Program Structures and Algorithms

Spring 2023(SEC 01)

NAME: Shivani Datar

NUID: 002772160

**Task:**

1. Implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) of a class called Timer.
2. Implement InsertionSort(in the InsertionSort class) by simply looking up the insertion code used byArrays.sort*.* If you have the *instrument = true* setting in *test/resources/config.ini*, then you will need to use the *helper* methods for comparing and swapping (so that they properly count the number of swaps/compares). The easiest is to use the helper.swapStableConditional method, continuing if it returns true, otherwise breaking the loop. Alternatively, if you are not using instrumenting, then you can write (or copy) your own compare/swap code. Either way, you must run the unit tests in InsertionSortTest.
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. I suggest that your arrays to be sorted are of type *Integer*. Use the doubling method for choosing *n*and test for at least five values of *n.*Draw any conclusions from your observations regarding the order of growth.

**Relationship Conclusion:**

For implementing the benchmarking for the insertion sort, I have added a main method inside the Benchmark\_Timer class. Here we are creating objects for Random class and InsertionSort class. We first create a List and then convert it to Array and apply the insertion sort logic on it. We will execute this from N = 250 to N = 16000(using doubling method, that is multiplying N by 2).

Text

Description automatically generated

For the random array I used rand.nextInt() for creating random Integers in the list. For the ordered array I have add ordered elements from 1 to N in the list. For reverse array from N to 1 added elements in the reverse order.

Text

Description automatically generated

Lastly for the partially ordered array, for the first half of array, it has randomly generated integers and for the second half it has ordered array.Text

Description automatically generated

Following is the code snippet implementing the Insertion sort algorithm :-

Here I am using 2 methods from the helper interface. We are given the from and to points; in between we need to implement the insertion sort. We compare a xs[j] with xs[j-1], if xs[j] is less than xs[j-1] it means the array is not sorted and I am swapping the elements.

Text

Description automatically generated

Following are the timings taken by different types of array :-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Random | Ordered | Reverse | Partial |
| 250 | 1.0855876 | 0.0059414 | 0.3526124 | 0.1133542 |
| 500 | 0.2844707 | 0.0037336 | 0.5390208 | 0.1603084 |
| 1000 | 1.0477832 | 0.0154541 | 2.0889583 | 0.4991041 |
| 2000 | 4.1703041 | 0.0125042 | 8.2576502 | 2.124529 |
| 4000 | 16.5528458 | 0.026325 | 38.4466748 | 8.2852959 |
| 8000 | 66.4618998 | 0.047554 | 152.741333 | 32.8193914 |
| 16000 | 292.428913 | 0.0903001 | 612.394675 | 137.337408 |

The graph below shows that insertion sort takes the most time when the array is reversely sorted because it needs to perform swapping for every subsequent element in the array, which is the worst case scenario for an insertion sort algorithm. It also takes the least amount of time when the array is already sorted.

Chart, line chart

Description automatically generated

Hence for 4 types of array we can conclude following relation:-

t (Ordered Array) < t (Partially Ordered) < t (Random Array) < t (Reverse Array)

**Evidence to support that conclusion:**

Output for the timings of all 4 types of array :-

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

**Graphical Representation:**

Graph for Random Array :-

Observations:-

|  |  |
| --- | --- |
| N | Random |
| 250 | 1.0855876 |
| 500 | 0.2844707 |
| 1000 | 1.0477832 |
| 2000 | 4.1703041 |
| 4000 | 16.5528458 |
| 8000 | 66.4618998 |
| 16000 | 292.428913 |

Chart, line chart

Description automatically generated

Graph for Ordered Array:-

Observations:-

|  |  |
| --- | --- |
| N | Ordered |
| 250 | 0.0059414 |
| 500 | 0.0037336 |
| 1000 | 0.0154541 |
| 2000 | 0.0125042 |
| 4000 | 0.026325 |
| 8000 | 0.047554 |
| 16000 | 0.0903001 |

Chart, line chart

Description automatically generated

Graph for Reverse Array:-

Observations:-

|  |  |
| --- | --- |
| N | Reverse |
| 250 | 0.3526124 |
| 500 | 0.5390208 |
| 1000 | 2.0889583 |
| 2000 | 8.2576502 |
| 4000 | 38.4466748 |
| 8000 | 152.741333 |
| 16000 | 612.394675 |

Chart, line chart

Description automatically generated

Graph for Partially Ordered Array:-

|  |  |
| --- | --- |
| N | Partial |
| 250 | 0.1133542 |
| 500 | 0.1603084 |
| 1000 | 0.4991041 |
| 2000 | 2.124529 |
| 4000 | 8.2852959 |
| 8000 | 32.8193914 |
| 16000 | 137.337408 |

Chart, line chart

Description automatically generated

Graph For All array comparison: -

Chart, line chart

Description automatically generated

**Unit Test Screenshots:**

Screenshot for timer test cases running :-

Graphical user interface, text

Description automatically generated

Screenshot for BenchMark\_Timer testcases running :-

Graphical user interface, text

Description automatically generated

Screenshot for InsertionSort testcases running :-

Text

Description automatically generated