STUDENT OUTLINE

Lesson 25 – Merge and Mergesort

INTRODUCTION: In Lesson 23, we studied the quadratic sorting algorithms. We saw how the number of steps required increased as an N^2 factor when sorting N elements. In the next two lessons we will study two recursive sorts, mergesort and quicksort, which work by dividing lists in half. In this lesson, 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. <u>A Non-Recursive Mergesort</u>

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.

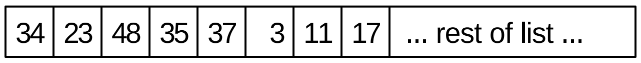
```
/* List A is unsorted, with A.length values in the array.
   first is the index position of the first value; last
   is the index position of the last value in the array;
   first < last.
 */
void mergeSort (int A[], int first, int last)
{
  int  mid;

  mid = (first + last) / 2;
  selectionSort (A, first, mid);
  selectionSort (A, mid+1, last);
  merge (A, first, mid, last);
}
```

See Transparency T.A.25.1, ***Example of Mergesort.***

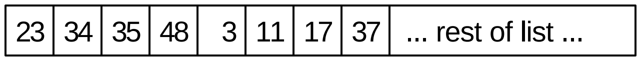
3. A modified selection sort would have to be written to sort a range of values in list A. Likewise, the `merge` method will also have to be modified to internally merge two halves of the array into one ordered array.

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

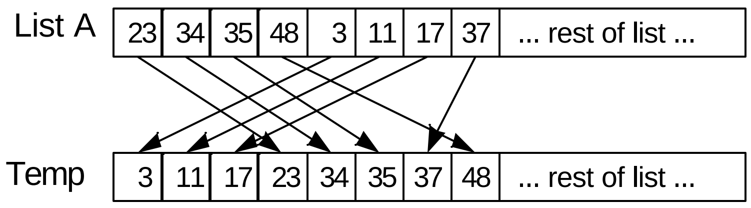
Unsorted List



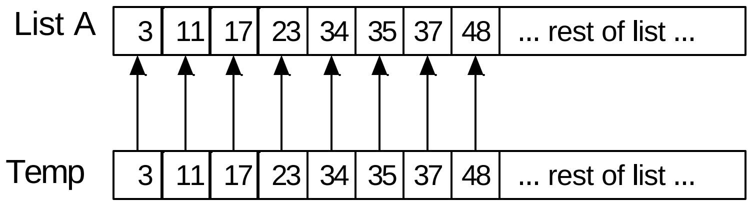
List After Sorting Each Half



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 List A. Here is a proposed method 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 (int A[], int first, int mid, int last)
```

7. You will need to write the code for this version of the `merge` method to support a recursive mergesort. To assist you in that task, we next present a non-recursive mergesort algorithm.

B. A Merge Algorithm

1. The mergesort algorithm requires a merge algorithm that we will solve first.

2. The method header and the specifications of the merge routine are given below. You may assume the array definitions from the sorting template program in Lesson 23 apply.

```
/* Preconditions: Lists A and B are sorted in nondecreasing
                  order.
   Action: Lists A and B are merged into one list, C.
   Postcondition: List C contains all the values from
                  Lists A and B, in nondecreasing order.
 */
void merge (int[] A, int[] B, int[] C)
```

3. The `merge` method breaks down into four cases:

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

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

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

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

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

See T.A.25.2, *Merging Two Lists.*

5. Example of method Merge:

   A: 2 6 11 15 21

   B: 4 5 9 13 17 25 29

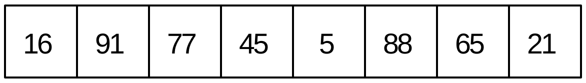
   C: 2 4 5 6 9 11 13 15 17 21 25 29

C. Recursive Mergesort

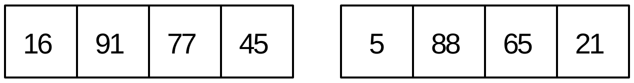
1. Instead of dividing the list once, a recursive mergesort will keep dividing the list in half until the sublists are one or two values in length.

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.

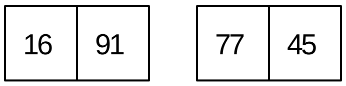
3. Let's try and work through the recursive mergesort of a list of eight values.



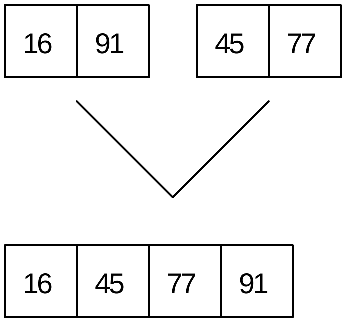
The list is divided into two sublists:



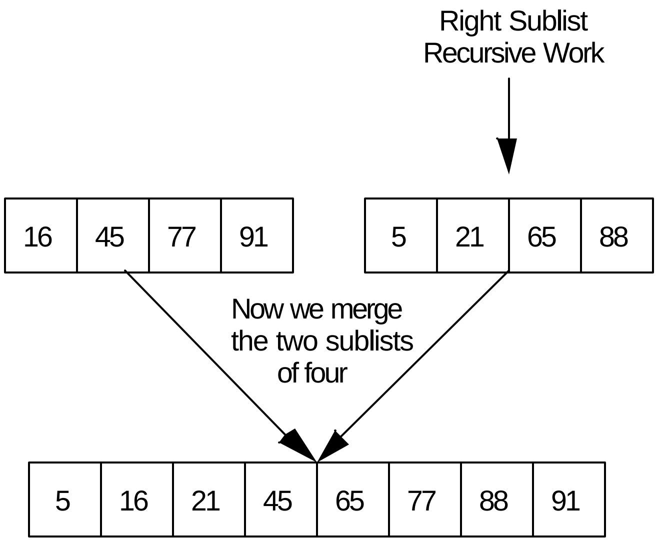
Now let's work on the left sublist. It will be divided into lists of two.



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.


Right Sublist Recursive Work
16 45 77 91
5 21 65 88
Now we merge the two sublists of four
5 16 21 45 65 77 88 91

D. Order of Recursive Mergesort

1. Suppose 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 $\log_2 N$. 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 $\log_2 N$ steps to N elements in the list. The order of recursive Mergesort is $O(N * \log_2 N)$ 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 N^2 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 Exercise L.A.25.1, *Merge*
Lab Exercise L.A.25.2, *Mergesort (Recursive)*

LAB EXERCISE

Merge

Assignment:

1. As explained in the student outline, write a method to merge two sorted lists into one sorted list.

2. Add the code for your merge method to the provided merge template program, *MergeTemplate.java*.

Instructions:

1. The merge algorithm is prone to logic errors. The most common error is dealing with cases when you have reached the end of one list or the other. You are to test these 6 different input scenarios:

	List A		List B	
	Quantity	Largest Value	Quantity	Largest Value
Trial 1	20	100	40	100
Trial 2	40	100	20	100
Trial 3	20	100	40	50
Trial 4	20	50	40	100
Trial 5	40	50	20	100
Trial 6	40	100	20	50

2. Turn in your source code and the printed run output for trials 4 and 6. If possible, print only the merge function source code.

LAB EXERCISE

Mergesort (Recursive)

Assignment:

1. Using the merge program in lab exercise L.A.25.1, `Merge`, as a starting point, write a recursive `mergeSort` method as described in the student outline. Pseudocode for the recursive `mergeSort` method is given below.

   ```
   void mergeSort(int[] a, int first, int last)
   //  Recursively divides a list in half, over and over. When the
   //  sublist has one or two values, stop subdividing.
   {
      if (sublist has only one value)
         do nothing
      else if (sublist has two values)
         sort it if necessary
      else    // recursion, divide list into two halves
         Find midpoint of current sublist
         Call mergeSort and process left sublist
         Call mergeSort and process right sublist
         merge left and right sublists
   }
   ```

2. You will have to modify the `merge` method to fit the necessary calls of the `mergeSort` method.

Instructions:

1. After confirming that your mergesort works, paste the necessary routines into your sorting template program and count the number of steps for a recursive mergesort. Record the number of steps on the worksheet from Lesson 23, Worksheet W.A.23.1, *Comparison of Sorting Algorithms*.

2. Turn in your source code and a printed run output of 100 numbers, sized from 1-200. If possible, print only `merge` and `mergeSort` methods.