Design and Analysis of Algorithms

CSE 5311

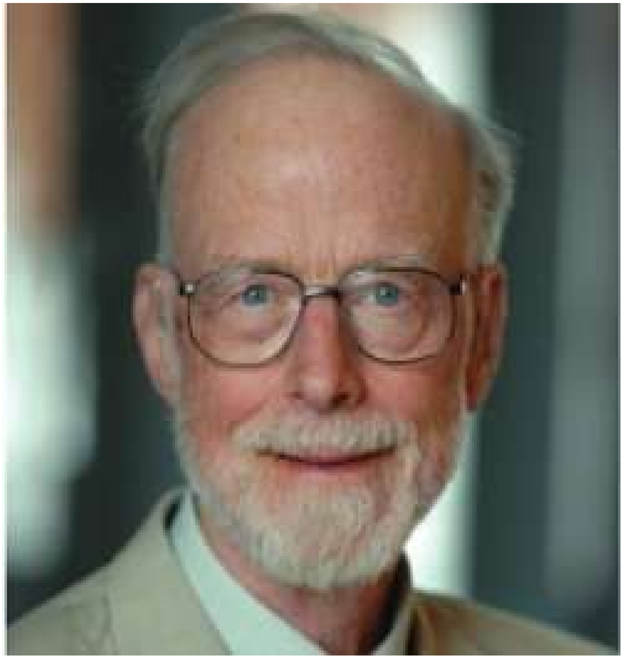
Lecture 7 Quick Sort

Junzhou Huang, Ph.D.

Department of Computer Science and Engineering

Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts "in place" (like insertion sort, but not like merge sort).
- Very practical (with tuning).



Divide and Conquer

Quicksort an n-element array:

1. ***Divide:*** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray $\leq x \leq$ elements in upper subarray.

$\leq x$	x	$\geq x$

2. ***Conquer:*** Recursively sort the two subarrays.
3. ***Combine:*** Trivial.

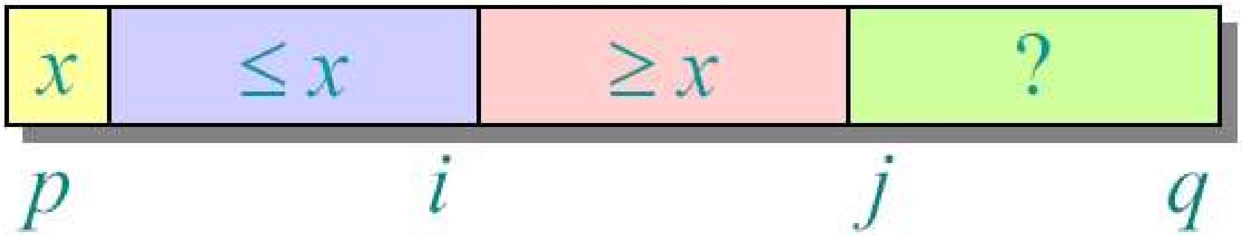
Key: *Linear-time partitioning subroutine.*

Partitioning Subroutine

PARTITION(A, p, q) ▷ $A[p \,.\,.\, q]$
    $x \leftarrow A[p]$ ▷ pivot $= A[p]$
    $i \leftarrow p$
    for $j \leftarrow p + 1$ **to** q
        do if $A[j] \leq x$
            then $i \leftarrow i + 1$
                exchange $A[i] \leftrightarrow A[j]$
    exchange $A[p] \leftrightarrow A[i]$
    return i

Running time $= O(n)$ for n elements.

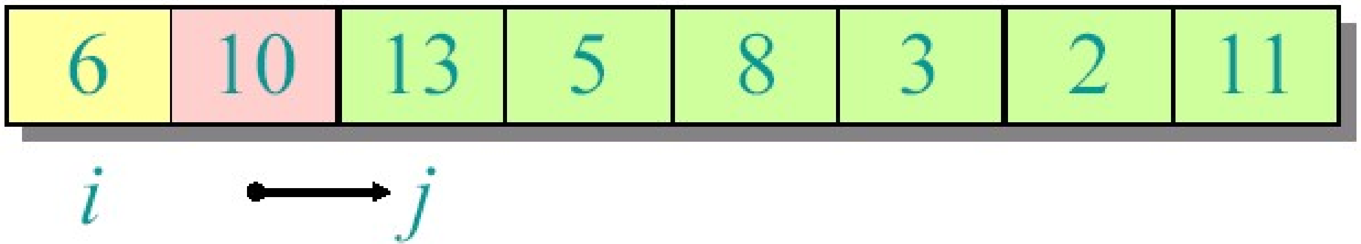
Invariant:



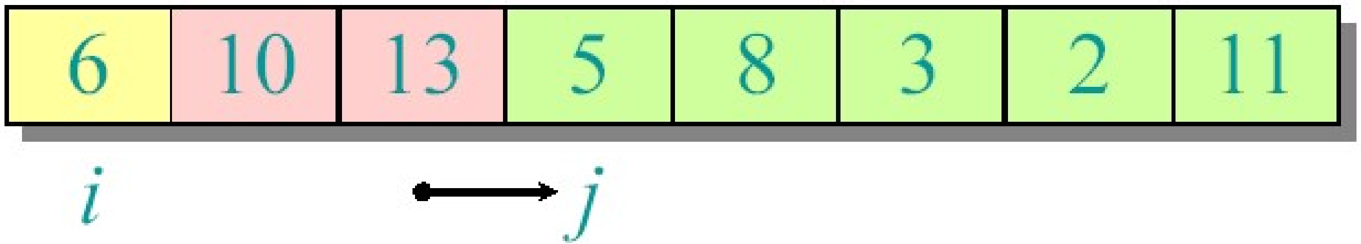
Example of Partitioning

6	10	13	5	8	3	2	11
i	j						

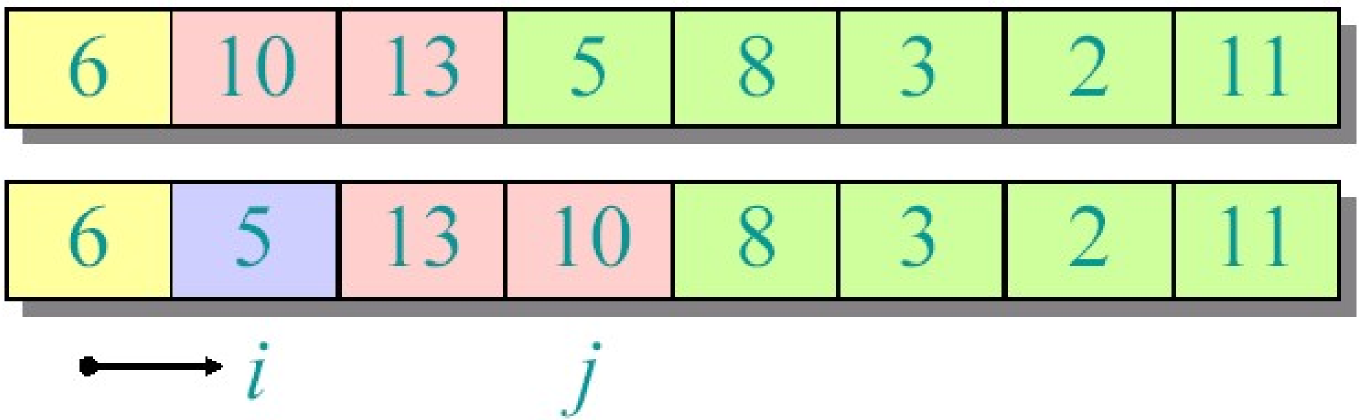
Example of Partitioning



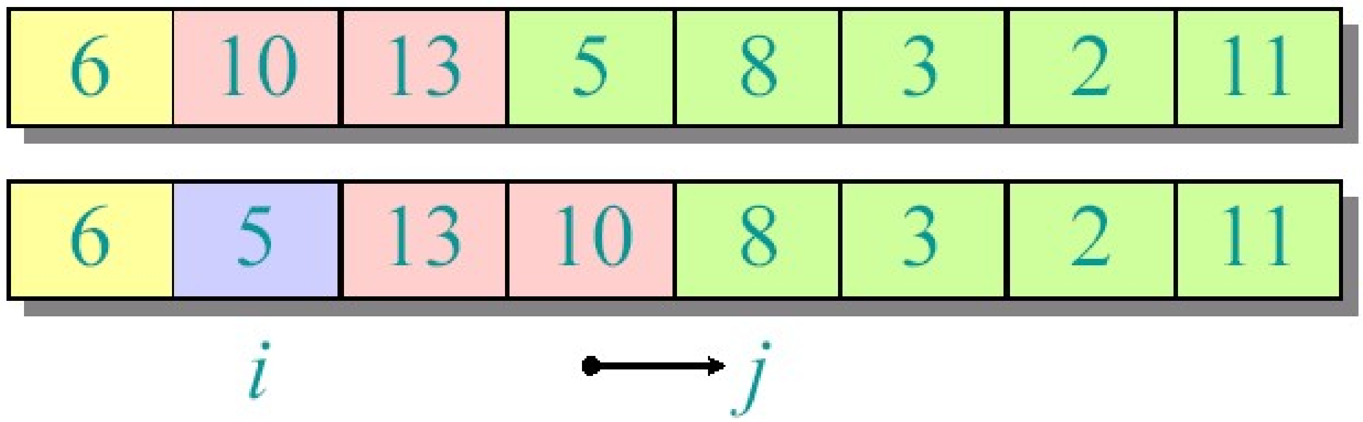
Example of Partitioning



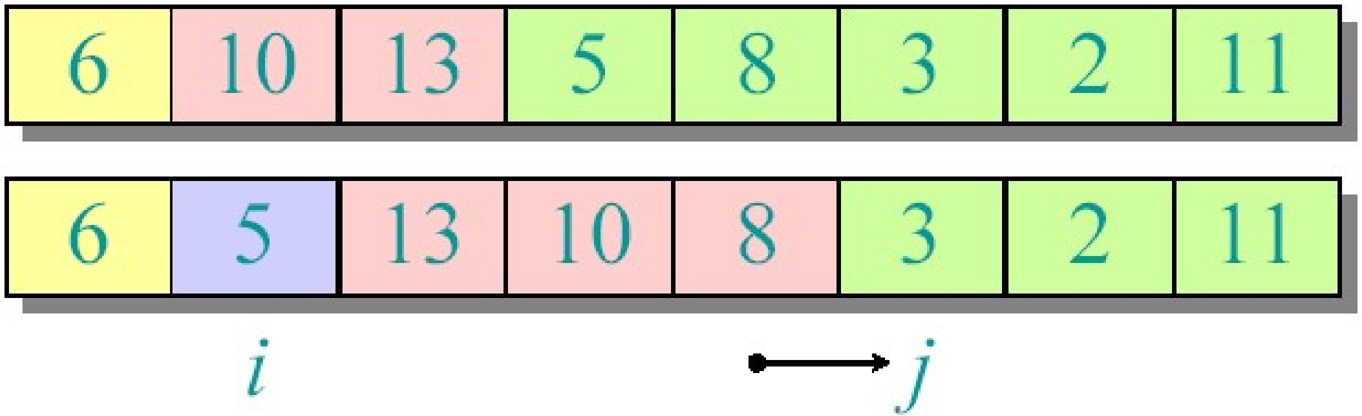
Example of Partitioning



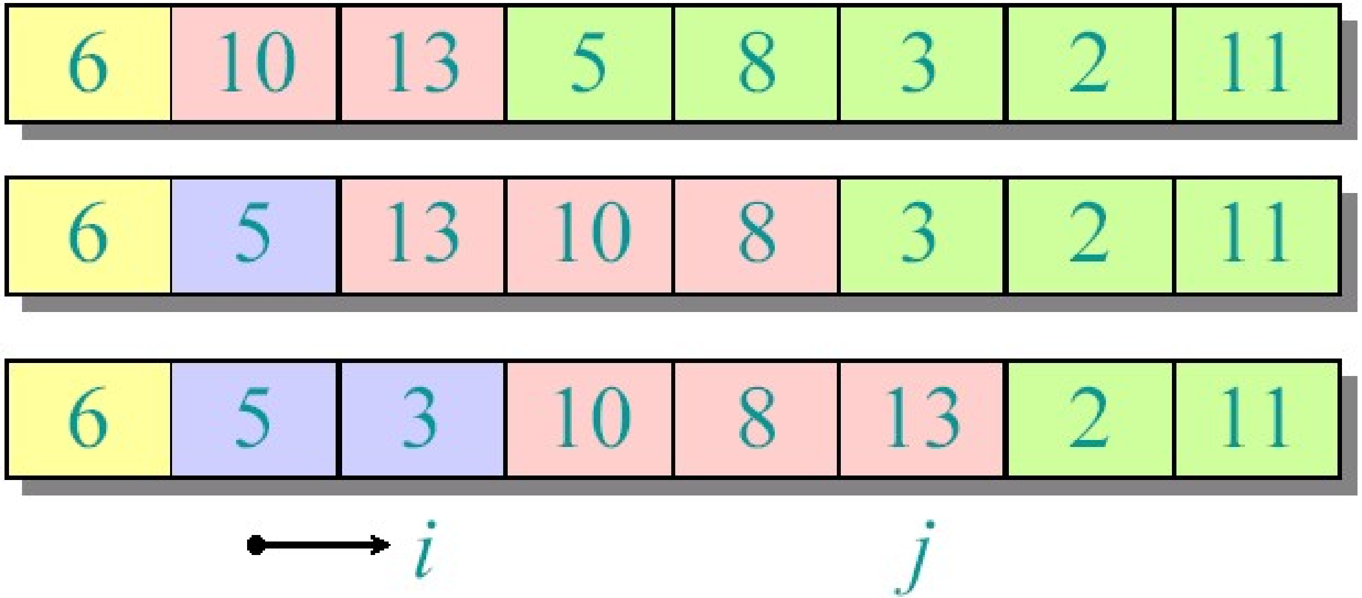
Example of Partitioning



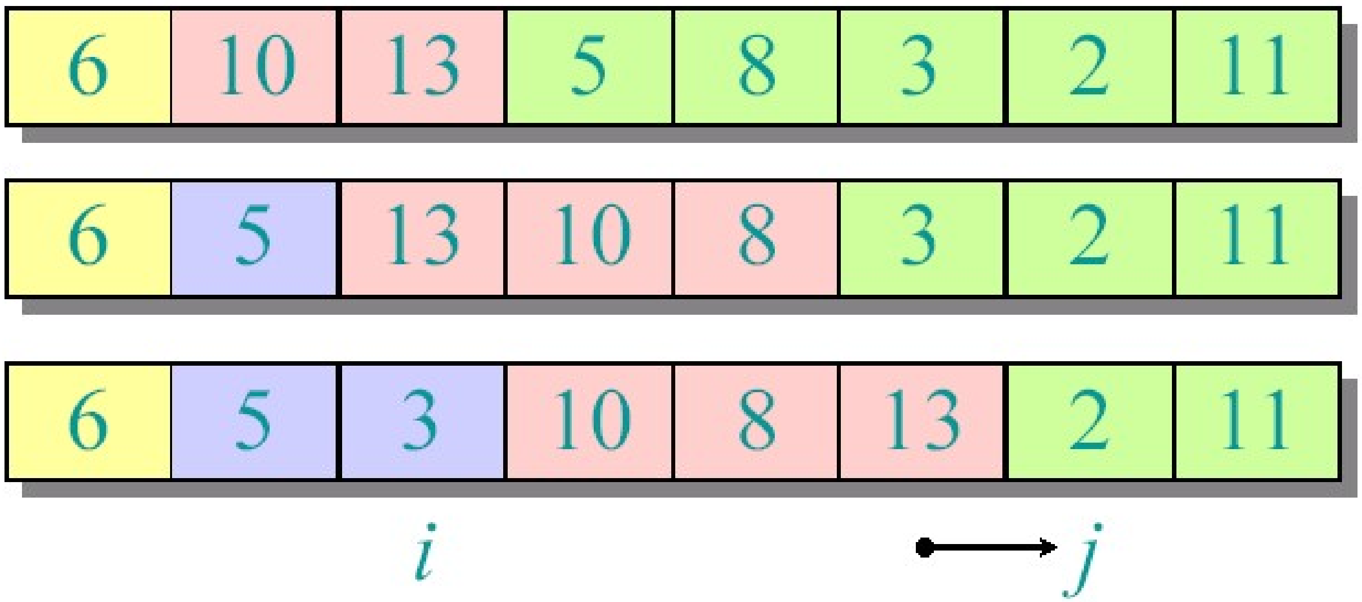
Example of Partitioning



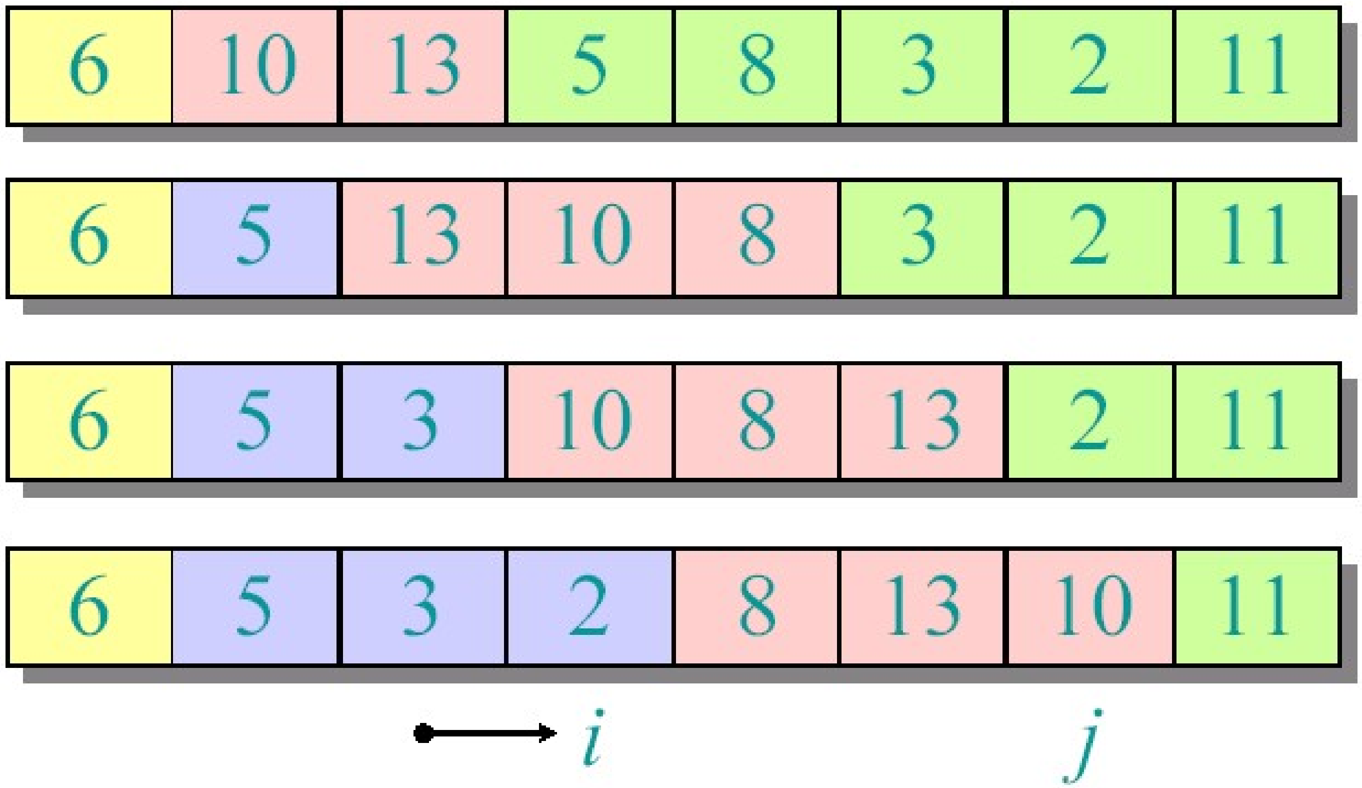
Example of Partitioning



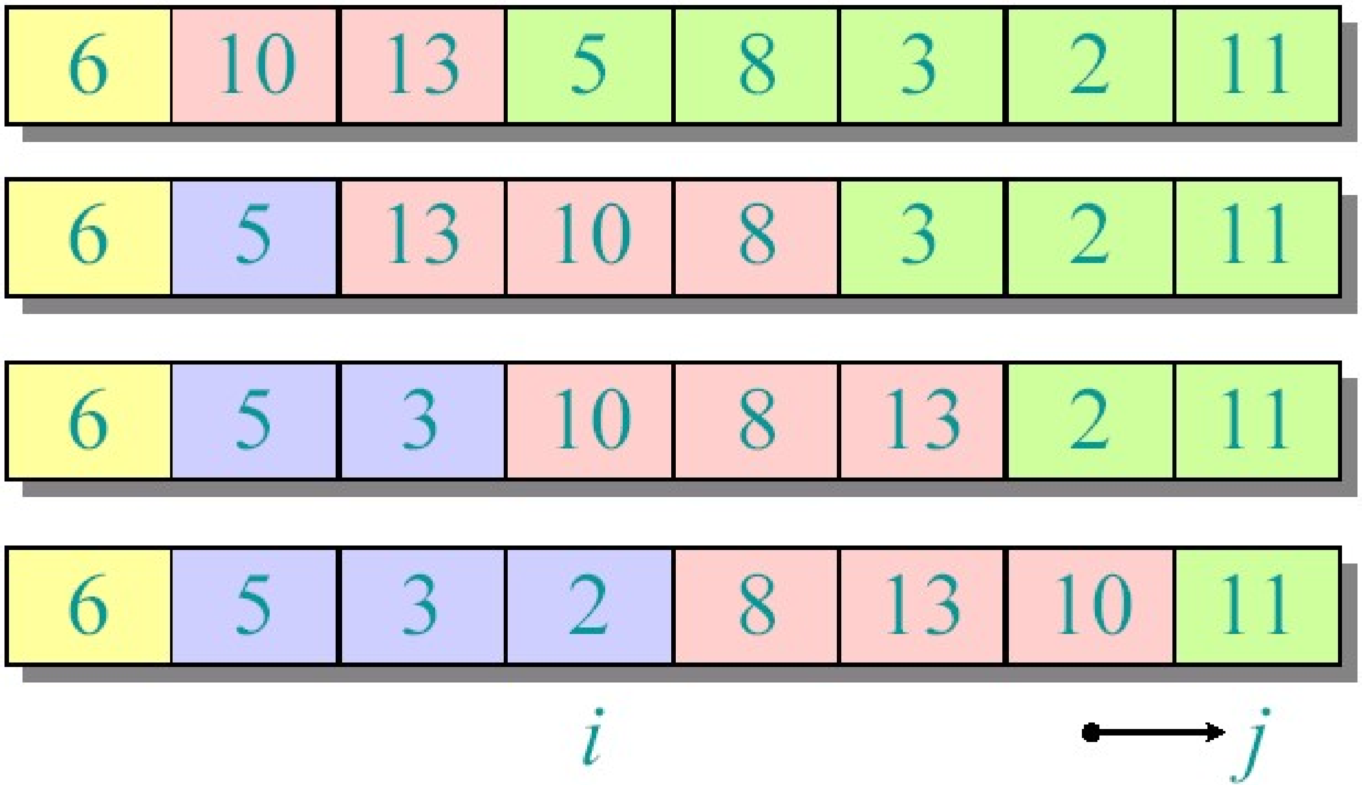
Example of Partitioning



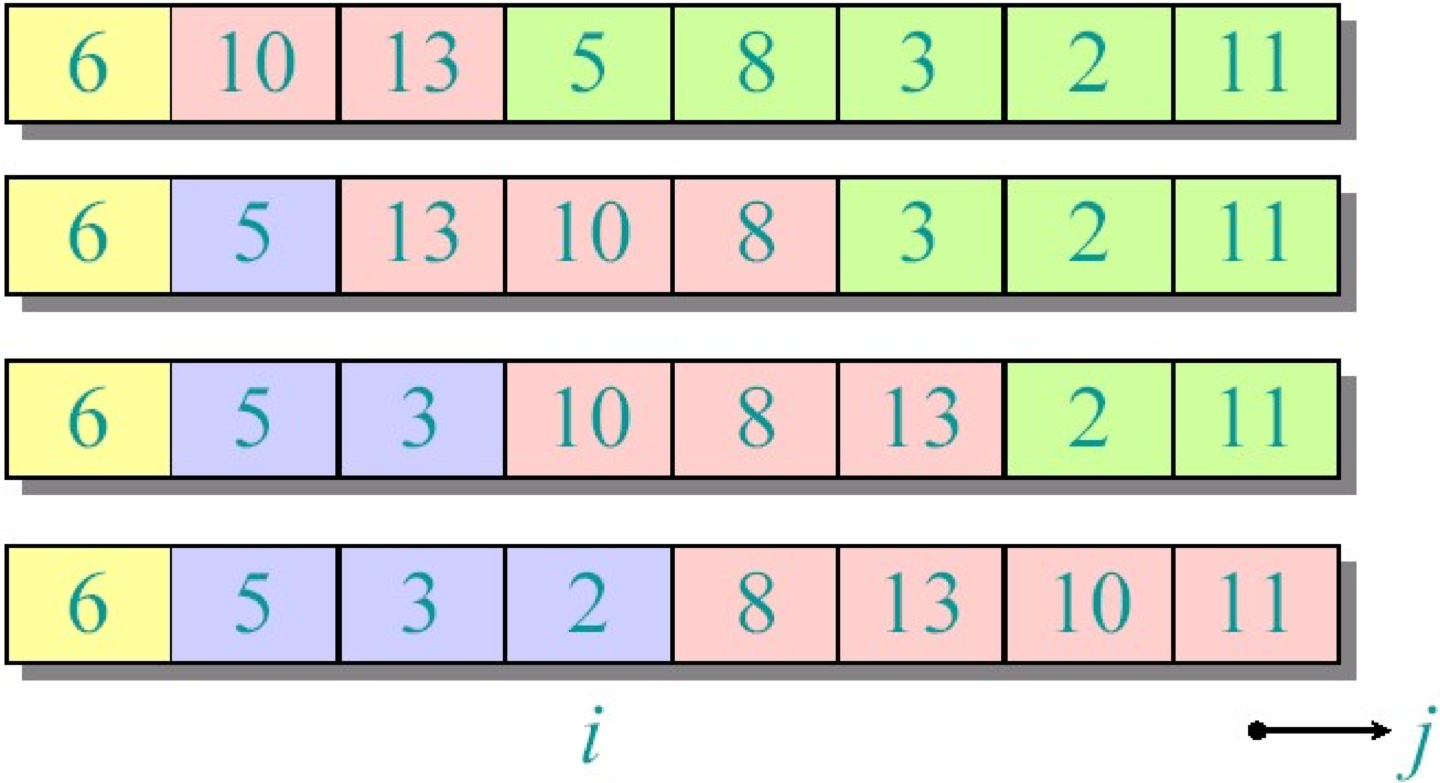
Example of Partitioning



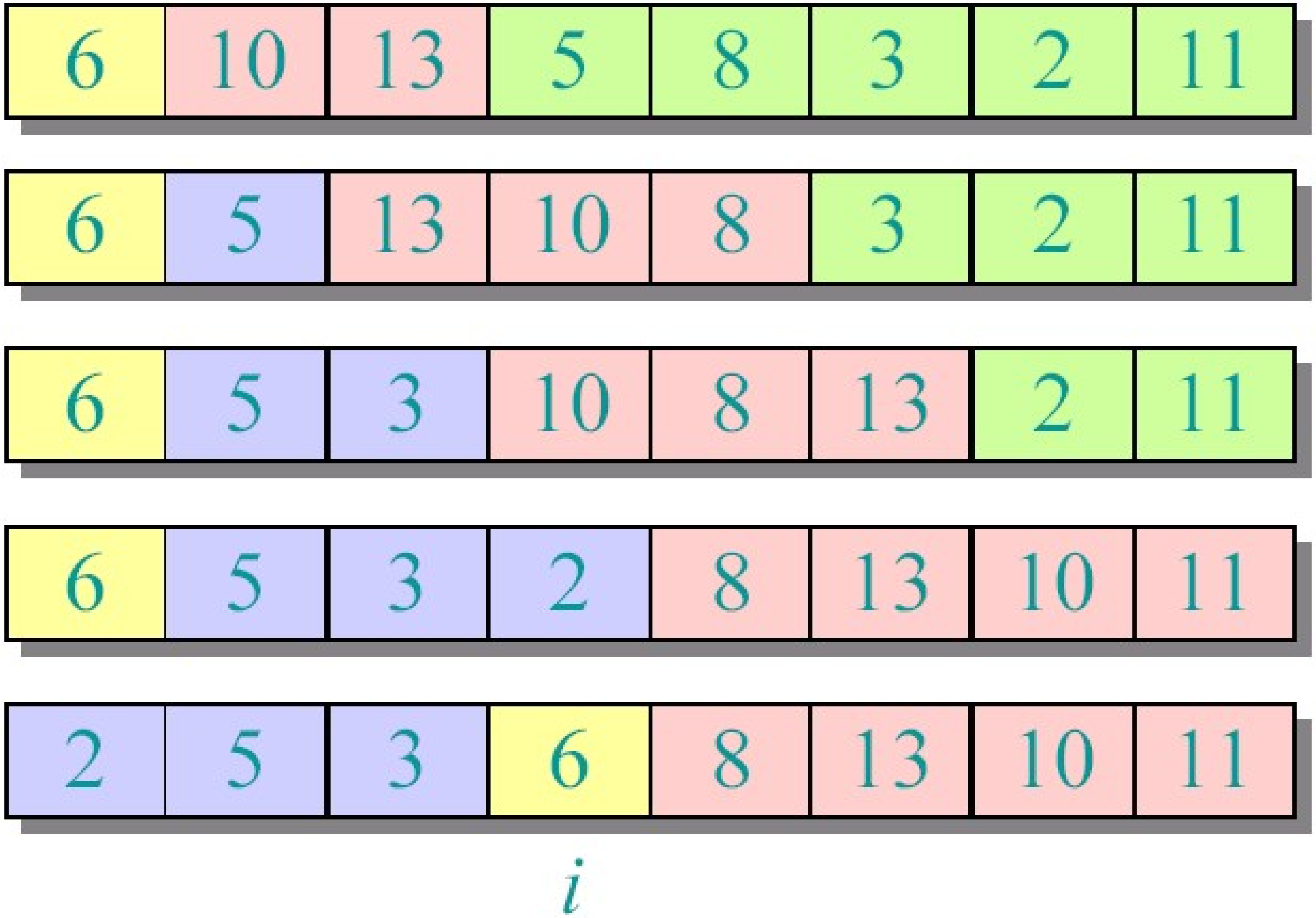
Example of Partitioning



Example of Partitioning



Example of Partitioning



Running Time for PARTITION

The running time of PARTITION on the subarray $A[p...r]$ is $\Theta(n)$ where $n=r-p+1$

Pseudo code for Quicksort

QUICKSORT(A, p, r)
  if $p < r$
    then $q \leftarrow$ PARTITION(A, p, r)
      QUICKSORT(A, p, $q-1$)
      QUICKSORT(A, $q+1$, r)

Initial call: QUICKSORT(A, 1, n)

Analysis of Quicksort

- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let $T(n)$ = worst-case running time on an array of n elements.

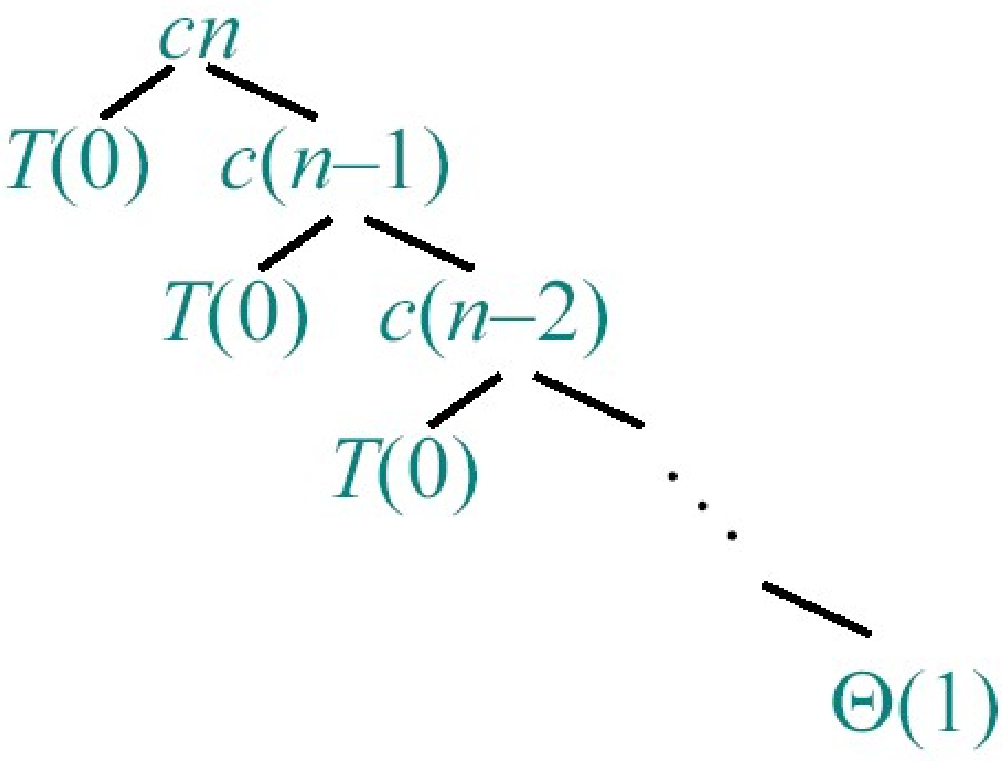
Worst-case of Quicksort

- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

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

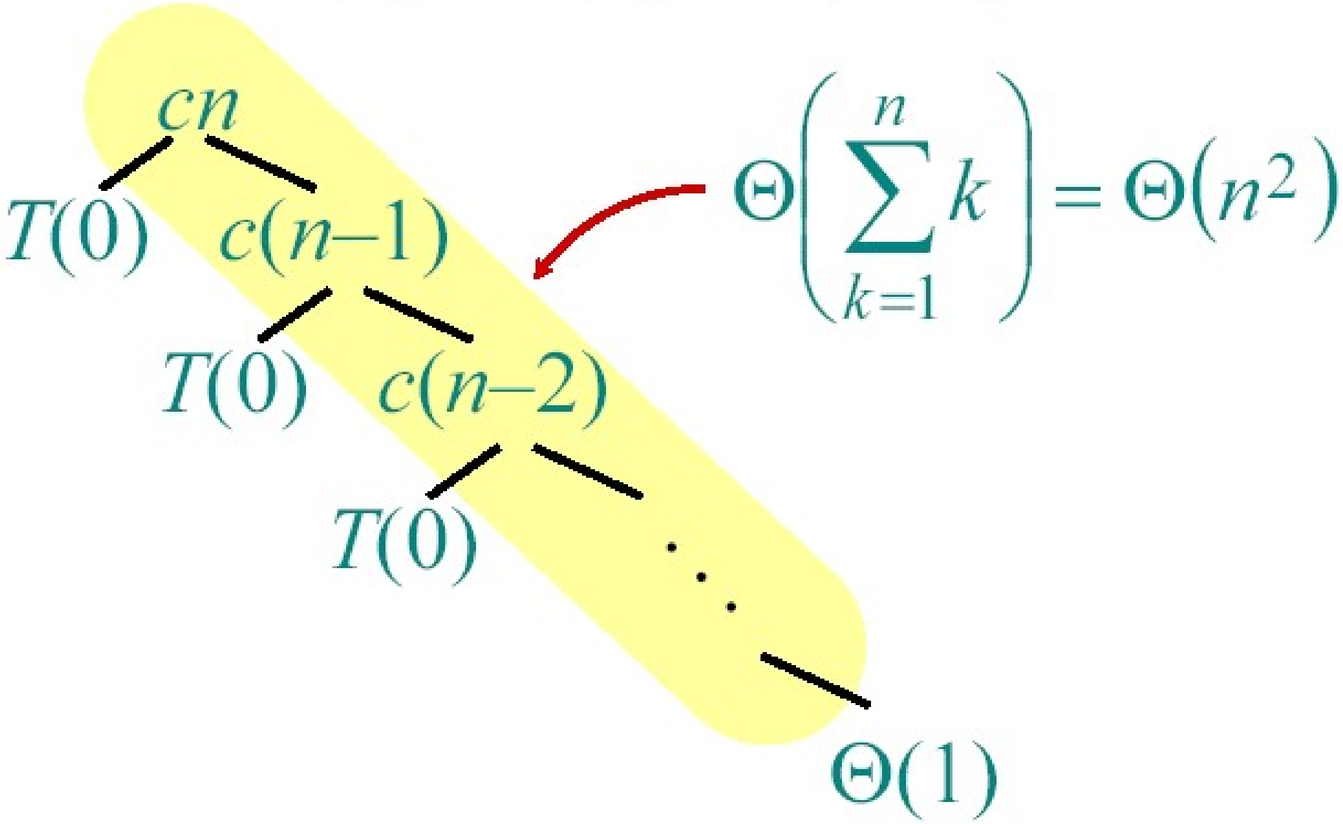
Worst-case Decision Tree

$$T(n) = T(0) + T(n-1) + cn$$



Worst-case Decision Tree

$$T(n) = T(0) + T(n-1) + cn$$



Worst-case Analysis

For the worst case, we can write the recurrence equation as

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

We guess that $T(n) \le cn^2$

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

Worst-case Analysis

This expression achieves a maximum at either end points:

***q*=0 or *q*=*n*-1.**

Using the maximum of *T*(*n*) we have

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

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

Thus

$$T(n) = O(n^2).$$

Worst-case Analysis

We can also show that the recurrence equation as

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

Has a solution of $T(n) = \Omega(n^2)$

We guess that $T(n) \ge cn^2$

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

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

Worst-case Analysis

Using the maximum of $T(n)$ we have

$$
\begin{aligned}
T(n) &\geq cn^2 - c(2n-1) + \Theta(n) \\
&= cn^2 - c(2n-1) + c_1 n \\
&= cn^2 + (c_1 - 2c)n + c \\
&\geq cn^2 + (c_1 - 2c)n
\end{aligned}
$$

We can pick the constant c_1 large enough so that $c_1 - 2c \geq 0$
and $T(n) \geq cn^2 = \Omega(n^2)$

Thus the worst case running time of quicksort is $\Theta(n^2)$

Best-case Analysis

(For intuition only!)

If we're lucky, PARTITION splits the array evenly:

$$T(n) = 2T(n/2) + \Theta(n)$$
$$= \Theta(n \lg n) \qquad \text{(same as merge sort)}$$

What if the split is always $\frac{1}{10} : \frac{9}{10}$?

$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?

Analysis of Almost Best-case

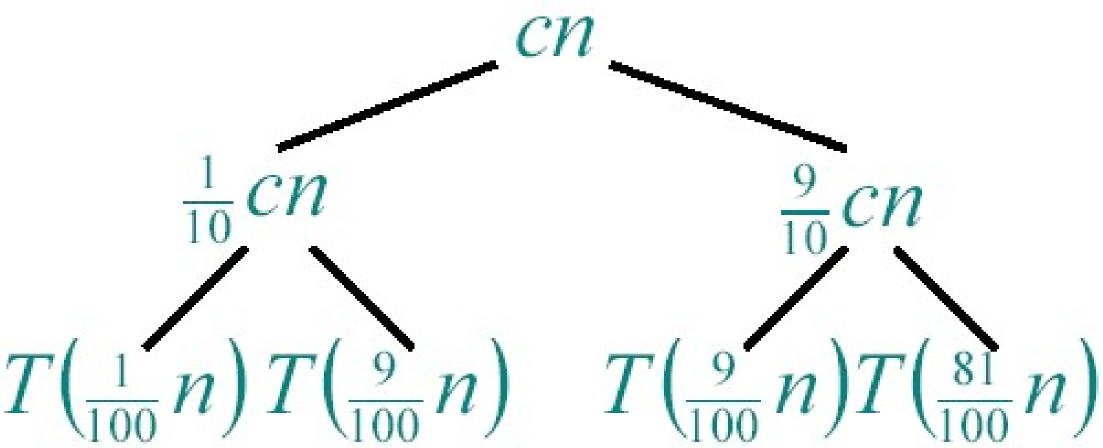
$$T(n)$$

Analysis of Almost Best-case

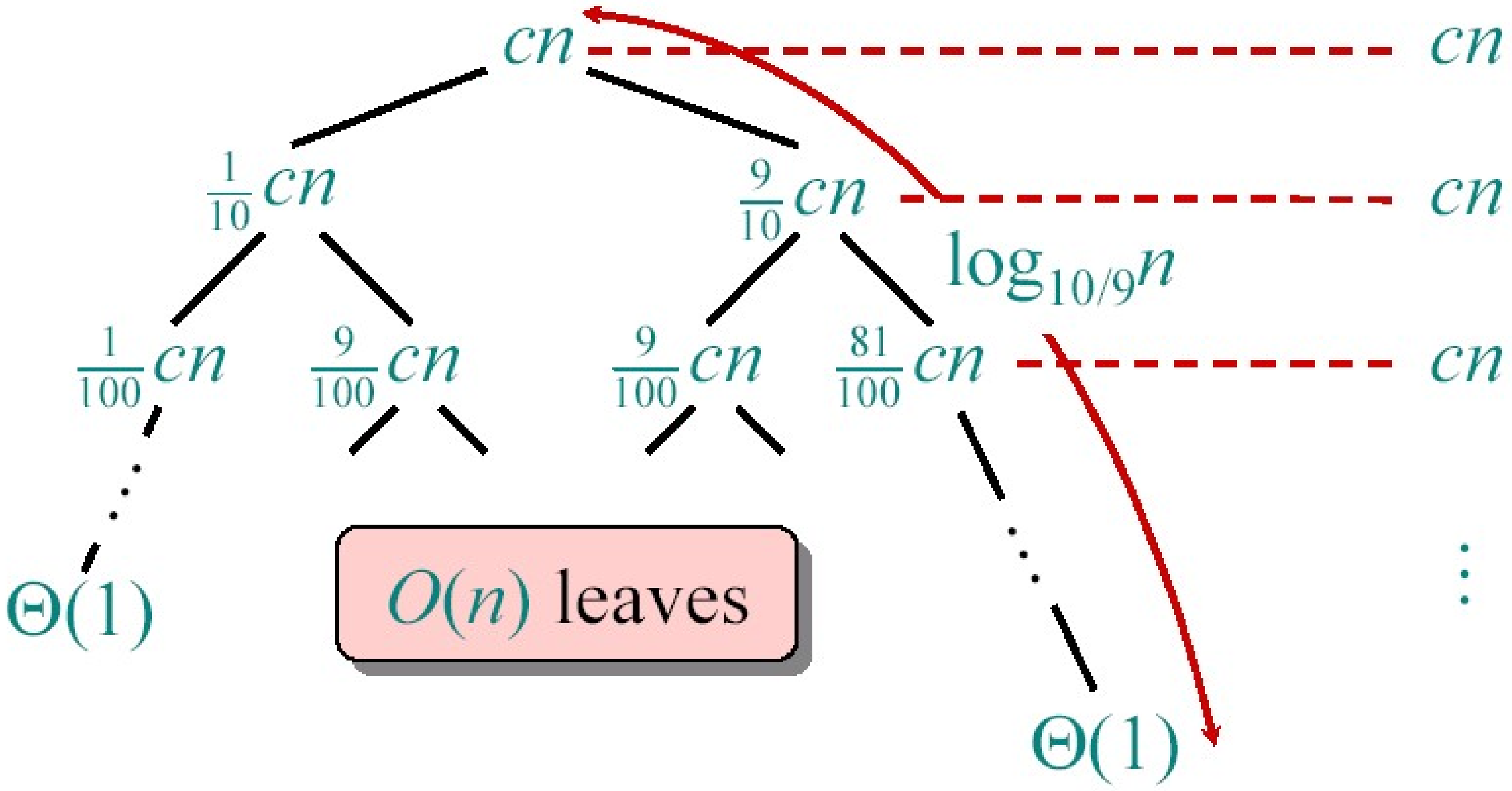
$$cn$$

$$T\left(\frac{1}{10}n\right) \qquad\qquad T\left(\frac{9}{10}n\right)$$

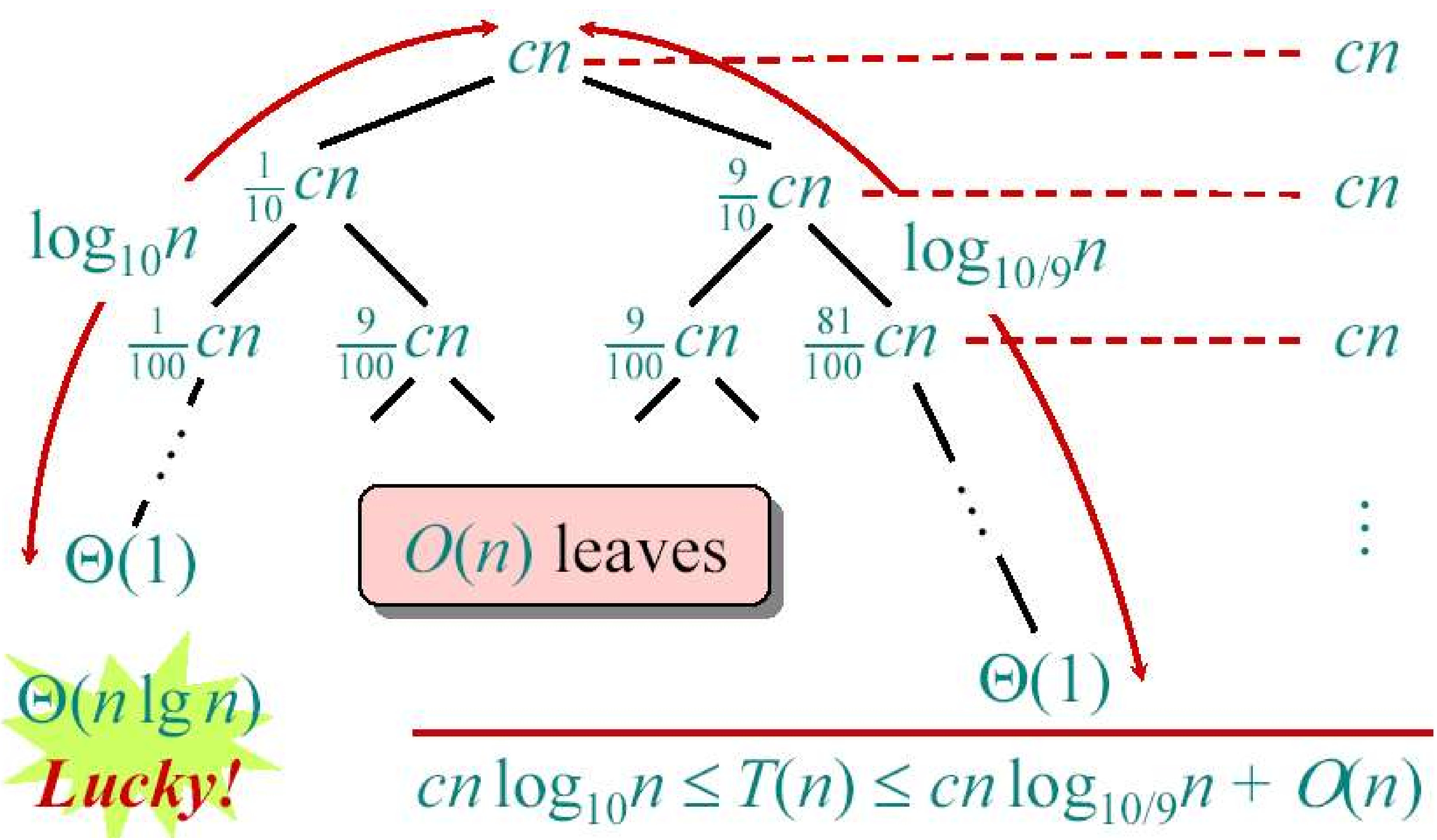
Analysis of Almost Best-case



Analysis of Almost Best-case



Analysis of Almost Best-case



$$cn \log_{10} n \le T(n) \le cn \log_{10/9} n + O(n)$$

Best-case Analysis

For the best case, we can write the recurrence equation as

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

We guess that

$$T(n) \ge cn \log n = \Omega(n \log n)$$

Best-case Analysis

$$T(n) \geq \min_{0 \leq q \leq n-1} \{cq \log q + c(n-q-1)\log(n-q-1)\} + \Theta(n)$$

$$= c \cdot \min_{0 \leq q \leq n-1} \{q \log q + (n-q-1)\log(n-q-1)\} + \Theta(n)$$

Let $Q = q \log q + (n-q-1)\log(n-q-1)$

$$dQ / dq = c\{\frac{q}{q} + \log q - \log(n-q-1) - \frac{(n-q-1)}{(n-q-1)}\} = 0$$

$$\Rightarrow q = \frac{n-1}{2}$$

Best-case Analysis

This expression achieves a minimum at

$$q = \frac{n-1}{2}$$

Using the minimum of *T*(*n*) we have

$$
\begin{aligned}
T(n) &\geq c\{\frac{n-1}{2}\log\frac{n-1}{2} + \frac{n-1}{2}\log\frac{n-1}{2}\} + \Theta(n) \\
&= cn\log(n-1) + c(n-1) + \Theta(n) \\
&= cn\log(n-1) + \Theta(n) \\
&\geq cn\log n \\
&= \Omega(n\log n)
\end{aligned}
$$

More Intuition

Suppose we alternate lucky, unlucky, lucky, unlucky, lucky, ….

$$L(n) = 2U(n/2) + \Theta(n) \quad \textit{lucky}$$

$$U(n) = L(n-1) + \Theta(n) \quad \textit{unlucky}$$

Solving:

$$\begin{aligned} L(n) &= 2(L(n/2-1) + \Theta(n/2)) + \Theta(n) \\ &= 2L(n/2-1) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

Lucky!

How can we make sure we are usually lucky?

Randomized Quicksort

IDEA: Partition around a ***random*** element.

- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.

Randomized Quicksort

Standard Problematic Algorithm :

QUICKSORT(A, p, r)
  if $p < r$
    then $q \leftarrow$ PARTITION(A, p, r)
      QUICKSORT($A, p, q-1$)
      QUICKSORT($A, q+1, r$)

Initial call: QUICKSORT(A, 1, n)

Randomized Quicksort

```
RANDOMIZED-PARTITION (A, p, r)
    i←RANDOM(p, r)
    exchange A[r]↔A[i]
    return PARTITION(A, p, r)
```

```
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)
```

Randomized Quicksort

Let $T(n)$ = the random variable for the running time of randomized quicksort on an input of size n, assuming random numbers are independent.

For $k = 0, 1, \ldots, n-1$, define the ***indicator random variable***

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

$E[X_k] = \Pr\{X_k = 1\} = 1/n$, since all splits are equally likely, assuming elements are distinct.

Randomized Quicksort Analysis

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \quad\vdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n) \right).$$

Calculating Expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k\left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Take expectation in both side

$$= \sum_{k=0}^{n-1} E\left[X_k\left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)]$$

Independence of X_k from other random choice

$$= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)$$

Linearity of expectation; $E[X_k] = 1/n$.

Calculating Expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k\left(T(k)+T(n-k-1)+\Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k\left(T(k)+T(n-k-1)+\Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E[X_k]\cdot E[T(k)+T(n-k-1)+\Theta(n)]$$

$$= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1}\Theta(n)$$

$$= \frac{2}{n}\sum_{k=1}^{n-1} E[T(k)] + \Theta(n)$$

Summations have identical terms.

Calculating Expectation

$$E[T(n)] = \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

(The $k = 0, 1$ terms can be absorbed in the $\Theta(n)$.)

Prove: $E[T(n)] \le a n \lg n$ for constant $a > 0$.

- Choose a large enough so that $a n \lg n$ dominates $E[T(n)]$ for sufficiently small $n \ge 2$.

Use fact: $\sum_{k=2}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$ (exercise).

Calculating Expectation

$$\sum_{k=1}^{n-1} k \lg k = \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg k$$

The $\lg k$ in the first summation is bounded above by $\lg (n/2) = \lg n - 1$

The $\lg k$ in the second summation is bounded above by $\lg n$

$$\le (\lg n - 1) \sum_{k=1}^{\lceil n/2 \rceil - 1} k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

$$= \lg n \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k$$

$$\le \frac{1}{2} n(n-1) \lg n - \frac{1}{2} (\frac{n}{2} - 1) \frac{n}{2}$$

$$\le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$$

For n ≥ 2 this is the bound.

Substitution Method

$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

$$= \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

$$= an\lg n - \left(\frac{an}{4} - \Theta(n)\right)$$

Desired - Residual

$$\le an\lg n,$$

if a is chosen large enough so that $an/4$ dominates the $\Theta(n)$.

Quicksort: Summary

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from ***code tuning***.
- Quicksort behaves well even with caching and virtual memory.