Quicksort

Review of insertion sort and merge sort

- Insertion sort
  - Worst case number of comparisons = O(?)
- Selection sort
  - Worst case number of comparisons = O(?)
- Merge sort
  - Worst case number of comparisons = O(?)

Sorting Algorithms

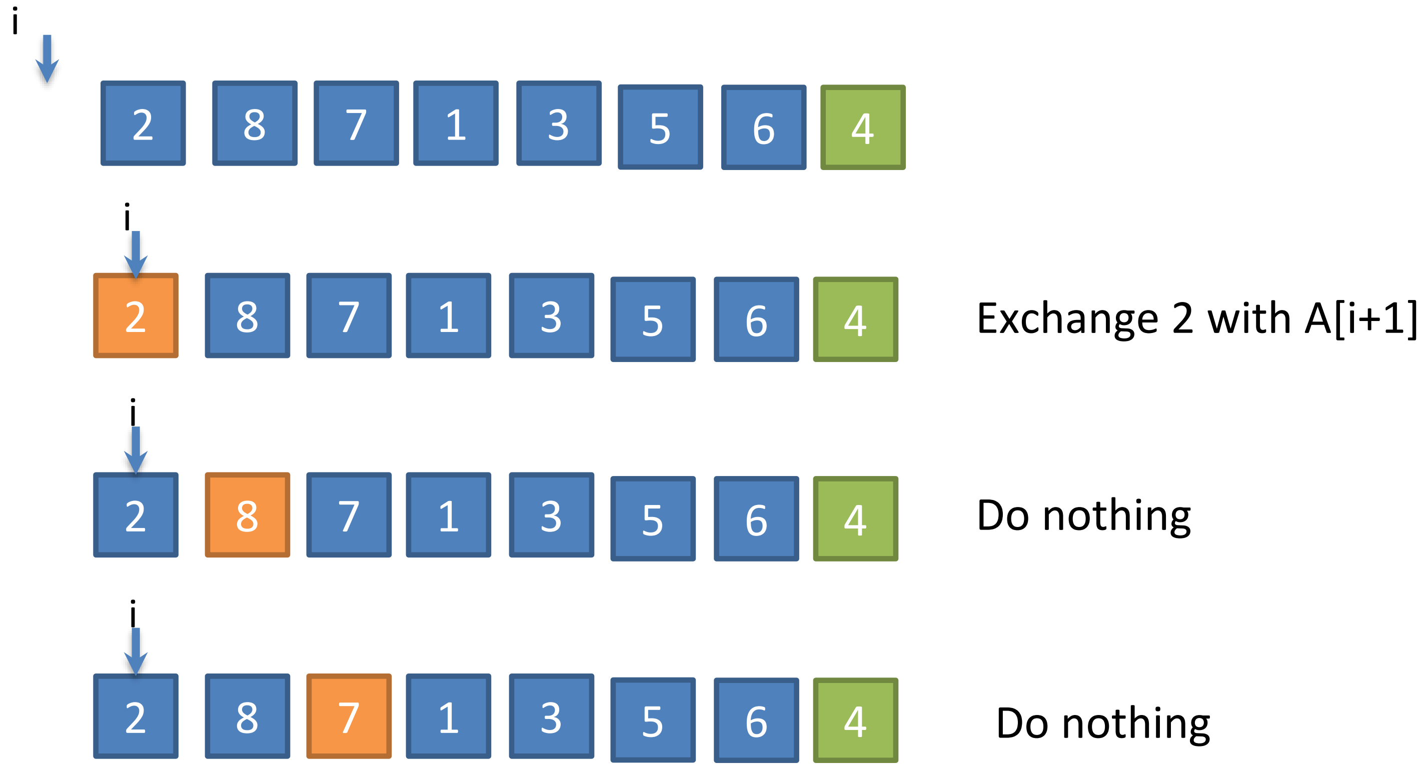
Algorithm	Worst Time	Expected Time	Extra Memory
Insertion sort	$O(n^2)$	$O(n^2)$	O(1)
Selection sort	$O(n^2)$	$O(n^2)$	O(1)
Merge sort	$O(n\ lgn)$	$O(n\ lgn)$	O(n)
Quick sort	$O(n^2)$	$O(n\ lgn)$	O(1)
Heap sort	$O(n\ lgn)$	$O(n\ lgn)$	O(1)

Quicksort Algorithm

- Input: A[1, …, n]
- Output: A[1, .., n], where A[1] ≤ A[2]…≤ A[n]
- **Quicksort**:
  1. if(n ≤ 1) **return**;
  2. Choose the pivot p = A[n]
  3. Put all elements less than p on the left; put all elements lager than p on the right; put p at the middle. (**Partition**)
  4. **Quicksort**(the array on the left of p)
  5. **Quicksort**(the array on the right of p)

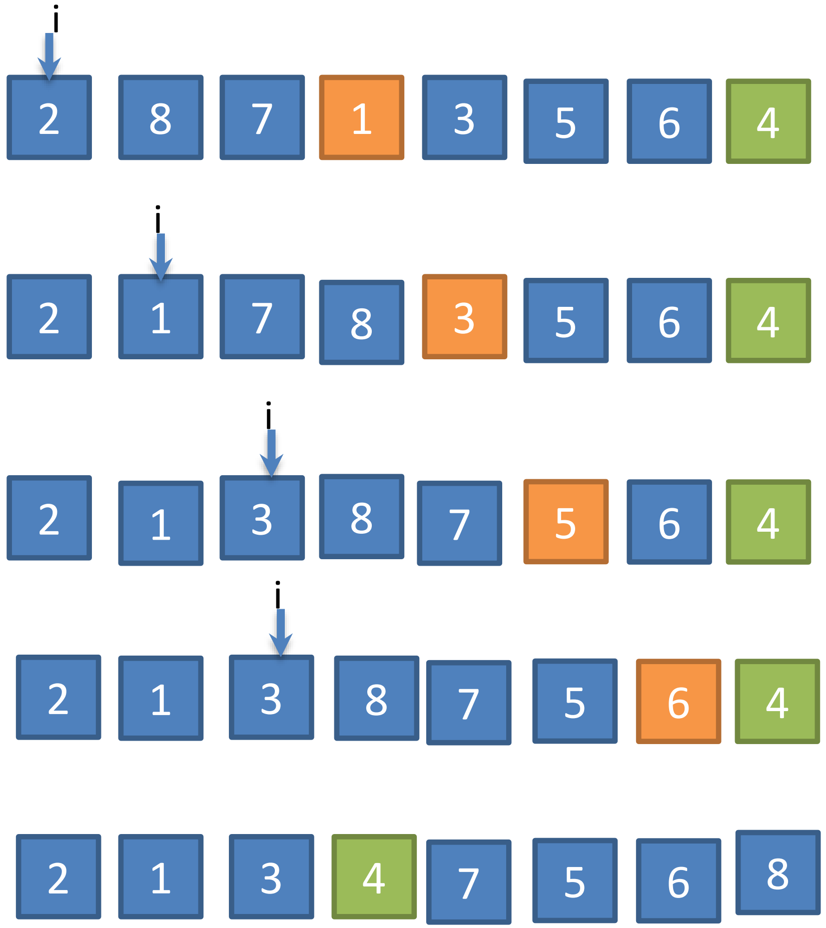
Partition

- Partition example



Partition

- Partition example



Exchange 1 with A[i+1]

Exchange 3 with A[i+1]

Do nothing

Do nothing

Final step: exchange A[n] with A[i+1]

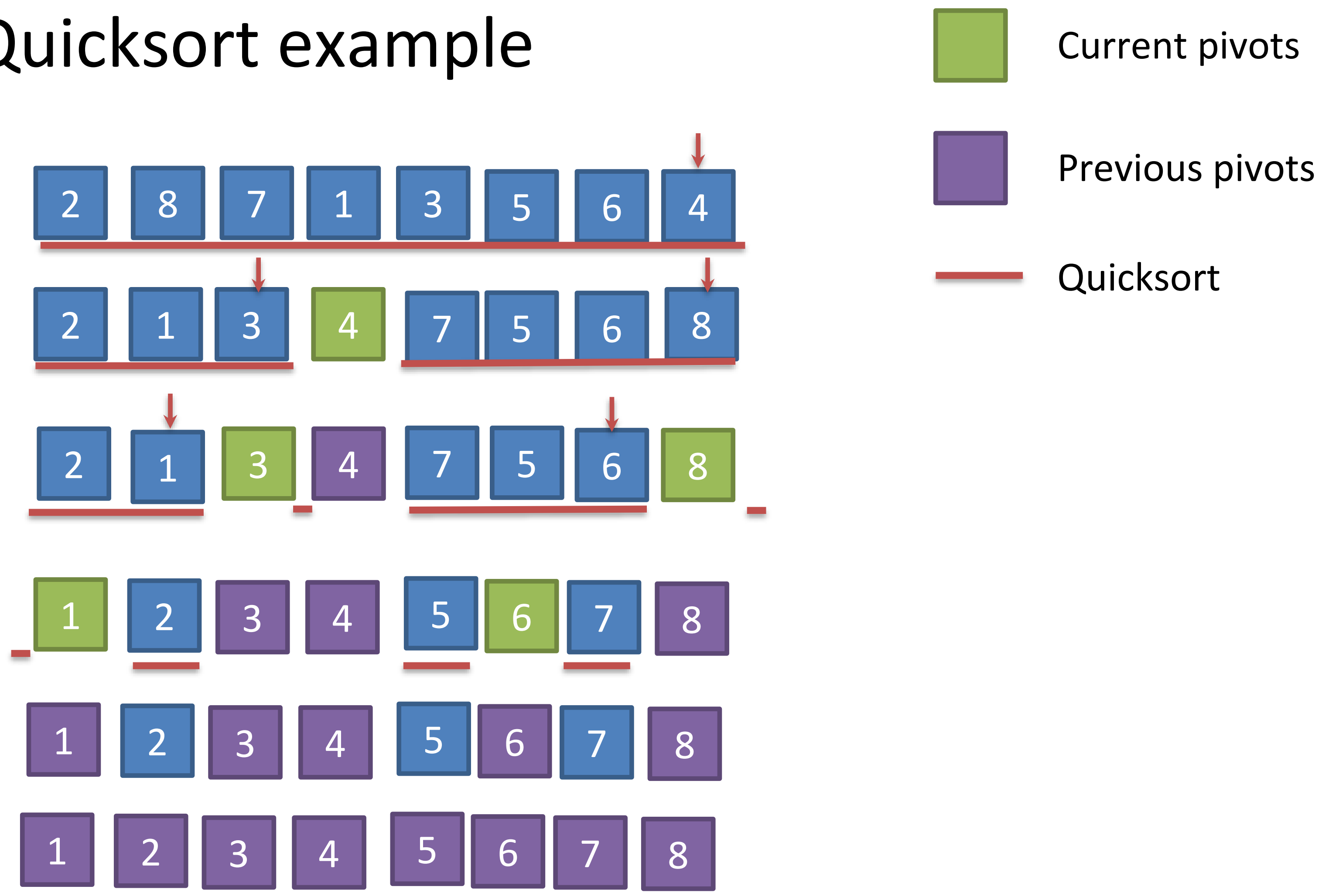
Partition

```
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
```

$T(n)=O(n)$

Quicksort Algorithm

- Quicksort example



Quicksort Algorithm

```
Quicksort(A, p, r){
    if(p<r){                        // if(n<=1) return
        q = partition(A, p, r) //small ones on left
                               //larger ones on right
        Quicksort(A, p, q-1)
        Quicksort(A, q+1, r)
    }
}
```

Sorting the entire array: Quicksort(A, 1, length(A))

Analysis of Quicksort

- Time complexity
  - Worst case
  - Expected
  - Best case

Worst-Case Analysis

- What will be the worst case?
  - Occur when PARTITION produces one subproblem with n-1 elements and one with 0 elements
    - When the input array is already sorted increasingly or decreasingly
  - The pivot is the smallest element, all the time
  - Partition is always unbalanced

$$
\begin{aligned}
T(N) &= T(N-1) + cN \\
T(N-1) &= T(N-2) + c(N-1) \\
T(N-2) &= T(N-3) + c(N-2) \\
&\vdots \\
T(2) &= T(1) + c(2) \\
T(N) &= T(1) + c\sum_{i=2}^{N} i = O(N^2)
\end{aligned}
$$

Best-case Analysis

- What will be the best case?
  - Partition is perfectly balanced.
  - Pivot is always in the middle (median of the array)

$$
\begin{aligned}
T(N) &= 2T(N/2) + cN \\
\frac{T(N)}{N} &= \frac{T(N/2)}{N/2} + c \\
\frac{T(N/2)}{N/2} &= \frac{T(N/4)}{N/4} + c \\
\frac{T(N/4)}{N/4} &= \frac{T(N/8)}{N/8} + c \\
&\vdots \\
\frac{T(2)}{2} &= \frac{T(1)}{1} + c \\
\frac{T(N)}{N} &= \frac{T(1)}{1} + c\log N \\
T(N) &= cN\log N + N = O(N\log N)
\end{aligned}
$$

Worst-case and Best-case Partitioning

- The behavior of quicksort is determined by the relative ordering of the values in the array
- Worst case
  - Occur when PARTITION produces one subproblem with n-1 elements and one with 0 elements
    - When the input array is already sorted increasingly or decreasingly
  - $T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$
    - By substitution: $T(n) = \Theta(n^2)$
- Best case – one of size $\lfloor n/2 \rfloor$, and one of size $\lceil n/2 \rceil -1$
  - $T(n) \le 2T(n/2) + \Theta(n)$ $\quad$ $T(n) = O(n \lg n)$

Balanced Partitioning

- The average-case running time of quicksort is much closer to the best case than to the worst case
  - Suppose $T(n) \le T(9n/10) + T(n/10) + cn$
    - By recursion tree: $O(n \lg n)$
      - Every level of the tree has cost cn
      - Boundary condition reaches at depth $\log_{10} n = \Theta(\lg n)$
      - Recursion terminates at depth
  - Even a 99-to-1 split yields an $O(n \lg n)$ running time $\log_{10/9} n = \theta(\lg n)$
    - Any split of constant proportionality yields a recursion tree of depth $\Theta(\lg n)$, where the cost at each level is $O(n)$

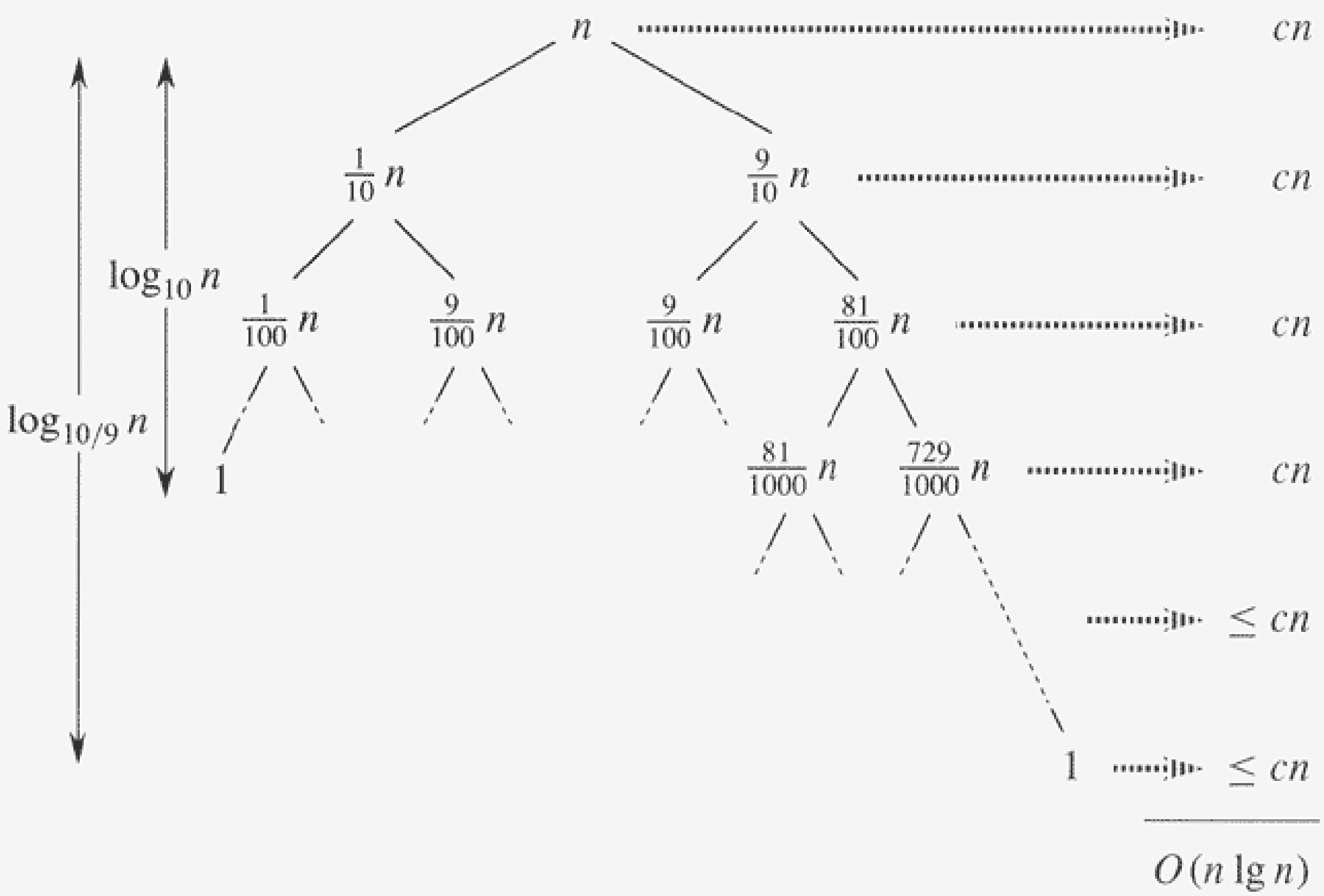


Figure 7.4 A recursion tree for QUICKSORT in which PARTITION always produces a 9-to-1 split, yielding a running time of $O(n \lg n)$. Nodes show subproblem sizes, with per-level costs on the right. The per-level costs include the constant c implicit in the $\Theta(n)$ term.

Improved Pivot Selection

- Pick median value of three elements from data array:
  - A[0], A[n/2], and A[n-1]
  - Use this median value as pivot.

Strict proof of the worst-case time complexity

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

Proof that $T(n) = O(n^2)$

1. When n=1, $T(1) = T(0) + T(0) + \theta(1)$= $O(n^2)$
2. Hypothesis: when $k \le n, T(k) = O(k^2)$;

induction statement: when $k = n+1, T(k) = O(k^2)$;

Strict proof of the worst-case time complexity

- $T(k) = T(n+1)$
  $= \max_{0 \le q \le n} \big(T(q) + T(n-q)\big) + \theta(n+1)$

Since $q \le n, n-1 \le n,$

$T(k) \le \max_{0 \le q \le n} (cq^2 + c(n-q)^2) + \theta(n+1)$

$= c \max_{0 \le q \le n} (q^2 + (n-q)^2) + \theta(n+1)$

$\le c(n^2) + \theta(n+1) = c(k-1)^2 + \theta(k)$

$= O(k^2);$

Strict proof of the expected time complexity

- Given A[1, …, n], after sorting them to $A[1] \leq A[2] \ldots \leq A[n]$, the chance for 2 elements, A[i] and A[j], to be compared is $\frac{2}{j-i+1}$.
- The total comparison is calculated as:
- $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = O(nlgn)$

- Slides and figures have been collected from various publicly available Internet sources for preparing the lecture slides of IT2001 course. I acknowledge and thank all the original authors for their contribution to prepare the content.