CURTIN UNIVERSITY (CRICOS number: 00301J)

Department of Computing, Faculty of Engineering and Science

Data Structures and Algorithms (COMP1002)

PRACTICAL 2 - SORTING

AIMS

- To implement and test Bubble Sort, Insertion Sort, Selection Sort, MergeSort and Quicksort

BEFORE THE PRACTICAL:

- Read this practical sheet fully before starting.
- Copy the class Sorts from Blackboard. Use this as a starting point to writing the sorting methods

Use the data from RandomNames7000.csv to test ALL of your sorting algorithms

ACTIVITY 1: BUBBLE SORT IMPLEMENTATION

Time to write some code.

- In Sorts.java, implement the bubbleSort() method.
- Note that the method works on an int[] array. You will need to "hard code" this array in your test harness, or use the data from RandomNames7000.csv.
  - Don't forget to include a check to stop bubble sort if it doesn't do any swaps during a pass (*i.e.,* the array has finished being sorted).

ACTIVITY 2: SELECTION SORT IMPLEMENTATION

Do the same for Selection Sort as you did for Bubble sort

- Selection sort differs from bubble sort in that it only swaps *once* per pass. Instead, what it does is searches for the smallest value, updating the *index* of the smallest value until the end of the pass. Only then does it swap the smallest value with the first value.
  - Remember that in the second pass, the first value has already been sorted. So don't include the first value in the second pass.
  - The same is true for subsequent passes (ie: the third pass should ignore the first two values).

ACTIVITY 3: INSERTION SORT IMPLEMENTATION

Do the same for Insertion Sort as you did for Bubble sort

ACTIVITY 4: MERGE SORT IMPLEMENTATION

Do the same for Merge Sort as you had done for Bubble sort. MergeSort is quite a bit more complex than Bubble Sort and Selection Sort

- Note that a second, private function called mergeRecurse is available. This adds parameters for leftIdx and rightIdx to define the part of the array A that the mergeSort is dealing with during the recursive splitting. You don't physically split the array – you just 'narrow' your focus to the area bounded by leftIdx and rightIdx.
- Use the algorithm in the lectures or in the book (LaFore) to guide you in your implementation.

You will see that you need to create another function to perform the merge – so you will have: mergeSort, mergeRecurse and merge.

ACTIVITY 5: QUICKSORT IMPLEMENTATION

Add QuickSort to your Sorts.java. QuickSort requires the following broad steps:

- Select a pivot
- Re-organise the array so that all values to the left of the pivot are smaller than the pivot and all values to its right are larger than the pivot. This is called *partitioning*.
- If the array is of size more than 1, for each side (left and right) perform a recursive call to quickSort(). As with mergeSort(), define the left array via indexes and similarly for the right array. Do not include the pivot in either of the arrays – it's already in the right position.
- Else, (the array is of size 1 or less) you can return immediately because it is now sorted (base case). In your implementation, do not include this - it is a do nothing step, and is included here for clarity.

For an alternative pivot partitioning algorithm to that in the lectures, have a look at the book (LaFore). The way it works is that it starts from both sides of the array and searches inwards for any values that are on the 'wrong' side of the pivot. Upon finding one from each side, it swaps the two. However, note that LaFore selects the rightmost as the pivot – you may want to use the middlemost (just swap it with the rightmost and the rest of the algorithm is the same) to avoid the worst-case problem with rightmost (or leftmost) pivots and sorted arrays.

SUBMISSION DELIVERABLE:

Your classes and UML are due before the beginning of your next tutorial. Also include any other relevant classes that you may have created. Please only submit the *.java files.

You must submit your version of the test harness and any test data you've been using.

SUBMIT ELECTRONICALLY VIA BLACKBOARD, under the *Assessments* section.

MARKING GUIDE

Your submission will be marked as follows:

- [2] x5 Each Sort is implemented properly and tests correctly.