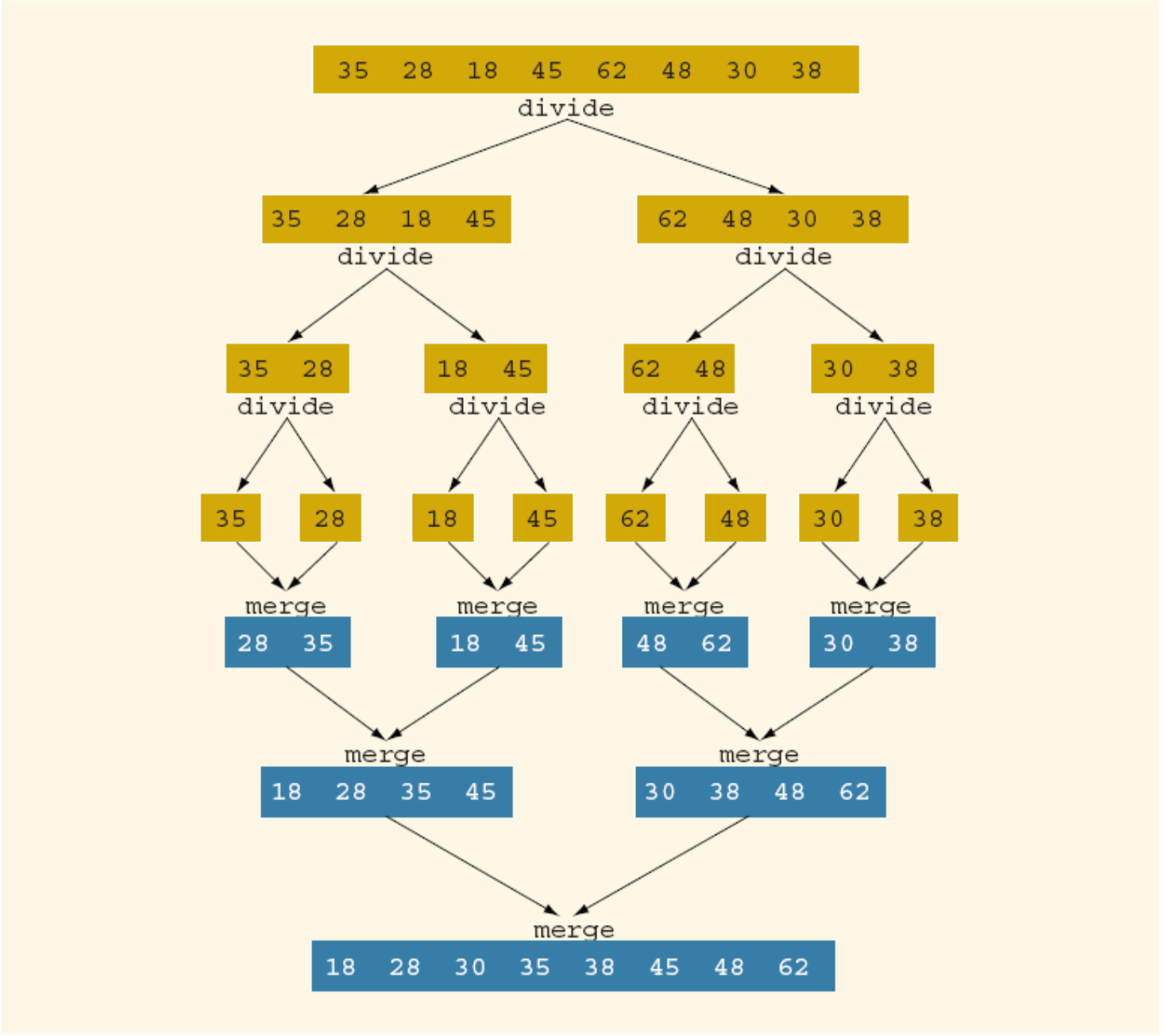

FIGURE 19-48 Merge sort algorithm

Merge Sort: Summary

- General algorithm:

```
if the list is of a size greater than 1
{
   a. Divide the list into two sublists.
   b. Merge sort the first sublist.
   c. Merge sort the second sublist.
   d. Merge the first sublist and the second sublist.
}
```

- Basic analysis:
  - Divide in half *log(n)* times, merge *log(n)* times = *2log(n)*
  - Merging touches each value once, so it is linear
    - At each level, there are *n* items to merge
  - Complexity is *2log(n) * n*, so it is *O(n log(n))*

Array vs. Linked List

- Conceptually no difference
- In practice, merging into an array is simpler
  - Allocate another array and copy into it
  - No pointers to mess around with
- Array: simpler code, easier to write, debug, maintain
- Linked List: doesn't require additional memory!
  - This is called *in-place* sorting

- In-place sorting is the one advantage of Merge Sort over *Quick Sort*, which is faster in practice

Divide

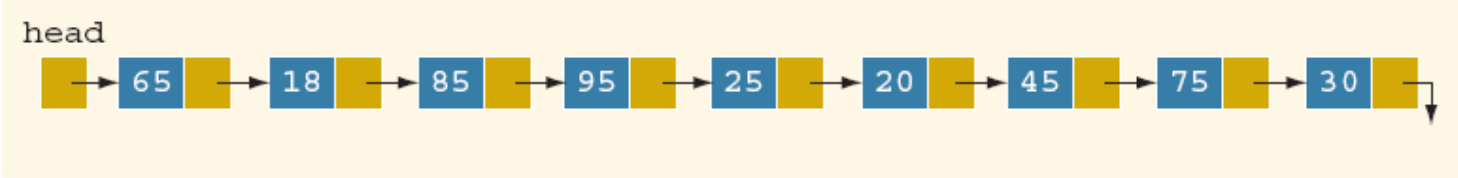

FIGURE 19-49 Unsorted linked list

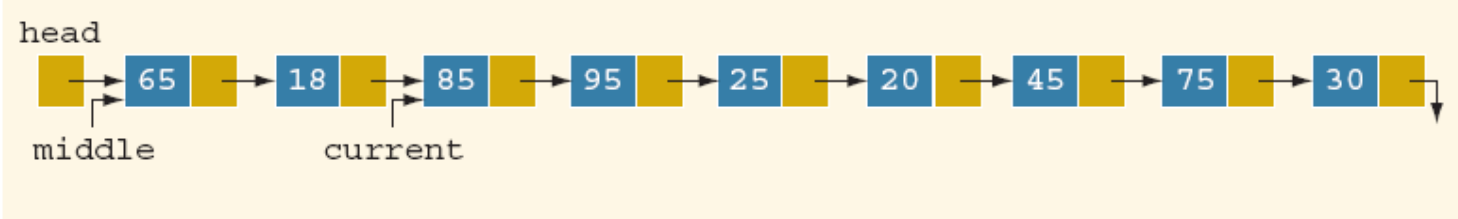

FIGURE 19-50 `middle` and `current` before traversing the list

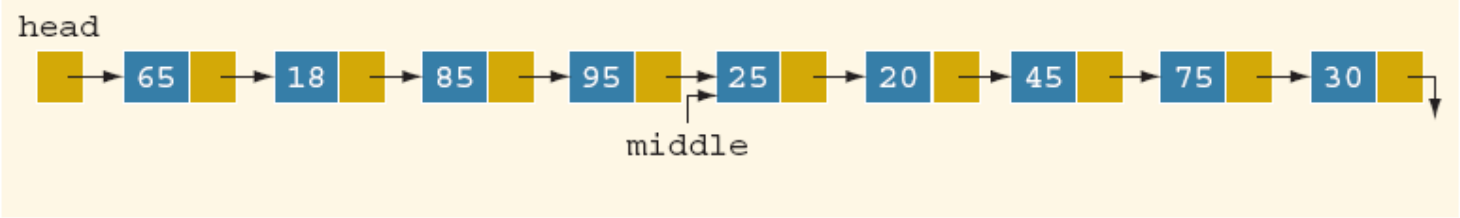

FIGURE 19-51 `middle` after traversing the list

Divide (continued)

- Every time we advance `middle` by one node, we advance `current` by one node
- After advancing `current` by one node, if it is not `NULL`, we again advance it by one node
  - Eventually, `current` becomes `NULL` and `middle` points to the last node of first sublist

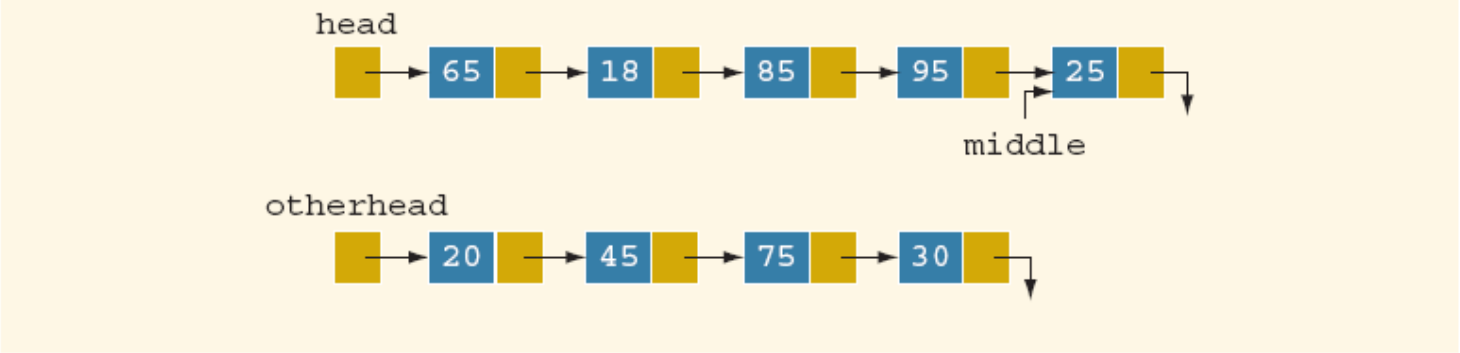

FIGURE 19-52 List after dividing it into two lists

Merge

- Sorted sublists are merged into a sorted list by comparing the elements of the sublists and then adjusting the pointers of the nodes with the smaller `info`

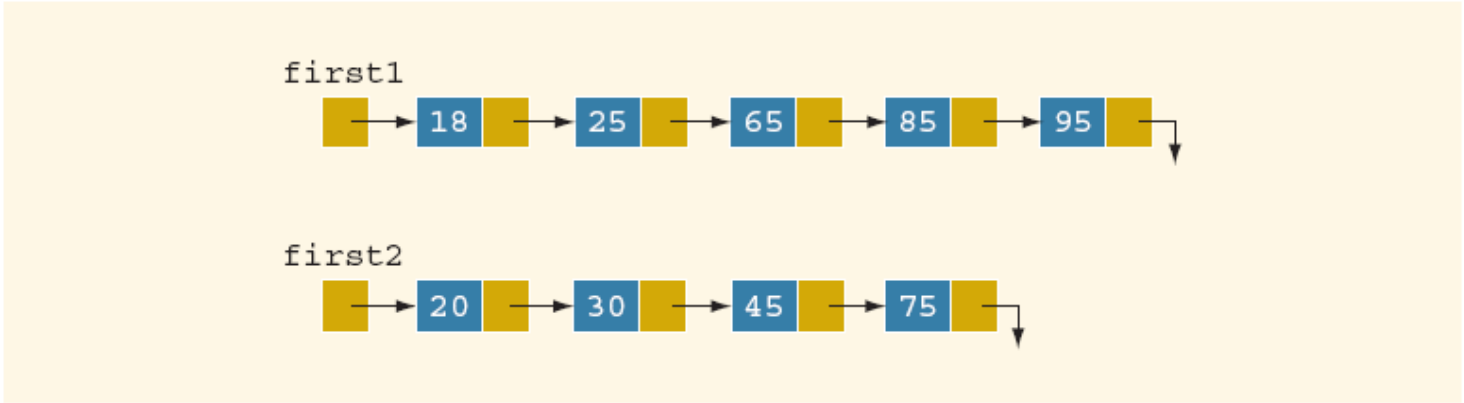

FIGURE 19-53 Sublists before merging

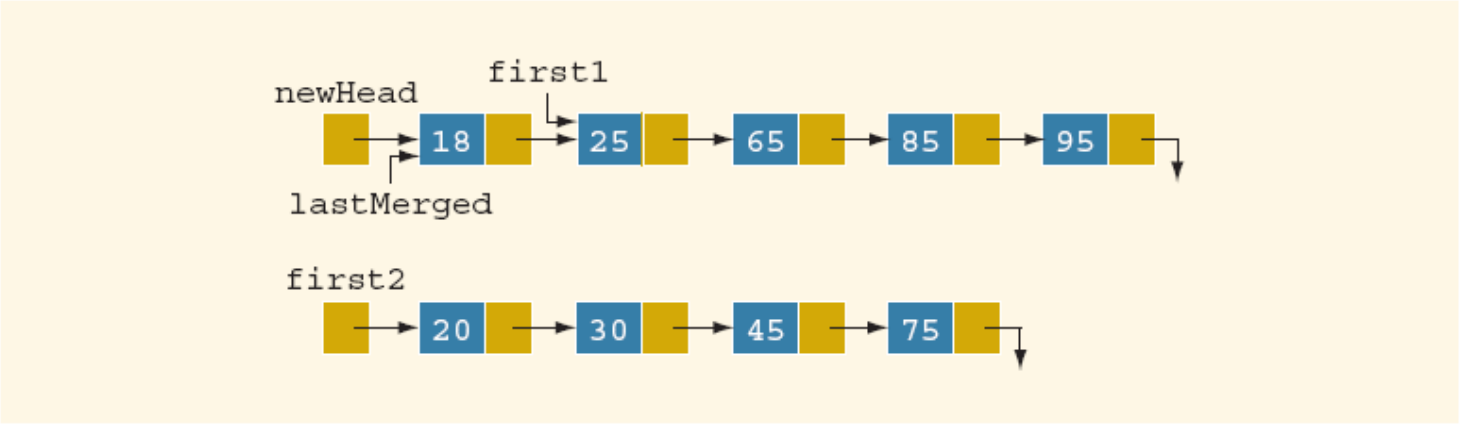

FIGURE 19-54 Sublists after setting `newHead` and `lastMerged` and advancing `first1`

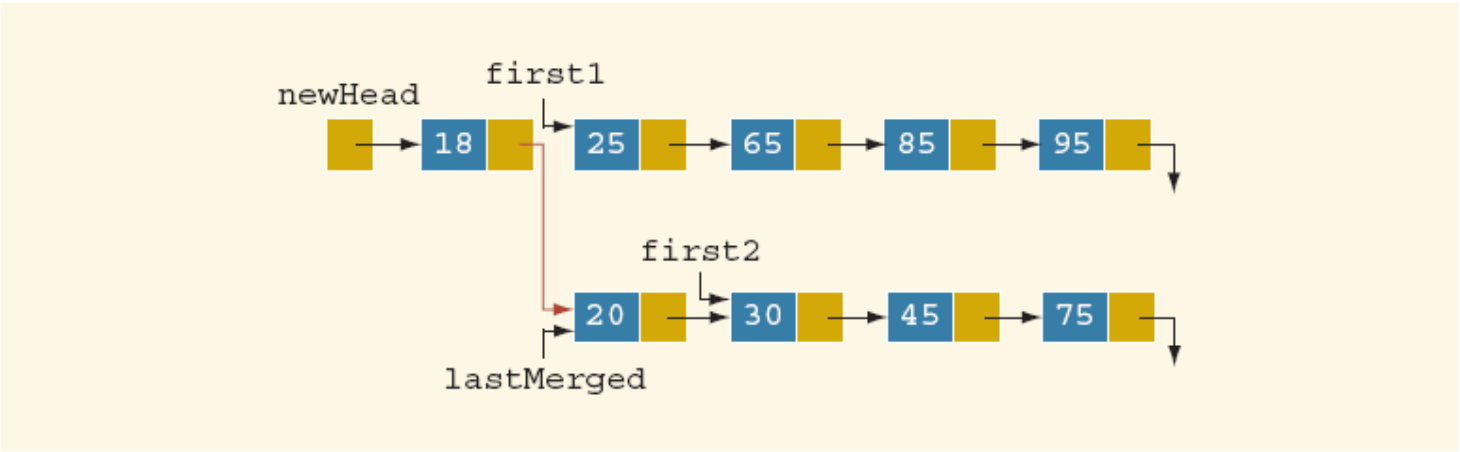

FIGURE 19-55 Merged list after putting the node with `info` 20 at the end of the merged list

Analysis: Merge Sort

- Suppose that *L* is a list of *n* elements, where $n > 0$
- Suppose that *n* is a power of 2; that is, $n = 2^m$ for some nonnegative integer *m*, so that we can divide the list into two sublists, each of size:

$$\frac{n}{2} = \frac{2^m}{2} = 2^{m-1}$$

  - *m* is the number of recursion levels

Analysis: Merge Sort (continued)

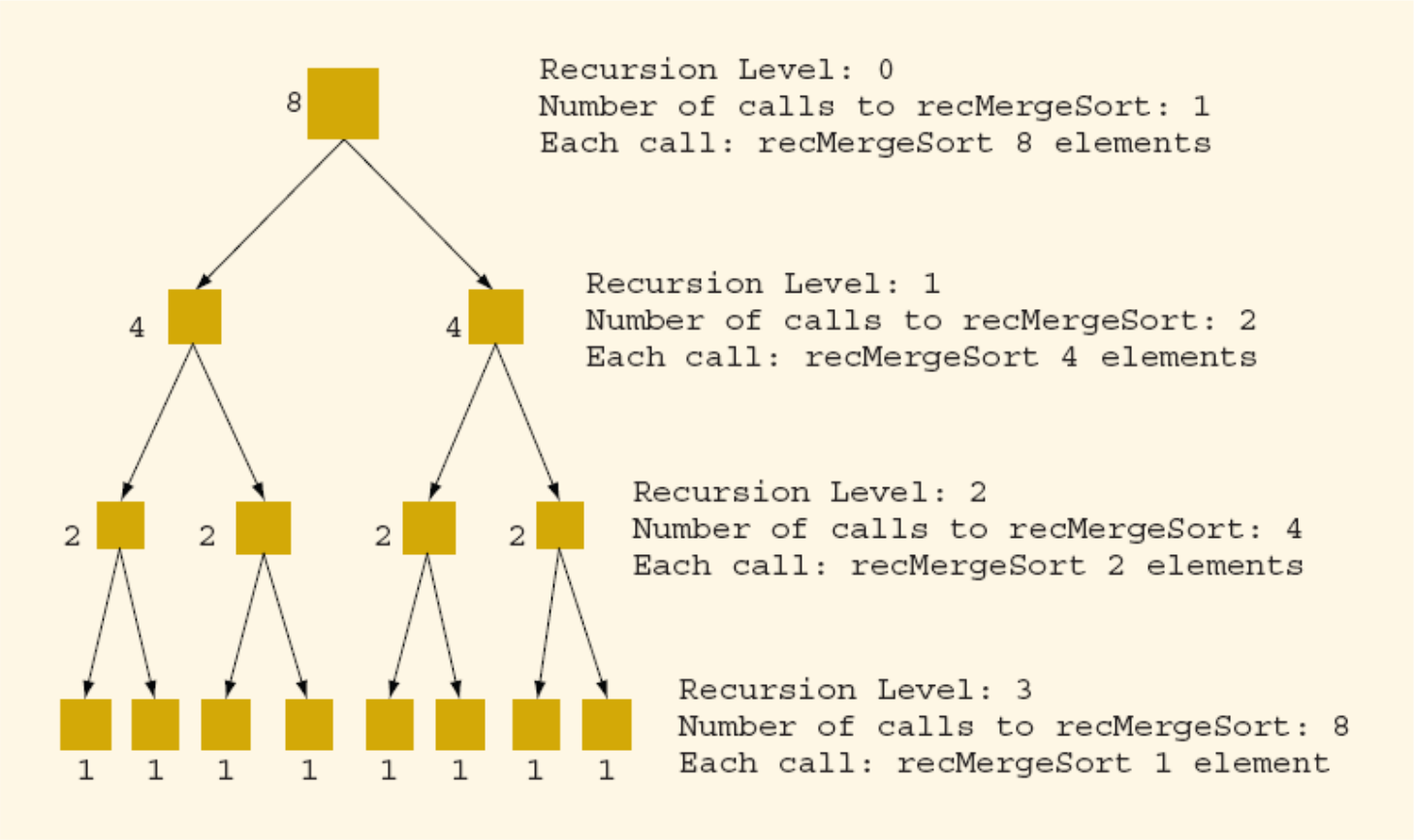

FIGURE 19-56 Levels of recursion to `recMergeSort` for a list of length 8

Analysis: Merge Sort (continued)

- To merge a sorted list of size *s* with a sorted list of size *t*, the maximum number of comparisons is *s* + *t* – 1
- The function `mergeList` merges two sorted lists into a sorted list
  - This is where the actual work (comparisons and assignments) is done
  - Max. # of comparisons at level *k* of recursion:

$$2^k\left(\frac{n}{2^k}-1\right) = n - 2^k = O(n).$$

Analysis: Merge Sort (continued)

- The maximum number of comparisons at each level of the recursion is $O(n)$
  - The maximum number of comparisons is $O(nm)$, where m is the number of levels of the recursion; since $n = 2^m$ ➔ $m = \log_2 n$
  - Thus, $O(nm) \equiv O(n \log_2 n)$
- $W(n)$: # of key comparisons in the worst case
- $A(n)$: # of key comparisons in average case

$$W(n) = O(n \log_2 n).$$

$$A(n) = n \log_2 n - 1.25n = O(n \log_2 n).$$