Akash Bharadwaj Karthik (002787011)

**Program Structures & Algorithms**

**Spring 2023 (Sec -8)**

**Assignment No. 3**

**Task**

* (Part 1) You are to implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) of a class called *Timer*. Also run the Benchmark\_Timer and Timer unit tests.
* (Part 2) Implement *InsertionSort* (in the *InsertionSort* class) by simply looking up the insertion code used by *Arrays.sort. Also runs its unit tests.*
* (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.

**Relationship Conclusion:**

The order of growth of the running time of Insertion Sort (Randomly ordered array of size *N*) is found to be .

The order of growth of the running time of Insertion Sort (Ordered array of size *N*) is found to be .

The order of growth of the running time of Insertion Sort (Partially ordered array of size *N*) is .

The order of growth of the running time of Insertion Sort (Reverse ordered array of size *N*) is .

Insertion Sort Time Complexity in order of increasing growth rates can be classified based on the condition of the array as follows:

**Evidence to support the conclusion:**

A table of values recording array size - ‘N’, number of runs - ‘runs’, and time in milliseconds - ‘time’, is observed for insertion sort on each type of array, and slope from log-log plot is generated to approximate the time complexity for each input scenario:

We already know from theoretical analysis of insertion sort that only polynomial terms are involved in its time complexity, and hence that ) is of the form:

Where,

N = input size

a = machine dependent constant

b = slope of the log-log scale graph

Therefore, the slope of the lg-lg plot will be used to approximate the power ‘b’.

Values of ‘N’ are doubled, and increment as follows: 1000,2000,4000,8000,16000.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | Runs | **Randomly Ordered** | | | |
|  |  | Time(milliseconds) | lg(Time) | lg(N) | Slope |
| 1000 | 30 | 2.64 | 1.40053793 | 9.965784285 | - |
| 2000 | 30 | 5.78 | 2.531069493 | 10.96578428 | 1.130531563 |
| 4000 | 30 | 24.06 | 4.588564737 | 11.96578428 | 2.057495245 |
| 8000 | 30 | 85.39 | 6.415995221 | 12.96578428 | 1.827430484 |
| 16000 | 30 | 548.4 | 9.099084761 | 13.96578428 | 2.68308954 |

**Analysis of experimental data (the running time of insertion sort with random ordered input)**

**lg(N)-lg(Time) plot for insertion sort on randomly ordered array**

**Hence, for randomly sorted array : T(n) since the slope is approximately 1.92**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | Runs | **Ordered** | | | |
|  |  | Time(milliseconds) | lg(Time) | lg(N) | Slope |
| 1000 | 30 | 0.011 | -6.506352666 | 9.965784285 | - |
| 2000 | 30 | 0.021 | -5.573466862 | 10.96578428 | 0.932885804 |
| 4000 | 30 | 0.043 | -4.53951953 | 11.96578428 | 1.033947332 |
| 8000 | 30 | 0.098 | -3.351074441 | 12.96578428 | 1.188445089 |
| 16000 | 30 | 0.22 | -2.184424571 | 13.96578428 | 1.166649869 |

**Analysis of experimental data (the running time of insertion sort with an ordered input)**

**lg(N)-lg(Time) plot for insertion sort on ordered array**

**Hence, for array in order : T(n) since the slope is approximately 1.08**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | Runs | **Partially Ordered** | | | |
|  |  | Time(milliseconds) | lg(Time) | lg(N) | Slope |
| 1000 | 30 | 1.42 | 0.50589093 | 9.965784285 | - |
| 2000 | 30 | 5.47 | 2.451540833 | 10.96578428 | 1.945649903 |
| 4000 | 30 | 24.32 | 4.604071324 | 11.96578428 | 2.152530491 |
| 8000 | 30 | 76.95 | 6.265849421 | 12.96578428 | 1.661778098 |
| 16000 | 30 | 277.38 | 8.115719958 | 13.96578428 | 1.849870537 |

**Analysis of experimental data (the running time of insertion sort with partially ordered input)**

**lg(N)-lg(Time) plot for insertion sort on partially ordered array**

**Hence, for partially sorted array : T(n) since the slope is approximately 1.90**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | Runs | **Reverse Ordered** | | | |
|  |  | Time(milliseconds) | lg(Time) | lg(N) | Slope |
| 1000 | 30 | 3.64 | 1.86393845 | 9.965784285 | - |
| 2000 | 30 | 17.43 | 4.123500664 | 10.96578428 | 2.259562214 |
| 4000 | 30 | 79.09 | 6.305423389 | 11.96578428 | 2.181922725 |
| 8000 | 30 | 296.8 | 8.213347282 | 12.96578428 | 1.907923892 |
| 16000 | 30 | 1291.54 | 10.33487661 | 13.96578428 | 2.121529328 |

**Analysis of experimental data (the running time of insertion sort with reverse ordered input)**

**lg(N)-lg(Time) plot for insertion sort on reverse ordered array**

**Hence, for reverse ordered array : T(n) since the slope is approximately 2.10**

**Analysis:**

Insertion Sort with random sorted data:

We expect that there will be an average of inversions when elements are randomly ordered, and hence, insertion sort will need to make at least compares, and so we expect a quadratic time complexity. The time complexity term observed scaling with is a reasonable figure close to quadratic.

Insertion Sort with ordered data:

We expect that there will be N – 1 compares and 0 swaps when the data is completely ordered, hence a linear time complexity is expected; the time complexity term observed scaling with is a reasonable figure close to linear.

Insertion Sort with partially ordered data:

We expect that there will be half the number of inversion of the average case of randomly ordered data, and hence a quadratic expression for time complexity is still expected. The time complexity term observed scaling with is a reasonable figure close to quadratic.

Insertion Sort with reverse ordered data:

We expect that there will maximum number of comparisons ( and swaps, and hence this is likely to be the highest growing time complexity term and also asymptotically quadratic. The time complexity term observed scaling with is a reasonable figure close to quadratic.

**Console Output:**

**Text

Description automatically generated**

Text

Description automatically generated

**Unit Tests:**

TimerTest.java

Text

Description automatically generated

Text

Description automatically generated

BenchmarkTest.java

Text

Description automatically generated

Text

Description automatically generated

InsertionSortTest.java

Text

Description automatically generated

Text

Description automatically generated