Theory: Merge sort in Java

38 minutes 0 / 4 problems solved

Merge sort is an efficient comparison-based sorting algorithm. It divides an unsorted array of size n into n single-element subarrays which are already sorted, and then repeatedly merges these subarrays to produce newly sorted subarrays until there is only 1 sorted subarray remaining. In the algorithm, **merge** is the main operation. It produces a new sorted array from two input sorted arrays.

For an array of size n, merge sort works in $O(n \log n)$, which is better comparing with such sorting algorithms as bubble sort, insertion sort, and selections sort. Thus, merge sort can be used in practice to sort even large arrays.

§1. The top-down implementation in Java

Given below is an implementation of the top-down version of merge sort in Java. The `mergeSort` method takes an array and a range of elements (`left` is inclusive, `right` is exclusive):

```
  public static void mergeSort(int[] array, int leftIncl, int rightExcl) {

   // the base case: if subarray contains <= 1 items, stop dividing because it's
sorted
        if (rightExcl <= leftIncl + 1) {
            return;
        }

        /* divide: calculate the index of the middle element */
        int middle = leftIncl + (rightExcl - leftIncl) / 2;



    mergeSort(array, leftIncl, middle);  // conquer: sort the left subarray


    mergeSort(array, middle, rightExcl); // conquer: sort the right subarray



        /* combine: merge both sorted subarrays into sorted one */

        merge(array, leftIncl, middle, rightExcl);

    }
```

The `merge` method performs merging of two subarrays using a temporary array:

Current topic:

Merge sort in Java

Topic depends on:

- ✓ Merge sort
- ✗ Algorithms in Java
- ✗ Insertion sort in Java

Table of contents:

```
 private static void merge(int[] array, int left, int middle, int right) {
     int i = left;   // index for the left subarray
     int j = middle; // index for the right subarray
     int k = 0;      // index for the temp subarray

     int[] temp = new int[right - left]; // temporary array for merging

     /* get the next lesser element from one of two subarrays

and then insert it in the array until one of the subarrays is empty */

     while (i < middle && j < right) {

         if (array[i] <= array[j]) {

             temp[k] = array[i];

             i++;

         } else {

             temp[k] = array[j];

             j++;

         }

         k++;

     }



 /* insert all the remaining elements of the left subarray in the array */

     for (;i < middle; i++, k++) {

         temp[k] = array[i];

     }



 /* insert all the remaining elements of the right subarray in the array */

     for (;j < right; j++, k++) {

         temp[k] = array[j];

     }



     /* effective copying elements from temp to array */

     System.arraycopy(temp, 0, array, left, temp.length);

 }
```

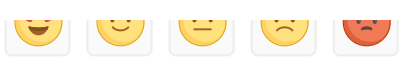
Below are several examples of how the implemented method can be used:

```
int[] array1 = { 30, 21, 23, 19, 28, 11, 23 };

mergeSort(array1, 0, array1.length); // { 11, 19, 21, 23, 23, 28, 30 }

int[] array2 = { 30, 20, 10, 10, 20, 10 };
mergeSort(array2, 0, array2.length); // { 10, 10, 10, 20, 20, 30 }
```

We can optimize the implementation and create only a single temporary array for all merge operation. Try to modify the code in this way.

Report a typo

53 users liked this theory. **5** didn't like it. **What about you?**



Start practicing

Comments (6) Hints (0) Useful links (1) Show discussion