Advanced Sorting

Updated: 21st July, 2023

Aims

- To implement and experiment with variations of MergeSort and QuickSort.

Before the Practical

- Read this practical sheet fully before starting.
- Ensure you have completed the activities from Practical 1 and are confident with recursion.

Activities

1. Revisit Practical 1

We will be extending on the sorting code from Practical 1. As a first step, copy across your `Sorts.java/py` and `SortsTestHarness.java/py` into your current working directory. You should have already implemented Bubble Sort, Insertion Sort and Selection Sort.

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

2. MergeSort Implementation

Using the recursive algorithm supplied in the lecture slides, implement MergeSort in the appropriate place in `Sorts.java/py`. Test it with SortsTestHarness.java/py.

Note: 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:

(a) `mergeSort() - public`

(b) `mergeSortRec() - private`

(c) `merge() - private`

3. Initial QuickSort Implementation

Using the algorithm supplied in the lecture slides, implement QuickSort in the appropriate place in `Sorts.java/py`. We will implement the (not recommended) *left-most* pivot selection strategy in this activity. Test it with `SortsTestHarness.java/py`.

Note: 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 lecture slides, have a look at the textbook (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.

4. Median of Three QuickSort Implementation

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

Add code to `SortsTestHarness.java/py` to allow the function to be tested.

Note: You will be able to re-use the private methods from Activity 3 as the pivot strategy is isolated in `QuickSortMedian3()`.

5. Random QuickSort Implementation

Copy your `QuickSort()` into `QuickSortRandom()` and implement a *random* pivot selection.

Add code to `SortsTestHarness.java/py` to allow the function to be tested.

Note: You will be able to re-use the private methods from Activity 3 as the pivot strategy is isolated in `QuickSortRandom()`.

6. Exploring Sorting Runtimes

`SortsTestHarness.java/py` lets you easily test your sorts code for various types of data.

- Create a table of runtime results, using at least four array sizes and at least three of the ascending/ descending/random/nearly sorted options across each of the implemented sorting algorithms.
- Write a paragraph discussing the results in terms of time complexity and other characteristics of the sorting algorithms.

Submission Deliverable

- Your code is due 2 weeks from your current tutorial session.
  - You will demonstrate your work to your tutors during that session
  - If you have completed the practical earlier, you can demonstrate your work during the next session
- You must **submit** your code and any test data that you have been using **electronically via Blackboard** under the *Assessments* section before your demonstration.
  - Java students, please do not submit the *.class files

Marking Guide

Your submission will be marked as follows:

- [4] MergeSort implementation
- [4] QuickSort implementation with left-most pivot selection
- [4] QuickSort implementation with median of three pivot selection
- [4] QuickSort implementation with random pivot selection
- [4] Sorts performance investigation

End of Worksheet