

Program Structures & Algorithms
Fall 2021
Assignment No. 2

Tasks:

1. Implemented `getClock()` function in `Timer.java` to return `System.nanoTime`
2. Implemented `toMillisecs()` function in `Timer.java` to convert nanoseconds to milliseconds
3. Implemented `repeat()` to iterate over 'n' runs and to time the execution of 'function' and return the `meanLapTime()` returned value.
4. Ran `TimerTest.java` and `BenchmarkTest.java` unit tests to test implementation (Passed)
5. Implemented `sort()` function in `InsertionSort.java` using `helper.swapStableConditional` method
6. Ran `InsertionSortTest.java` unit tests to test the implementation (Passed)
7. Implemented a main program named `Assignment2Driver.java` in `edu.neu.coe.info6205.assignment2` to be able to run the benchmarks with 4 required scenarios: random array, ordered array, reverse array, partially ordered array
8. Conducted experiments for each scenario after a warmup of 5-10 runs and recorded values
9. Tabulated readings and plotted relevant graphs

Screenshots:

Unit tests:

1. TimerTest

The screenshot displays an IDE window titled "INFO6205-Assignments - Timer.java". The main editor shows the `repeat` method in `Timer.java`, which is a generic function that repeats a given function `n` times, returning the average time per repetition. The method signature is:

```
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> function, UnaryOperator<T> preFunction, Consumer<U> postFunction) {
```

The test results panel at the bottom shows that all 10 tests passed in 2 seconds and 51 milliseconds. The tests include:

- `testPauseAndLapResume0` (174 ms)
- `testPauseAndLapResume1` (300 ms)
- `testLap` (200 ms)
- `testPause` (201 ms)
- `testStop` (100 ms)
- `testMillisecs` (100 ms)
- `testRepeat1` (159 ms)
- `testRepeat2` (205 ms)
- `testRepeat3` (511 ms)
- `testPauseAndLap` (101 ms)

The overall test summary is "Tests passed: 10 of 10 tests - 2 sec 51 ms".

2. BenchmarkTest

The screenshot shows an IDE window titled "INFO6205-Assignments - BenchmarkTest.java". The editor displays the source code for the `BenchmarkTest` class, which includes a `testWaitPeriods` method and a `GoToSleep` helper method. The `testWaitPeriods` method uses `Benchmark_Timer` to measure the execution time of a loop that calls `GoToSleep` with a delay of 100L. The `GoToSleep` method is a private static method that uses `Thread.sleep` to pause the execution for a specified duration.

The Run window at the bottom shows the execution results for the `BenchmarkTest` class. The tests passed: 2 of 2 tests - 1 sec 419 ms. The test results are as follows:

Test Name	Duration	Output
<code>testWaitPeriods</code>	1 sec 419 ms	2021-09-25 23:59:49 INFO Benchmark_Timer - Begin run: testWaitPeriods with 2 runs
<code>getWarmupRuns</code>	0 ms	Process finished with exit code 0

The status bar at the bottom indicates "Tests passed: 2 (moments ago)".

3. InsertionSortTest

The screenshot shows an IDE window titled "INFO6205-Assignments - InsertionSortTest.java". The editor displays the source code for the `InsertionSortTest` class, which includes a `sort` method and a `testStaticInsertionSort` method. The `sort` method is a public static method that sorts an array of integers using the insertion sort algorithm. The `testStaticInsertionSort` method is a public static method that tests the `sort` method with a specific array of integers.

The Run window at the bottom shows the execution results for the `InsertionSortTest` class. The tests passed: 6 of 6 tests - 143 ms. The test results are as follows:

Test Name	Duration	Output
<code>testMutatingInsertionSort</code>	88 ms	2021-09-26 00:07:41 DEBUG Config - Config.get(helper, instrument) = true
<code>sort0</code>	49 ms	2021-09-26 00:07:41 DEBUG Config - Config.get(helper, seed) = 0
<code>sort1</code>	1 ms	2021-09-26 00:07:41 DEBUG Config - Config.get(instrumenting, copies) = true
<code>sort2</code>	3 ms	2021-09-26 00:07:41 DEBUG Config - Config.get(instrumenting, swaps) = true
<code>sort3</code>	1 ms	2021-09-26 00:07:41 DEBUG Config - Config.get(instrumenting, compares) = true
<code>testStaticInsertionSort</code>	1 ms	2021-09-26 00:07:41 DEBUG Config - Config.get(instrumenting, inversions) = 1

The status bar at the bottom indicates "Tests passed: 6 (moments ago)".

Outputs

1. Random Array

```
package edu.neu.coe.info6205.assignment2;

import java.util.*;

/**
 * Run the benchmark for insertion sort
 * choice indicates the order of array. 1= random, 2= ordered, 3= reverse-ordered, 4= partially ordered
 */
public class Assignment2Driver {
    private static Config config;
    private static int N = 500;
    public static void main(String[] args) {
        int choice = 1;
        if(choice == 1){
            for(int i = 0; i < 6; i++){
                Helper<Integer> helper = new BaseHelper<>("insertion sort", N, config);
                Supplier<Integer[]> supplier = () -> helper.random(Integer.class, Random::nextInt);
                Assignment2Driver.benchmarkTarget(helper, supplier, i, "orderType: random");
                N*=2;
            }
        }
        else if(choice == 2) {
            Integer[] arr = new Integer[N];
            for(int i = 0; i < N; i++){
                arr[i] = i;
            }
        }
    }
}
```

Run: Assignment2Driver

```
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
2021-09-26 00:21:08 INFO Benchmark_Timer - Begin run: Insertion sort run: 1 for random array for 500 integers with 100 runs
0.26316616 ms
2021-09-26 00:21:08 INFO Benchmark_Timer - Begin run: Insertion sort run: 2 for random array for 1000 integers with 100 runs
0.9266157800000001 ms
2021-09-26 00:21:08 INFO Benchmark_Timer - Begin run: Insertion sort run: 3 for random array for 2000 integers with 100 runs
3.6331228199999996 ms
2021-09-26 00:21:08 INFO Benchmark_Timer - Begin run: Insertion sort run: 4 for random array for 4000 integers with 100 runs
14.36563823 ms
2021-09-26 00:21:10 INFO Benchmark_Timer - Begin run: Insertion sort run: 5 for random array for 8000 integers with 100 runs
58.83978765 ms
2021-09-26 00:21:16 INFO Benchmark_Timer - Begin run: Insertion sort run: 6 for random array for 16000 integers with 100 runs
248.86508187000001 ms

Process finished with exit code 0
```

2. Ordered Array

```
package edu.neu.coe.info6205.assignment2;

import java.util.*;

/**
 * Run the benchmark for insertion sort
 * choice indicates the order of array. 1= random, 2= ordered, 3= reverse-ordered, 4= partially ordered
 */
public class Assignment2Driver {
    private static Config config;
    private static int N = 500;
    public static void main(String[] args) {
        int choice = 2;
        if(choice == 1){
            for(int i = 0; i < 6; i++){
                Helper<Integer> helper = new BaseHelper<>("insertion sort", N, config);
                Supplier<Integer[]> supplier = () -> helper.random(Integer.class, Random::nextInt);
                Assignment2Driver.benchmarkTarget(helper, supplier, i, "orderType: random");
                N*=2;
            }
        }
        else if(choice == 2) {
            Integer[] arr = new Integer[N];
            for(int i = 0; i < N; i++){
                arr[i] = i;
            }
        }
    }
}
```

Run: Assignment2Driver

```
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
2021-09-26 00:24:40 INFO Benchmark_Timer - Begin run: Insertion sort run: 1 for ordered array for 500 integers with 100 runs
0.02497608 ms
2021-09-26 00:24:40 INFO Benchmark_Timer - Begin run: Insertion sort run: 2 for ordered array for 1000 integers with 100 runs
0.021207319999999998 ms
2021-09-26 00:24:40 INFO Benchmark_Timer - Begin run: Insertion sort run: 3 for ordered array for 2000 integers with 100 runs
0.03278076 ms
2021-09-26 00:24:40 INFO Benchmark_Timer - Begin run: Insertion sort run: 4 for ordered array for 4000 integers with 100 runs
0.00917586 ms
2021-09-26 00:24:40 INFO Benchmark_Timer - Begin run: Insertion sort run: 5 for ordered array for 8000 integers with 100 runs
0.01274484 ms
2021-09-26 00:24:40 INFO Benchmark_Timer - Begin run: Insertion sort run: 6 for ordered array for 16000 integers with 100 runs
0.02522791 ms

Process finished with exit code 0
```

3. Reverse Array

```
17  * choice indicates the order of array. 1= random, 2= ordered, 3= reverse-ordered, 4= partially ordered
18  */
19
20  public class Assignment2Driver {
21      private static Config config;
22      private static int N = 500;
23      public static void main(String[] args) {
24          int choice = 3;
25          if(choice == 1){
26              for(int i = 0; i < 6; i++){
27                  Helper<Integer> helper = new BaseHelper<>("insertion sort", N, config);
28                  Supplier<Integer[]> supplier = () -> helper.random(Integer.class, Random::nextInt);
29                  Assignment2Driver.benchmarkTarget(helper, supplier, i, orderType: "random");
30                  N*=2;
31              }
32          }
33          else if(choice == 2) {
34              Integer[] arr = new Integer[N];
35              for(int i = 0; i < 6; i++){
36                  Helper<Integer> helper = new BaseHelper<>("insertion sort", N, config);
37                  Supplier<Integer[]> supplier = () -> generateOrderedArray(N, asc: true);
38                  Assignment2Driver.benchmarkTarget(helper, supplier, i, orderType: "ordered");
39                  N*=2;
40              }
41          }
42      }
43  }
```

Run: Assignment2Driver

```
2021-09-26 00:26:47 INFO Benchmark_Timer - Begin run: Insertion sort run: 1 for reverse array for 500 integers with 100 runs
0.46865851999999997 ms
2021-09-26 00:26:47 INFO Benchmark_Timer - Begin run: Insertion sort run: 2 for reverse array for 1000 integers with 100 runs
1.81589446 ms
2021-09-26 00:26:48 INFO Benchmark_Timer - Begin run: Insertion sort run: 3 for reverse array for 2000 integers with 100 runs
7.1812128600000005 ms
2021-09-26 00:26:48 INFO Benchmark_Timer - Begin run: Insertion sort run: 4 for reverse array for 4000 integers with 100 runs
28.20000735 ms
2021-09-26 00:26:51 INFO Benchmark_Timer - Begin run: Insertion sort run: 5 for reverse array for 8000 integers with 100 runs
113.278841269999999 ms
2021-09-26 00:27:04 INFO Benchmark_Timer - Begin run: Insertion sort run: 6 for reverse array for 16000 integers with 100 runs
453.86547511 ms
Process finished with exit code 0
```

4. Partially Ordered Array

```
1 package edu.neu.coe.info6205.assignment2;
2
3 import java.util.*;
4
5 /**
6  * Run the benchmark for insertion sort
7  *
8  * choice indicates the order of array. 1= random, 2= ordered, 3= reverse-ordered, 4= partially ordered
9  */
10
11 public class Assignment2Driver {
12     private static Config config;
13     private static int N = 500;
14     public static void main(String[] args) {
15         int choice = 4;
16         if(choice == 1){
17             for(int i = 0; i < 6; i++){
18                 Helper<Integer> helper = new BaseHelper<>("insertion sort", N, config);
19                 Supplier<Integer[]> supplier = () -> helper.random(Integer.class, Random::nextInt);
20                 Assignment2Driver.benchmarkTarget(helper, supplier, i, orderType: "random");
21             }
22         }
23     }
24 }
```

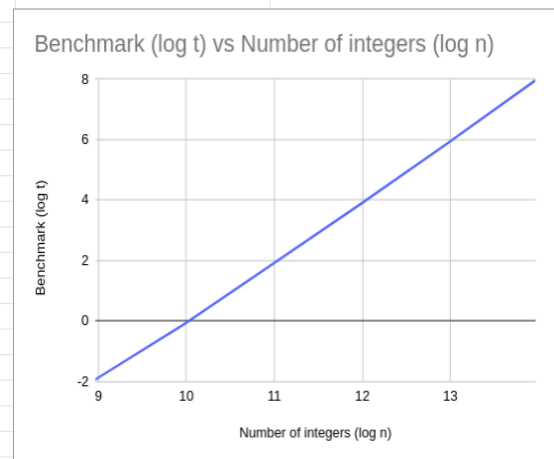
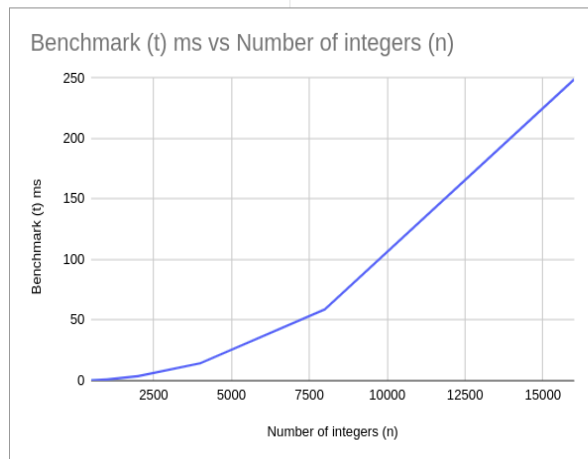
Run: Assignment2Driver

```
2021-09-26 20:52:24 INFO Benchmark_Timer - Begin run: Insertion sort run: 1 for partially ordered array for 500 integers with 100 runs
0.27578509 ms
2021-09-26 20:52:24 INFO Benchmark_Timer - Begin run: Insertion sort run: 2 for partially ordered array for 1000 integers with 100 runs
1.0010526 ms
2021-09-26 20:52:24 INFO Benchmark_Timer - Begin run: Insertion sort run: 3 for partially ordered array for 2000 integers with 100 runs
3.98963145 ms
2021-09-26 20:52:24 INFO Benchmark_Timer - Begin run: Insertion sort run: 4 for partially ordered array for 4000 integers with 100 runs
16.27799137 ms
2021-09-26 20:52:26 INFO Benchmark_Timer - Begin run: Insertion sort run: 5 for partially ordered array for 8000 integers with 100 runs
66.56569241 ms
2021-09-26 20:52:33 INFO Benchmark_Timer - Begin run: Insertion sort run: 6 for partially ordered array for 16000 integers with 100 runs
261.97588743 ms
Process finished with exit code 0
```

Tabulation and graphs: (t vs n[left] & log t vs log n[right])

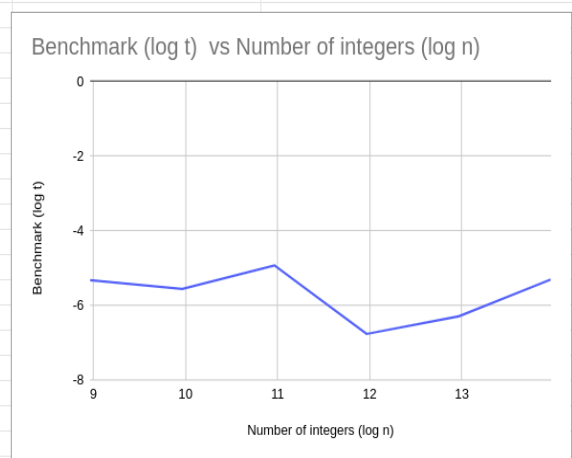
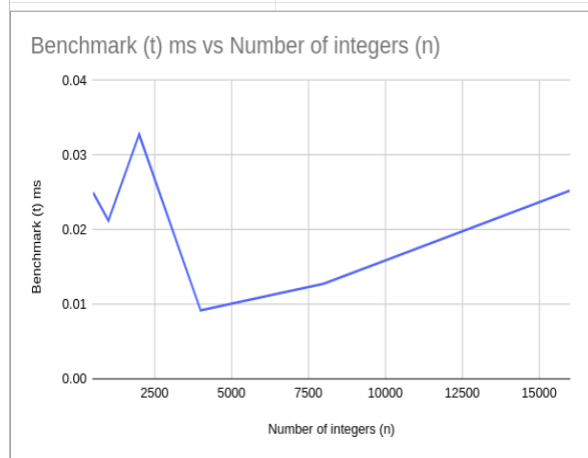
1. RandomArray

Number of integers (n)	Benchmark (t) ms	Number of integers (log n)	Benchmark (log t)
500	0.26316616	8.965784285	-1.925954107
1000	0.92661578	9.965784285	-0.1099568436
2000	3.63312282	10.96578428	1.861210138
4000	14.36563823	11.96578428	3.844550185
8000	58.83978765	12.96578428	5.878720135
16000	248.8650019	13.96578428	7.959219547



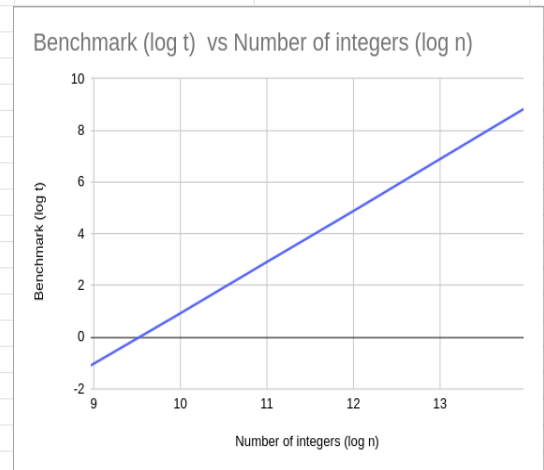
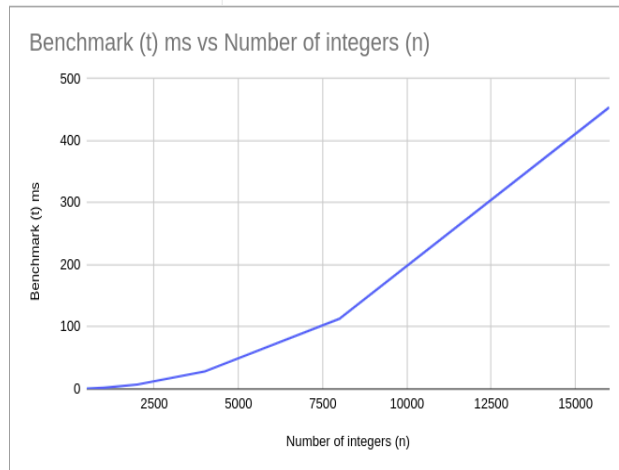
2. Ordered Array

A	B	C	D	E
Number of integers (n)	Benchmark (t) ms		Number of integers (log n)	Benchmark (log t)
500	0.02497608		8.965784285	-5.323309126
1000	0.02120732		9.965784285	-5.559293873
2000	0.03278076		10.96578428	-4.931006887
4000	0.00917506		11.96578428	-6.768066692
8000	0.01274484		12.96578428	-6.293942928
16000	0.02522791		13.96578428	-5.308835498



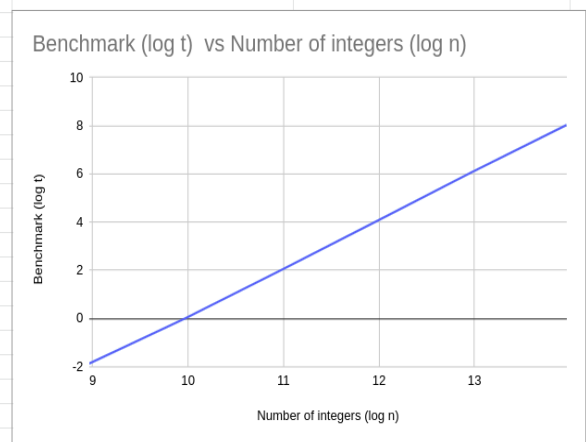
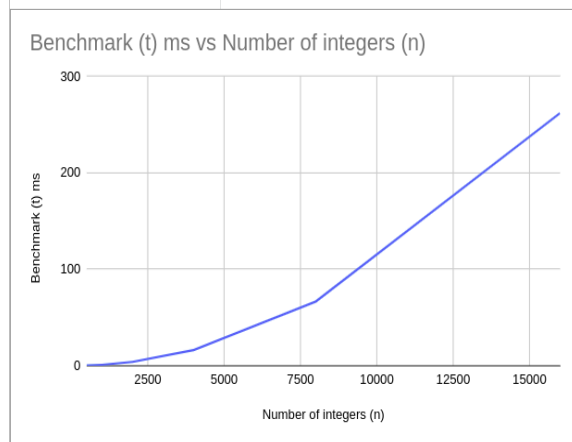
3. Reverse Array

A	B	C	D	E
Number of integers (n)	Benchmark (t) ms		Number of integers (log n)	Benchmark (log t)
500	0.46865852		8.965784285	-1.093390984
1000	1.81589446		9.965784285	0.8606803555
2000	7.18121286		10.96578428	2.844227526
4000	28.20000735		11.96578428	4.817623634
8000	113.2788413		12.96578428	6.823734603
16000	453.8654751		13.96578428	8.826120939

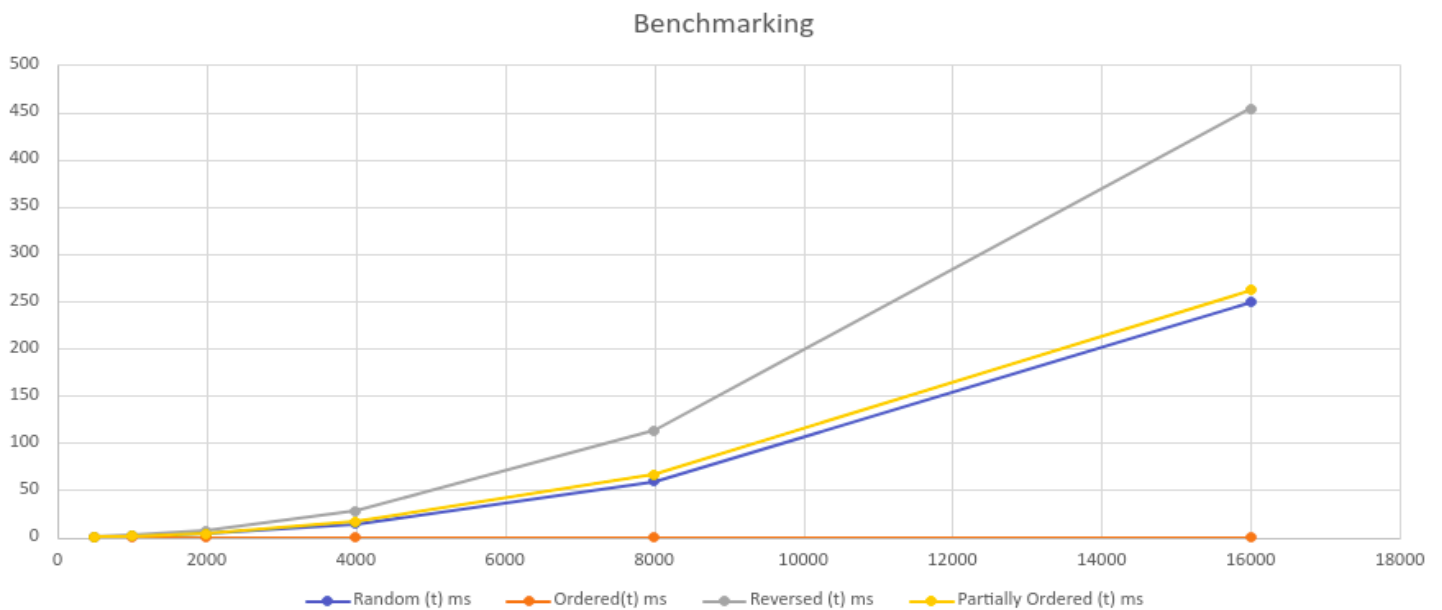


4. Partially Ordered Array

A	B	C	D	E
Number of integers (n)	Benchmark (t) ms		Number of integers (log n)	Benchmark (log t)
500	0.27578509		8.965784285	-1.858383634
1000	1.0010526		9.965784285	0.001517782131
2000	3.98963145		10.96578428	1.996255481
4000	16.27799137		11.96578428	4.024850783
8000	66.56569241		12.96578428	6.056706907
16000	261.9750874		13.96578428	8.033285815



Consolidated Graph:



Conclusions:

1. An ordered array has the fastest benchmark time by a long margin.
2. The general trend of the log graphs indicates the order of growth for Partially Ordered array and Random Array is closer to $O(n \log n)$.
3. The trend for Reverse ordered array indicates order of growth is closer to $O(n^2)$
4. The trend for Ordered Array indicates order of growth is closer to $O(1)$
5. The benchmarks in ascending order of time are Ordered, Partially Ordered, Random and Reverse order