CS1112-206/211/212–Fall 2009

Section 13 Solution

Friday, November 27

When dealing with recursions, it is best to keep track of where we are and what we still have to do, and make sure to transfer the values between recursive calls correctly. This can be done using lots of scratch paper so that values from different recursive calls do not mix up, because they do not mix up in the real program anyway.

1 Recursion

The (transposed) output for (a) is `5 2 1 3 9 243`. Observe that the first three lines are the exponents, where the last three lines are the power of 3 to the reverse sequence of the exponents.

The (transposed) output for (b) is `10 5 2 1 2 4 32 1024`.

2 Counting Comparisons

A comparison is defined as when we take a value in u and a value in v and compare them together. The only place this is done is in the `if` block in the first `while` loop. We determine whether $u(i) \leq v(j)$. If it is, we did one comparison. Otherwise, we also did one comparison (to answer that it is not). Hence, we can simply increment `count` after the `if` block's `end`. Where do we initialize `count`? Answer: Before we begin counting, i.e., initialize it before the first `while` loop.

3 Counting Comparisons and Recursion

If $N = 1$, we do not have to compare anything, so `countOUT` will be the same as `countIN`. Hence, we can assign `countOUT=countIN` in the `if` block. Otherwise, we need to call `MergeSortR` recursively. c_1 is `countIN` plus the number of comparisons used to sort the left half of the array. c_2 is `countIN` plus the number of comparisons used to sort the right half of the array. c_3 is the number of comparisons used to merge the two halves. Therefore, the number of comparisons done to sort the array is

$$(c_1 - \mathit{countIN}) + (c_2 - \mathit{countIN}) + c_3.$$

Since the desired `countOUT` is `countIN` plus the number of comparisons, we can assign

```
countOUT=c1+c2+c3-countIN;
```

at the end of the `else` block (but before the keyword `end`).

4 Counting Recursive Calls

The number of lines represents the number of recursive calls to `MergeSortR`. We can do this by hand by running through the code carefully to calculate the length of the vector to be sorted in each recursive call. The solution is posted on the course webpage (`Section13.m`).