CURTIN UNIVERSITY (CRICOS number: 00301J)
Faculty of Engineering and Science
Department of Computing
Data Structures and Algorithms

Practical 6

Aims: Sorting, Part I

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

Before the Practical:

- Read this practical sheet fully before starting.
- Copy the class Sorts at the end of this practical worksheet to a java class file. Use this as a guide to writing the sorting methods

You will have this week and next week to get all of the sorting algorithms working.

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

Activity 1: Implementing Bubble Sort

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 (ie: the array has finished being sorted).

Activity 2: Implementing Selection Sort and Insertion Sort

Do the same for Selection Sort and Insertion Sort as you had done 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 4: Implementing MergeSort

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.

Submission Deliverable:

Your completed `sorts.java` class is due at the beginning of your tutorial 7 (after completing quicksort.

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

If you finish early, use the rest of the practical to start the next worksheet, because that will be due later on.

```
/*  Data Structures and Algorithms
** Class to hold various static sort methods.
*/
class Sorts
{
    // bubble sort
    public static void bubbleSort(int[] A)
    {
    }//bubbleSort()

    // selection sort
    public static void selectionSort(int[] A)
    {
    }// selectionSort()

    // insertion sort
    public static void insertionSort(int[] A)
    {
    }// insertionSort()

    // mergeSort - wrapper method for kick-starting the recursive algorithm
    public static void mergeSort(int[] A)
    {
    }//mergeSort()
    private static void mergeSortRecurse(int[] A, int leftIdx, int 
rightIdx)
    {
    }//mergeSortRecurse()
    private static void merge(int[] A, int leftIdx, int midIdx, int 
rightIdx)
    {
    }//merge()


    // quickSort - wrapper method for kick-starting the recursive algorithm
    public static void quickSort(int[] A)
    {
    }//quickSort()
    private static void quickSortRecurse(int[] A, int leftIdx, int 
rightIdx)
    {
    }//quickSortRecurse()
    private static int doPartitioning(int[] A, int leftIdx, int rightIdx, 
int pivotIdx)
    {
           return 0;  // TEMP - Replace this when you implement QuickSort
    }//doPartitioning


}//end Sorts class
```