Data Structures and Algorithms, Spring '23 section 1 Dr. Hristescu
Lab 9

Assigned: Thursday, March 30
Due: 10pm Monday, April 3

Solutions to problems **different than specified in the description** of this lab or that **do not implement the methods as described in class** will not receive any credit. Follow the instructions carefully.

Problem 1: Design, implement and test the following sorting methods that works with ***integer* (i.e. type int)** data and sorts the data in **ascending/increasing order**. You **HAVE TO** use an **array** to store this data. Assume the items are unique.
In each of your methods you need to keep track of the following *counters* and *display* them at the end of the method:

- the **number of item comparisons** and
- the **number of swaps/shifts** (where applicable)

that your method performs on the considered input sequences.

Develop a testing suite that tests your methods on best case, worst case and randomly chosen input. Clearly **label** the **worst** and **best** case input sequences for each method.
Compare the theoretical analysis we have done in class with the results you get from running your code on various input sequences.
Problem 1: Implement these methods **as presented in class**.

A. Bubble Sort
B. Improved Bubble Sort
C. Selection Sort
D. Improved Selection Sort*
E. Insertion Sort (**shifting**, not swapping) (grab+sequential(Search&Shift)+drop)
Note: The Search and the Shift can be done either concomitantly or separately (Search followed by Shift)

Extra Credit:

F. Improved **Insertion Sort (grab+BinarySearch+Shift+drop)**
Note: The BinarySearch has to be performed before the Shifting

//End Extra Credit

Problem 2: Analyze and express the time complexity of Improved Insertion Sort using the big-O notation.

Input data: number of integers, input data specified one integer at a time.
Note that the worst case input sequence may not be the same for all the methods.

Make sure that:

- All methods sort in **increasing/ascending order.**
- All methods work **the way they were described in class** (not textbook, internet or other external source).
- For all methods the largest **number of item comparisons** is All Pairwise (**AP**) on any possible input and not more.
- Any array index or additional bookkeeping counter comparisons are not included in the item comparison count.
- Methods do not pass over any known already sorted part of the array.
- If your testing calls several methods that your counters are reset between calls.

*Hint for 1. D. (ImprovedSelectionSort): finding the index of the largest item in the unsorted part of the collection can be done using several different methodical approaches. Some

approaches may be more efficient than others at detecting if the collection is already in sorted order. Reminder that index comparison should be used for this as no **additional** item comparisons are allowed for this bookkeeping.

Use the display format shown in the sample run below. Feel free to organize your code either in separate drivers for the individual sorting methods or using a menu-driven driver with methods named **BubbleSort, ImprovedBubbleSort, SelectionSort, ImprovedSelectionSort, InsertionSort,** [EC: **ImprovedInsertionSort**].
Example of a sample run: Input read in in red

```
elvis %java BubbleSort
Enter number of integers: 6
Enter integer number 0: 1
Enter integer number 1: 0
Enter integer number 2: 5
Enter integer number 3: 2
Enter integer number 4: 3
Enter integer number 5: 4

Input data:
1    0    5    2    3    4
Sorted data:
0    1    2    3    4    5
Number of comparisons: 15
Number of swaps: 4


elvis %java BubbleSort
Enter number of integers: 6
Enter integer number 0: 0
Enter integer number 1: 1
Enter integer number 2: 2
Enter integer number 3: 3
Enter integer number 4: 4
Enter integer number 5: 5

Input data:
0    1    2    3    4    5
Sorted data:
0    1    2    3    4    5
Number of comparisons: 15
Number of swaps: 0

elvis %java BubbleSort
Enter number of integers: 6
Enter integer number 0: 5
Enter integer number 1: 4
Enter integer number 2: 3
Enter integer number 3: 2
Enter integer number 4: 1
Enter integer number 5: 0

Input data:
5    4    3    2    1    0
Sorted data:
0    1    2    3    4    5
Number of comparisons: 15
Number of swaps: 15
```

Problem3: Lab9 Conclusions: Summarize in a few sentences what you have learned from working on this lab.

Submit: Follow the instructions to turn in the lab electronically using **Lab9** for the name of your directory.
Develop your own extensive input suite and test your solutions thoroughly on it. Submit all the code developed for this lab and all sample runs on the input suite you developed.