Announcements

- **Weekly Reading Assignment:**
  - this week: Chapter 7 & 8 (CLRS)
  - next week: Chapter 9 (CLRS)

- **Extra Help Session: SN014, starting at 4:50pm this Thursday, September 29, 2005 in SN014**

QuickSort

- ***Divide***: $A[p \ldots r]$ is partitioned (rearranged) into two nonempty subarrays $A[p \ldots q]$ and $A[q+1 \ldots r]$ s.t. each element of $A[p \ldots q]$ is less than or equal to each element of $A[q+1 \ldots r]$. Index q is computed here.
- ***Conquer***: two subarrays are sorted by recursive calls to quicksort.
- ***Combine***: no work needed since the subarrays are sorted in place already.

Quicksort (A, p, r)

1. if $p < r$
2. then $q \leftarrow$ Partition(A, p, r)
3. Quicksort(A, p, q)
4. Quicksort(A, $q+1$, r)

* In place, not stable

Partition(A, p, r)

1. $x \leftarrow A[p]$
2. $i \leftarrow p - 1$
3. $j \leftarrow r + 1$
4. while TRUE
5. do repeat $j \leftarrow j - 1$
6. until $A[j] \leq x$
7. repeat $i \leftarrow i + 1$
8. until $A[i] \geq x$
9. if $i < j$
10. then exchange $A[i] \leftrightarrow A[j]$
11. else return j

Example: Partitioning Array

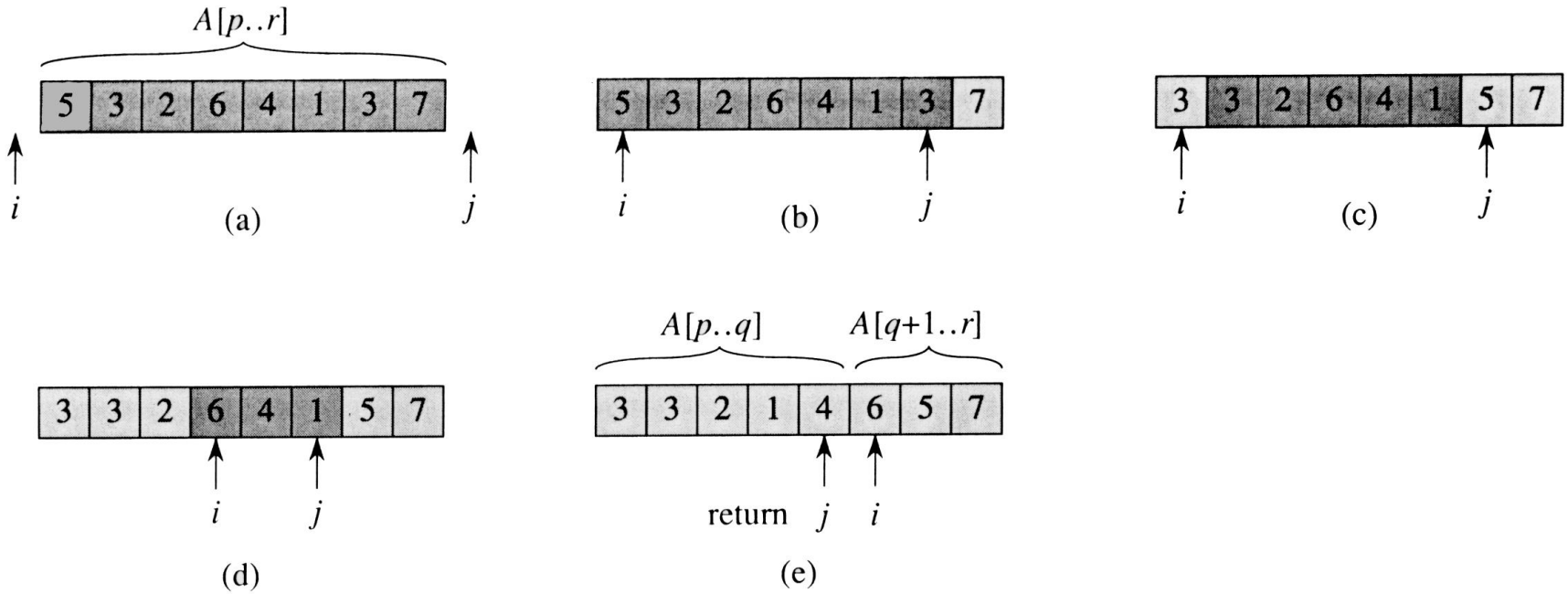

Figure 8.1 The operation of PARTITION on a sample array. Lightly shaded array elements have been placed into the correct partitions, and heavily shaded elements are not yet in their partitions. (**a**) The input array, with the initial values of i and j just off the left and right ends of the array. We partition around $x = A[p] = 5$. (**b**) The positions of i and j at line 9 of the first iteration of the **while** loop. (**c**) The result of exchanging the elements pointed to by i and j in line 10. (**d**) The positions of i and j at line 9 of the second iteration of the **while** loop. (**e**) The positions of i and j at line 9 of the third and last iteration of the **while** loop. The procedure terminates because $i \geq j$, and the value $q = j$ is returned. Array elements up to and including $A[j]$ are less than or equal to $x = 5$, and array elements after $A[j]$ are greater than or equal to $x = 5$.

Algorithm Analysis

The running time of quicksort depends on whether the partitioning is balanced or not.

- **Worst-Case Performance (unbalanced):**

$$T(n) = T(1) + T(n-1) + \Theta(n) \Leftarrow \textit{partitioning takes } \Theta(n)$$

$$= \sum_{k=1 \text{ to } n} \Theta(k) \Leftarrow T(1) \textit{ takes } \Theta(1) \text{ time \& reiterate}$$

$$= \Theta\left(\sum_{k=1 \text{ to } n} k\right)$$

$$= \Theta(n^2)$$

* This occurs when the input is completely sorted.

Worst Case Partitioning

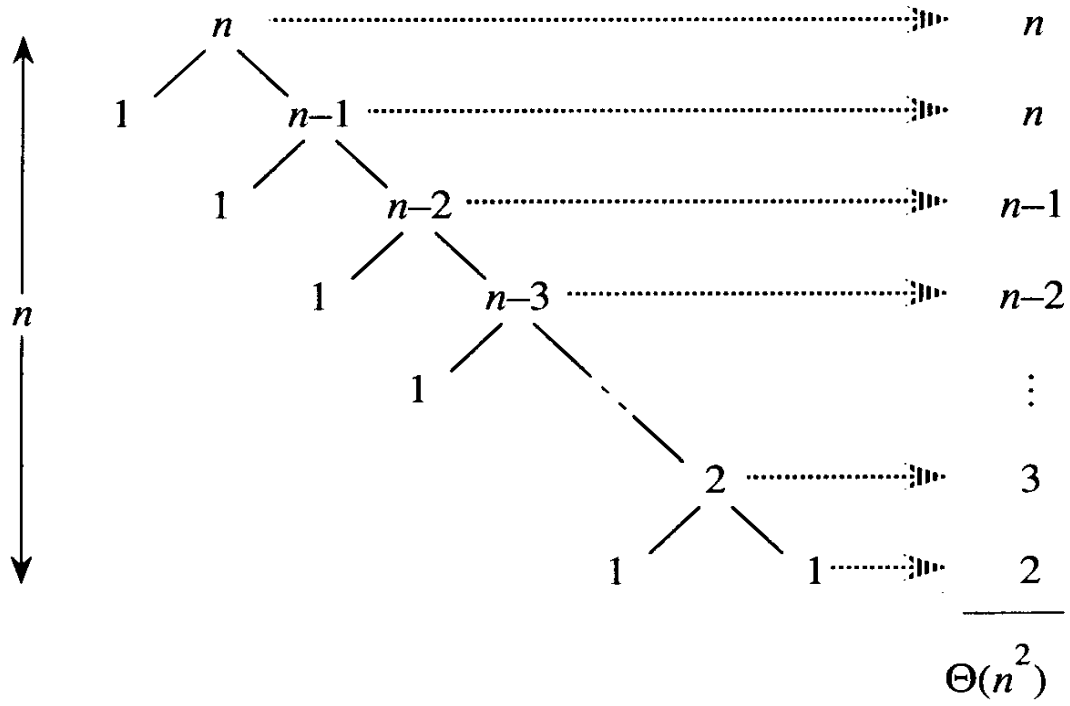

Figure 8.2 A recursion tree for QUICKSORT in which the PARTITION procedure always puts only a single element on one side of the partition (the worst case). The resulting running time is $\Theta(n^2)$.

Best Case Partitioning

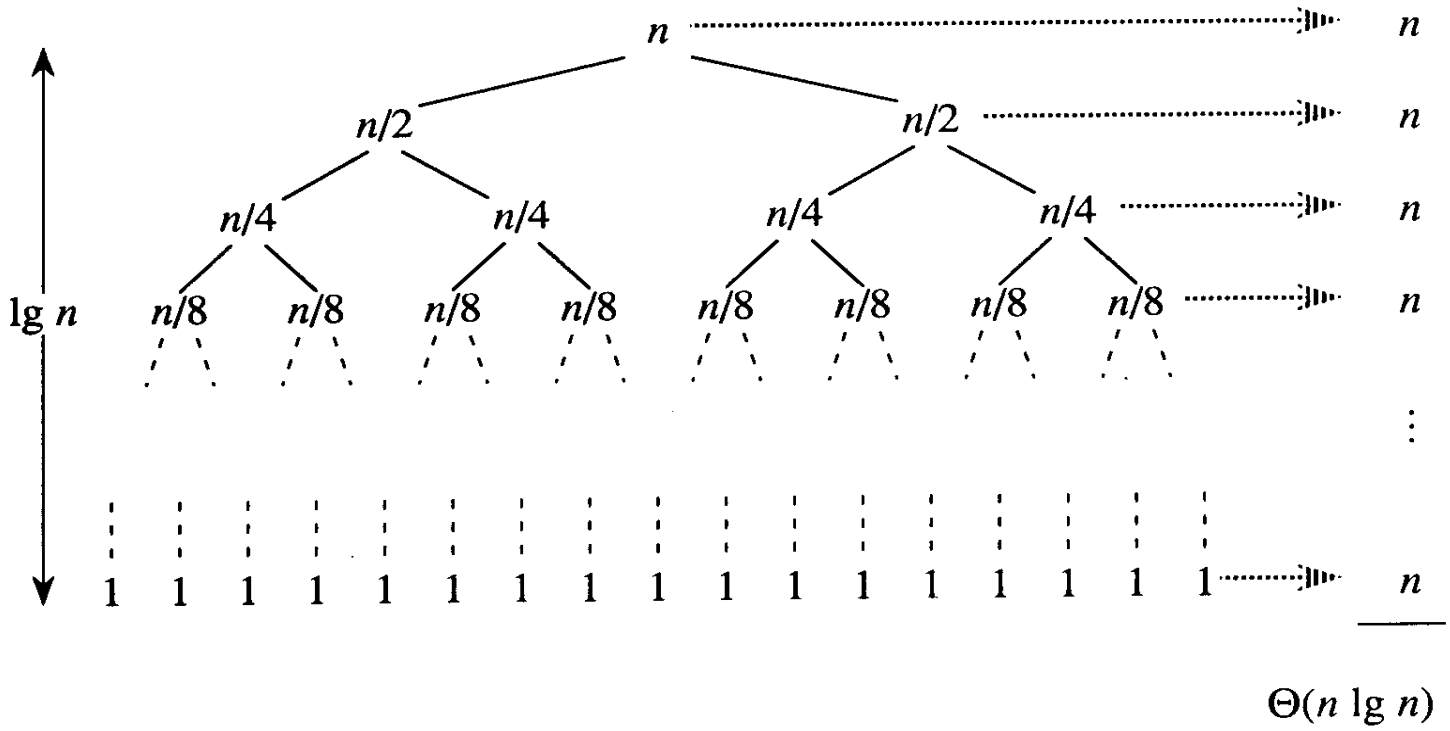

Figure 8.3 A recursion tree for QUICKSORT in which PARTITION always balances the two sides of the partition equally (the best case). The resulting running time is $\Theta(n \lg n)$.

Analysis for Best Case Partition

- When the partitioning procedure produces two regions of size $n/2$, we get the a balanced partition with best case performance:

$$T(n) = 2T(n/2) + \Theta(n)$$

$$\text{So, } T(n) = \Theta(n \log n)$$

- *Can it perform better than O(n logn) on any input?*

Average-Case Behavior

- For example, when the partitioning algorithm always produces a 7-to-3 proportional split:

$$T(n) = T(7n/10) + T(3n/10) + n$$

Solve the recurrence by visualizing recursion tree, each level has a cost of n with the height of $lg\ n$. So, we get $T(n) = \Theta(n\ log\ n)$ when the split has constant proportionality.

- For a split of proportionality α, where $0 \le \alpha \le 1/2$, the minimum depth of the tree is $- lg\ n / lg\ \alpha$ and the maximum depth is $- lg\ n / lg\ (1- \alpha)$.

Average-Case Splitting

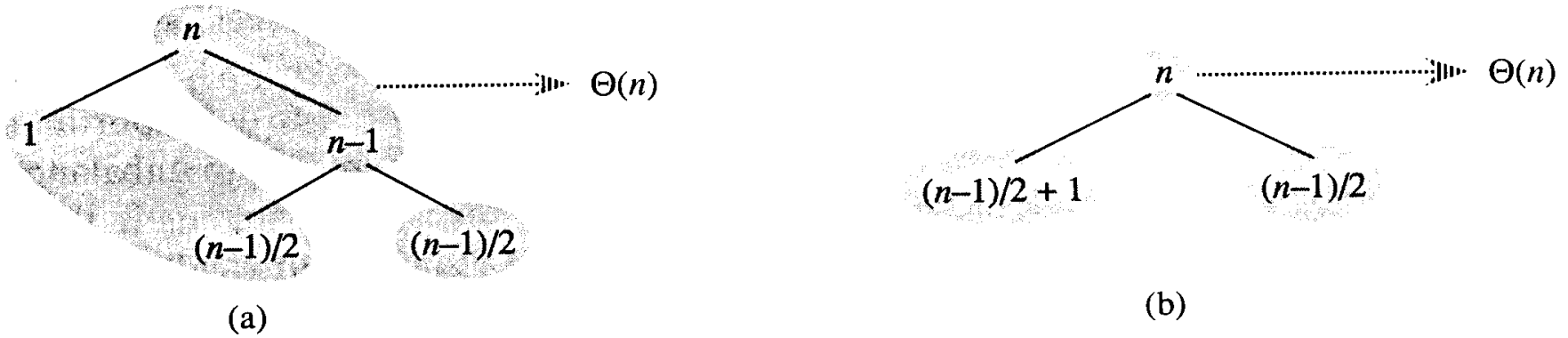

Figure 8.5 **(a)** Two levels of a recursion tree for quicksort. The partitioning at the root costs n and produces a "bad" split: two subarrays of sizes 1 and $n - 1$. The partitioning of the subarray of size $n - 1$ costs $n - 1$ and produces a "good" split: two subarrays of size $(n - 1)/2$. **(b)** A single level of a recursion tree that is worse than the combined levels in (a), yet very well balanced.

The combination of good and bad splits would result in $T(n) = \Theta(n \log n)$, but with slightly larger constant hidden by the O-notation. (Rigorous average-case analysis later)

Randomized Quicksort

- An algorithm is *randomized* if its behavior is determined not only by the input but also by values produced by a *random-number generator*. No particular input elicits worst-case behavior. Two possible versions of quicksort:
  - Impose a distribution on input to ensure that every permutation is equally likely. This does not improve the worst-case running time, but makes run time independent of input ordering.
  - Exchange $A[p]$ with an element chosen at random from $A[p \ldots r]$ in Partition. This ensures that the pivot element is equally likely to be any of input elements.

Randomized-Partition(*A*, *p*, *r*)

1. $i \leftarrow$ *Random (p, r)*
2. exchange $A[p] \leftrightarrow A[i]$
3. return Partition*(A, p, r)*

Randomized-Quicksort (A, p, r)

1. if $p < r$
2. then $q \leftarrow$ Randomized-Partition(A, p, r)
3. Randomized-Quicksort(A, p, q)
4. Randomized-Quicksort(A, $q+1$, r)

Worst-Case Analysis

- $T(n) = \max_{1 \le q \le n-1} (T(q) + T(n - q)) + \Theta(n)$

- **Substitution method: Guess** $T(n) \le cn^2$

$$T(n) \le \max_{1 \le q \le n-1} (cq^2 + c(n - q)^2) + \Theta(n)$$

$$= c \cdot \max_{1 \le q \le n-1} (q^2 + (n - q)^2) + \Theta(n)$$

Take derivatives to get maximum at $q = 1, n-1$**:**

$$T(n) \le cn^2 - 2c(n - 1) + \Theta(n) \le cn^2$$

Therefore, the worst case running time is $\Theta(n^2)$

Average-Case Analysis

- Partitioning depends only on the rank of $x = A[p]$.
- When $\text{rank}(x) = 1$, index i stops at $i = p$ and j stops $j = p$. $q = p$ is returned. So, the probability of the low side has one element is $1/n$.
- When $\text{rank}(x) \geq 2$, there is at least one element smaller than $x = A[p]$. When the "Partition" terminates, each of the $\text{rank}(x)$-1 elements in the low side of the partition is strictly less than x. For each $i = 1, \ldots, n-1$, the probability is $1/n$ that the low side has i elements.

Recurrence for Average-Case

- Combining two cases, the size $q - p + 1$ of low side partition is 1 with probability of $2/n$, and the size is i with probability of $1/n$ for $i = 2,\ldots,n-1$. So,

$$T(n) = 1/n(\mathrm{T}(1) + \mathrm{T}(n-1) + \sum_{q=1 \text{ to } n-1}(T(q)+T(n-q))) + \Theta(n)$$

$$= 1/n\ (\Theta(1) + \mathrm{O}(\mathrm{n}^2) + \sum_{q=1 \text{ to } n-1} T(q)+T(n-q)) + \Theta(n)$$

$$= 1/n\ (\sum_{q=1 \text{ to } n-1} T(q)+T(n-q)) + \Theta(n)$$

$$= 2/n\ (\sum_{k=1 \text{ to } n-1} T(k)) + \Theta(n)$$

Solving Recurrence

- **Substitution Method: Guess** $T(n) \le a\, n \lg n + b$

$$
\begin{aligned}
T(n) &= 2/n \left(\sum_{k=1 \text{ to } n-1} T(k)\right) + \Theta(n) \\
&\le 2/n \left(\sum_{k=1 \text{ to } n-1} a\, k \lg k + b\right) + \Theta(n) \\
&= \left(2a/n \sum_{k=1 \text{ to } n-1} k \lg k\right) + 2b(n-1)/n + \Theta(n) \\
&\le 2a/n(n^2 \lg n/2 - n^2/8) + 2b(n-1)/n + \Theta(n) \\
&\le a\, n \lg n + b + (\Theta(n) + b - an/4) \\
&\le a\, n \lg n + b \qquad \Leftarrow \text{if we choose } a \text{ large enough}
\end{aligned}
$$