**Timer class Enhancements and Insertion Sort Analysis**

**Task:**

1. Implement 3 methods of the Timer class: repeat(), getClock(), toMillisecs() functions and run benchmark test and timer test.
2. Implement Insertion sort code inside Insertion.sort() with the helper function helper.swap and run InsertionSortTest.
3. Implement a main program with insertion sort for four types of the array and drawing conclusions of my observations regarding the order of growth:-
4. Random
5. Sorted
6. Partially-sorted
7. Reverse-sorted

**Part 1:** Implemented following methods to pass all test cases in BenchmarkTest & TimerTest().

1. Repeat()
2. getClock()
3. toMillisecs()

Text

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

BenchmarkTest - Passed all test cases

Text

Description automatically generated

TimerTest - Passed all test cases

A screenshot of a computer

Description automatically generated with medium confidence

**Part 2** - Implement Insertion Sort code using Helper.swap() function and pass all test cases for InsertionSortTest.

Text

Description automatically generated

InsertionSortTest - all test cases passed

A screenshot of a computer

Description automatically generated with medium confidence

**Part 3:** Implement a main program with insertion sort for four types of the array and drawing conclusions of my observations regarding the order of growth.

**Output for BenchMarkDriver.java (Main method implementation):**

**Text

Description automatically generated**

**Graph showing observed relationship:**

**Table showing final avg time recorded with observations:**

|  |  |
| --- | --- |
| **Array Type** | **Time taken (ms)** |
| Sorted array | 0.150386456 |
| Partially sorted array | 0.1902602 |
| Random array | 0.208063326 |
| Reverse sorted | 4.577925873 |

**All observed values for running test case 5 times for varied lengths and finding avg time taken (ms):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Type** | **Array length** | **Run** | **Time 1** | **Time 2** | **Time 3** | **Time 4** | **Time 5** | **Avg time taken (ms)** |
| Sorted array | 1000 | 500 | 0.00849 | 0.00745 | 0.00524 | 0.00545 | 0.00651 | 0.00662732 |
| Partially sorted array | 1000 | 500 | 0.007183 | 0.0072 | 0.00537 | 0.00544 | 0.00643 | 0.006324858 |
| Random array | 1000 | 500 | 0.008962 | 0.0082 | 0.00535 | 0.01313 | 0.01314 | 0.009756469 |
| Reverse array | 1000 | 500 | 0.02975 | 0.02912 | 0.00655 | 0.00645 | 0.0057 | 0.015514159 |
| Sorted array | 2000 | 500 | 0.011275 | 0.01053 | 0.01075 | 0.01045 | 0.00945 | 0.010490732 |
| Partially sorted array | 2000 | 500 | 0.010405 | 0.01054 | 0.01066 | 0.01057 | 0.00944 | 0.010324286 |
| Random array | 2000 | 500 | 0.010546 | 0.01039 | 0.01029 | 0.01046 | 0.00949 | 0.010235504 |
| Reverse array | 2000 | 500 | 0.011355 | 0.01122 | 0.01151 | 0.01197 | 0.00992 | 0.011195595 |
| Sorted array | 4000 | 500 | 0.022947 | 0.02058 | 0.02225 | 0.02009 | 0.02235 | 0.021641388 |
| Partially sorted array | 4000 | 500 | 0.0226 | 0.01945 | 0.02242 | 0.01987 | 0.02188 | 0.021242944 |
| Random array | 4000 | 500 | 0.021274 | 0.01981 | 0.02268 | 0.01997 | 0.02248 | 0.021242277 |
| Reverse array | 4000 | 500 | 0.02253 | 0.02073 | 0.02086 | 0.02154 | 0.02295 | 0.021723607 |
| Sorted array | 8000 | 500 | 0.040695 | 0.05534 | 0.05292 | 0.04433 | 0.04862 | 0.048379867 |
| Partially sorted array | 8000 | 500 | 0.062092 | 0.04712 | 0.05835 | 0.04476 | 0.04966 | 0.05239742 |
| Random array | 8000 | 500 | 0.044958 | 0.06685 | 0.05086 | 0.04677 | 0.04461 | 0.050811034 |
| Reverse array | 8000 | 500 | 0.056709 | 0.04335 | 0.04937 | 0.03993 | 0.04429 | 0.04672983 |
| Sorted array | 12000 | 500 | 0.077713 | 0.0802 | 0.08101 | 0.07974 | 0.07478 | 0.078688205 |
| Partially sorted array | 12000 | 500 | 0.083552 | 0.08296 | 0.08418 | 0.08309 | 0.07766 | 0.41144754 |
| Random array | 12000 | 500 | 0.077317 | 0.07999 | 0.08068 | 0.08672 | 0.07451 | 0.399220038 |
| Reverse array | 12000 | 500 | 0.078993 | 0.08124 | 0.08107 | 0.08201 | 0.0708 | 0.394115654 |
| Sorted array | 24000 | 500 | 0.148173 | 0.13489 | 0.15526 | 0.14538 | 0.15278 | 0.736491226 |
| Partially sorted array | 24000 | 500 | 0.150804 | 0.15543 | 0.18546 | 0.14538 | 0.15356 | 0.639824152 |
| Random array | 24000 | 500 | 0.156646 | 0.13784 | 0.15634 | 0.15255 | 0.15374 | 0.757114636 |
| Reverse array | 24000 | 500 | 0.152784 | 0.14938 | 0.19022 | 0.15329 | 0.15265 | 0.79833201 |

**Conclusion:** From the above noted observations and the bar chart, we can conclude that Insertion Sort takes the longest to sort a reverse sorted array whereas the least amount of time for an already sorted array. A partially sorted array takes the 2nd longest time and random array takes 3rd longest to sort.

**The order of sorting time is as follows:**

Sorted array < Partially sorted array < Random array < Reverse array