CURTIN UNIVERSITY (CRICOS number: 00301J)

Department of Computing, Faculty of Engineering and Science

Data Structures and Algorithms (COMP1002)

PRACTICAL 8 – ADVANCED SORTING

AIMS

- To implement and experiment with the variations of Mergesort and Quicksort

BEFORE THE PRACTICAL:

- Ensure you have completed the activities from Practical 1 and are confident with recursion before starting

ACTIVITY 1: REVISIT PRACTICAL 1

We will be extending on the sorting code from Practical 1, so as a first step, copy across the code from the first practical.

You should already have implemented:

- Bubble sort
- Insertion sort
- Selection sort

Re-familiarise yourself with running the code for the different sorts and the output it gives as an indication of the scaling of the algorithms.

ACTIVITY 2: MERGESORT IMPLEMENTATION

Using the recursive algorithm given in the lecture slides, implement MergeSort in the appropriate place in sorts.py/java.

You will need to create a wrapper method, mergesort(A), that will be called to run the sort. This method will add 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.

You will see that you need to create another function to perform the merge – so you will Have three functions: mergeSort, mergeSortRec and merge. The first is public, and the others are only used by the mergesort, so they are private. Python students should use an underscore to indicate private methods (_mergeSortRec(…) and _merge(….).

ACTIVITY 3: INITIAL QUICKSORT IMPLEMENTATION

Add QuickSort to your Sorts.java. The algorithm is provided in the lecture slides. We will implement the (not recommended) left-most pivot selection strategy in this activity.

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.

ACTIVITY 4: ALTERNATE QUICKSORT IMPLEMENTATION

Copy your QuickSort into QuickSortMedian3 and implement a **median of three** pivot selection.

Add code to SortsTestHarness to allow the function to be tested. Note that you'll be able to re-use the private methods from Activity 3 as the pivot strategy is isolated in QuickSortM.

ACTIVITY 5: 3-WAY QUICKSORT IMPLEMENTATION

Copy your QuickSort into QuickSort3way and implement a **3-way QuickSort**. Look online for a suitable algorithm, e.g. https://www.tutorialspoint.com/3-way-quicksort-dutch-national-flag

Add code to SortsTestHarness to allow the function to be tested. You should add another data option to have repeating values (i.e. not just random, ascending, descending, almost sorted – have another for repeating ("R")) .

SUBMISSION DELIVERABLE:

Your source code and any input files (all that are required for this practical) are due before the beginning of your next tutorial. Also include any other relevant classes that you may have created.

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

MARKING GUIDE

Your submission will be marked as follows:

- [2] MergeSort implementation – works with SortsTestHarness
- [2] QuickSort implementation with leftmost pivot selection
- [2] QuickSort implementation with median of three pivot selection
- [2] QuickSort 3-way implementation
- [2] Notes and discussion on the performance of the four implementations (above), showing how you've tested them against each other and with various input data sets