5. Divide-and-Conquer

Not in book: Quicksort

How?

Quicksort

Divide:

- Choose e from input A
- Partition A into A^-, e, and A^+

Conquer:

- Quicksort(A^-)
- Quicksort(A^+)

Combine:

- Output sorted A^-, then e, then sorted A^+

Mergesort: easy divide, hard combine
Quicksort: hard divide, easy combine

Quicksort pseudo-code

Precondition: A an array of elements

```
Quicksort(A)
if |A| <= 1
   return A          base case
Choose a splitter a_i from A                 Partition
A^- = A^+ = {}
foreach element a_j of A do
    LI: A- U A+ U aj = A and e in A- ≤ aj and e in A+ ≤ j
   if aj < ai then
      Add a_j to A^-
   else
      Add a_j to A^+
end foreach
 Assert: e in A- ≤ aj; e in A+ ≥ j
Quicksort(A^-)    Assert: A- in sorted order (inductive hyp.)
Quicksort(A^+)    Assert: A+ in sorted order (inductive hyp.)
Return A^- followed by a_i followed by A^+
```

Precondition: Returns A in sorted order

Quicksort in-action

A	L	G	O	R	I	T	H	M	S

Quicksort in-place pseudo-code

Precondition: A[l..r] is sub-array of items

```
procedure Quicksort(A, l, r)
if l + 1 >= r
   return A
Choose a splitter a_i from A[l]..A[r]
k = Partition(A, l, r, j)
Quicksort(A, l, k-1)
Quicksort(A, k+1, r)
```

A- = A[l..k-1]

A+ = A[k+1..r]

Postcondition: A[l..r] contains original elements in sorted order

Index where splitter was put

Partition pseudo-code

Precondition: A[l..r] is sub-array of items. j is in range [l..r]

```
function Partition(A, l, r, j)
Swap(A[j], A[r])
ll = rr = i = l-1;
loop
```

LI: A[l..ll] ≤ A[r], A[rr.i] ≥ A[r] A[l..r] contains original elements of A

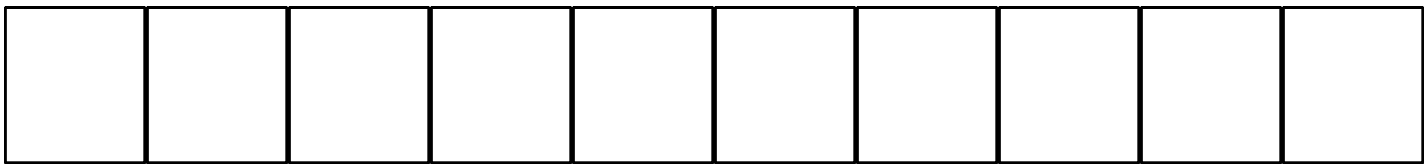
```
   LI: A[l..ll] ≤ A[r], A[rr.i] ≥ A[r] A[l..r] contains original elements of A
   exit when i = r
   i = i+1;
   if A[i] < A[r] then
      Swap(A[rr], A[i])
      rr = rr + 1;
      ll = ll + 1;
endloop
```

Assert: A[l..r] contains original elements of A.
A[l.ll] ≤ A[r], A[rr..r-1] ≥ A[r]

```
Swap(A[r], A[rr]);
Return k = rr.
```

Postcondition: A[l..r] contains original elements of A.
Returns k s.t. A[l..k-1] ≤ A[k] (=A[j]) ≤ A[k+1]..A[r]

In-place quicksort in action



Runtime analysis of Quicksort

What is $T(n)$?

Divide: Partition: $O(n)$

Conquer: max(
$T(1) + T(n-2)$
$T(2) + T(n-3)$
...
$T(n-2) + T(1)$)

$$T(n) = T(n-1) + O(n)$$
$$T(n) = T(n-2) + O(n) + O(n)$$
$$T(n) = T(1) + (n-2)*O(n)$$
$$T(n) = O(n^2)$$

Combine: $O(1)$

Worst case: $O(n^2)$

Worst-case input?

Quicksort

How to choose the splitter

- First
- Last
- Middle
- Best of three
- Random
- Best of best of three

Average-case complexity of Quicksort

Observations:

- After an element ai is chosen as a splitter value and compared to other elements in its range, it is ____never compared again____
- If two elements are ____separated into different partitions____, they will never be compared to each other again

If we look at a range of sorted elements $z_i..z_j$, z_i and z_j are compared to each other only if ____z_i or z_j____ chosen as a splitter before ____any elements between z_i and z_j____

Average-case complexity of Quicksort

Let x and y be two elements with $x \le y$

- If Quicksort picks any splitter z with $x \le y \le z$ __before it picks__ x or y then it never compares x to y
- Assume that it picks the splitters randomly from all candidates in an unpartition group
- If x and y are k places apart in the final sorted order, the chance of picking x or y before picking z between them is __2/(k+1)__

The farther apart two elements in the final order, the less likely they are to be compared

Average-case complexity for Quicksort

"If x and y are k places apart in the final sorted order, the chance of picking x or y before picking z between them is 2/(k+1)"

1 pair	n-1 apart	1*(2/n)
2 pairs	n-2 apart	2*(2/n-1)
3 pairs	n-3 apart	
...		
n-2 pairs	2 apart	(n-2)*(2/3)
n-1 pairs	1 apart	(n-1)*(2/2)

Expected time

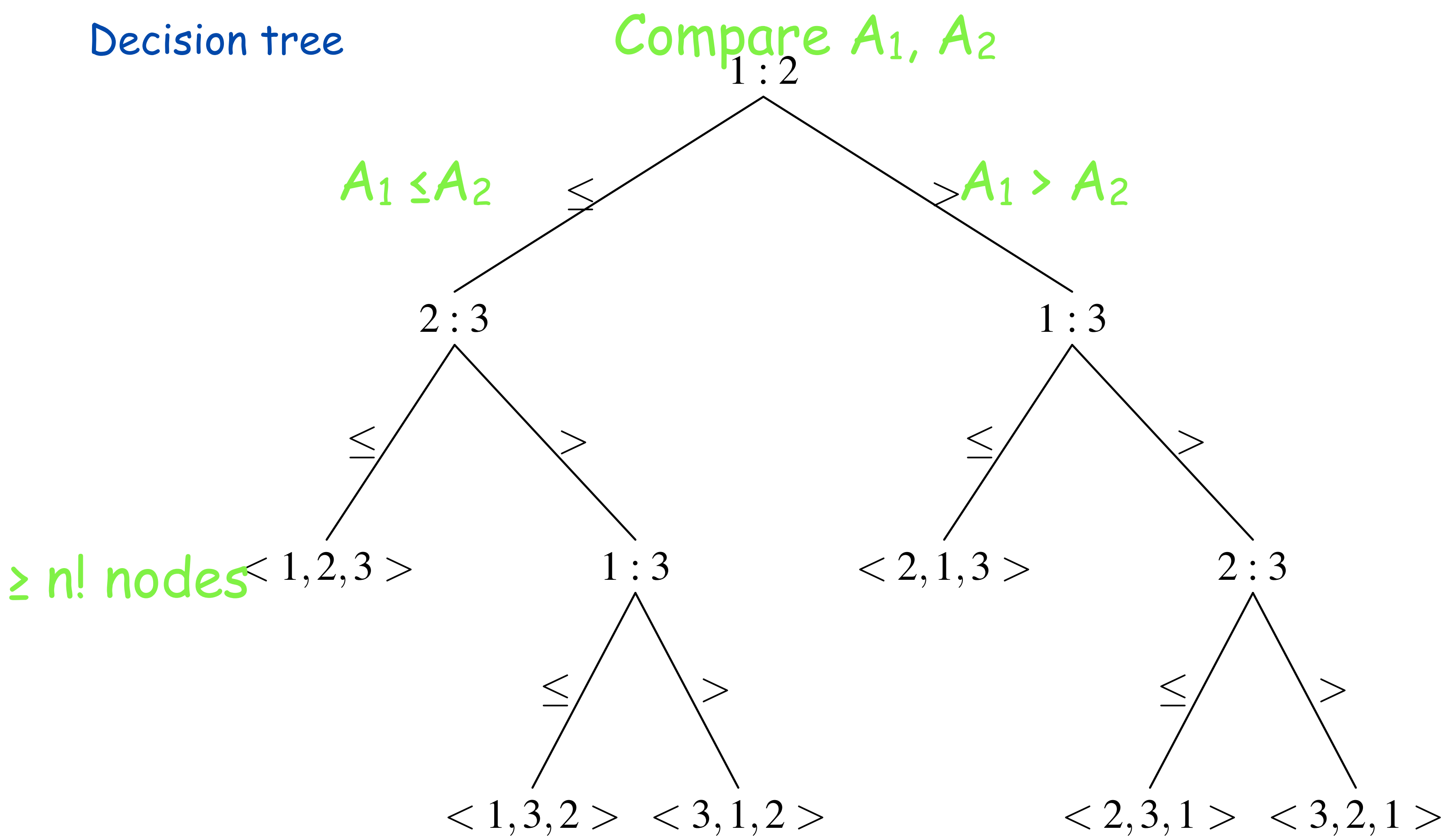
θ(n log n)

=sum_i=1^n-1 sum_k=1^n-i 2/k+1 ≤ sum_i=1^n sum_k=1^n 2/k ≤ sum_i=1^n 2(log n + 1) = O(n log n)

Averaged across all ____ inputs, Quicksort takes time θ(n log n)

Lower bound for comparison-based sorting

Decision tree



Sorted order is A_1, A_3, A_2

Model ANY sorting alg based on compares as decision tree

Lower-bound for comparison based sorting

Min height of tree with k leaves = log k

Min height of tree with n! leaves = log(n!)

= θ(n log n)

Lower-bound for comparison-based sorting

Any algorithm based on comparisons between elements is in $\Omega(n \log n)$ time

For any algorithm, there's an instance that takes at least n log n

Can there be algorithms with instances that sort in O(n) time?

Can there be algorithms that sort in O(n log n) time for every instance?

Can there be algorithms that sort in O(n log n) time for most instances?