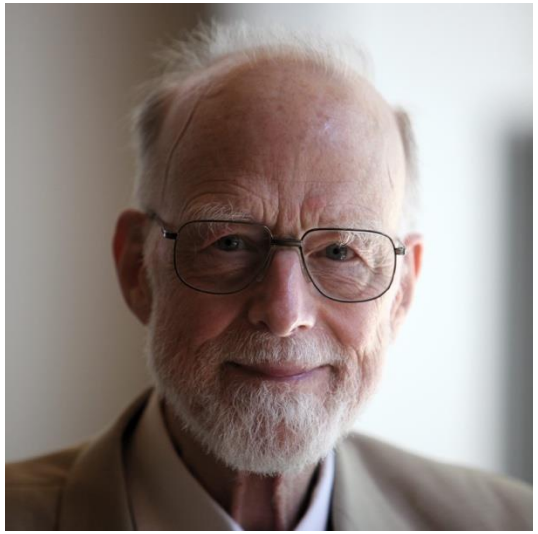
Quicksort (1959)

Adapted from the CLRS book slides

OVERVIEW

Goals

- Developed by Tony Hoare in 1959
- Worst-case running time is $\Theta(n^2)$.
- Randomized version has expected running time $\Theta(n \lg n)$ assuming that all elements to be sorted are distinct.
- Constants hidden in $\Theta(n \lg n)$ are small.
- Sorts in place.



DESCRIPTION OF QUICKSORT

Quicksort is based on the three-step process of divide-and-conquer.

- To sort the subarray $A[p:r]$:

  Divide: Partition $A[p:r]$, into two (possibly empty) subarrays $A[p:q-1]$ and $A[q+1:r]$, such that each element in the first subarray $A[p:q-1]$ is $\leq A[q]$ and $A[q]$ is $\leq$ each element in the second subarray $A[q+1:r]$.

  Conquer: Sort the two subarrays by recursive calls to QUICKSORT.

  Combine: No work is needed to combine the subarrays, because they are sorted in place.

- Perform the divide step by a procedure PARTITION, which returns the index q that marks the position separating the subarrays.

PSEUDOCODE

```
QUICKSORT(A, p, r)
1  if p < r
2      // Partition the subarray around the pivot, which ends up in A[q].
3      q = PARTITION(A, p, r)
4      QUICKSORT(A, p, q − 1)  // recursively sort the low side
5      QUICKSORT(A, q + 1, r)  // recursively sort the high side
```

Initial call is QUICKSORT$(A, 1, n)$.

PARTITIONING

Partition subarray $A[p:r]$ by the following procedure:

PARTITION(A, p, r)

```
1  x = A[r]                          // the pivot
2  i = p − 1                         // highest index into the low side
3  for j = p to r − 1                // process each element other than the pivot
4      if A[j] ≤ x                   // does this element belong on the low side?
5          i = i + 1                 // index of a new slot in the low side
6          exchange A[i] with A[j]   // put this element there
7  exchange A[i + 1] with A[r]       // pivot goes just to the right of the low side
8  return i + 1                      // new index of the pivot
```

PARTITION always selects the last element $A[r]$ in the subarray $A[p:r]$ as the ***pivot***—the element around which to partition.

PARTITIONING (continued)

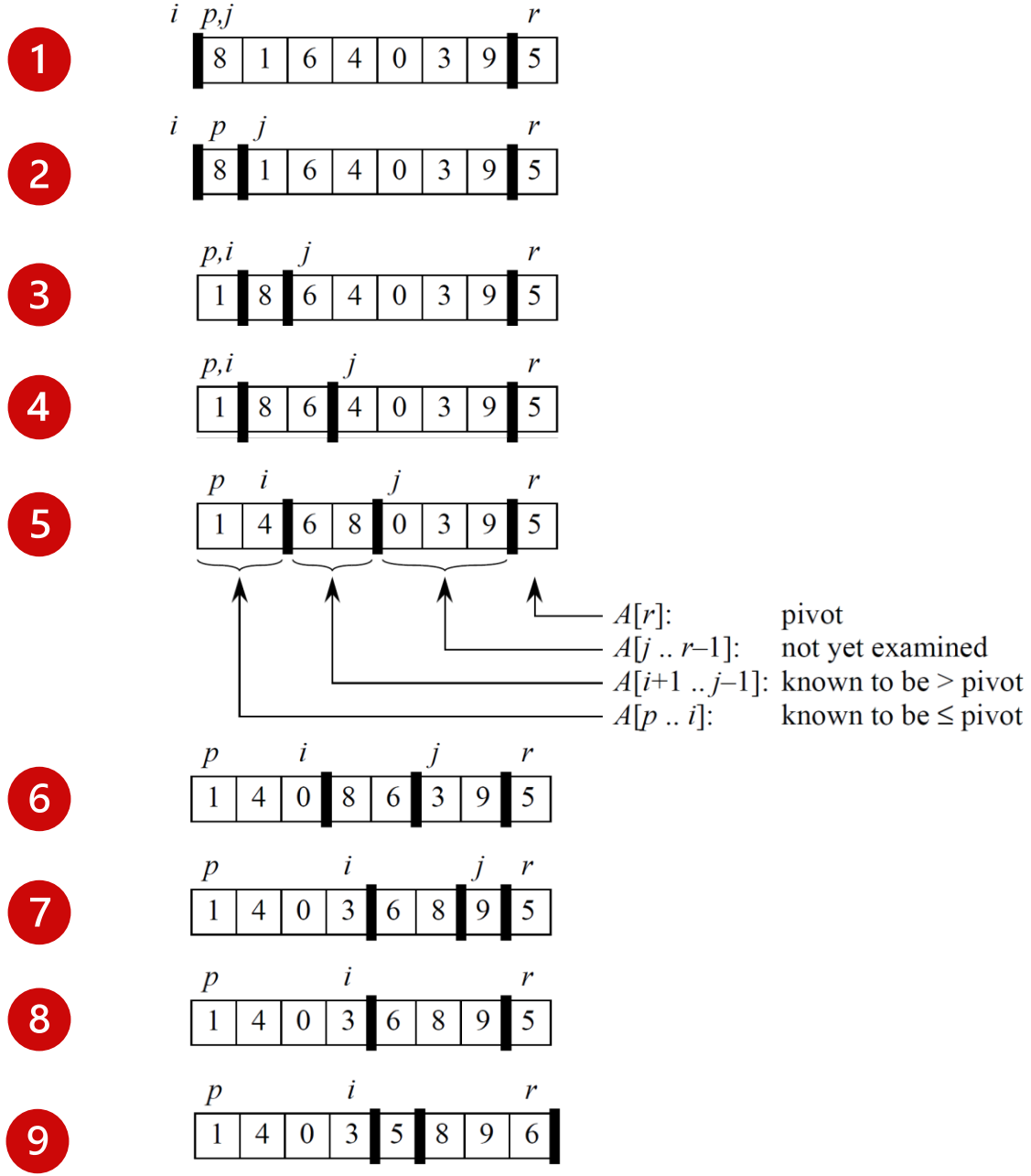
As the procedure executes, the array is partitioned into four regions, some of which may be empty:

Loop invariant:

1. All entries in $A[p : i]$ are $\leq$ pivot.
2. All entries in $A[i + 1 : j - 1]$ are $>$ pivot.
3. $A[r] =$ pivot.

It's not needed as part of the loop invariant, but the fourth region is $A[j : r - 1]$, whose entries have not yet been examined, and so we don't know how they compare to the pivot.

EXAMPLE



Time for partitioning

$\Theta(n)$ to partition an n-element subarray.

PERFORMANCE OF QUICKSORT

The running time of quicksort depends on the partitioning of the subarrays:

- If the subarrays are balanced, then quicksort can run as fast as mergesort.
- If they are unbalanced, then quicksort can run as slowly as insertion sort.

- ***Worst case***
  - Occurs when the subarrays are completely unbalanced.
  - Have 0 elements in one subarray and $n-1$ elements in the other subarray.
  - Get the recurrence

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) \ . \end{aligned}$$

PERFORMANCE OF QUICKSORT (continued)

- ***Best case***
  - Occurs when the subarrays are completely balanced every time.
  - Each subarray has $\leq n/2$ elements.
  - For an upper bound, get the recurrence

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \ . \end{aligned}$$

BALANCED PARTITIONING

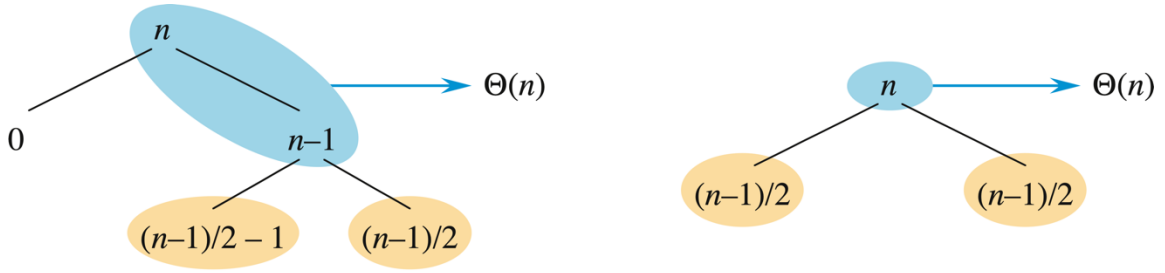
- Quicksort's average running time is much closer to the best case than to the worst case.
- Imagine that PARTITION always produces a 9-to-1 split.
- Get the recurrence

$$T(n) \leq T(9n/10) + T(n/10) + \Theta(n)$$
$$= O(n \lg n) .$$

- Intuition: look at the recursion tree.
- It's like the one for $T(n) = T(n/3) + T(2n/3) + O(n)$ in the recursion tree lecture
- Except that here the constants are different: $\log_{10} n$ full levels and $\log_{10/9} n$ levels that are nonempty.
- As long as it's a constant, the base of the log doesn't matter in asymptotic notation.

INTUITION FOR THE AVERAGE CASE

- Splits in the recursion tree will not always be constant.
- There will usually be a mix of good and bad splits throughout the recursion tree.
- To see that this doesn't affect the asymptotic running time of quicksort, assume that levels alternate between best-case and worst-case splits.



- The extra level in the left-hand figure only adds to the constant hidden in the Θ-notation.
- There are still the same number of subarrays to sort, and only twice as much work was done to get to that point.
- The number of layers is still $\Theta(\lg n)$ (unless we are extremely unlucky!)

RANDOMIZED VERSION OF QUICKSORT

- The extremely unlucky case is indeed possible (e.g., an already-sorted array as input)
- To avoid this, we add randomization to quicksort.
- We could randomly permute the input array.
- Instead, we use ***random sampling***, or picking one element at random.
- Don't always use $A[r]$ as the pivot. Instead, randomly pick an element from the subarray that is being sorted.
- Randomization of quicksort stops any specific type of array from causing worst case behavior.

RANDOMIZED-PARTITION(A, p, r)

1 i = RANDOM(p, r)
2 exchange $A[r]$ with $A[i]$
3 **return** PARTITION(A, p, r)

AVERAGE-CASE ANALYSIS

- Assume that all values being sorted are distinct. (No repeated values.)
- The dominant cost of the algorithm is partitioning.
- Assume that RANDOMIZED-PARTITION makes it so that the pivot selected by PARTITION is selected randomly from the subarray passed to these procedures.
- PARTITION removes the pivot element from future consideration each time.
- Thus, PARTITION is called at most n times.
- QUICKSORT recurses on the partitions.
- The amount of work that each call to PARTITION does is a constant plus the number of comparisons that are performed in its **for** loop.
- Let $X =$ the total number of comparisons performed in all calls to PARTITION.
- Therefore, the total work done over the entire execution is $O(n + X)$.

AVERAGE-CASE ANALYSIS (continued)

Need to compute a bound on the overall number of comparisons.

For ease of analysis:

- Rename the elements of A as $z_1, z_2, \ldots, z_n$, with z_i being the ith smallest element. (So that the output order is $z_1, z_2, \ldots, z_n$.)
- Define the set $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ to be the set of elements between z_i and z_j, inclusive.

Each pair of elements is compared at most once, because elements are compared only with the pivot element, and then the pivot element is never in any later call to PARTITION.

Let $X_{ij} = \mathrm{I}\{z_i \text{ is compared with } z_j\}$.
(Considering whether z_i is compared with z_j at any time during the entire quicksort algorithm, not just during one call of PARTITION.)

AVERAGE-CASE ANALYSIS (continued)

Since each pair is compared at most once, the total number of comparison performed by the algorithm is

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \ .$$

Take expectations of both sides, use linearity of expectation:

$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared with } z_j\} \ . \end{aligned}$$

AVERAGE-CASE ANALYSIS (continued)

- Now all we have to do is find the probability that two elements are compared.
  - Think about when two elements are *not* compared.

  - For example, numbers in separate partitions will not be compared.
  - In the previous example, $\langle 8, 1, 6, 4, 0, 3, 9, 5 \rangle$ and the pivot is 5, so that none of the set $\{1, 4, 0, 3\}$ will ever be compared with any of the set $\{8, 6, 9\}$.
- Once a pivot x is chosen such that $z_i < x < z_j$, then z_i and z_j will never be compared at any later time.
- If either z_i or z_j is chosen before any other element of Z_{ij}, then it will be compared with all the elements of Z_{ij}, except itself.
- The probability that z_i is compared with z_j is the probability that either z_i or z_j is the first element chosen.
- There are $j - i + 1$ elements, and pivots are chosen randomly and independently. Thus, the probability that any particular one of them is the first one chosen is $1/(j - i + 1)$.

AVERAGE-CASE ANALYSIS (continued)

$$
\begin{aligned}
\Pr\{z_i \text{ is compared with } z_j\} &= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\} \\
&\quad + \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\
&= \frac{2}{j-i+1} .
\end{aligned}
$$

AVERAGE-CASE ANALYSIS (continued)

- The expected running time of quicksort, using RANDOMIZED-PARTITION, is $O(n \lg n)$ if all values being sorted are distinct.

$$
\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n) \; .
\end{aligned}
$$