STUDENT OUTLINE

Lesson 24 – Merge and Mergesort

INTRODUCTION:

In Lesson 22, 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.

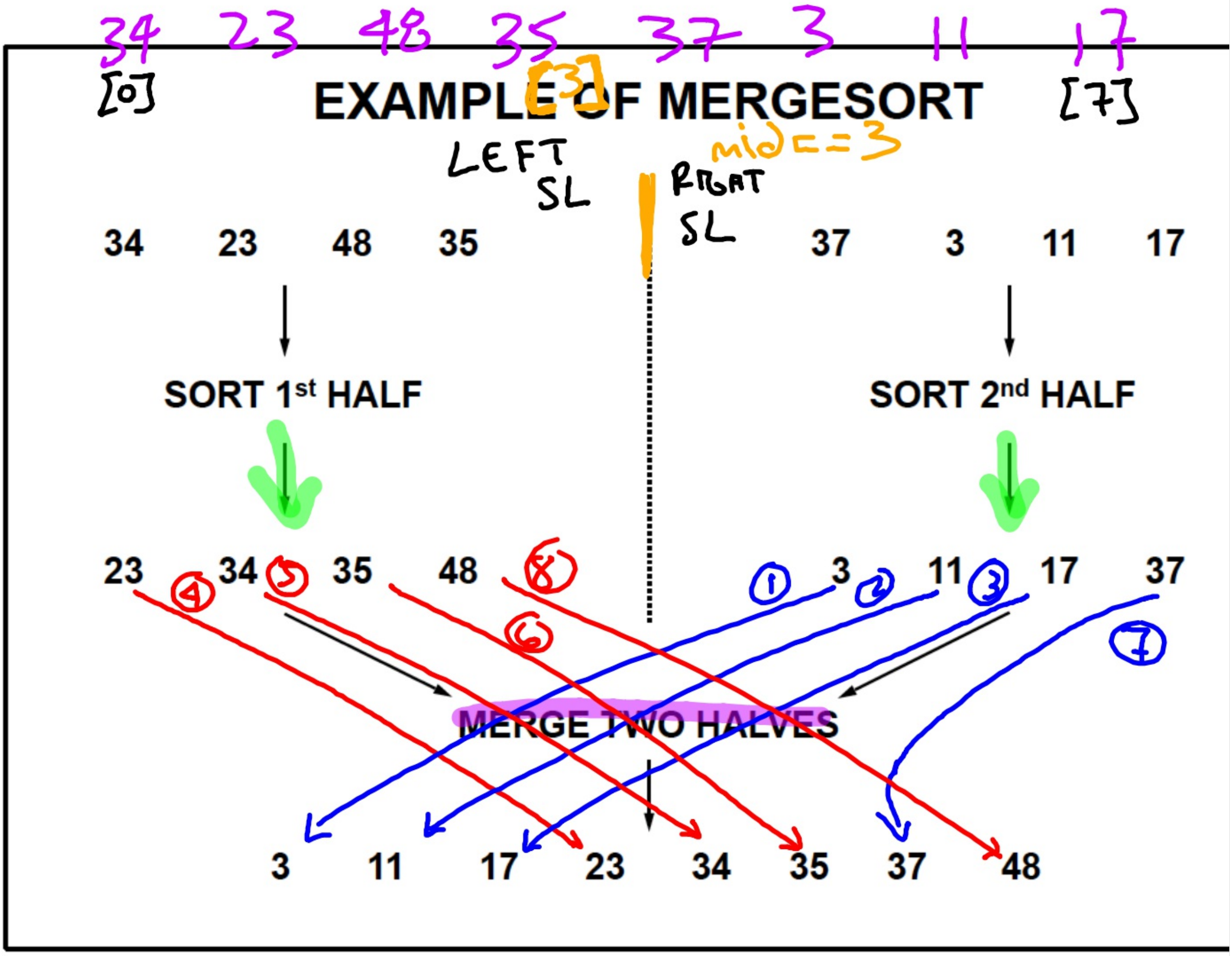
```
/* List A is unsorted, with A.length values in the array.
   first is the index position of the first value; last
   is the index position of the last value in the array;
   first < last.
 */
void mergeSort (int A[], int first, int last)
{
  int  mid;

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

THE LEFT SUBLIST

THE RIGHT SUBLIST

See Transparency T.A. 24.1, *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.



EXAMPLE OF MERGESORT

34 23 48 35 | 37 3 11 17

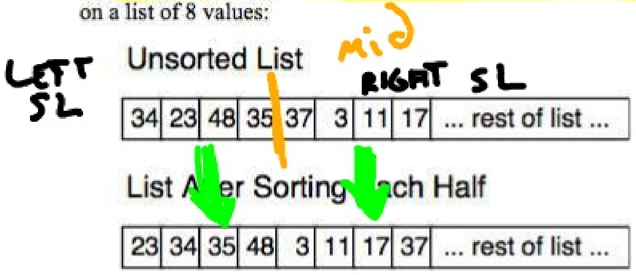
SORT 1st HALF | SORT 2nd HALF

23 34 35 48 | 3 11 17 37

MERGE TWO HALVES

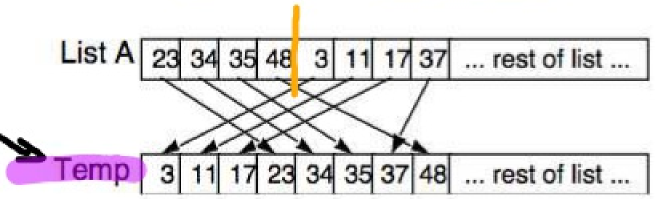
3 11 17 23 34 35 37 48

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


LEFT SL
Unsorted List
mid
RIGHT SL
34 23 48 35 37 3 11 17 ... rest of list ...
List After Sorting Each Half
23 34 35 48 3 11 17 37 ... rest of list ...

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.

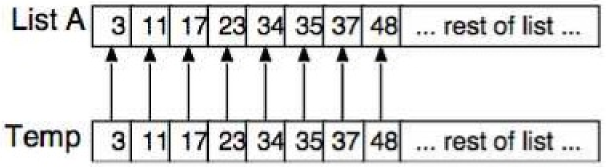
TEMPORARY ALLOWS MERGE DUE TO C.B.R.


List A
23 34 35 48 3 11 17 37 ... rest of list ...
Temp
3 11 17 23 34 35 37 48 ... rest of list ...

SAME AS PREVIOUS PAGE

Then copy `Temp` back into List A:

DONT FORGET ME!!


List A
3 11 17 23 34 35 37 48 ... rest of list ...
Temp
3 11 17 23 34 35 37 48 ... rest of list ...

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`:

SUBSCRIPT BOUNDARIES ON THE ARRAY

```
/* 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 ~~[illegible]~~ 22 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)
```

LAB 24.1

3. The merge method breaks down into four cases. WHILE

A NESTED if...else MAY WORK WELL

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

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

BE SURE TO ADVANCE!

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.24.2, *Merging Two Lists.*

5. Example of method Merge:

A: 2 6 11 15 21

B: 4 5 9 13 17 24 29

C: 2 4 5 6 9 11 13 15 17 21 24 29

SEE NEXT PAGE

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.

YAY!!!

i is RED
j is BLUE

MERGING TWO LISTS

A: 2 6 11 15 21 *

B: 4 5 9 13 17 25 29 6

WHERE IT CAME FROM →	0	0	1	1	2	2	3	3	4	4	5	6
C:	2	4	5	6	9	11	13	15	17	21	25	29

CONCEPTUAL BREAK

mid==3

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

[0] [7]

16	91	77	45	5	88	65	21

The list is divided into two sublists:

mid==1

LEFT SL

16	91	77	45

[0] [3]

5	88	65	21

RIGHT SL

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

FOUND BASE CASE

16	91

77	45

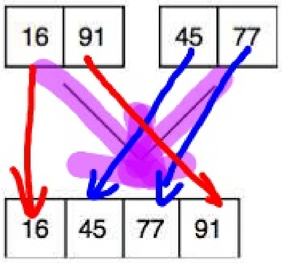
Each list of two is now very easy to sort. After each list of two is sorted, we then merge these two lists back into a list of four.

Swap!!

16	91

45	77

MERGE



16	45	77	91

ORDERED LEFT SUB-LIST

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

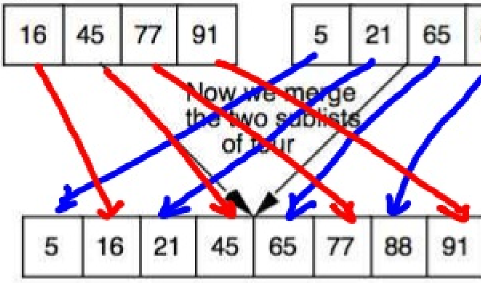
THEN PROCESS

Right Sublist Recursive Work

16	45	77	91

5	21	65	88

MERGE!



5	16	21	45	65	77	88	91

A PERFECT ZIPPER!

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.

ON N ELEMENTS...

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 $O(N * \log N)$.

APPLY THIS ALGORITHM

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.

WE PICK THE LEAST EFFICIENT CATEGORY PRESENT

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.24.1, *Merge*
Lab Exercise L.A.24.2, *Mergesort (Recursive)*

FRESH TEMPLATE

USE SAME TEMPLATE FROM LAB 22.

ADD IT IN