2021-09-26 22:22:51 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 10000 mean time is 0.076 for a sorted array

2021-09-26 22:22:51 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 10000 mean time is 0.08 for a partial sorted array
```

```
2021-09-26 22:26:10 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 20000 mean time is 0.2 for a reverse array

2021-09-26 22:26:12 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 20000 mean time is 0.124 for a random array

2021-09-26 22:26:12 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 20000 mean time is 0.118 for a sorted array

2021-09-26 22:26:12 INFO  Benchmark_Timer - Begin run: Benchmarking with 500 runs
When n is 20000 mean time is 0.122 for a partial sorted array
```

Order of Growth:

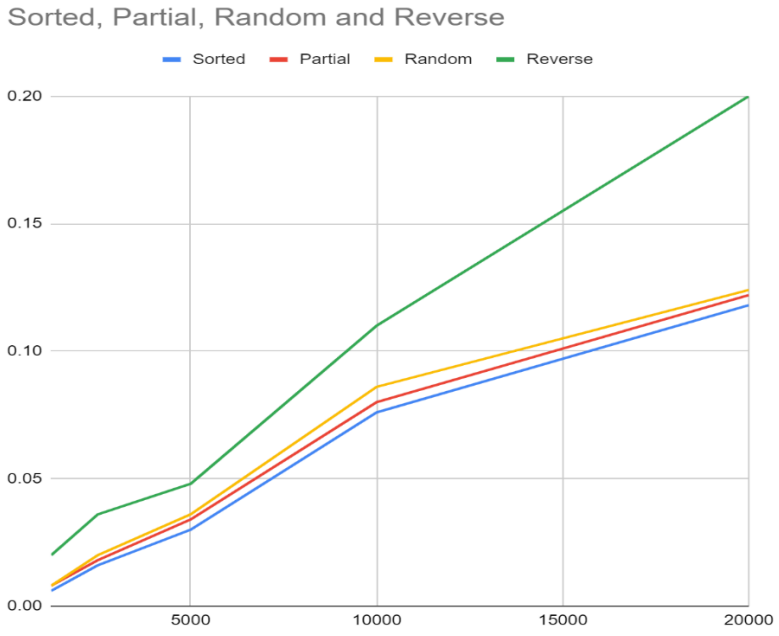
Reverse Ordered array:

Time $\propto n^2$

Sorted array:

Time $\propto n$

Evidence:



Ordered< Partially Ordered< Random Sorted< Reverse Ordered

Conclusion:

From the analysis we can say that insertion sort takes less time to sort a sorted array and more time to sort a reversed array.

Partially sorted arrays take a little more time than sorted arrays and randomly sorted arrays on an average takes more time than partially sorted arrays.

The graph is almost linear for a sorted array(~n) and almost quadratic(~n2) for reverse order array, the other partial and random are in-between the sorted and reverse order.