CH08-320201
Algorithms and Data Structures

Lecture 9/10 — 6 Mar 2018

Prof. Dr. Michael Sedlmair

Jacobs University

Spring 2018

This & that

- Don't forget: midterm in 2 weeks

2.2 Quicksort

Divide & Conquer

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

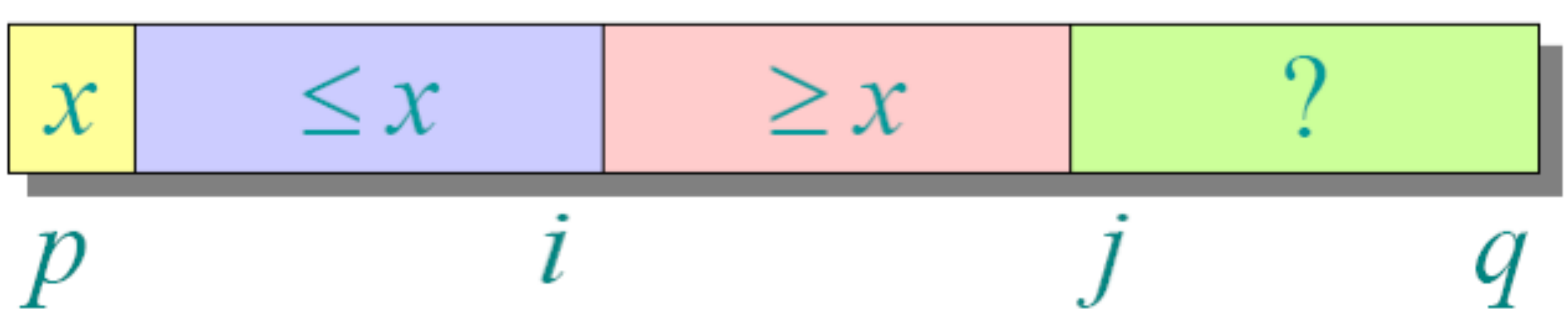
2. Conquer:
   Recursively sort the two subarray
3. Combine:
   Nothing to be done.

Key: Linear-time partitioning subroutine.

Divide

```
PARTITION (A, p, q)                //A[p. . q]
  x := A[p]                         //pivot = A[p]
  i := p
  for j := p + 1 to q
    if A[j] ≤ x
    then i:=i+1
       exchange A[i] ↔ A[j]
  exchange A[p] ↔ A[i]
  return i
```

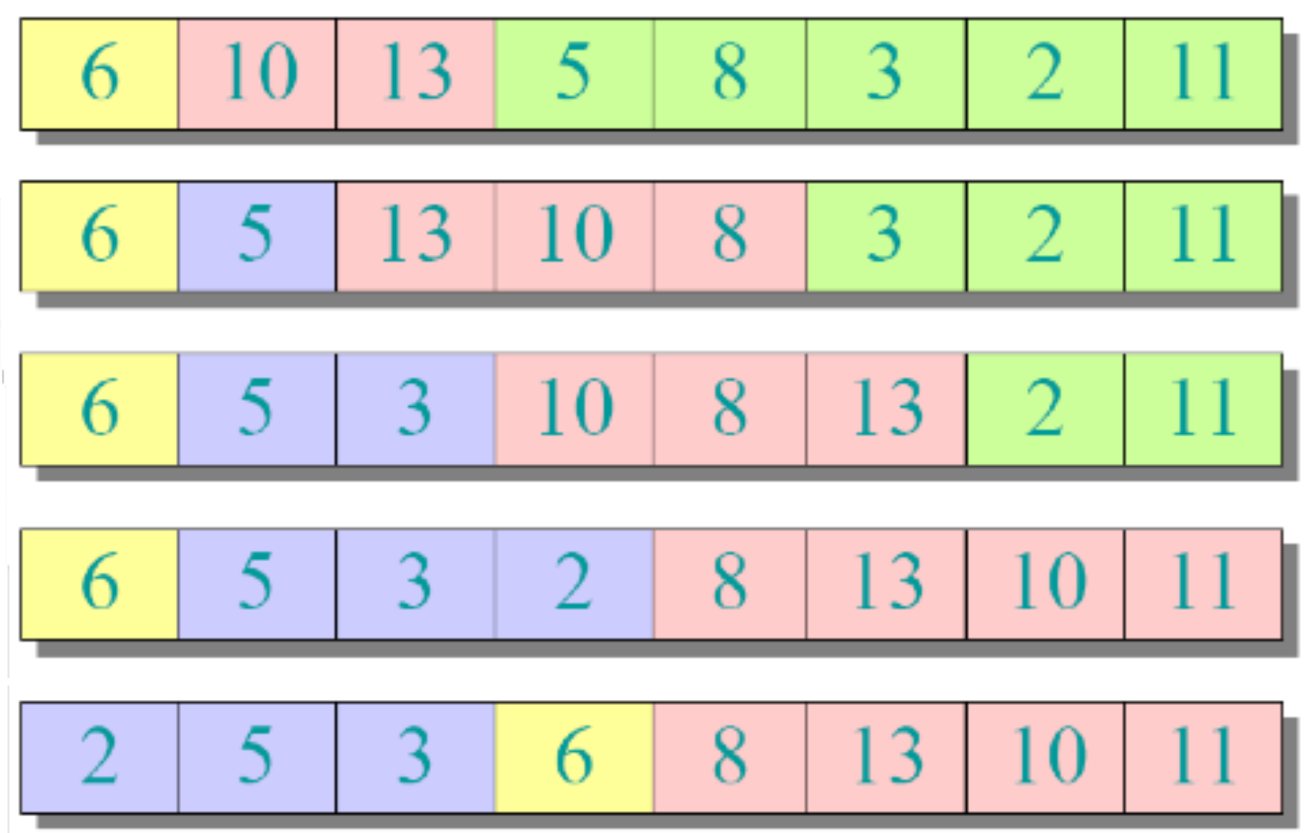
Invariant:



Divide

- Example for PARTITION:

```
PARTITION (A, p, q)
   x := A[p]
   i := p
   for j := p + 1 to q
      if A[j] ≤ x
      then i:=i+1
           exchange A[i] ↔ A[j]
   exchange A[p] ↔ A[i]
   return i
```



Divide

```
PARTITION (A, p, q)              //A[p. . q]
  x := A[p]                      //pivot = A[p]
  i := p
  for j := p + 1 to q
    if A[j] ≤ x
    then i:=i+1
       exchange A[i] ↔ A[j]
  exchange A[p] ↔ A[i]
  return i
```

Running time for $n = q - p + 1$ elements:

T (n) = Θ(n)

Conquer

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

Initial Call: QUICKSORT (A, 1, n)

Runtime Analysis

- Assume all input elements are distinct.
  - (In practice, there are better partitioning algorithms for when duplicate input elements may exist.)
- Let *T(n)* be the worst-case running time for *n* elements.
- Worst case:
  - Input sorted or reverse sorted.
  - Partition around min or max element.
  - One side of partition always has no elements.

Runtime Analysis

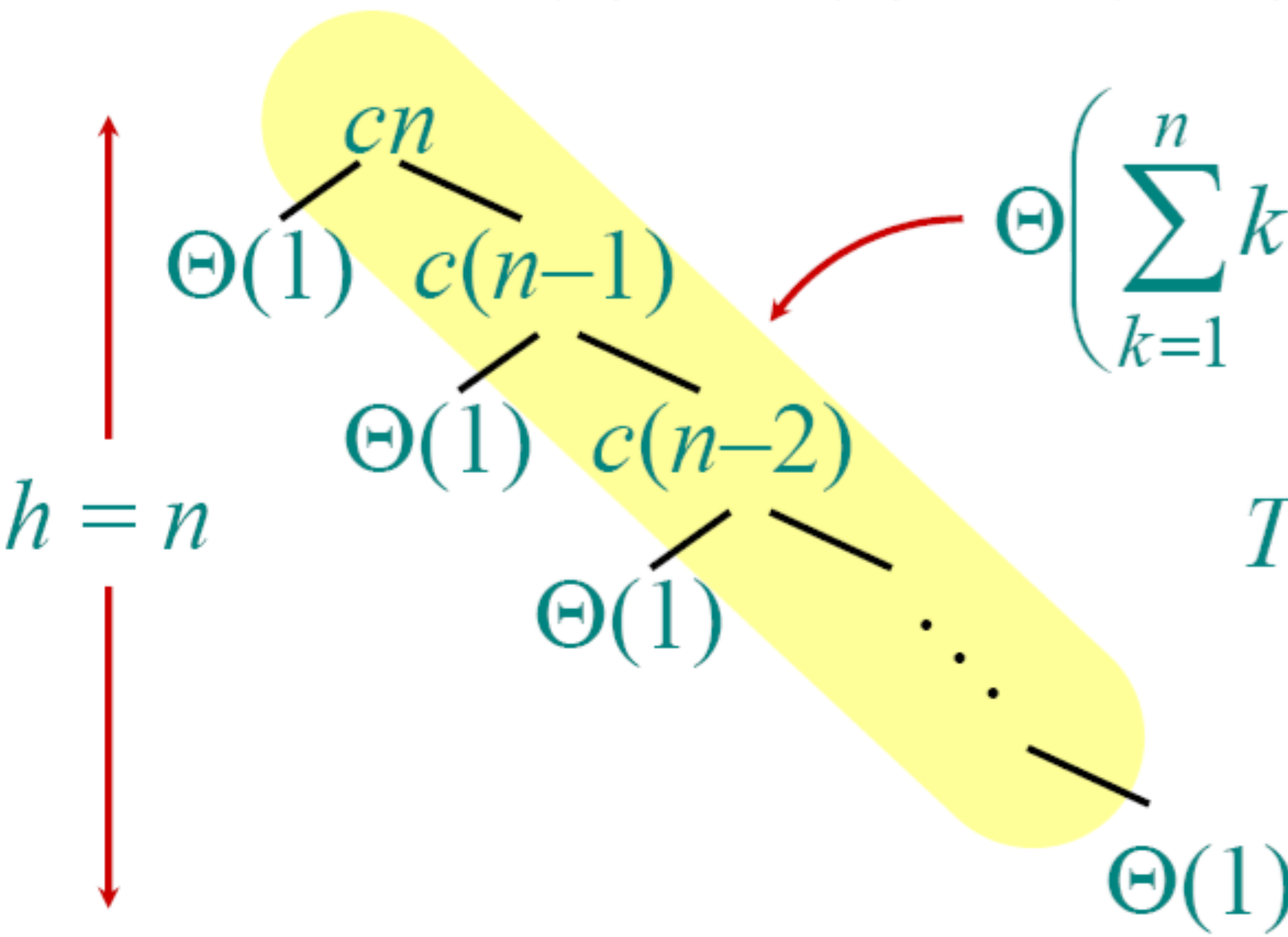
Worst case:

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

Runtime Analysis

Worst-case recursion tree:

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



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

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

Runtime Analysis

Best case:

If we are lucky, PARTITION splits the array evenly:

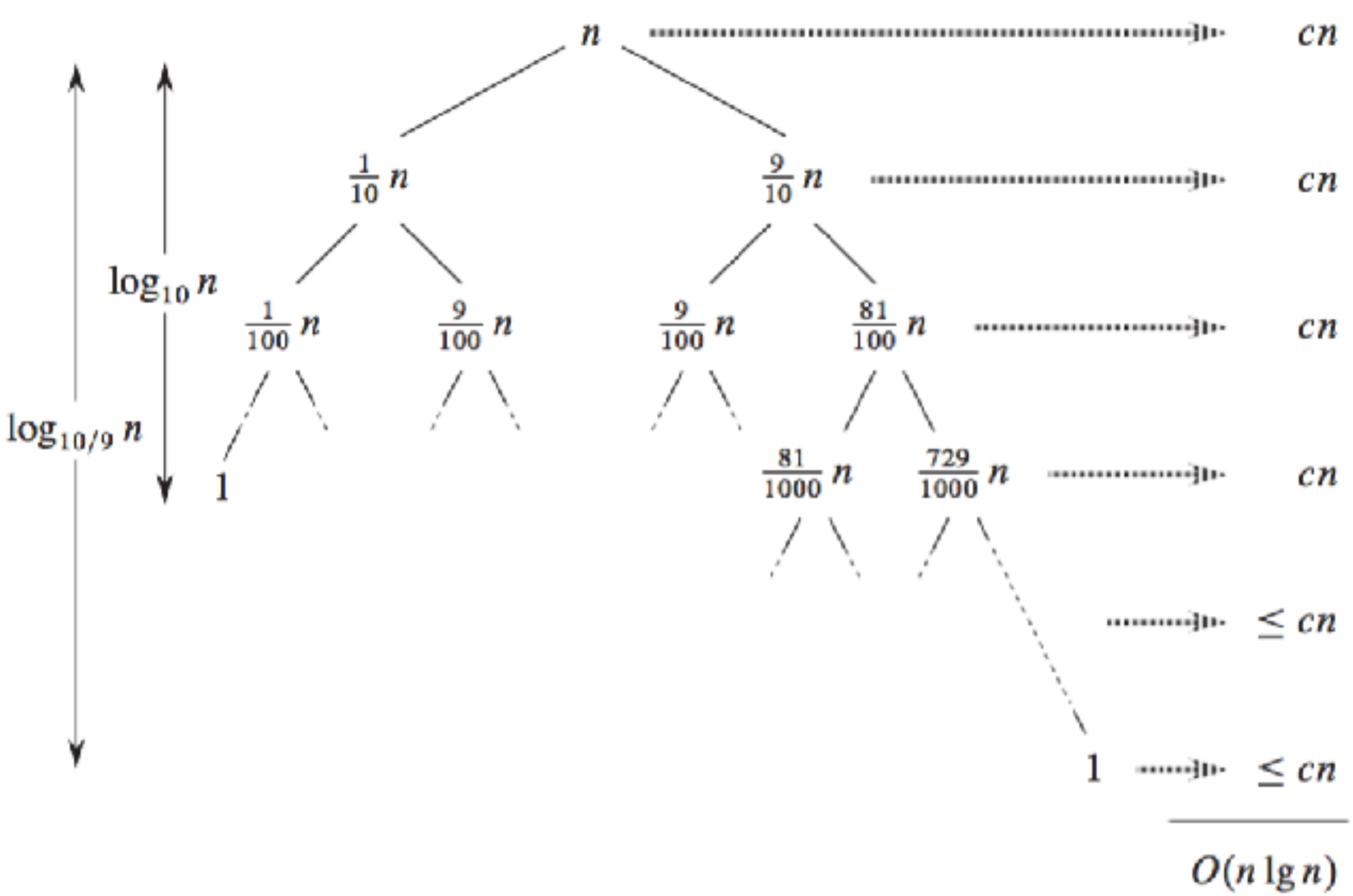
$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)$

This is the same as Merge Sort.

Runtime Analysis

What if the split is 1/10 : 9/10?

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



Runtime Analysis

What if we alternate bw. lucky and unlucky choices

$L(n) = 2U(n/2) + \Theta(n)$ **lucky**

$U(n) = L(n-1) + \Theta(n)$ **unlucky**

Solving:

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

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

$= \Theta(n \lg n)$

How can we make sure that this is usually happening?

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

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

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

Randomized Quicksort

- Let $T(n)$ be the random variable for the running time of the randomized quicksort on an input of size n (assuming random numbers are independent).
- For k = 0, 1 ,… , n–1, define 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

Recurrence:

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

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

Randomized Quicksort

Calculating expectations:

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E[X_k] \cdot E\left[T(k) + T(n-k-1) + \Theta(n)\right] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n) \\
&= \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n)
\end{aligned}
$$

Randomized Quicksort

- Use substitution method to solve recurrence.
- Guess: $E[T(n)] = \Theta(n \lg n)$.
- Prove: $E[T(n)] \leq a\, n \lg n$ for constant $a>0$.
- Use:

$$\sum_{k=2}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$$

(proof by induction)

Randomized Quicksort

Proof:
$$\begin{aligned}
E[T(n)] &= \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n) \\
&\le \frac{2}{n}\sum_{k=2}^{n-1} ak\lg k + \Theta(n) \\
&= \frac{2a}{n}\left(\frac{1}{2}n^2\lg n - \frac{1}{8}n^2\right) + \Theta(n) \\
&= an\lg n - \left(\frac{an}{4} - \Theta(n)\right) \\
&\le an\lg n,
\end{aligned}$$

if a is chosen large enough.

Conclusion

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is often the best practical choice because its expected runtime is $\Theta(n \lg n)$ and the constant is quite small.
- Quicksort is typically over twice as fast as MergeSort.
- Quicksort is an in-situ sorting algorithm (debatable).
- Quicksort has a worst-case runtime of $\Theta(n^2)$ when the array is already sorted.
- Visualization(RandomizedQuicksort): http://www.sorting-algorithms.com/quick-sort

2.3 Lower bounds for sorting

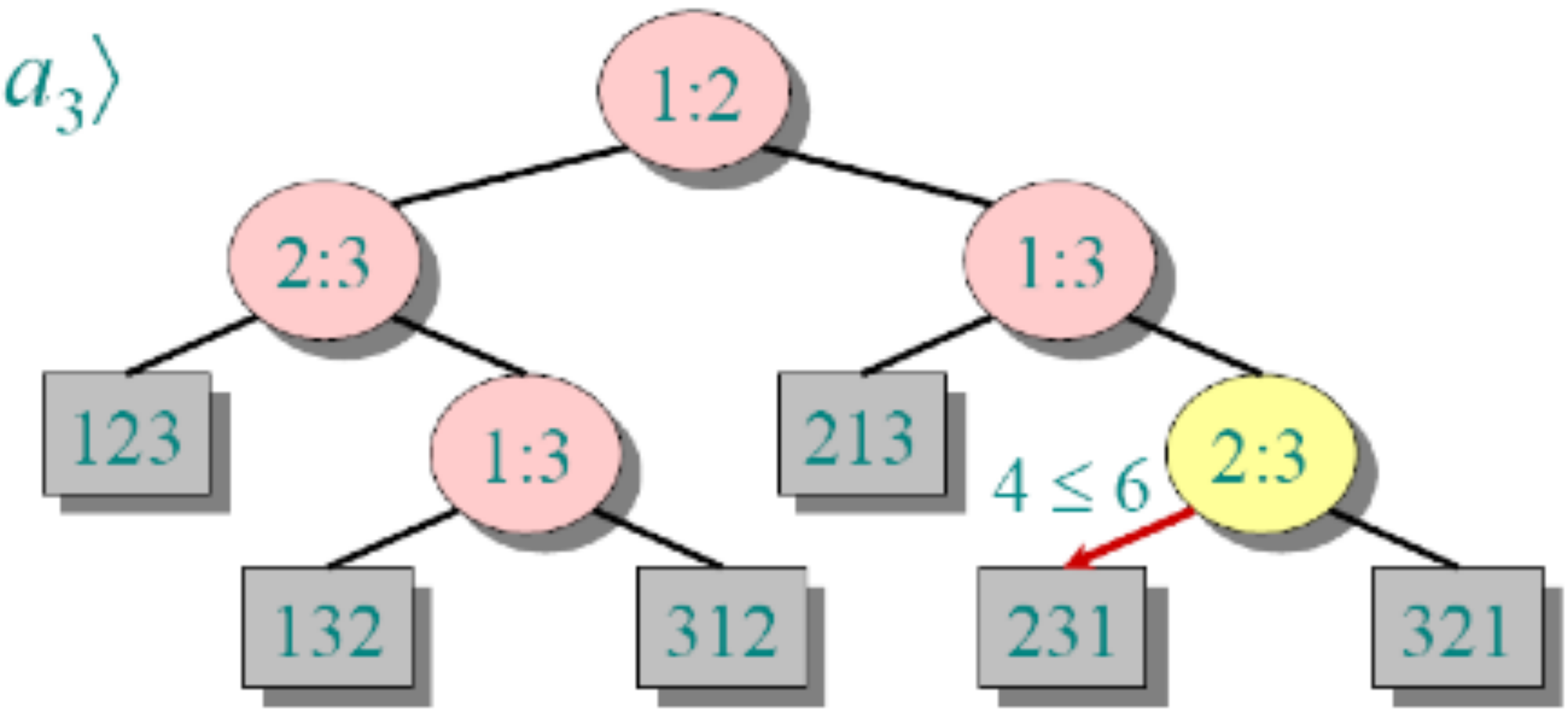
Comparison sorts

- All sorting algorithms we have seen so far are comparison sorts.
- A comparison sort only uses comparisons to determine the relative order of elements.
- The best worst-case running time we encountered for comparison sorting was $O(n \lg n)$.
- Is $O(n \lg n)$ the best we can do?

Decision tree

Example:

Sort $\langle a_1, a_2, a_3 \rangle$
$= \langle 9, 4, 6 \rangle$:



Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ indicating the order $a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$.

Decision tree model

A decision tree can model the execution of any comparison sort:

- One tree for each input size *n*.
- View the algorithm as splitting whenever it compares two elements.
- The tree contains the comparisons along all possible instruction traces.
- The running time of the algorithm = the length of the path taken.
- Worst-case running time = height of tree.

Decision tree sorting

Theorem:

Any decision tree that can sort n elements must have height $\Omega(n \lg n)$.

Proof:

The tree must contain $\geq n!$ leaves,
since there are $n!$ possible permutations.
A height-h binary tree has $\leq 2^h$ leaves.

Thus, $n! \leq 2^h$.

Then,
$$
\begin{aligned}
h &\geq \lg(n!) \\
&\geq \lg((n/e)^n) \\
&= n \lg n - n \lg e \\
&= \Omega(n \lg n).
\end{aligned}
$$

$$n! \sim \sqrt{2\pi n}\ \left(\frac{n}{e}\right)^n; \quad n \to \infty$$

(Stirling's formula)

Lower bound for comparison sorting

- The lower bound for comparison sorting $\Omega(n \lg n)$.
- Heapsort and Merge Sort are asymptotically optimal comparison sorting algorithms.