10. Sorting Algorithms :: Merge Sort :: Time Complexity Analysis

How efficient is merge sort? To simplify our analysis we will assume the size of the list being sorted is a power of 2, i.e., $n = 2^p$. This assumption will not alter the final result.

First, let's analyze the time complexity of the $merge()$ procedure. On input to $merge()$, let the size of $list_L$ and $list_R$ be $n = 2^p$. The key operations (the ones that dominate the time) are the accessing of elements in $list_L$ and $list_R$, i.e., reads and writes.

During each pass of the while loop we access $list_L$, $list_R$, and $list$ one time for a total of 3 list accesses. Assume that during the while loop we copy all of the elements from $list_L$ to $list$; then the while loop would execute $n/2$ times and we would perform $3(n/2)$ list accesses. Since all of the elements of $list_R$ still remain to be copied to $list$, the $copyRest()$ method would perform $2(n/2) = n$ list accesses, so for this case, the total number of list accesses in $merge()$ would be $3(n/2) + n = 5n/2$.

A similar argument can be made by assuming that during the while loop we copy all of the elements from $list_R$ to $list$ and then call $copyRest()$ to copy the all of the elements from $list_L$.

In either case, we would perform $5n/2 = 2.5n$ list accesses during the merge and because constants in Big O notation are irrelevant and we are trying to determine an upper bound, it will simplify our math to say that $merge()$ performs $3n$ list accesses.

10. Sorting Algorithms :: Merge Sort :: Time Complexity Analysis (continued)

Now we consider *recursiveMergeSort*(). On the first call, let the size of the original list being sorted be $n = 2^p > 2$. In creating sublist $list_L$ we would copy $n/2$ elements from *list* to $list_L$ which would involve $2(n/2) = n$ list accesses. Creating sublist $list_R$ would also require n list accesses.

Let's let $a(n)$ be the number of list accesses that are performed in *recursiveMergeSort*() for an input list of size n. We now have:

$$a(n) = n + n + 3n = 5n$$

but $a(n)$ does not include the number of list accesses that are performed in each recursive call; those need to be counted as well. Since we are letting $a(n)$ be the number of list accesses during *recursive MergeSort*() for a list of size n, then for each recursive call, the number of list accesses will be $a(n/2)$. Thus:

$$a(n) = 5n + a(n/2) + a(n/2) = 2a(n/2) + 5n$$

An equation of this form—where the value of $a(n)$ depends on the value of $a(n/2)$ and the value of $a(n/2)$ depends on $a(n/4)$—is known as a **recurrence relation** and recurrence relations commonly arise when analyzing the time complexity of recursive methods.