13. Sorting Algorithms :: Merge Sort :: Java Implementation

This implementation sorts an ArrayList<Integer> list into ascending order. With a few minor changes it can be made to sort into descending order.

```
//*************************************************************************************
// CLASS: MergeSorter (MergeSorter.java)
//*************************************************************************************
import java.util.ArrayList;

/**
 * Implements the recursive merge sort algorithm.
 */
public class MergeSorter {

    /**
     * Creates and returns a new ArrayList consisting of the elements from pList at indices
     * pFromIndex (inclusive) to pToIndex (exclusive).
     */
    private static ArrayList<Integer> arrayListCopy(ArrayList<Integer> pList, int pFromIndex,
    int pToIndex) {
        ArrayList<Integer> pCopy = new ArrayList<>();
        for (int i = pFromIndex; i < pToIndex; ++i) {
            pCopy.add(pList.get(i));
        }
        return pCopy;
    }
```

13. Sorting Algorithms :: Merge Sort :: Java Implementation (continued)

```
/**
 * Copies elements from pSrcList[pSrcIndex..pSrcList.size - 1] to pSstList starting at
 * index pSstIndex.
 */
private static void copyRest(ArrayList<Integer> pSrcList, int pSrcIndex,
ArrayList<Integer> pDstList, int pDstIndex) {
    while (pSrcIndex < pSrcList.size()) {
        pDstList.set(pDstIndex, pSrcList.get(pSrcIndex));
        ++pSrcIndex;
        ++pDstIndex;
    }
}
```

13. Sorting Algorithms :: Merge Sort :: Java Implementation (continued)

```
/**
 * The primary recursive routine that merge sorts pList.
 */
public static void mergeSort(ArrayList<Integer> pList) {
    // The base case is reached when the size of pList is 1. The list is trivially
    // sorted so we have nothing to do.
    if (pList.size() < 2) return;

    // Otherwise, split the list into two halves: a left half and a right half. Recursively
    // merge sort each half.
    ArrayList<Integer> listL = arrayListCopy(pList, 0, pList.size() / 2);
    ArrayList<Integer> listR = arrayListCopy(pList, pList.size() / 2, pList.size());
    mergeSort(listL);
    mergeSort(listR);

    // On return from the two method calls above, both listL and listR will be sorted.
    // Merge them to form a sorted list.
    merge(listL, listR, pList);
}
```

13. Sorting Algorithms :: Merge Sort :: Java Implementation (continued)

```
    /**
     * Merges pListL and pListR into pList.
     */
    private static void merge(ArrayList<Integer> pListL, ArrayList<Integer> pListR,
    ArrayList<Integer> pList) {
        int leftIndex = 0, rightIndex = 0, listIndex = 0;
        while (leftIndex < pListL.size() && rightIndex < pListR.size()) {
            if (pListL.get(leftIndex).compareTo(pListR.get(rightIndex)) <= 0) {
                pList.set(listIndex, pListL.get(leftIndex));
                ++leftIndex;
            } else {
                pList.set(listIndex, pListR.get(rightIndex));
                ++rightIndex;
            }
            ++listIndex;
        }
        if (leftIndex < pListL.size()) {
            copyRest(pListL, leftIndex, pList, listIndex);
        } else {
            copyRest(pListR, rightIndex, pList, listIndex);
        }
    }
}
```