CST 311
Algorithm Analysis & Design

Al Lake
Oregon Institute of Technology

Chapter 7

Quicksort

Quicksort

- Quicksort is a sorting algorithm whose worst-case running time is $\Theta(n^2)$ on an input array of n numbers.
- In spite of the worst-case running time, quicksort is often the most practical choice for sorting because of it's average running time:

  $\Theta(n \lg n)$

Quicksort

- A divide-and-conquer algorithm which does the following:
  - Divide: partition (rearrange) the array into two (possibly empty) subarrays
  - Conquer: sort the two subarrays by recursive calls to quicksort.
  - Combine: pull the two subarrays together.

Quicksort

```
QUICKSORT(A, p, r)
if p < r
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q − 1)
         QUICKSORT(A, q + 1, r)
```

Partition the array

```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
     do if A[j] ≤ x
           then i ← i + 1
                exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

Partition the array

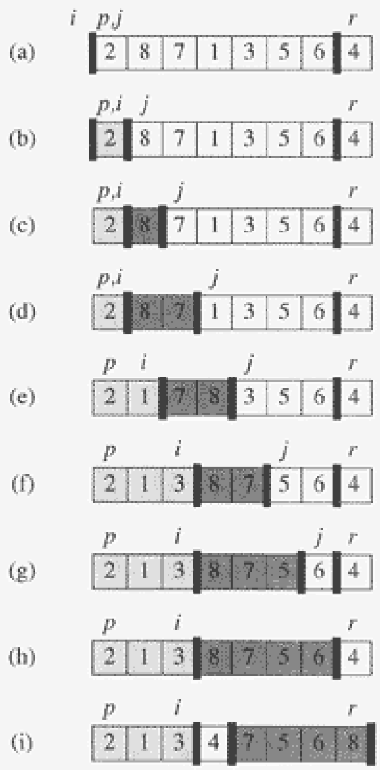

Figure 7.1 The operation of PARTITION on a sample array. Lightly shaded array elements are all in the first partition with values no greater than x. Heavily shaded elements are in the second partition with values greater than x. The unshaded elements have not yet been put in one of the first two partitions, and the final white element is the pivot. **(a)** The initial array and variable settings. None of the elements have been placed in either of the first two partitions. **(b)** The value 2 is "swapped with itself" and put in the partition of smaller values. **(c)–(d)** The values 8 and 7 are added to the partition of larger values. **(e)** The values 1 and 8 are swapped, and the smaller partition grows. **(f)** The values 3 and 8 are swapped, and the smaller partition grows. **(g)–(h)** The larger partition grows to include 5 and 6 and the loop terminates. **(i)** In lines 7–8, the pivot element is swapped so that it lies between the two partitions.

Partition the array

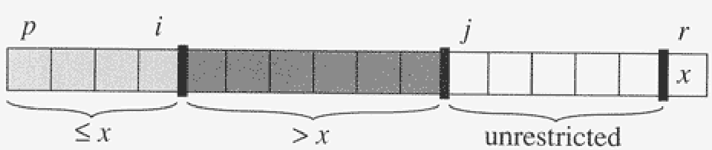

Figure 7.2 The four regions maintained by the procedure PARTITION on a subarray $A[p\,..\,r]$. The values in $A[p\,..\,i]$ are all less than or equal to x, the values in $A[i+1\,..\,j-1]$ are all greater than x, and $A[r] = x$. The values in $A[j\,..\,r-1]$ can take on any values.

Partition the array

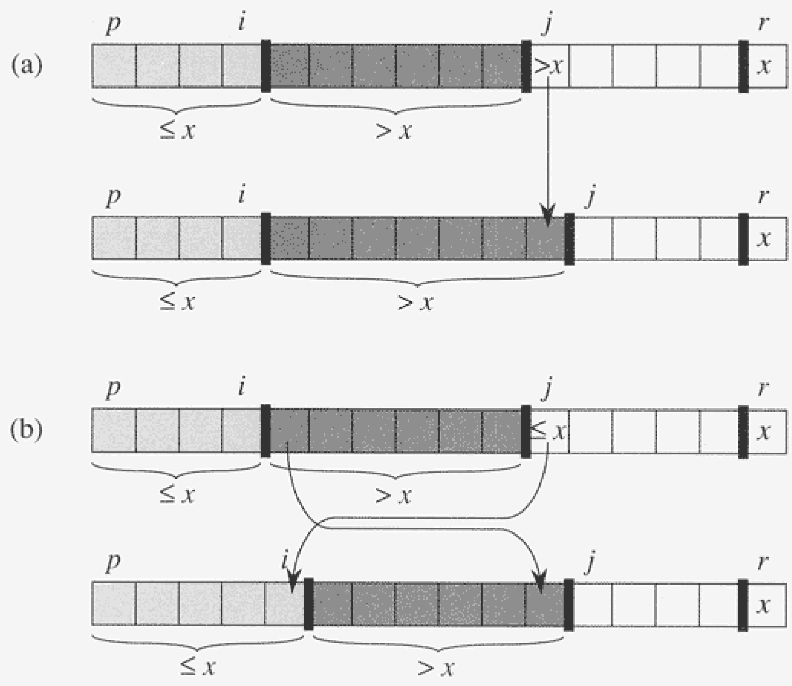

Figure 7.3 The two cases for one iteration of procedure PARTITION. **(a)** If $A[j] > x$, the only action is to increment j, which maintains the loop invariant. **(b)** If $A[j] \leq x$, index i is incremented, $A[i]$ and $A[j]$ are swapped, and then j is incremented. Again, the loop invariant is maintained.

Recursion tree

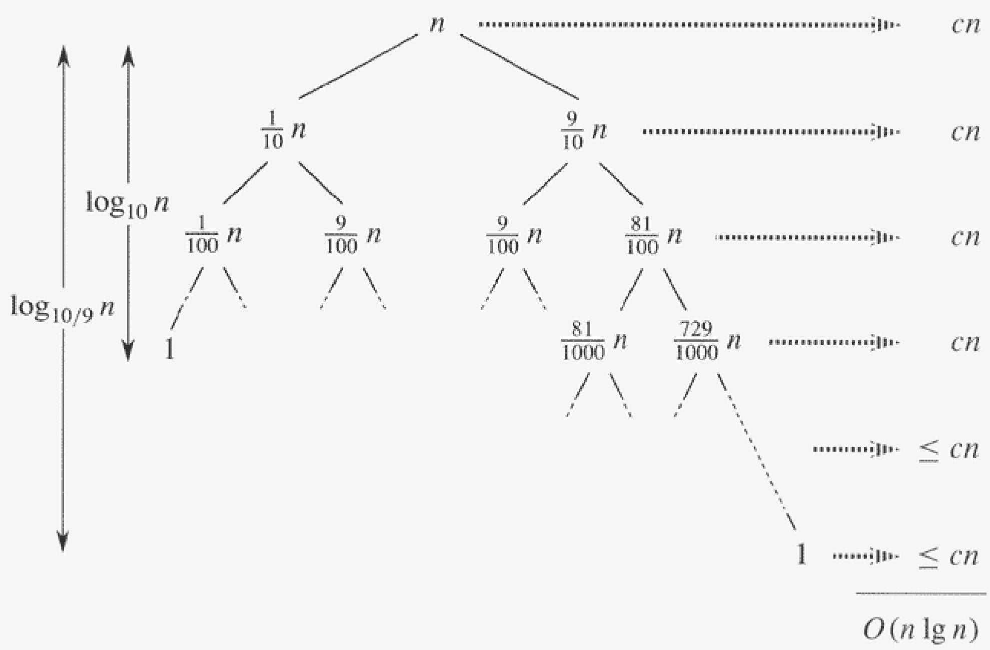

Figure 7.4 A recursion tree for QUICKSORT in which PARTITION always produces a 9-to-1 split, yielding a running time of $O(n \lg n)$. Nodes show subproblem sizes, with per-level costs on the right. The per-level costs include the constant c implicit in the $\Theta(n)$ term.

Recursion tree

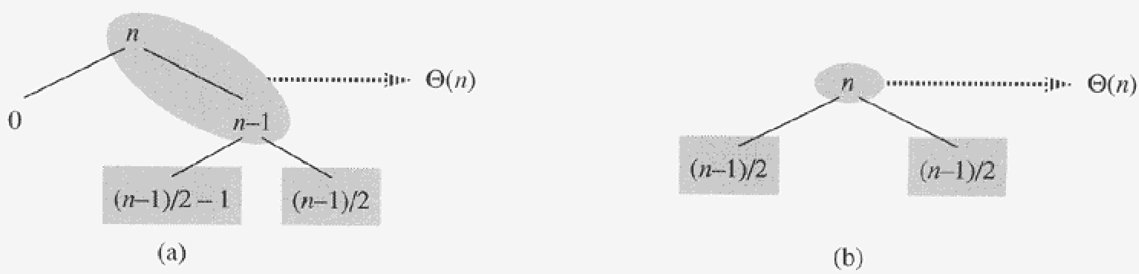

Figure 7.5 **(a)** Two levels of a recursion tree for quicksort. The partitioning at the root costs n and produces a "bad" split: two subarrays of sizes 0 and $n-1$. The partitioning of the subarray of size $n-1$ costs $n-1$ and produces a "good" split: subarrays of size $(n-1)/2-1$ and $(n-1)/2$. **(b)** A single level of a recursion tree that is very well balanced. In both parts, the partitioning cost for the subproblems shown with elliptical shading is $\Theta(n)$. Yet the subproblems remaining to be solved in (a), shown with square shading, are no larger than the corresponding subproblems remaining to be solved in (b).

Analysis of quicksort

- **The text shows the proof. What is the worst case behavior for quicksort? Can you demonstrate this?**
- **What is the dominating element for quicksort?**