COM2031 Advanced Algorithms, Autumn Semester 2019

Lab 2: MergeSort

Purpose of the lab

Your task is to implement MergeSort. You can use any programming language you like. A template Java file, and a sample solution will be given in Java.

You will notice that it is a fairly long way from the pseudo code in the lectures / book to a real implementation as you will need to create all the data structures and take care of special cases, parameter checking etc that a pseudo code representation dismisses hand-wavingly.

You will probably not finish the coding for this exercise within the 2hrs allocated to the lab. This is not unexpected. You will need to work on finalising the algorithms also after the lab to fully profit from the exercise. The lab serves to get you started, settle any questions and put you on the right track.

Always work through the exercise sheet in conjunction with the lecture slides / textbook!

Part 1 (Fundamental): MergeSort on Integers

Implement the **`MergeSort`** algorithm from the lecture/tutorials. You may use Java or C++ (or any other programming language of your choice).

Implement `MergeSort` initially to sort arrays of int. Use the description of the algorithm in the lecture slides as a starting point. The core of your implementation shall be a method `MSort` as follows:

```
/**
 *  recursive part of MergeSort.  This method returns the elements
 *  between positions start and end in sorted order in a new list
 *
 * @param S
 * array of elements to sort
 * @param start
 * first element of S to be included in sort (inclusive)
 * @param end
 * last element to S to be included in sort (exclusive)
 * @return new array of length end-start that contains the elements
 * of S between start and end in sorted order
 */
```

```
public static int [] MSort (final int [] S, final int start, final
int end){ TODO }
```

It also makes sense to implement a helper method that is utilised by the core recursive method:

```
/** takes two sorted lists and returns a sorted merged list
 *  Prerequisite: left and right *must* be in sorted order
 *  @param L
 *  a list that is already sorted
 *  @param R
 *  another list that is already sorted
 *  @return
 *  new list with all elements of L and R in sorted order
 */
public static int [] merge (final int [] L, final int [] R){
  TODO
}
```

While `merge` will end up having many more lines of code than `MergeSort`, it is conceptually simpler as it does not contains a recursion. I suggest therefore to start implementing `merge` first, and only after that work on the recursive structure `MergeSort` needs to implement (ie how it calls itself on the sublists). `MergeSort` will essentially be a call of `MSort` on the whole array.

The template file also contains code for testing your implementation.

Part 2 (Advanced): MergeSort on `Comparable`

Extend the implementation to not only apply to sort arrays of int, but arrays of any type that implements the interface `Comparable`. Alternatively extend the implementation so that it takes an array of a given type along with a `Comparator` object that tells it how to compare to elements of the array. (This will need the use of Java generics).

Part 3 (Fundamental): Library Sort on Arrays

Find the source code (eg via the web) of Java's library function `Arrays.sort` and understand how sorting is implemented in the library -- which sorting algorithm is ultimately used? And what is the function of all the boilerplate infrastructure code around it?