# Student lesson

19 – Merge and MergeSort

**INTRODUCTION:** In Lesson A17, *Quadratic Sorting Algorithms*, we saw how the number of steps required increased N2 when sorting N elements. In this lesson, we will study a recursive sort, called mergeSort that works by dividing lists in half. After solving a preliminary merge problem, you will code a recursive mergeSort.

The key topics for this lesson are:

A. A Non-Recursive MergeSort

B. A Merge Algorithm

C. Recursive MergeSort

D. Order of Recursive MergeSort

**VOCABULARY:** MERGE MERGESORT

**DISCUSSION:** A. A Non-Recursive MergeSort

1. The overall logic of mergeSort is to "divide and conquer." A list of random integers will be split into two or more equal-sized lists (each with the same number of elements, plus or minus one), with each partial list or “sublist” sorted independently of the other. The next step will be to merge the two sorted sublists back into one big sorted list.

2. Here is a non-recursive mergeSort method. We divide the list into two equal-sized parts and sort each with the selection sort, then merge the two using an algorithm to be discussed in part B.

/\* List A is unsorted, with A.size() values in the ArrayList.

first is the index of the first value; last

is the index of the last value in the ArrayList;

first < last.

\*/

**void** mergeSort (**ArrayList <Integer> A,** **int** first, **int** last){

**int** mid;

mid = (first + last) / 2;

selectionSort (A, first, mid);

selectionSort (A, mid+1, last);

merge (A, first, mid, last);

}

|  |  |
| --- | --- |
| **See Transparency A18.1, *MergeSort Example*** | 3. A modified selection sort will have to be written to sort a range of values in list A. Likewise, themerge method will have to be modified to internally merge two halves of the array into one ordered array. |

4. The following example will illustrate the action of a non-recursive mergeSort on a section of a list containing 8 values:



5. Merging the two halves of the array in the modified merge method will require the use of a local temporary array. Because the two sublists are stored within one array, the easiest approach is to merge the two sublists into another array, then copy the temp array back to the original.



Then copy Temp back into List A:



6. This version of merge will need to be able to work with sections of ArrayList A. Here is a proposed parameter list for merge:

/\* will merge the two sorted sublists within A into

one continuous sublist from A[first] .. A[last].

The left list ranges from A[first]..A[mid]. The

right list ranges from A[mid+1]..A[last].

\*/

**void** merge (**ArrayList <Integer>**  A, **int** first, **int** mid, **int** last)

7. The recursive version of mergeSort will require the above version of merge. However, to help you understand how to write a merge method, we next present a simpler merge algorithm.

B. A Merge Algorithm

1. The mergeSort algorithm requires a merge algorithm that we will design first.

2. The method header and the specifications of the merge method are given below. You may assume the ArrayList definitions from the sorting template program in Lesson 17 apply.

/\* Preconditions: Lists *A* and *B* are non-empty and in sorted nondecreasing order.

Action: Lists *A* and *B* are merged into one ArrayList, *C*.

Postcondition: List *C* contains all the values from

Lists *A* and *B*, in nondecreasing order.

\*/

**void** merge (**ArrayList <Integer> A**, **ArrayList <Integer> B**, **ArrayList <Integer> C**)

3. The merge method breaks down into four cases:

a. ArrayList A is done, so pull a value from ArrayList B.

b. ArrayList B is done, so pull a value from ArrayList A.

c. Neither is done, and if A[i] < B[j] (where i & j are index markers in lists A and B) then pull from ArrayList A.

d. Neither is done, and if B[j] <= A[i] then pull from ArrayList B.

4. It is important to deal with the four cases in the order described above. For example, if you are done with ArrayList A (if i > A.length-1), you do not want to inspect any values past A[i]*.*

|  |  |
| --- | --- |
| **See Transparency A18.2, *Merging Two Lists*** | 5. Example of method merge results: |

A: 2 6 11 15 21

B: 4 5 9 13 17 25 29

C: 2 4 5 6 9 11 13 15 17 21 25 29

C. Recursive MergeSort

1. Instead of dividing the list once, a recursive mergeSort will keep dividing the list in half until the sublists are one or two values in length.

2. When developing a recursive solution, a key step is identifying the base case of the solution. What situation will terminate the recursion? In this case, a sublist of one or two values will be our two base cases.

3. Let's try and work through the recursive mergeSort of a list of eight values.



The list is divided into two sublists:



Now let's work on the left sublist. It will be divided into lists of two.



Each list of two is now very easy to sort. After each list of two is sorted, we then merge these two lists back into a list of four.



Now the algorithm proceeds to solve the right sublist (positions 5-8) recursively. Then the two lists of four are merged back together.



D. Order of Recursive MergeSort

1. Suppose that we have a list of 8 numbers. If we trace the migration of one value, it will be a member of the following sizes of lists: eight, four, two. The number of calls of mergeSort needed to sort one value into its final resting spot is log2N. If N = 8, then it will take three calls of the algorithm for one value to find its final resting spot.

2. We must apply log2N steps to sort N elements in the list. The order of recursive Mergesort is O(N \* log2N) or O(N \* log N).

3. What about the cost of merging the fragments of the list? The merge algorithm is a linear one, so when combined with the mergeSort routine, we have a O (N \* log N) + O(N), which remains in the category of an O(N \* log N) algorithm.

|  |  |
| --- | --- |
| **SUMMARY/ REVIEW:** | The recursive mergeSort produces a dramatic increase in efficiency in comparison with the N2 order of the quadratic sorts. This concept of dividing the problem in two is used in several other classic algorithms. Once again, recursion makes it easier to define a problem and code the solution. |

**ASSIGNMENT:** Lab Assignment A18.1, *Merge*

Lab Assignment A18.1, Starter Code, *MergeTemplate.java*

Lab Assignment A18.2, *Recursive MergeSort*

Transparency A18.1*, MergeSort Example*

Transparency A18.2*, Merging Two Lists*

Worksheet A18.1*, Non-recursive Merge and MergeSort*