Mergesort

DPV Ch 2.3

Let L be a given list of length n. We wish to analyze the running time of the recursive procedure MERGESORT(L) that calls the procedure MERGE($\cdot$, $\cdot$) as a subroutine.

```
/* sort input list L and return the sorted list */
MERGESORT(L) {
    if n = 1 then return L                          /* base case */
    else {
        A ← any ⌊n/2⌋ elements of L                 /* divide */
        B ← the remaining ⌈n/2⌉ elements of L
        S_A ← MERGESORT(A)                          /* recurse */
        S_B ← MERGESORT(B)
        return MERGE(S_A, S_B)                      /* combine */
    }
}

/* merge the sorted lists S_1 and S_1 and return the merged list */
MERGE(S_1, S_2) {
    S ← empty list
    while both S_1 and S_2 are not empty do {
        s_1 ← front element of S_1
        s_2 ← front element of S_2
        if s_1 < s_2 then remove s_1 from S_1 and add it to the end of S
        else remove s_2 from S_2 and add it to the end of S
    }
    if S_1 is empty then append S_2 to S
    if S_2 is empty then append S_1 to S
    return S
}
```

Let $T(n)$ denote the worst-case running time of MERGESORT(L). We give two methods for asymptotically upper-bounding $T(n)$.

1. Iteration method. We first assume $n = 2^k$ for some integer k. (We may assume) $T(n)$ satisfies

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ n + 2T(n/2) & \text{if } n > 1. \end{cases}$$

We iterate the recurrence to get

$$\begin{aligned} T(n) &= n + 2T(n/2) \\ &= n + 2(n/2) + 4T(n/4) \\ &= n + 2(n/2) + 4(n/4) + \cdots + 2^i T(n/2^i) \\ &= n + 2(n/2) + 4(n/4) + \cdots + 2^i(n/2^i) + \cdots + 2^k(n/2^k) \qquad [T(n/2^k) = T(1) = 1] \\ &= n(1+k) \\ &= n(1 + \log n). \end{aligned}$$

Thus, $T(n) = O(n \log n)$ for all n that's a power of 2.

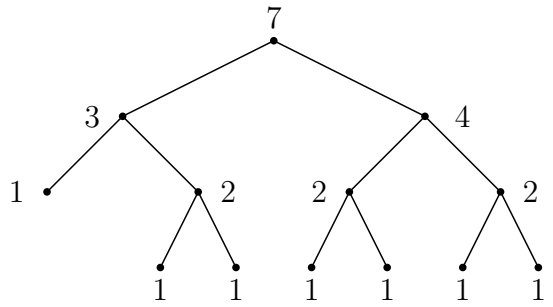
Now let n be an arbitrary positive integer. Let $p = 2^{\lceil \log n \rceil}$. We have $T(n) \le T(p) = p(1 + \log p) \le 2n(1 + \log 2n)$.

Therefore, $T(n) = O(n \log n)$ in general.

The above analysis is correct for a "simple algorithm" where we make n a power of 2 by padding with dummy numbers.

2. Recursion tree analysis. Any recursive algorithm has a *recursion tree* where the root represents the initial call, and the children of any node represent its recursive calls.

Let n be an arbitrary positive integer. Consider the recursion tree for MERGESORT. Each node has an associated input list length. For example,



Recursion tree analysis relates the time of an algorithm to its recursion tree. For MERGESORT, we reason as follows.

(i) The time per level is $O(n)$ since each node spends $O(1)$ time on each element in its list (in the MERGE step), and any given element is in no more than 1 node at each level.

(ii) The number of levels is $\leq \lceil \log n \rceil + 1$.

(i) & (ii) imply the total time for MERGESORT is $O(n \log n)$.

Remarks.

1. In practice, we implement MERGESORT as an iterative algorithm: *merge lists of length 1, 2, 4, ...* This corresponds to a bottom-up traversal of the recursion tree.

2. In the analysis of a divide-and-conquer algorithm, we usually ignore the time needed to set/clean up the recursive procedures. This is justifiable when this time is dominated by the time required for combining the solutions to subproblems.