CST238 Fall 2019

Homework 08 - Sorted, Folded Numbers and Sorting Queues

Part 0. For 2 points extra credit, complete the FYS survey, and note in your header for Part 1 that you did it:
FYS 156 Survey for CS Students

Part 1. (15 points) Write a program that will sort numbers and print them in folded order (defined below)
Implement the following functions:

```
// reads a file and returns an array of integers; first number is # of elements in file
int * readInts(string filename, int & size);

// uses insertion sort to sort number in ascending order
void sort(int * a, int size);

// prints a sorted array in folded format
void printFolded(int * a, int size);
```

1. Start by reading in files, as in the examples below
2. Sort the numbers in ascending order
3. Print the numbers in folded format; rather than printing 1, 2, 3, you print 1-3 *if the numbers are continuous*

Example Data	Sample Output
6 51 27 53 77 52 75	Enter filename: f1.txt 27 51-53 75 77
7 26 51 27 53 77 52 75	Enter filename: f2.txt 26-27 51-53 75 77
10 4 5 6 1 2 3 7 9 10 11	Enter filename: f3.txt 1-7 9-11

Submit this file as main.cpp

Part 2. (25 points)
All three sorting methods we've seen so far (bubble, insertion, selection) have the same Big-O, $O(n^2)$. However, they perform differently, not because of magic, but because they use different numbers of swaps and comparisons. We're going to compare the techniques for time, comparisons, and swaps.

1. Fork the Starting Code - Sorting Comparison from iLearn

2. The main.cpp and mysorts.h files are complete (unless you do the extra credit), so you will only modify mysort.cpp

3. Implement the following functions one at a time, in this order, testing throughout:

```
// copies from the source array into a new (hint, hint) array
int * copyArray(int * values, int size);

// determines if a given array is sorted in ascending order
bool isSorted(int * values, int size);

// bubble sorts the given array
void bubble_sort (int * values, int size);

// insertion sorts the given array
void insertion_sort (int * values, int size);

// selection sorts the given array
void selection_sort (int * values, int size);
```

4. Run the code, using ten_numbers.txt and sorted_numbers.txt for testing purposes

5. Verify that your code works correctly, using Examples 1-3 below

Extra Credit: In Example 3 below, insertion sort is significantly slower than selection sort. This shouldn't be the case, but we're using a simplified version that swaps too much! Instead of swapping, we should:

1. Store the value at the current index of the outer loop in a temp variable
2. Shift elements to the right until we find the place for that value
3. Write that value to the correct location

The Wikipedia page for insertion sort explains this (look for the second algorithm example). Since we only do about 1/3 of the work, we can divide swaps by 3.
For up to 3 points of extra credit, implement this approach *as a new function*, so there are four sorts. You must

- Modify main.cpp and the mysorts.h files to add a fast_insertion_sort function
- In the main function, make another copy of the array, and add the call to fast_insertion sort
- Implement fast_insertion sort
- NOTE CLEARLY in your mysorts.cpp file that you've implemented the extra credit

Submit *only the* mysorts.cpp file, which should contain your header information.

Example 1

```
Enter filename: ten_numbers.txt
Start the bubble sorting...
Bubble sort completed. Is sorted: true
Elapsed time: 4e-06 seconds
Swaps: 22
Comps: 45

Start the insertion sorting...
Insertion sort completed. Is sorted: true
Elapsed time: 2e-06 seconds
Swaps: 22
Comps: 29

Start the selection sorting...
Selection sort completed. Is sorted: true
Elapsed time: 2e-06 seconds
Swaps: 6
Comps: 45
```

Example 2

```
Enter filename: sorted_numbers.txt
Start the bubble sorting...
Bubble sort completed. Is sorted: true
Elapsed time: 2e-06 seconds
Swaps: 0
Comps: 45

Start the insertion sorting...
Insertion sort completed. Is sorted: true
Elapsed time: 1e-06 seconds
Swaps: 0
Comps: 9

Start the selection sorting...
Selection sort completed. Is sorted: true
Elapsed time: 2e-06 seconds
Swaps: 0
Comps: 45
```

Example 3

```
Enter filename: ten_thousand_numbers.txt
Start the bubble sorting...
Bubble sort completed. Is sorted: true
Elapsed time: 0.615245 seconds
Swaps: 25320039
Comps: 49995000

Start the insertion sorting...
Insertion sort completed. Is sorted: true
Elapsed time: 0.351148 seconds
Swaps: 25320039
Comps: 25330030

Start the selection sorting...
Selection sort completed. Is sorted: true
Elapsed time: 0.169737 seconds
Swaps: 9985
Comps: 49995000
```

Example 4

```
EEnter filename: ten_thousand_numbers.txt
Start the bubble sorting...
Bubble sort completed. Is sorted: true
Elapsed time: 0.677852 seconds
Swaps: 25320039
Comps: 49995000

Start the insertion sorting...
Insertion sort completed. Is sorted: true
Elapsed time: 0.384324 seconds
Swaps: 25320039
Comps: 25330030

Start the fast insertion sorting...
Fast insertion sort completed. Is sorted:
true
Elapsed time: 0.16179 seconds
Swaps: 8443346
Comps: 25330030

Start the selection sorting...
Selection sort completed. Is sorted: true
Elapsed time: 0.166085 seconds
Swaps: 9985
Comps: 49995000
```