Algorithms
Chapter 7 Quicksort

Assistant Professor: Ching-Chi Lin
林清池 助理教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **Description of Quicksort**
- Performance of Quicksort
- A Randomized Version of Quicksort
- Analysis of Quicksort

Quicksort

- Worst-case running time: $\Theta(n^2)$.
- Best practical choice:
  - Expected running time: $\Theta(n \lg n)$.
  - 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 $A[p \ldots r]$:
  - **Divide:** Partition $A[p \ldots r]$, into two (possibly empty) subarrays $A[p \ldots q-1]$ and $A[q+1 \ldots r]$, such that each element in the first subarray $A[p \ldots q-1]$ is $\leq A[q]$ and $A[q]$ is $\leq$ each element in the second subarray $A[q+1 \ldots r]$.
  - **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.

The Quicksort procedure

QUICKSORT(A, p, r)

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

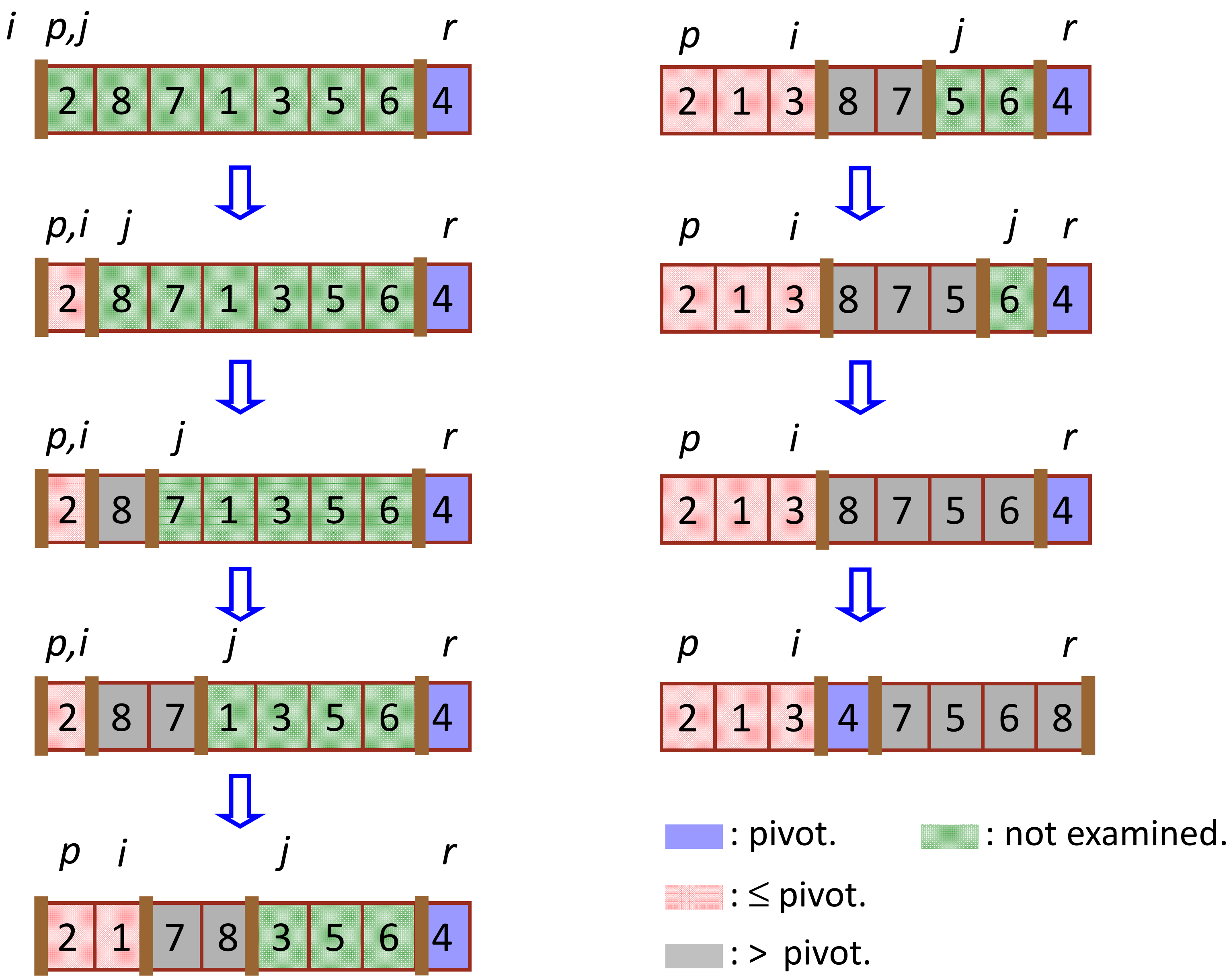
- Initial call is QUICKSORT(A, 1, n).
- Perform the divide step by a procedure PARTITION, which returns the index q.

Partitioning the array

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

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

- PARTITION always selects the last element $A[r]$ in the subarray $A[p\ldots r]$ as the pivot.
- Time: $O(n)$.



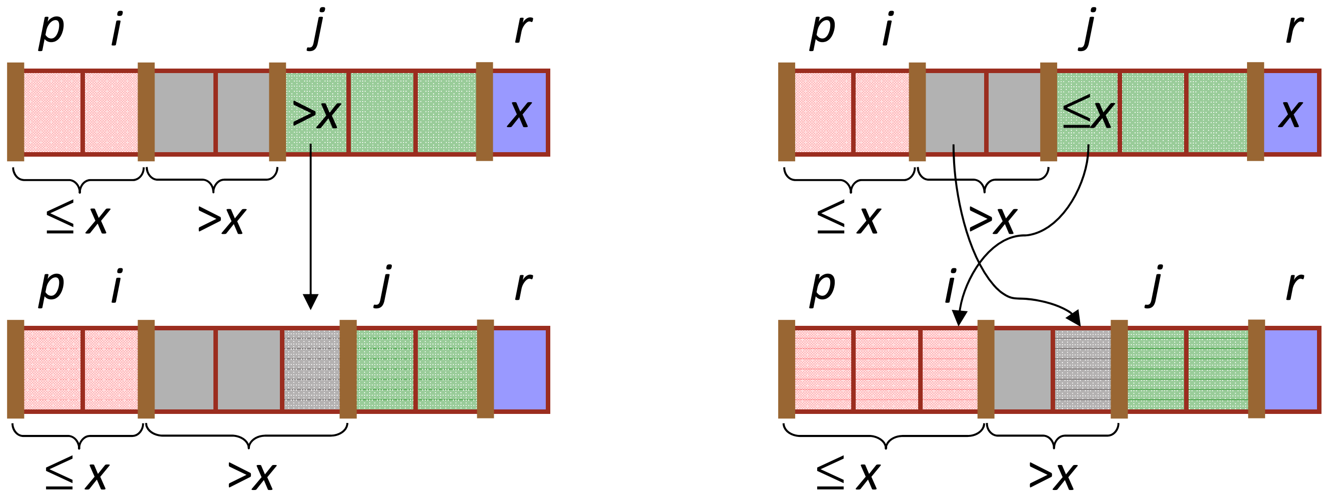
i p,j r
2 8 7 1 3 5 6 4
p,i j r
2 8 7 1 3 5 6 4
p,i j r
2 8 7 1 3 5 6 4
p,i j r
2 8 7 1 3 5 6 4
p i j r
2 1 7 8 3 5 6 4
p i j r
2 1 3 8 7 5 6 4
p i j r
2 1 3 8 7 5 6 4
p i r
2 1 3 8 7 5 6 4
p i r
2 1 3 4 7 5 6 8
: pivot.
: not examined.
: ≤ pivot.
: > pivot.

Loop invariant

- As the procedure executes, the array is partitioned into four regions, some of which may be empty.
- **Loop invariant:**
  - All entries in $A[p \ldots i]$ are $\leq$ pivot.
  - All entries in $A[i+1 \ldots j-1]$ are $>$ pivot.
  - $A[r]$ = pivot.
- It's not needed as part of the loop invariant, but the fourth region is $A[j \ldots r-1]$, whose entries have not yet been examined.

Correctness$_{1/2}$

- **Initialization:** Before the loop starts, all the conditions of the loop invariant are satisfied, because *r* is the pivot and the subarrays $A[p\ldots i\,]$ and $A[i+1\ldots j-1]$ are empty.
- **Maintenance:** While the loop is running:
  - If $A[j]$ > pivot, then increment only *j*.
  - If $A[j] \leq$ pivot, then $A[j]$ and $A[i+1]$ are swapped and then *i* and *j* are incremented.



Correctness$_{2/2}$

- **Termination:** When the loop terminates, $j = r$, so all elements in A are partitioned into one of the three cases: $A[p\ldots i] \leq$ pivot, $A[i+1\ldots r-1] >$ pivot, and $A[r] =$ pivot.
- The last two lines of PARTITION move the pivot element from the end of the array to between the two subarrays.
  - By swapping $A[i+1]$ and $A[r]$
- **Time for partitioning:** $\Theta(n)$ to partition an n-element subarray.

Outline

- Description of Quicksort
- **Performance of Quicksort**
- A Randomized Version of Quicksort
- Analysis of Quicksort

Performance of quicksort

- The running time of quicksort depends on the partitioning of the subarrays:
  - If the subarrays are balanced, then quicksort can run asymptotically as fast as mergesort.
  - If they are unbalanced, then quicksort can run asymptotically as slowly as insertion sort.

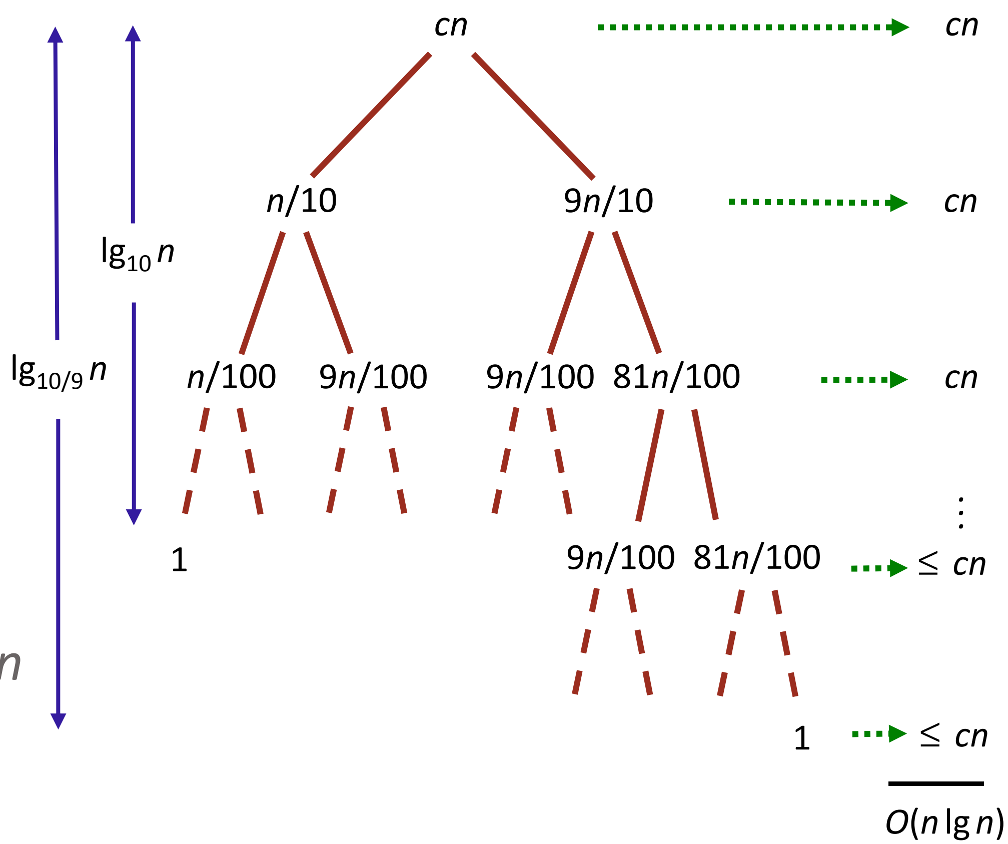
Performance of quicksort

- **Worst-case partitioning:**
  - Have 0 elements in one subarray and $n-1$ elements in the other subarray.
  - The recurrence is $T(n) = T(n-1) + T(0) + \Theta(n)$
    $$= T(n-1) + \Theta(n)$$
    $$= \Theta(n^2).$$
  - Occurs when the input array is sorted.

- **Best-case partitioning:**
  - Occurs when the subarrays are completely balanced every time.
  - Each subarray has $\leq n/2$ elements.
  - The recurrence is $T(n) \leq 2T(n/2) + \Theta(n)$
    $$= \Theta(n \log n).$$

Balanced partitioning$_{1/2}$

- **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, then the running time is
    $T(n) \le T(9n/10) + T(n/10) + cn$
    $= \Theta(n \log n)$.

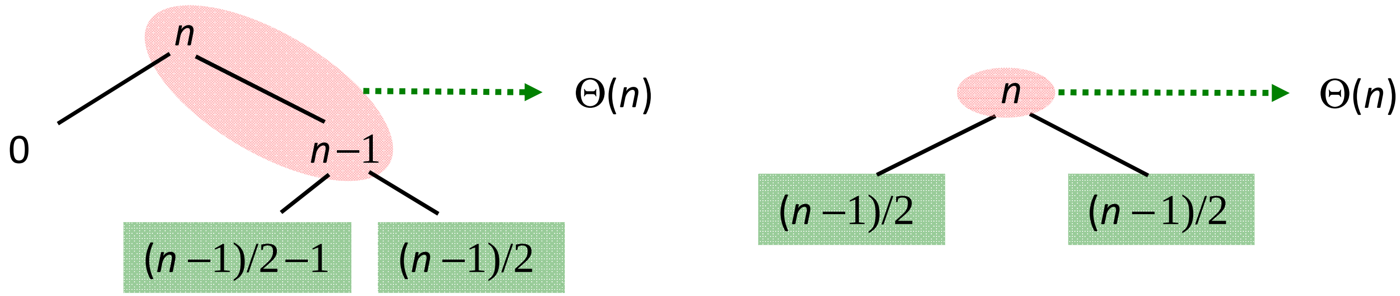


Balanced partitioning$_{2/2}$

- **Intuition:** look at the recursion tree.
  - It's like the one for $T(n) = T(n/3) + T(2n/3) + O(n)$ in Section 4.2.
  - Except that here the constants are different; we get $\log_{10} n$ 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.
  - Any split of **constant proportionality** will yield a recursion tree of depth $\Theta(\lg n)$.

Intuition for the average case

- There will usually be a mix of good and bad splits throughout the recursion tree.
- 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.
  - Only twice as much work was done to get to that point.
  - Both figures result in $O(n \lg n)$ time.

Outline

- Description of Quicksort
- Performance of Quicksort
- **A Randomized Version of Quicksort**
- Analysis of Quicksort

Randomized version of quicksort$_{1/2}$

- In exploring the average-case behavior of quicksort, we have assumed that all input permutations are equally likely.
- This is not always true.
- 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.

RANDOMIZED-PARTITION(A, p, r)

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

Randomized version of quicksort$_{2/2}$

- Randomly selecting the pivot element will, on average, cause the split of the input array to be reasonably well balanced.

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

Outline

- Description of Quicksort
- Performance of Quicksort
- A Randomized Version of Quicksort
- **Analysis of Quicksort**

Analysis of quicksort

- We will analyze
  - the worst-case running time of QUICKSORT and RANDOMIZED-QUICKSORT (the same), and
  - the expected (average-case) running time of RANDOMIZED-QUICKSORT.

- **Worst-case analysis:** $T(n) = O(n^2)$.
  - Recurrence for the worst-case:

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n).$$

  - Because PARTITION produces two subproblems, totaling size $n-1$, q ranges from 0 to $n-1$.

Worst-case analysis

- **Guess:** $T(n) \le cn^2$, for some c.
  - Substituting our guess into the recurrence:

$$T(n) \le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n)$$
$$= c \cdot \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n)$$

  - The maximum value of $(q^2+(n-q-1)^2)$ occurs – when q is either 0 or $n-1$. (Second derivative)
  - This means that $\max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) \le (n-1)^2$

$$= n^2 - 2n + 1.$$

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

$$\le cn^2$$
$$= O(n^2)$$

choose c so that $c(2n-1) \ge \theta(n)$

Average-case Analysis$_{1/5}$

- **Average-case analysis:** $T(n) = O(n \log n)$.
  - The dominant cost of the algorithm is partitioning.
  - PARTITION is called at most n times.
  - 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 is $O(n + X)$**.

Average-case Analysis$_{2/5}$

- We will now 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.
  - 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:
  - Elements are compared only to the pivot element, and
  - The pivot element is never in any later call to PARTITION.
- The expectation of total number of comparisons performed by the algorithm is $\mathrm{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$.

Average-case Analysis$_{3/5}$

- Now all we have to do is find the probability that two elements are compared.
- Consider the input: 2, 8, 7, 1, 3, 5, 6, 4 and the pivot is 4,
  - None of the set {2, 1, 3} will ever be compared to any of the set {5, 6, 7}.
- 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 to all the elements of Z_{ij}, except itself.

Average-case Analysis$_{4/5}$

- Therefore,

Pr{z_i is compared to z_j} = Pr{z_i or z_j is the first pivot chosen from Z_{ij}}

= Pr{z_i is the first pivot chosen from Z_{ij}}

+ Pr{z_j is the first pivot chosen from Z_{ij}}

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1}$$

$$= \frac{2}{j-i+1}.$$

- Substituting into the equation for E[X]:
  - $$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}.$$

Average-case Analysis$_{5/5}$

- Let $k = j - i$, then

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

- So the expected running time of quicksort, using RANDOMIZED-PARTITION, is $O(n \lg n)$.