Northeastern University

Lecture 28: Faster Sorting Methods - 1

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from

Data Structures and Abstractions with Java, 5th edition. By Frank M. Carrano and Timothy M. Henry.
ISBN-13 978-0-13-483169-5

Merge Sort

Merge Sort

- Divides an array into halves
- Sorts the two halves,
  - Then merges them into one sorted array.
- The algorithm for merge sort is usually stated recursively.
- Major programming effort is in the merge process.

Merging Arrays

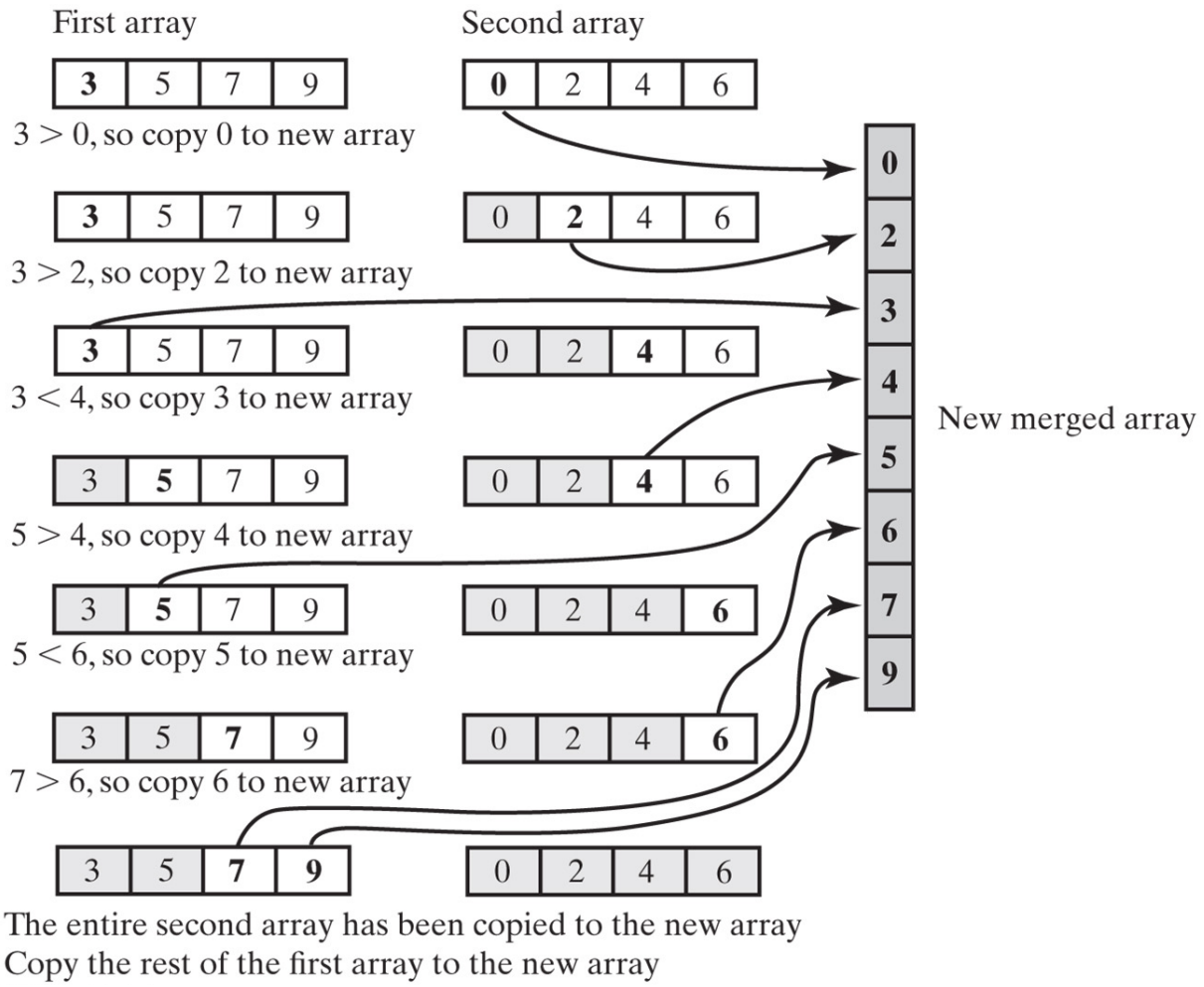


Figure 16-1: Merging two sorted arrays into one sorted array

Recursive Merge Sort

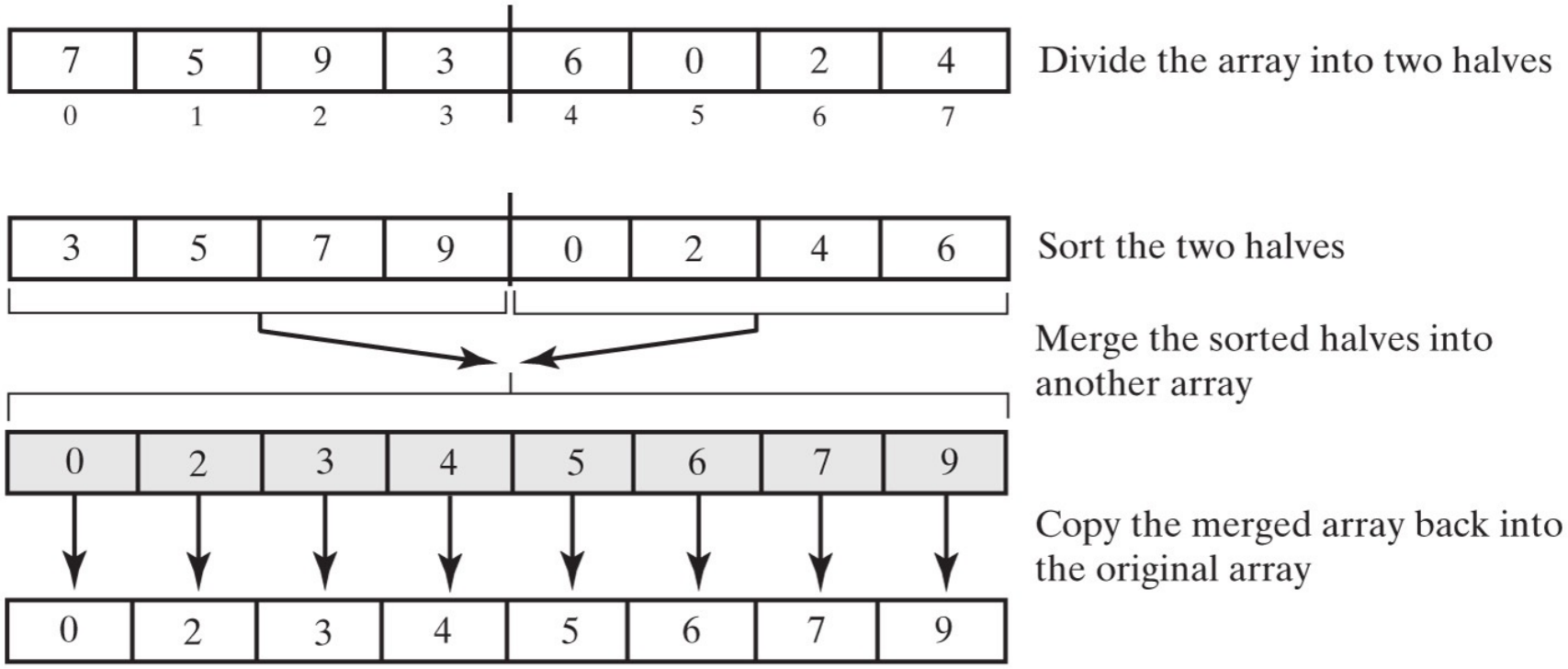


Figure 16-2: The major steps in a merge sort

Recursive Merge Sort

Algorithm mergeSort(a, tempArray, **first**, last)
// Sorts the array entries `a[first..last]` recursively.

```
if (first < last)
{
    mid = approximate midpoint between first and last
    mergeSort(a, tempArray, first, mid)
    mergeSort(a, tempArray, mid + 1, last)
    Merge the sorted halves a[first..mid] and a[mid+1..last] using the array tempArray
}
```

Recursive algorithm for merge sort

Algorithm merge(a, tempArray, **first, mid, last)**

// Merges the adjacent subarrays a[first..mid] and a[mid+1..last].
beginHalf1 = first, endHalf1 = mid, beginHalf2 = mid + 1, endHalf2 = last

// While both subarrays are not empty, compare an entry in one subarray with
// an entry in the other; then copy the smaller item into the temporary array

index = 0 //Next available location in `tempArray`

```
while ( (beginHalf1 <= endHalf1) and (beginHalf2 <= endHalf2) ){
        if (a[beginHalf1] <= a[beginHalf2]){
                tempArray[index]  =  a[beginHalf1]
                beginHalf1++
        } else{
                tempArray[index]  =  a[beginHalf2]
                beginHalf2++
        }
        index++
}
```

// Assertion: One subarray has been completely copied to `tempArray`.
Copy remaining entries from other subarray to `tempArray`
Copy entries from `tempArray` to array a

Pseudocode describes the merge step

Recursive Merge Sort

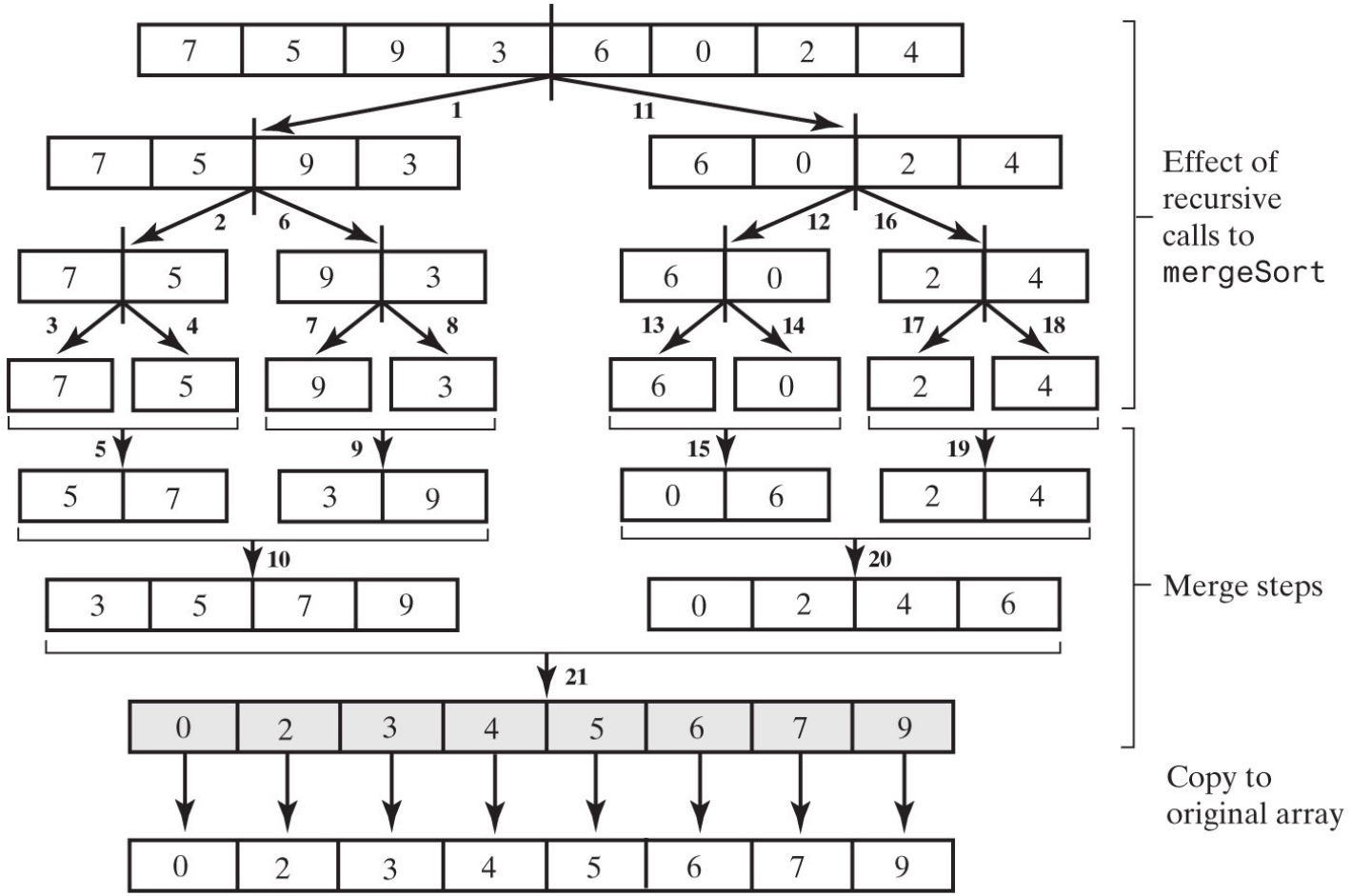


Figure 16-3: The effect of the recursive calls and the merges during a merge sort

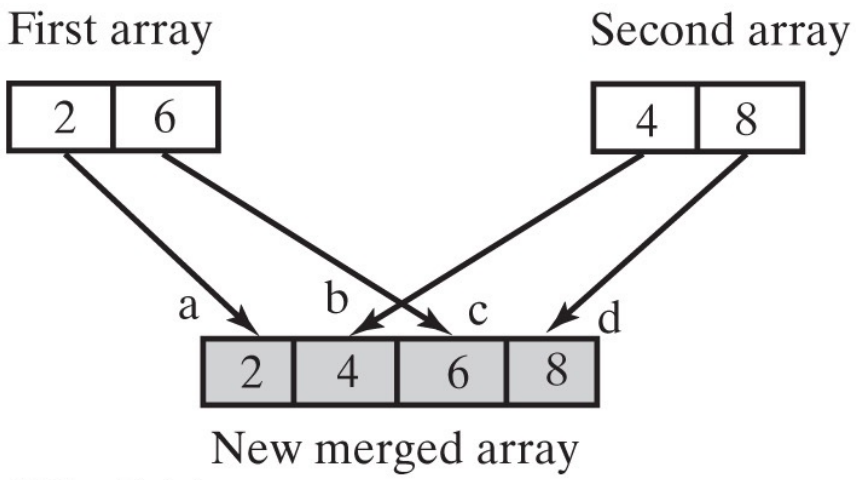
Recursive Merge Sort

```
public static <T extends Comparable<? super T>>
              void mergeSort(T[] a, int first, int last)
{
   // The cast is safe because the new array contains null entries
   @SuppressWarnings("unchecked")
   T[] tempArray = (T[])new Comparable<?>[a.length]; // Unchecked cast

   mergeSort(a, tempArray, first, last);
} // end mergeSort
```

Be careful to allocate the temporary array only once

Efficiency of Merge Sort



a. $2 < 4$, so copy 2 to new array

b. $6 > 4$, so copy 4 to new array

c. $6 < 8$, so copy 6 to new array

d. Copy 8 to new array

Figure 16-4: A worst-case merge of two sorted arrays. Efficiency is O(*n* log *n*).

Note that the proof is in section 16.6.

Iterative Merge Sort

- Less simple than recursive version
  - Need to control the merges
- Will be more efficient of both time and space
  - But, trickier to code without error

Iterative Merge Sort (cont.)

- Starts at beginning of array
  - Merges pairs of individual entries to form two-entry subarrays
- Returns to the beginning of array and merges pairs of the two-entry subarrays to form four-entry subarrays
  - And so on
- After merging all pairs of subarrays of a particular length, might have entries left over.

Exercise

- Suppose following list represents an array of Integer objects.
- Show the steps that a merge sort takes when sorting this array.

80 90 70 85 60 40 50 95

Answer

- Split the array in half until we reach single-element subarrays:

  80 90 70 85 60 40 50 95

  80 90 70 85 | 60 40 50 95

  80 90 | 70 85 | 60 40 | 50 95

  80 | 90 | 70 | 85 | 60 | 40 | 50 | 95

- Merge pairs of subarrays to produce larger sorted subarrays:

  80 90 | 70 85 | 40 60 | 50 95

  70 80 85 90 | 40 50 60 95

  40 50 60 70 80 85 90 95

Merge Sort in the Java Class Library

- Class `Arrays` in the package `java.util` defines versions of a static method `sort`

```
public static void sort(Object[] a)

public static void sort(Object[] a, int first, int after)
```

Exercise (10-15 mins)

- Please download "L28_E1" project from "In-class Exercises" on the Canvas.
- Please finish methods `mergeSort()` and `merge()`, and run Merge Sort to a given sequence of integers. (ArraySortRecursive.java)

Note: You should use `merge()` in `mergeSort()` to support the sorting.

Expected Result

```
Running Merge Sort:


Before sort:
11 2 5 7 9 21 8 18 24 10


After sort:
2 5 7 8 9 10 11 18 21 24
```

Answer

```
private static <T extends Comparable<? super T>>
void mergeSort(T[] a, T[] tempArray, int first, int last)
{

    if (first < last)
    {
        int mid = first + (last - first) / 2;  // Index of midpoint
        mergeSort(a, first, mid);              // Sort left half array
        mergeSort(a, mid + 1, last);           // Sort right half array

        merge(a, tempArray, first, mid, last);  // Merge the two halves
    }

}
```

Answer (cont.)

```
private static <T extends Comparable<? super T>>
   void merge(T[] a, T[] tempArray, int first, int mid, int last){

   int beginHalf1 = first;
   int endHalf1 = mid;
   int beginHalf2 = mid + 1;
   int endHalf2 = last;

   int index = beginHalf1;
   for (; (beginHalf1 <= endHalf1) && (beginHalf2 <= endHalf2); index++){
      if (a[beginHalf1].compareTo(a[beginHalf2]) < 0)
      {
         tempArray[index] = a[beginHalf1];
         beginHalf1++;
      } else {
         tempArray[index] = a[beginHalf2];
         beginHalf2++;
      } // end if
   } // end for

   for (; beginHalf1 <= endHalf1; beginHalf1++, index++)
      tempArray[index] = a[beginHalf1];

   for (; beginHalf2 <= endHalf2; beginHalf2++, index++)
      tempArray[index] = a[beginHalf2];

   for (index = first; index <= last; index++)
      a[index] = tempArray[index];

} // end merge
```