More on pointer-based merge sort for sorted lists. Suppose that in the merge step, we check if all elements in one list are bigger than all the elements in the other list (this is true if the last item in one list is smaller than the first item in the other list). We can check this in constant time if we have pointers to the first and last elements in each list, and we can merge the two lists in constant time if they are given to us as linked lists.

Then if we are given a sorted list, this version of merge sort will return in $O(n)$ time.

Proof. The recurrence relation is given by

$$T(n) = 2T(n/2) + c, T(1) = d,$$

where c and d are constants. Solving the recurrence via substitution, we get:

$$\begin{aligned} T(n) &= 2T(n/2) + c \\ &= 2(2T(n/4) + c) + c \\ &= 2(2(2T(n/8) + c) + c) + c \\ &= \dots \\ &= 2^k T(n/2^k) + 2^{k-1}c + 2^{k-2}c + \dots + 4c + 2c + c \\ &= 2^k T(n/2^k) + c\sum_{i=0}^{k-1} 2^i \end{aligned}$$

To solve the recurrence, we need to know what k causes $n/2^k = 1$. Multiplying both sides by 2^k and taking the $\log$ of both sides, we get $k = \log_2 n$.

All that remains is to solve $\sum_{i=0}^{k-1} 2^i$ and plug in k. This is in your book on page 4; it is $2^{k-1+1} - 1$. So we get

$$\begin{aligned} T(n) &= 2^k T(n/2^k) + c\sum_{i=0}^{k-1} 2^i \\ &= 2^{\log_2 n} T(n/2^{\log_2 n}) + c(2^{\log_2 n - 1 + 1} - 1) \\ &= nT(1) + c(n-1) \\ &= nd + nc - c \\ &= O(n) \end{aligned}$$

□