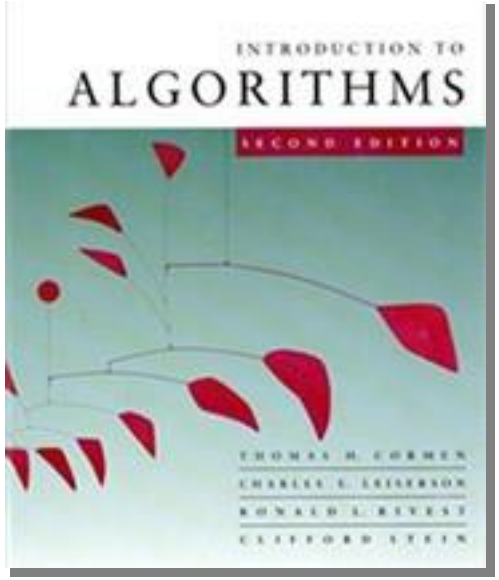
APPLIED ROOTS



Quicksort

- Divide and conquer
- Partitioning
- Worst-case analysis
- Intuition
- Randomized quicksort

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

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).

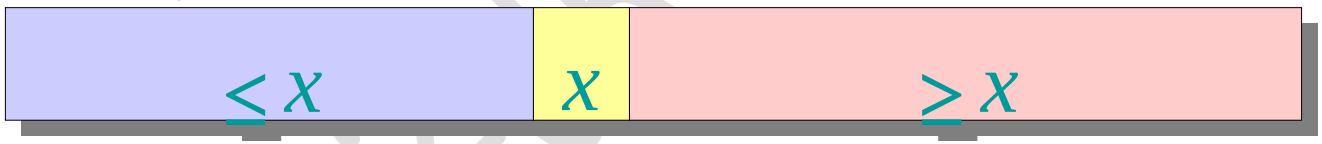
Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

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.



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

Key: *Linear-time partitioning subroutine.*

Source : MIT Introduction to Algorithms 6.046J/18.401J

Partitioning subroutine

PARTITION(A, p, q) $\rhd$ $A[p \mathinner{..} q]$ $x \leftarrow A[p]$ $\rhd$ pivot =

$A[p]$ Running timeRunning time $i \leftarrow p$ ==

$OO((nn))$ forfor nn **for** $j \leftarrow p + 1$ **to** q

elements.elements. **do if** $A[j] \leq x$

then $i \leftarrow i + 1$

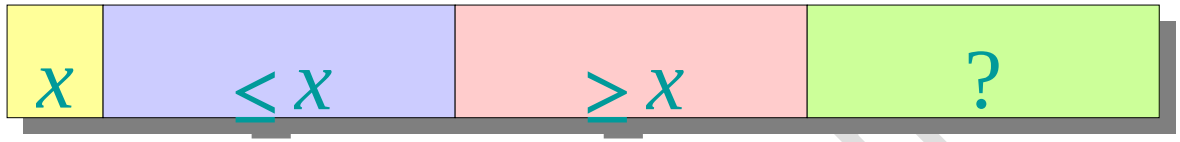
exchange $A[i] \leftrightarrow A[j]$

exchange $A[p] \leftrightarrow A[i]$ **return**

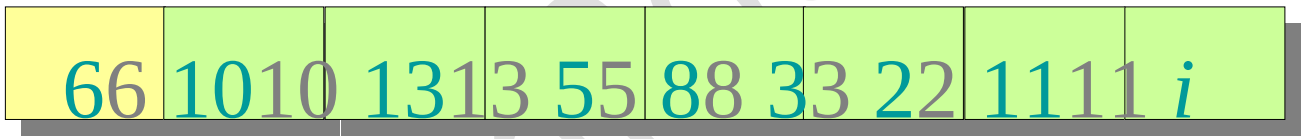
i

Invariant:

Source : MIT Introduction to Algorithms 6.046J/18.401J



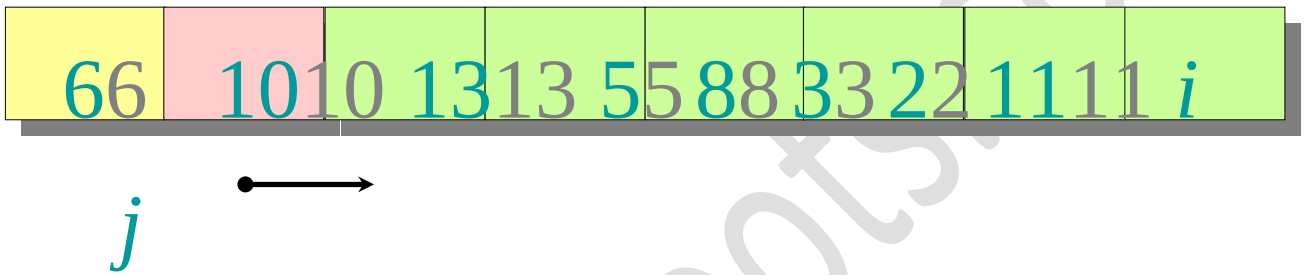
Example of partitioning



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

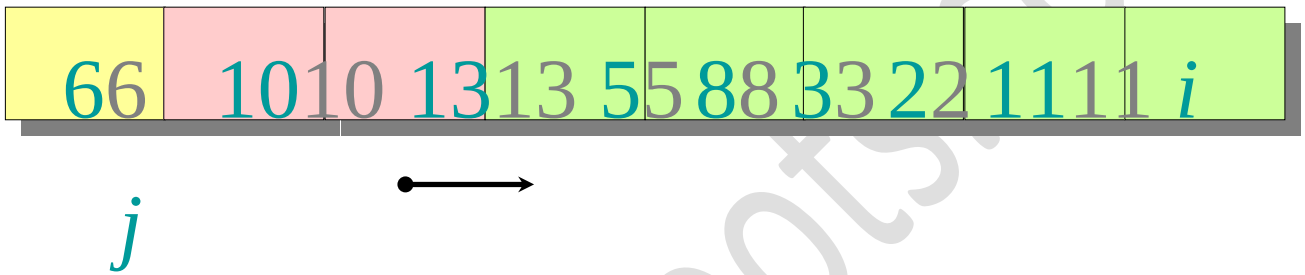
Example of partitioning



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

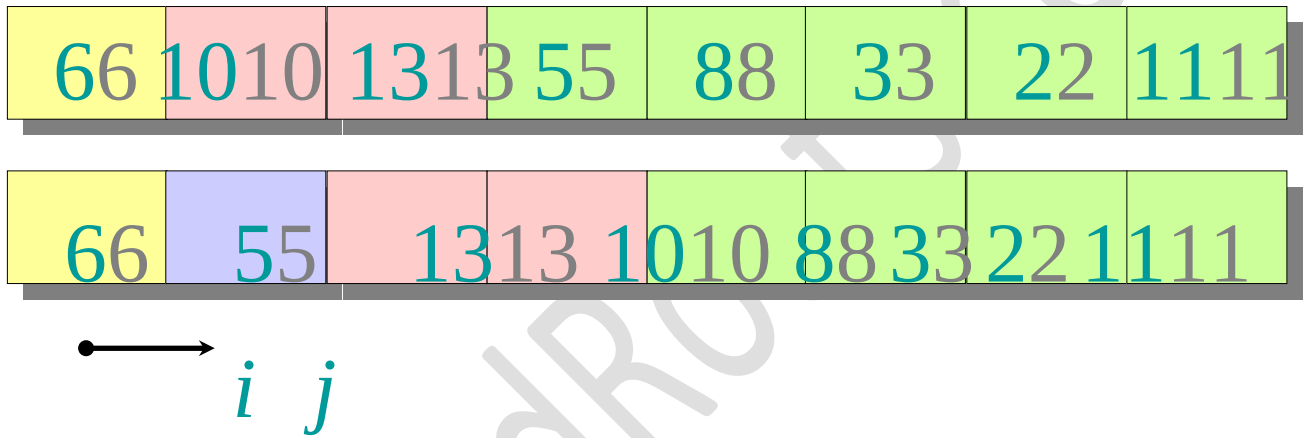
Example of partitioning



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

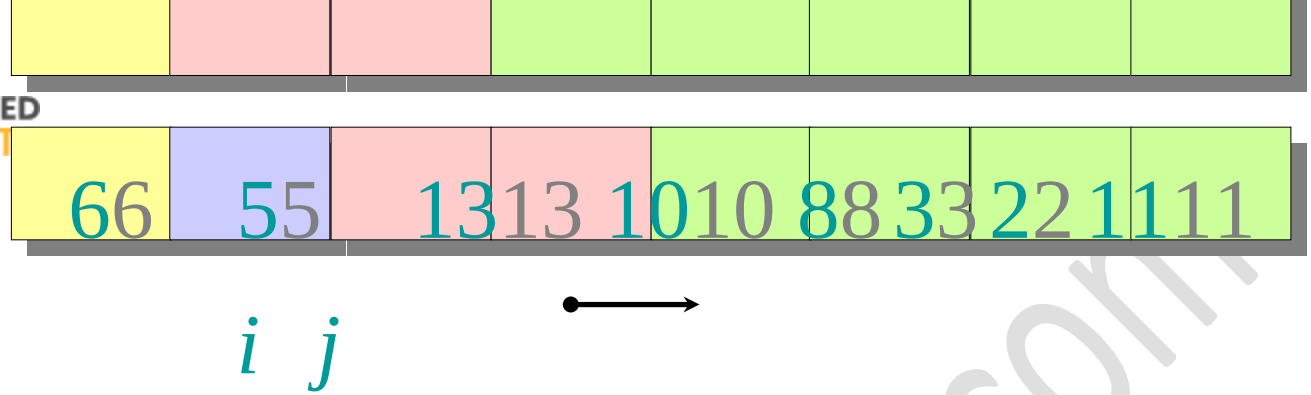
Example of partitioning



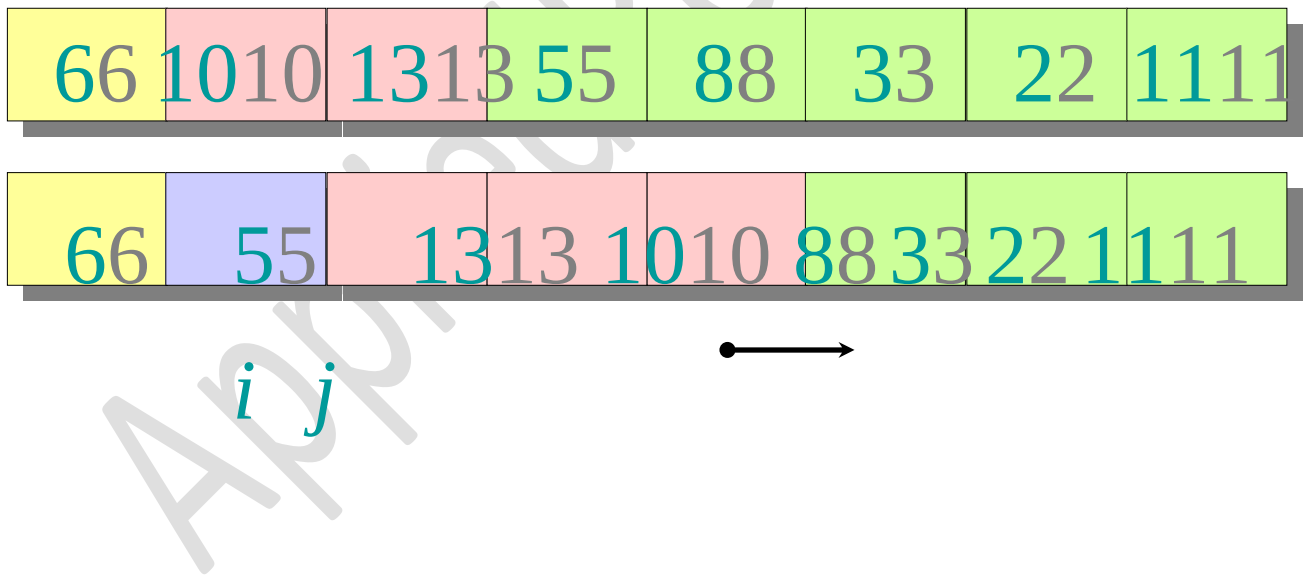
Example of partitioning

66 1010 1313 55 88 33 22 1111

Source : MIT Introduction to Algorithms 6.046J/18.401J



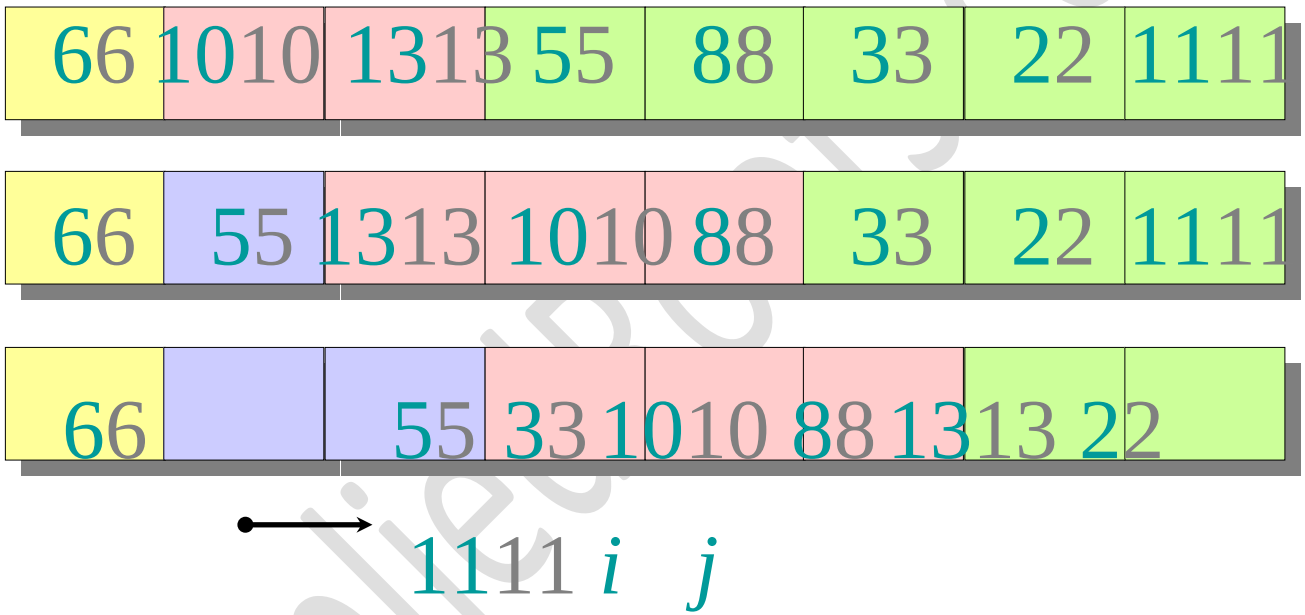
Example of partitioning



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

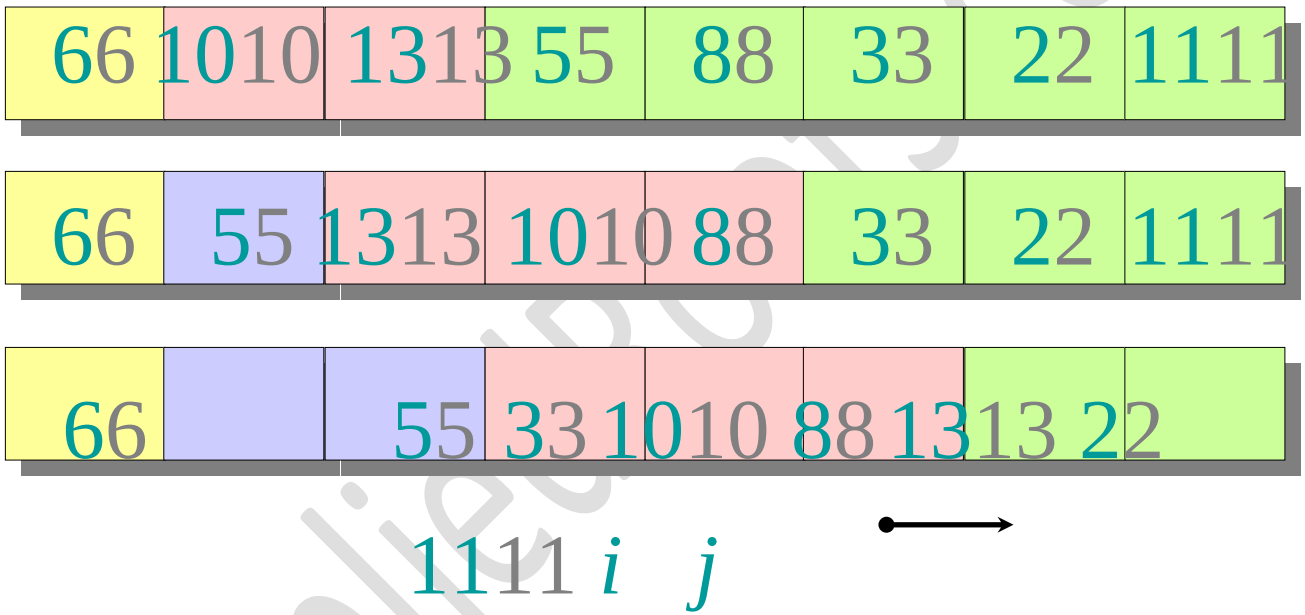
Example of partitioning



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

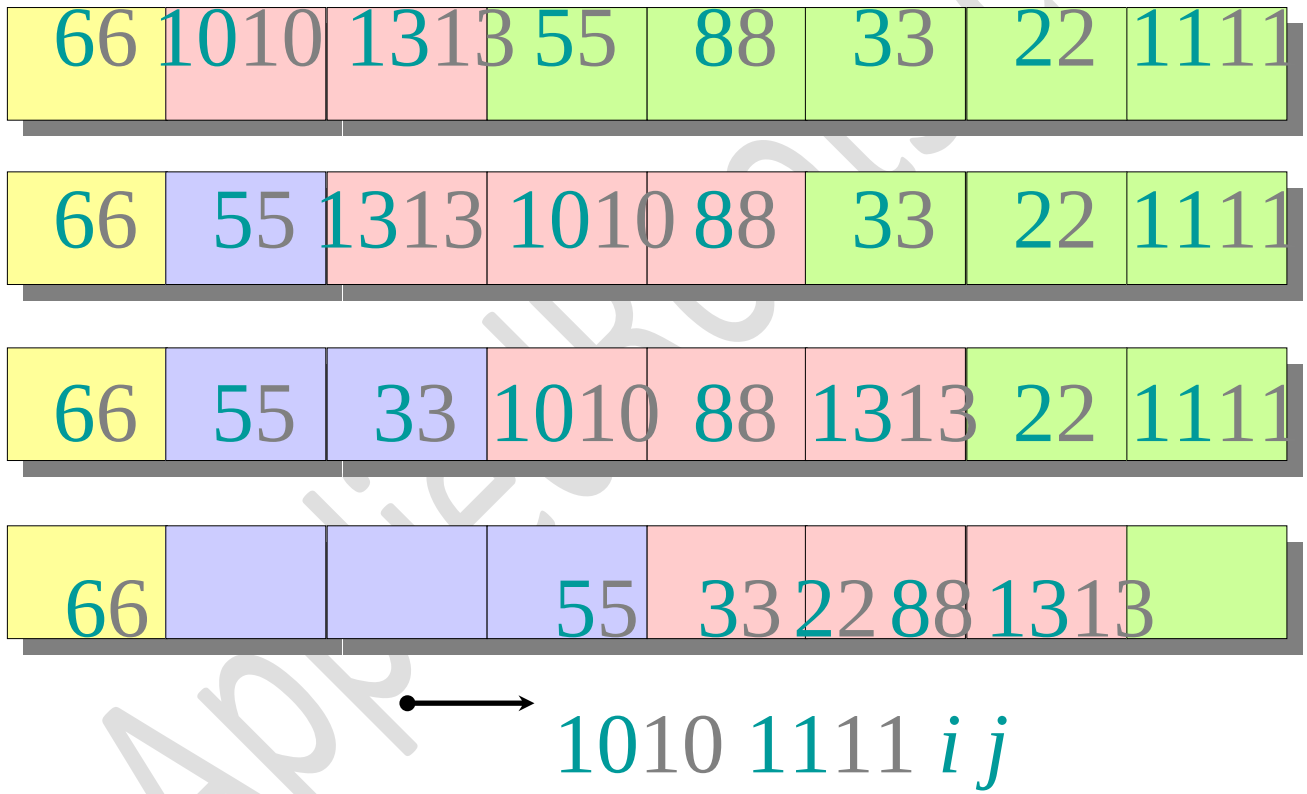
Example of partitioning



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

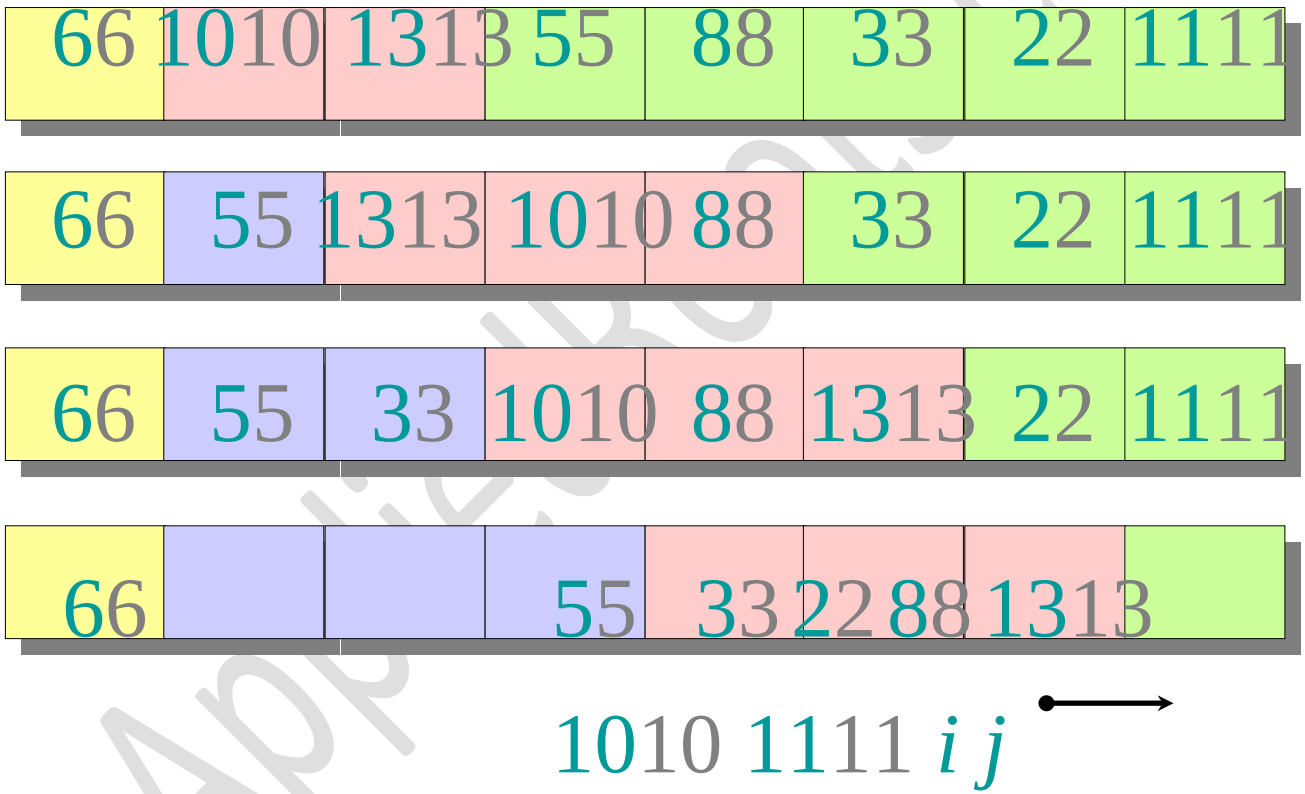
Example of partitioning



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

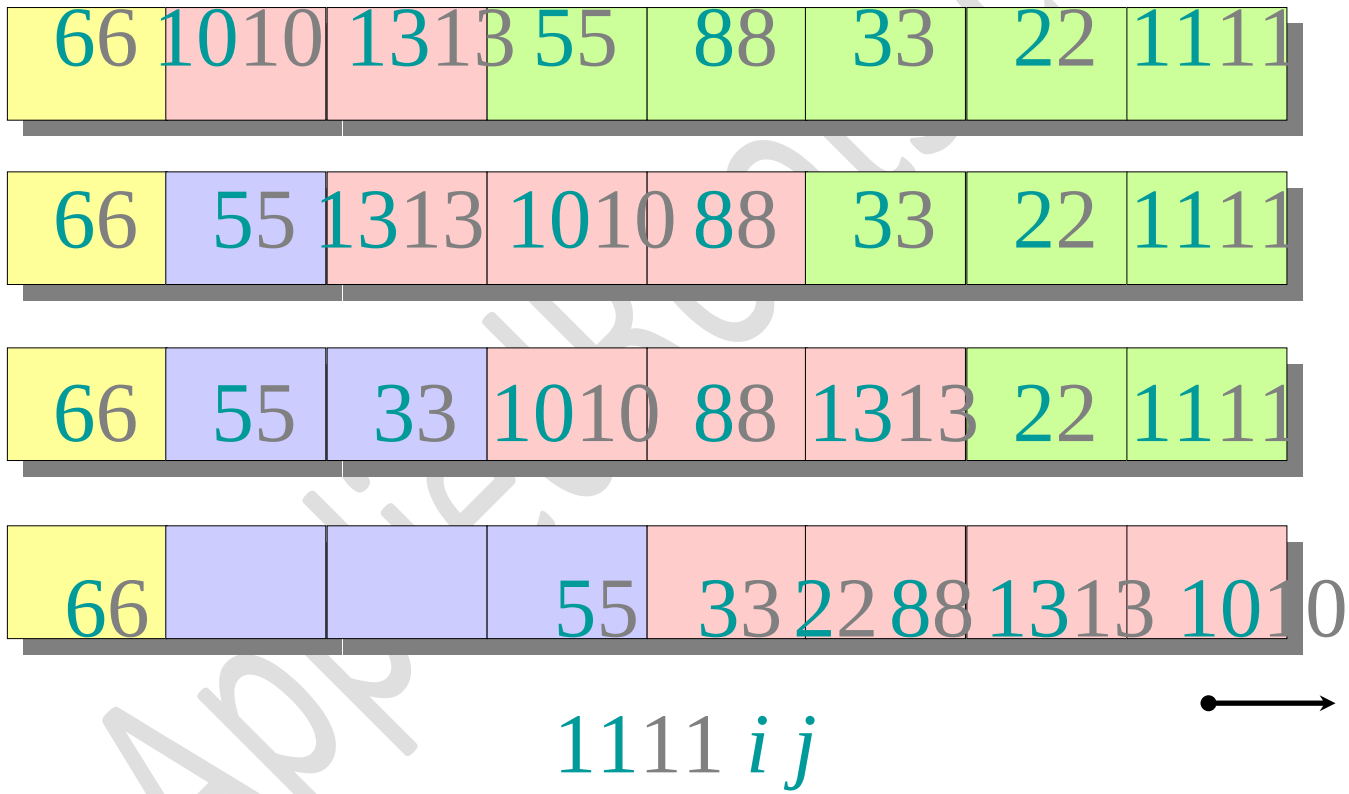
Example of partitioning



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

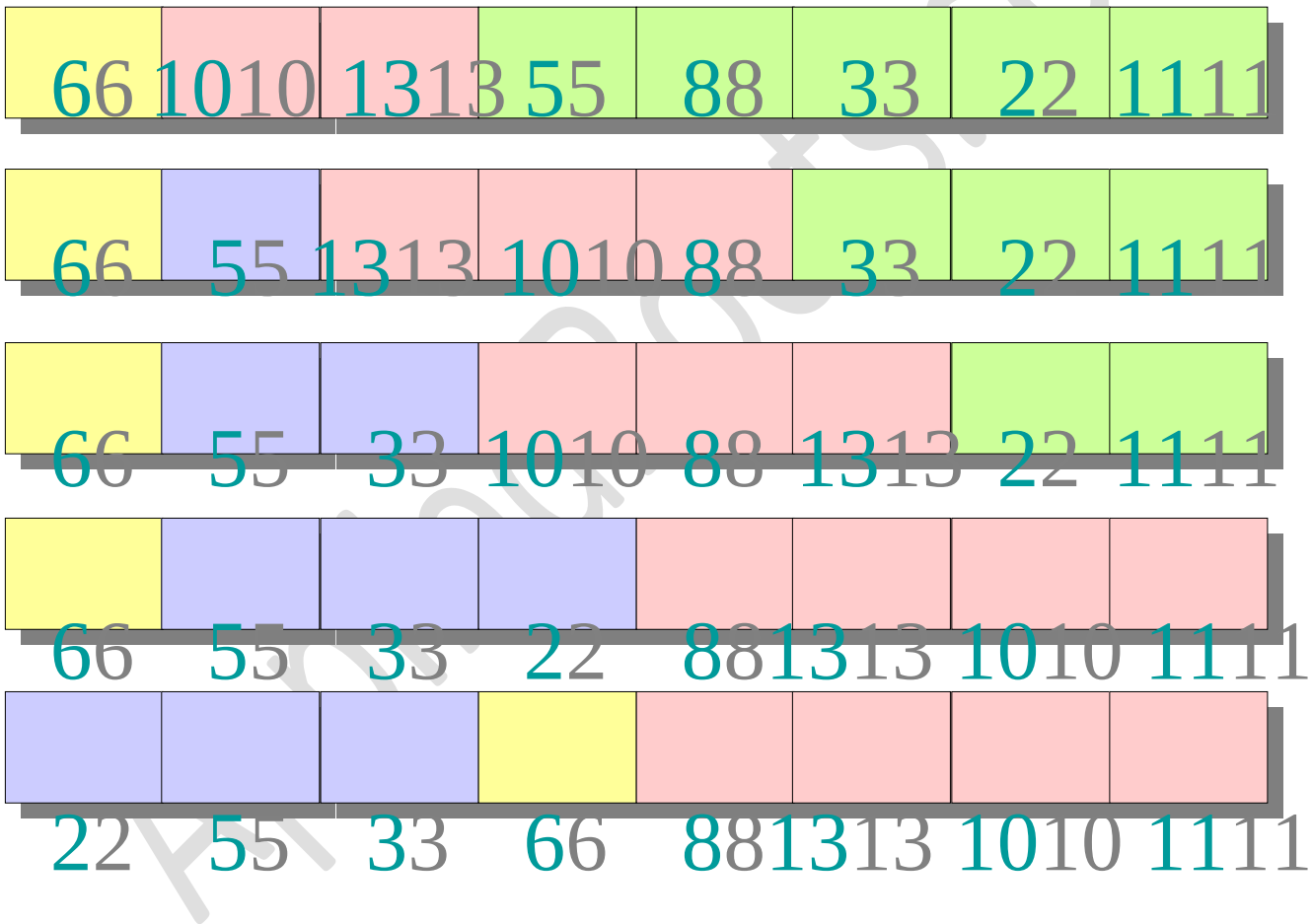
Example of partitioning



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

Example of partitioning



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

i

Pseudocode 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$(A, 1, n)$

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

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.

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

Worst-case of quicksort

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

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

Worst-case recursion tree

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

Worst-case recursion tree

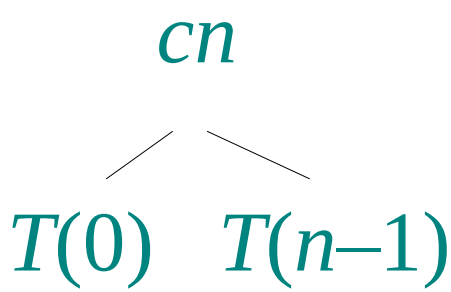
$$T(n) = T(0) + T(n-1) + cn$$

$T(n)$

Source : MIT Introduction to Algorithms 6.046J/18.401J

Worst-case recursion tree

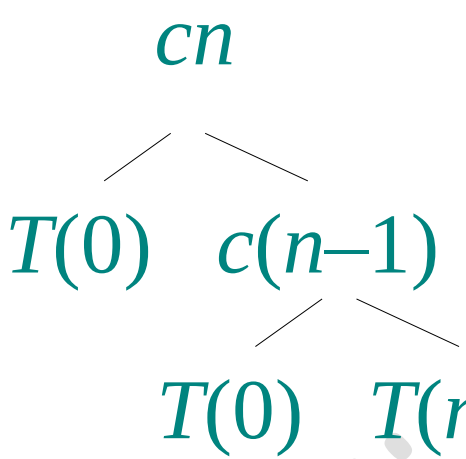
$$T(n) = T(0) + T(n-1) + cn$$



Source : MIT Introduction to Algorithms 6.046J/18.401J

Worst-case recursion tree

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

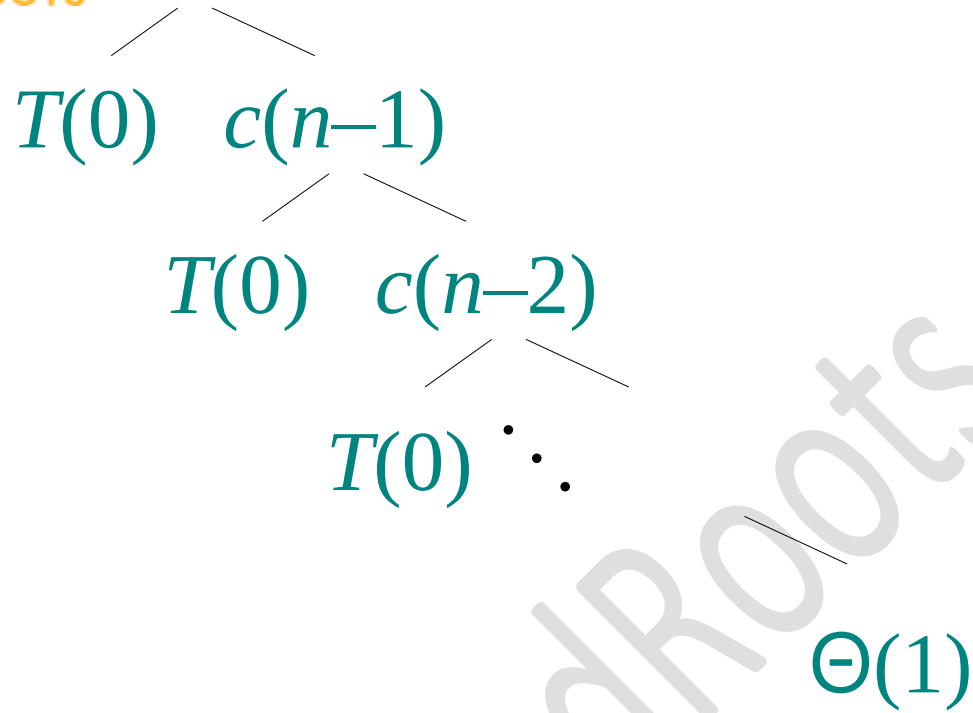


Worst-case recursion tree

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

cn

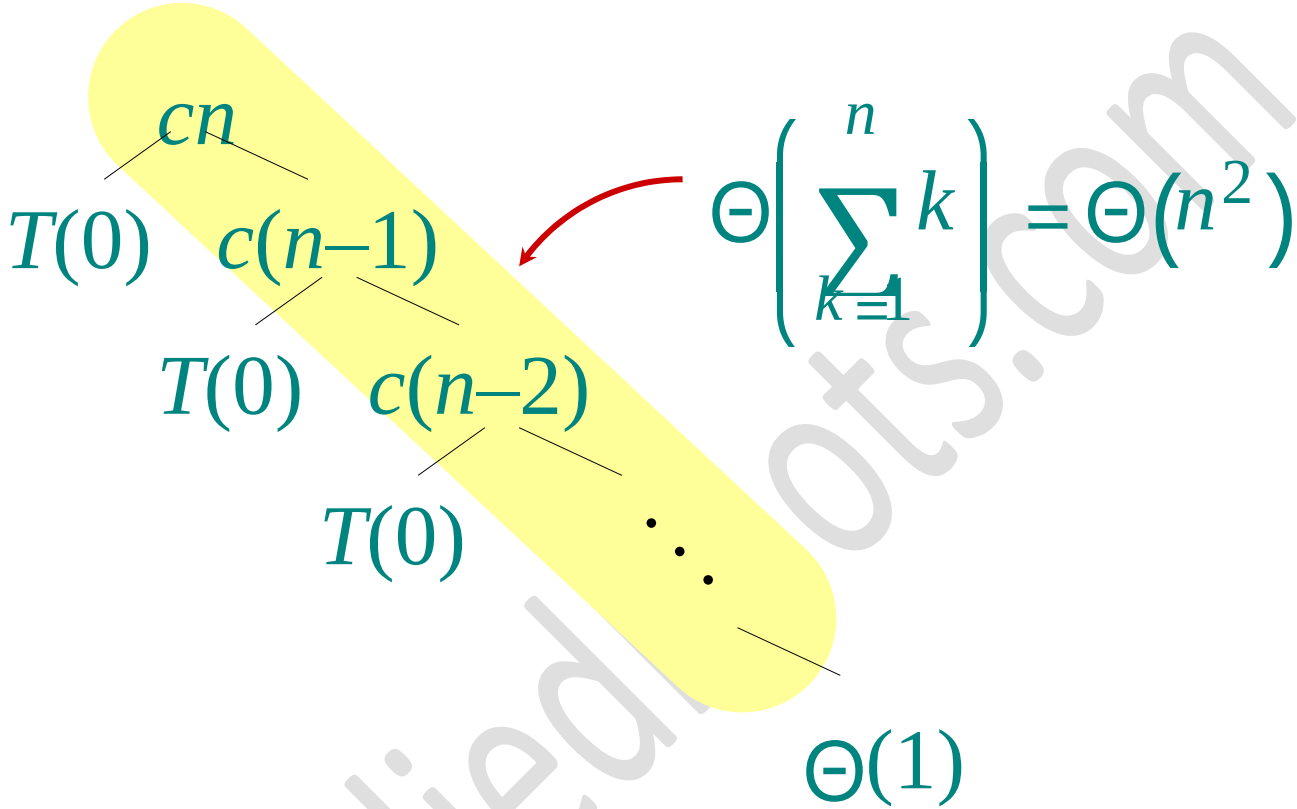
Source : MIT Introduction to Algorithms 6.046J/18.401J



Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

Source : MIT Introduction to Algorithms 6.046J/18.401J

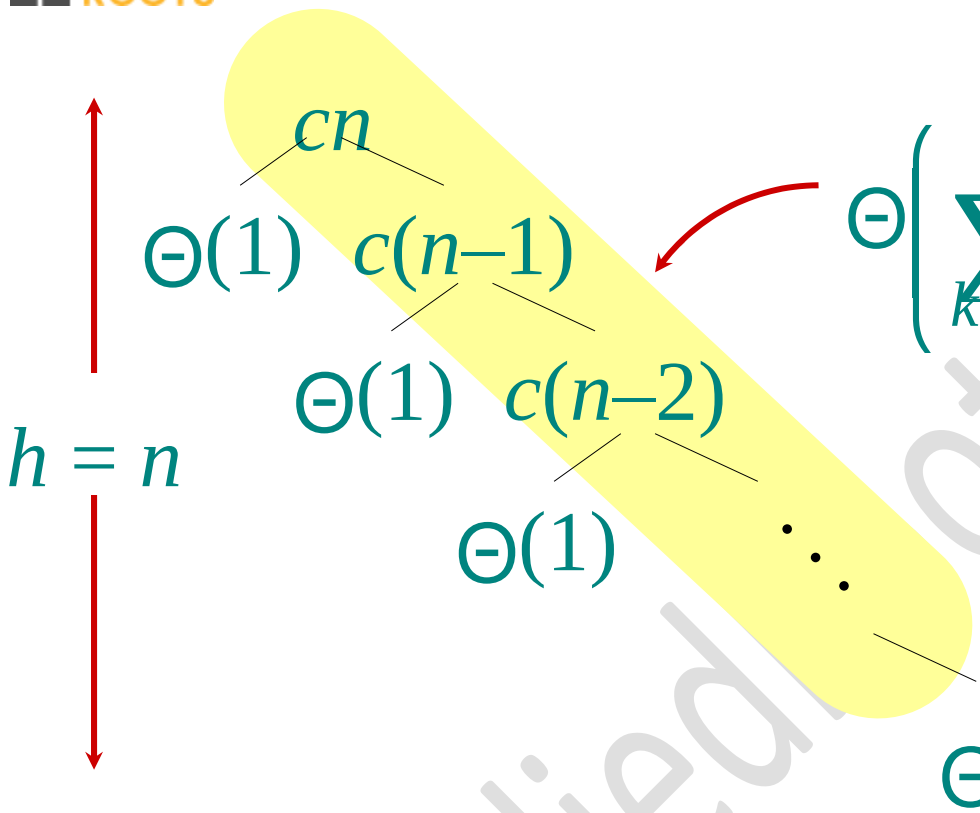


$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2)$$

Worst-case recursion tree

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

Source : MIT Introduction to Algorithms 6.046J/18.401J



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2)$$

$$T(n) = \Theta(n) + \Theta(n^2) = \Theta(n^2)$$

Best-case analysis

(For intuition only!)

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

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

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

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

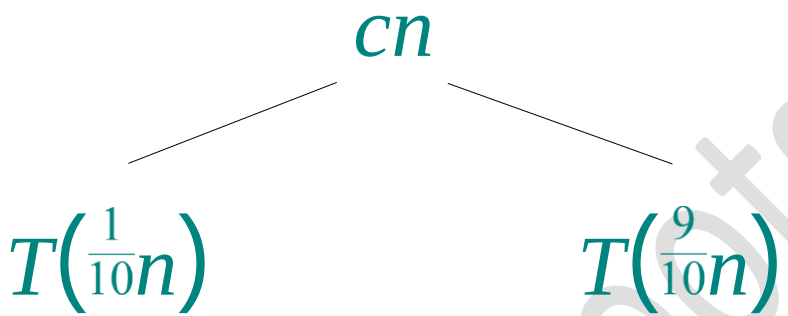
What is the solution to this recurrence?

Analysis of "almost-best" case

$$T(n)$$

Source : MIT Introduction to Algorithms 6.046J/18.401J

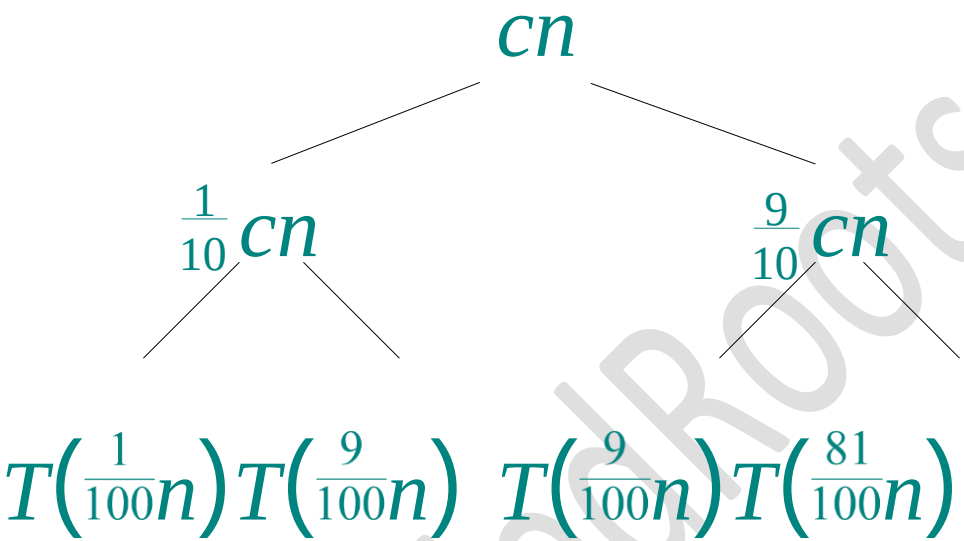
Analysis of “almost-best” case



Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

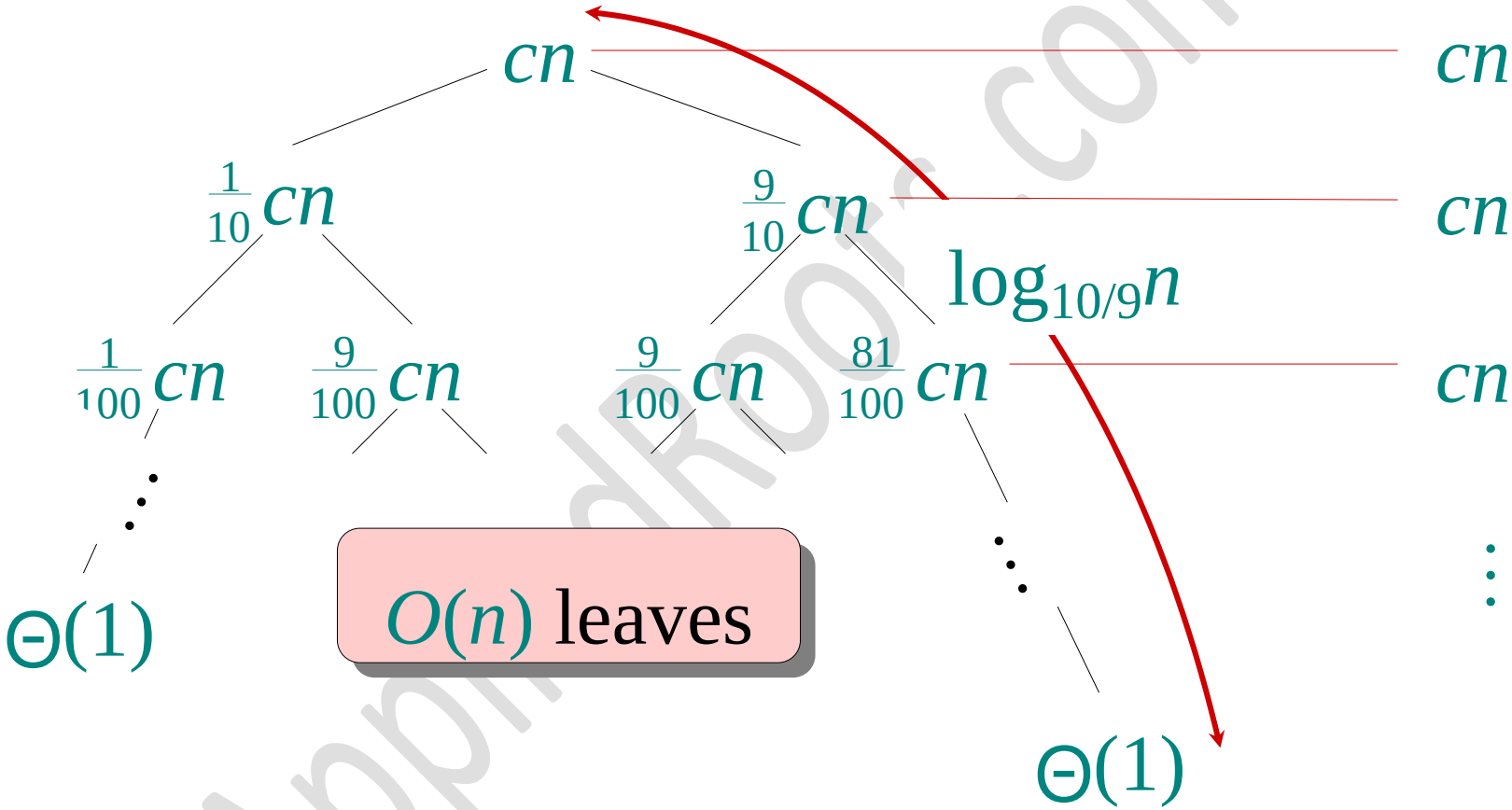
Analysis of “almost-best” case



Source : MIT Introduction to Algorithms 6.046J/18.401J

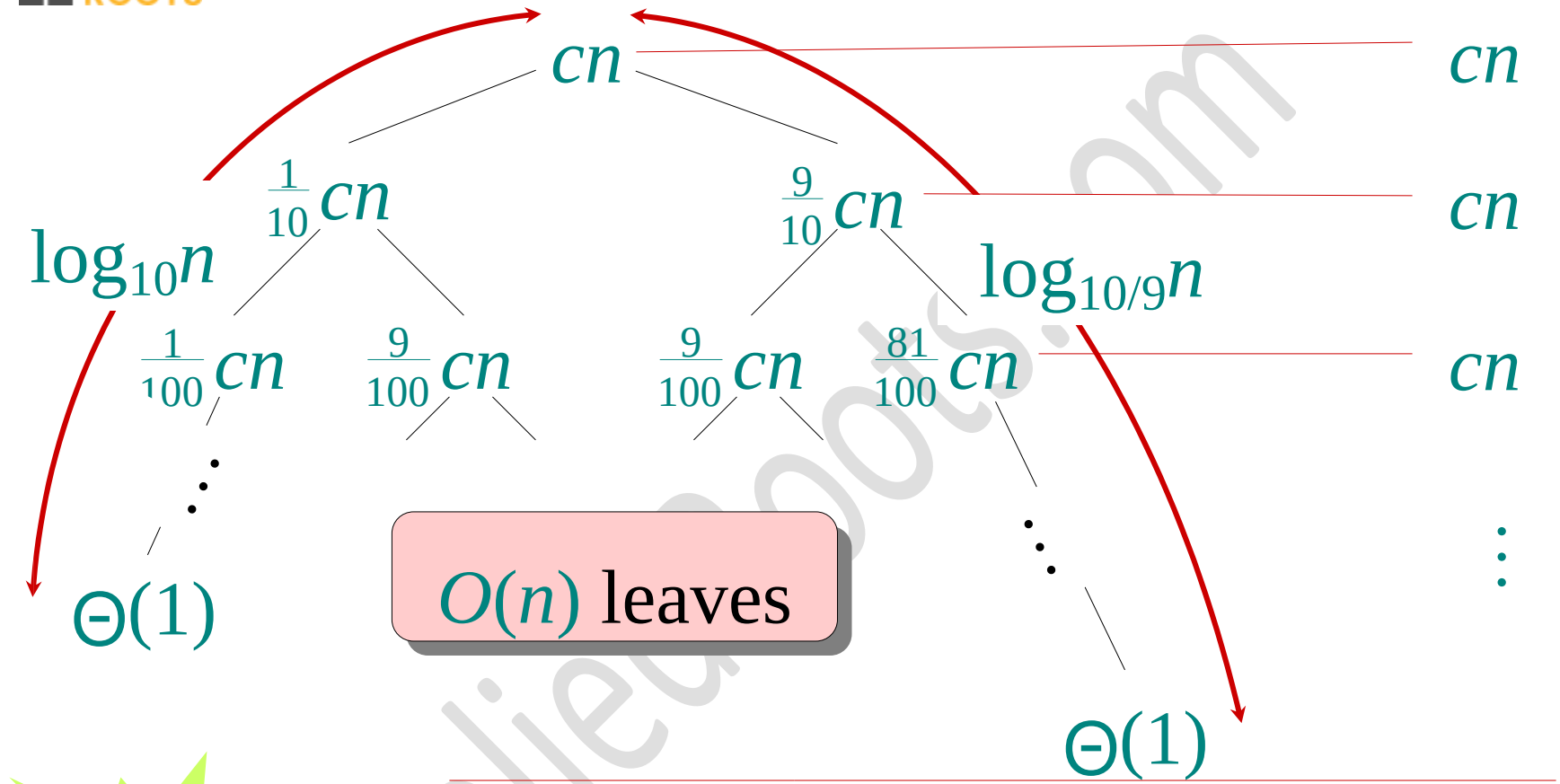
APPLIED ROOTS

Analysis of “almost-best” case



Analysis of “almost-best” case

Source : MIT Introduction to Algorithms 6.046J/18.401J



Θ(*n*lg*n*)

Lucky!

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

More intuition

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

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

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

Solving:

$$L(n) = 2(L(n/2-1) + \Theta(n/2)) + \Theta(n)$$

$$= 2L(n/2-1) + \Theta(n)$$

$$= \Theta(n \lg n)$$ ***Lucky!***

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

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.

Source : MIT Introduction to Algorithms 6.046J/18.401J

- The worst case is determined only by the output of a random-number generator.

Randomized quicksort analysis

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***

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

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

Analysis (continued)

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

$T(n-1) + T(0) + \Theta(n)$ if $n-1 : 0$ split,

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

Calculating expectation

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

Take expectations of both sides.

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

Calculating expectation

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

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

Linearity of expectation.

Source : MIT Introduction to Algorithms 6.046J/18.401J

Calculating expectation

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

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

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

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

Independence of X_k from other random choices.

Calculating expectation

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

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

$$= \sum E\,X_k \cdot \quad [\quad] \quad [\qquad\qquad] \quad ET(k)$$

$$+T(n-k-1) + \Theta(n)$$

$$k=0$$

$$\sum \quad [\quad] \quad \sum \quad [\qquad] \quad \sum$$

$$ET(k) + ET(n-k-1) + \Theta(n)$$

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

Calculating expectation

$$[\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad]$$

Source : MIT Introduction to Algorithms 6.046J/18.401J

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

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

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

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

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

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

Summations have identical terms.

Hairy recurrence

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

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

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

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

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

Substitution method

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

Substitute inductive hypothesis.

Source : MIT Introduction to Algorithms 6.046J/18.401J

Substitution method

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

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

Use fact.

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

Substitution method

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

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

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

$\lfloor 4 \rfloor$

Express as ***desired – residual***.

Substitution method

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

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

Source : MIT Introduction to Algorithms 6.046J/18.401J

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

$$\leq an\lg n,$$

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

Quicksort in practice

- Quicksort is a great general-purpose sorting algorithm.

Source : MIT Introduction to Algorithms 6.046J/18.401J

APPLIED ROOTS

- 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.

Source : MIT Introduction to Algorithms 6.046J/18.401J