Program Structures and Algorithms Spring 2023(SEC –1)
Assignment-3

Name: Naga Venkata Nishanth Sayana

NU ID: 002930970

Code Screenshots:

1)Timer Class:

```
xiaohuanlin *
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 input= supplier.get();
        if(preFunction!=null){
            input=preFunction.apply(supplier.get());
        }
        resume();
        U mid=function.apply(input);
        pauseAndLap();
        if(postFunction!=null) postFunction.accept(mid);
    }

    final double result=meanLapTime();
    resume();
    return result;
    // END
}
```

```
2 usages   xiaohuanlin *
private static long getClock() {
    long timeInNano=System.nanoTime();
    return timeInNano;
    // END
}

/**
 * NOTE: (Maintain consistency) There are two system methods for getting the clock time.
 * Ensure that this method is consistent with getTicks.
 *
 * @param ticks the number of clock ticks -- currently in nanoseconds.
 * @return the corresponding number of milliseconds.
 */
2 usages   xiaohuanlin *
private static double toMillisecs(long ticks) {
    double timeInMilli=ticks/1000000;
    return timeInMilli;
    // END
}
```

2)InsertionSort Class

```
1 override  xiaohuanlin *
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();
    for (int i = from; i < to-1; ++i) {
        int j=i+1;
        boolean temp=true;
        while(j>from && temp){
            temp=helper.swapStableConditional(xs,j);
            j--;
        }
    }
    // END
}
```

3)Main Method for Benchmarking Insertion Sort on different types of Arrays

```
public class BenchmarkInsertionSort {



    public static void main(String[] args) {
        Benchmark_Timer benchmark_timer = new Benchmark_Timer<Integer[]>( description: "Benchmark Insertion Sort",
                (Integer[] array) -> {
                    new InsertionSort<Integer>().sort(array,  makeCopy: true);
                });
        int [] arrayLengths={200,400,800,1600,3200};
        int m=100;

        System.out.println();
        System.out.println("Benchmarks for Random Array:");
        for(int i=0;i<arrayLengths.length;i++){
            Integer[] random=new Integer[arrayLengths[i]];
            Random rand = new Random();
            for(int j = 0; j <random.length ; j++){
                random[j] = rand.nextInt( bound: j+1);
            }
            double avgTime=benchmark_timer.run(random, m: 100);
            System.out.println("Avergae Time taken to sort the Random Array of length n="+arrayLengths[i]+" is T="+avgTime);
        }
```

```
System.out.println();
System.out.println("Benchmarks for Reverse Ordered Array:");
for(int i=0;i<arrayLengths.length;i++){
    Integer[] reverseOrderd=new Integer[arrayLengths[i]];
    int k= 0;
    for(int j =reverseOrderd.length-1 ; j >=0; j--){
        reverseOrderd[k] = j;
        k++;
    }
    double avgTime=benchmark_timer.run(reverseOrderd, m: 100);
    System.out.println("Avergae Time taken to sort the Reverse Ordered Array of length n="+arrayLengths[i]+" is T="+avgTime);
}

System.out.println();
System.out.println("Benchmarks for Partially Ordered Array:");
for(int i=0;i<arrayLengths.length;i++){
    Random rand = new Random();
    Integer[] partially=new Integer[arrayLengths[i]];

    for(int j = 0; j <= partially.length / 2; j++){
        partially[j] = j;
    }

    for(int j =  partially.length / 2 + 1 ; j < partially.length ; j++){
        partially[j] = rand.nextInt( bound: partially.length - j);
    }
    double avgTime=benchmark_timer.run(partially, m: 100);
    System.out.println("Avergae Time taken to sort the Partially Ordered Array of length n="+arrayLengths[i]+" is T="+avgTime);
}
```

```
System.out.println();
System.out.println("Benchmarks for Sorted Array:");
for(int i=0;i<arrayLengths.length;i++){
    Integer[] sorted=new Integer[arrayLengths[i]];

    for(int j = 0; j < sorted.length ; j++){
        sorted[j] = j;
    }
    double avgTime=benchmark_timer.run(sorted, m: 100);
    System.out.println("Avergae Time taken to sort the Sorted Array of length n="+arrayLengths[i]+" is T="+avgTime);
}
```

Test cases:

1)Timer Test Cases

```
xiaohuanlin *
public class TimerTest {

    xiaohuanlin
    @Before
    public void setup() {
        pre = 0;
        run = 0;
        post = 0;
        result = 0;
    }

    xiaohuanlin
    @Test
    public void testStop() {
        final Timer timer = new Timer();
        GoToSleep(TENTH,  which: 0);
```

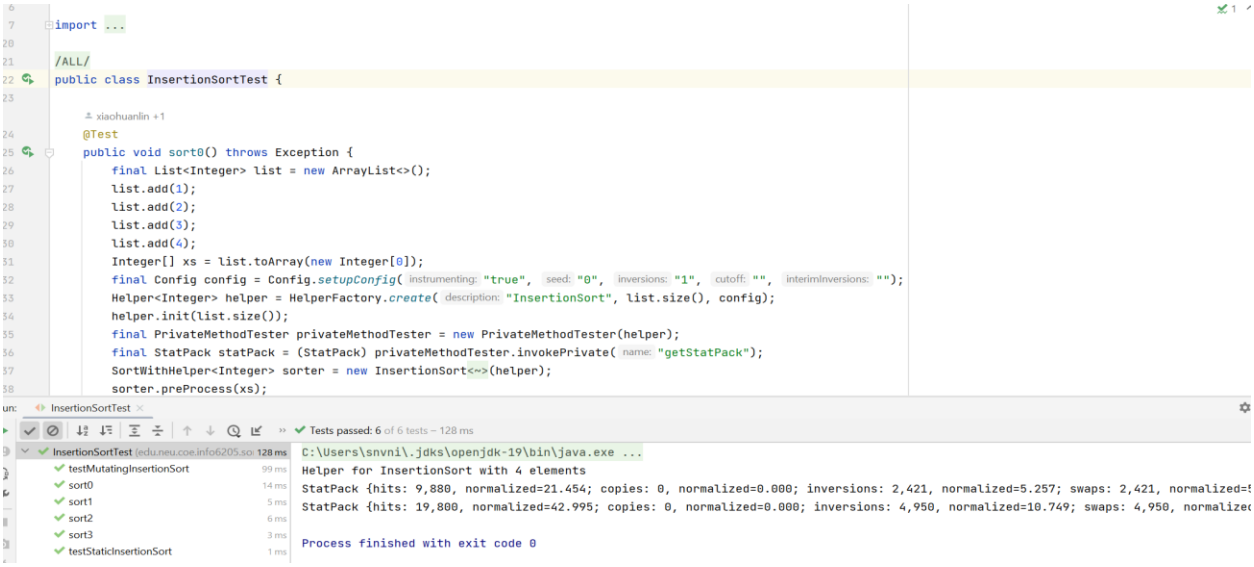
```
un:  TimerTest
Tests passed: 11 of 11 tests – 3 sec 12 ms
TimerTest (edu.neu.coe.info6205.util)  3 sec 12 ms
    testPauseAndLapResume0  199 ms
    testPauseAndLapResume1  327 ms
    testLap  217 ms
    testPause  218 ms
    testStop  108 ms
    testMillisecs  108 ms
    testRepeat1  157 ms
    testRepeat2  308 ms
    testRepeat3  776 ms
    testRepeat4  484 ms
    testPauseAndLap  110 ms

C:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...

Process finished with exit code 0
```

2)Benchmark Test Cases

```
/ALL/
public class BenchmarkTest {

    2 usages
    int pre = 0;
    2 usages
    int run = 0;
    2 usages
    int post = 0;

    xiaohuanlin
    @Test // Slow
    public void testWaitPeriods() throws Exception {
        int nRuns = 2;
        int warmups = 2;
        Benchmark<Boolean> bm = new Benchmark_Timer<>(
                description: "testWaitPeriods", b -> {
            GoToSleep( mSecs: 100L,  which: -1);
            return null;
        },
                b -> {
                    GoToSleep( mSecs: 200L,  which: 0);
```

```
Run:  BenchmarkTest
Tests passed: 2 of 2 tests – 1 sec 469 ms
BenchmarkTest (edu.neu.coe.info6205.  1 sec 469 ms
    testWaitPeriods  1 sec 469 ms
    getWarmupRuns  0 ms

C:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...
2023-02-04 16:42:05 INFO  Benchmark_Timer - Begin

Process finished with exit code 0
```

3)Insertion Sort Test Cases



```
import ...

/ALL/
public class InsertionSortTest {

    xiaohuanlin +1
    @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]);
        final Config config = Config.setupConfig( instrumenting: "true",  seed: "0",  inversions: "1",  cutoff: "",  interimInversions: "");
        Helper<Integer> helper = HelperFactory.create( description: "InsertionSort", list.size(), config);
        helper.init(list.size());
        final PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
        final StatPack statPack = (StatPack) privateMethodTester.invokePrivate( name: "getStatPack");
        SortWithHelper<Integer> sorter = new InsertionSort<>(helper);
        sorter.preProcess(xs);
```

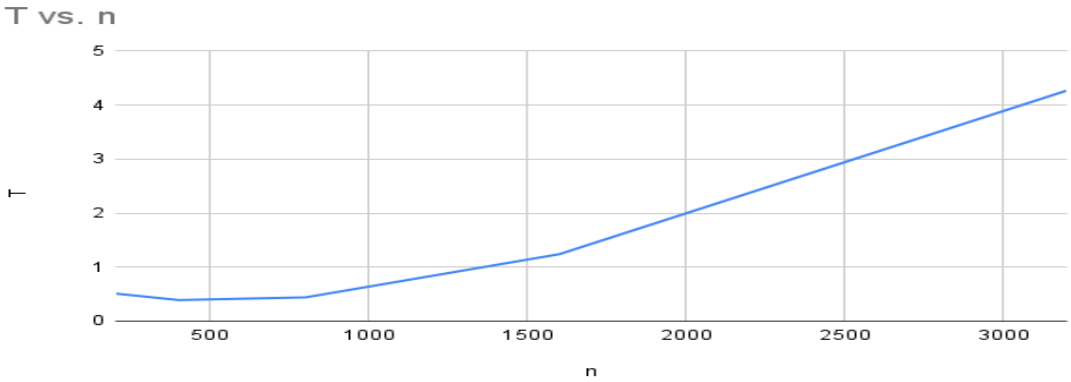
```
InsertionSortTest
Tests passed: 6 of 6 tests – 128 ms
InsertionSortTest (edu.neu.coe.info6205.so 128 ms
testMutatingInsertionSort 99 ms
sort0 14 ms
sort1 5 ms
sort2 6 ms
sort3 3 ms
testStaticInsertionSort 1 ms

C:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...
Helper for InsertionSort with 4 elements
StatPack {hits: 9,880, normalized=21.454; copies: 0, normalized=0.000; inversions: 2,421, normalized=5.257; swaps: 2,421, normalized=
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950, normalize

Process finished with exit code 0
```

Observations:

I have taken benchmarks of sorting Random, Reverse Ordered, Partially ordered, and Sorted array using Insertion Sort for different values of n using doubling method starting from 200, extending up to 3200. I have run the experiment m=100 times for each value of n for every type of array sorting. I have plotted the graphs for every type of sorted which illustrated below.

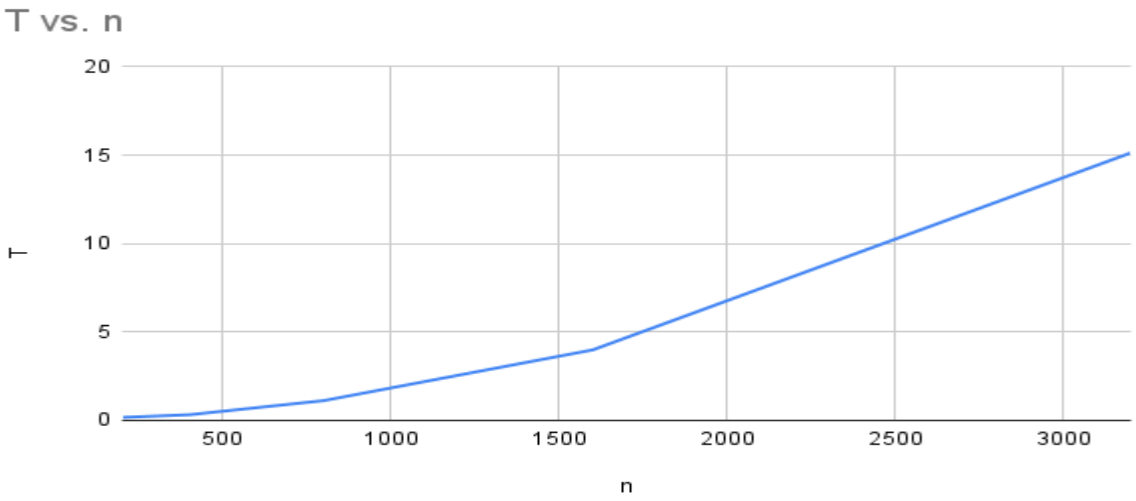
a) Tabulation of n values and Average Time for sorting Random Array

n	T
200	0.51
400	0.39
800	0.44
1600	1.24
3200	4.27



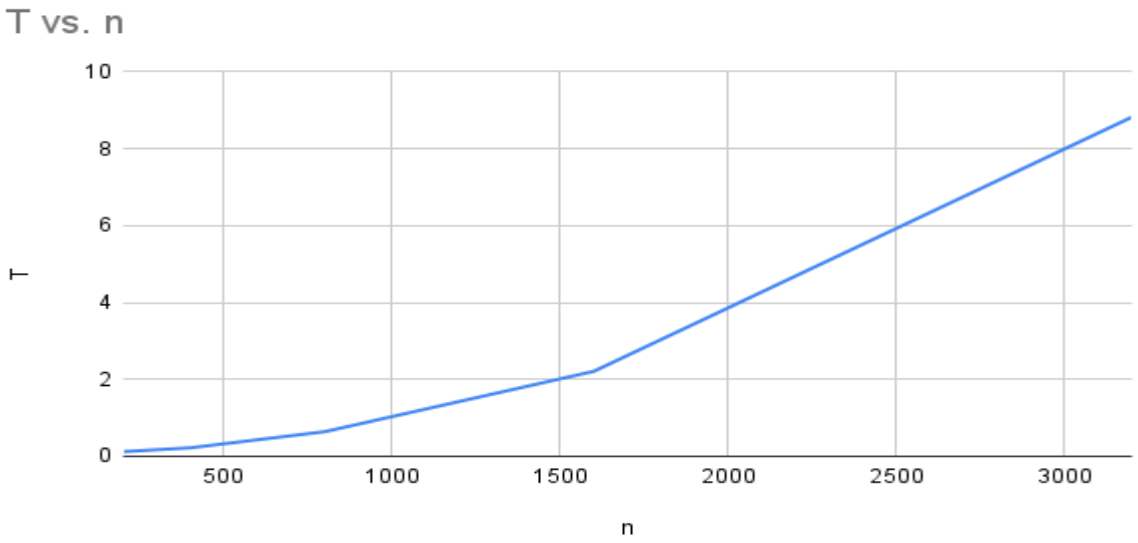
b) Tabulation of n values and Average Time for sorting Reverse Ordered Array

n	T
200	0.17
400	0.32
800	1.12
1600	3.99
3200	15.13



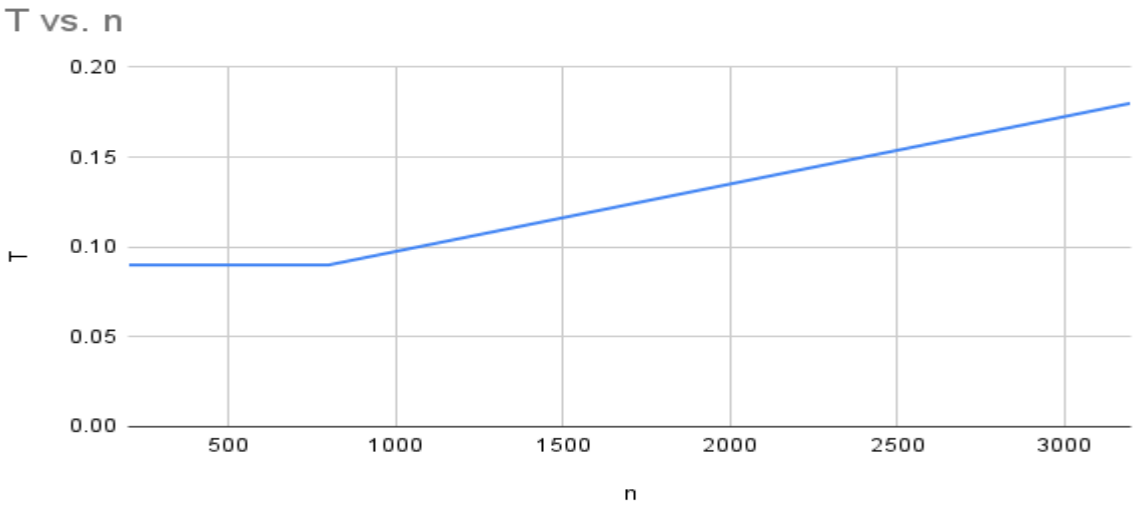
c) Tabulation of n values and Average Time for sorting Partially Ordered Array

n	T
200	0.12
400	0.22
800	0.64
1600	2.21
3200	8.82



d) Tabulation of n values and Average Time for Sorted Array

n	T
200	0.09
400	0.09
800	0.09
1600	0.12
3200	0.18



As we can see from the above tabulations and graphs, the time required for sorting the array is decreasing from Random Array to Sorted Array.