Sorting in Linear (?) Time

Sorting bounds

Heapsort, Mergsort: O(n log n)

Quicksort: O(n log n) on average

Can we do better?

Comparison-based sorting

Sorted order is determined based **only** on a comparison between input elements

- A[i] < A[j]
- A[i] > A[j]
- A[i] = A[j]
- A[i] ≤ A[j]
- A[i] ≥ A[j]

Do any of the sorting algorithms we've looked at use additional information?

- No
- All the algorithms we've seen are comparison-based sorting algorithms

Comparison-based sorting

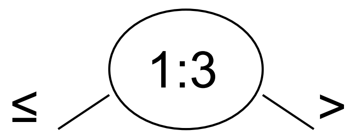
Sorted order is determined based **only** on a comparison between input elements

- $A[i] < A[j]$
- $A[i] > A[j]$
- $A[i] = A[j]$
- $A[i] \leq A[j]$
- $A[i] \geq A[j]$

Can we do better than $O(n \log n)$ for comparison based sorting approaches?

Decision-tree model

- *Full* binary tree representing the comparisons between elements by a sorting algorithm
- Internal nodes contain indices to be compared



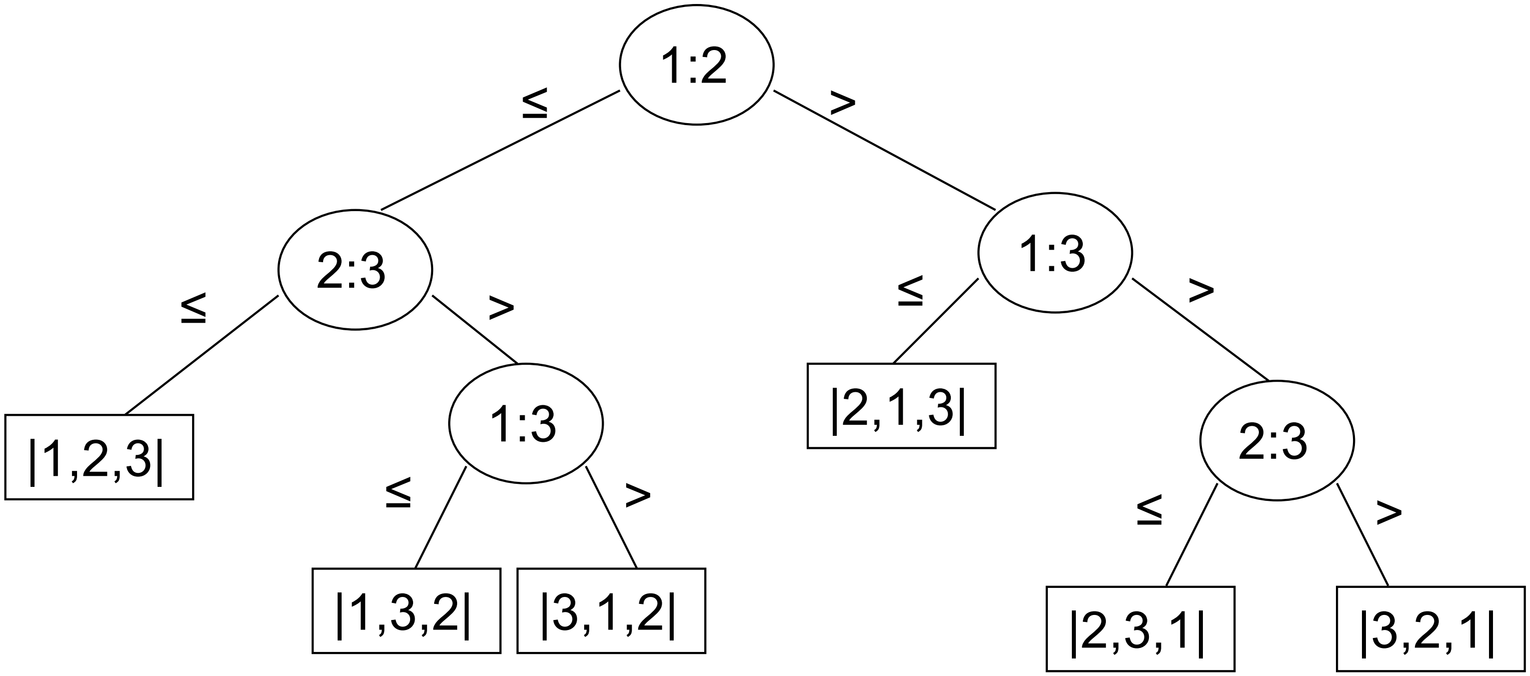
- Leaves contain a complete permutation of the input

[3, 12, 7] ⇨ | 1,3,2 | ⇨ [3, 7, 12]

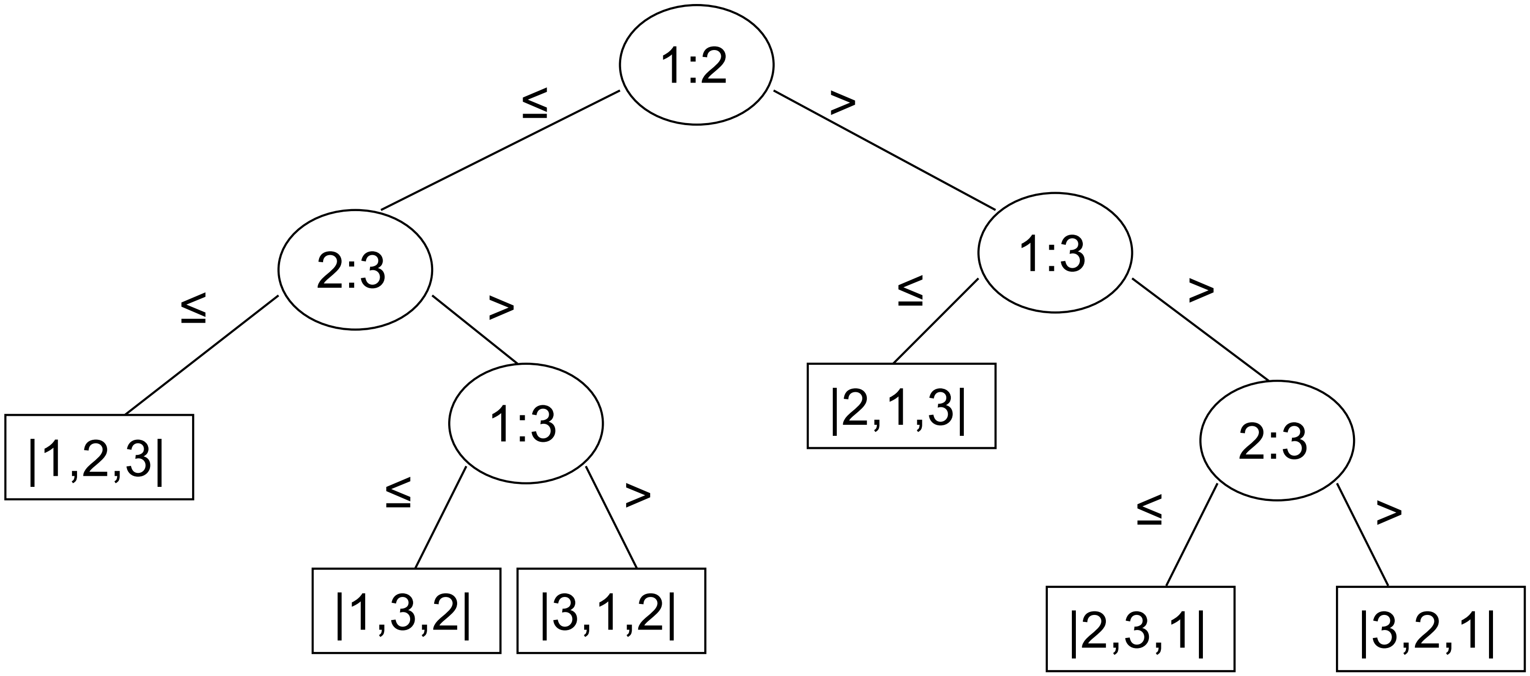
[7, 3, 12] ⇨ | 2,1,3 | ⇨ [3, 7, 12]

- Tracing a path from root to leave gives the correct reordering/permutation of the input for an input

A decision tree model

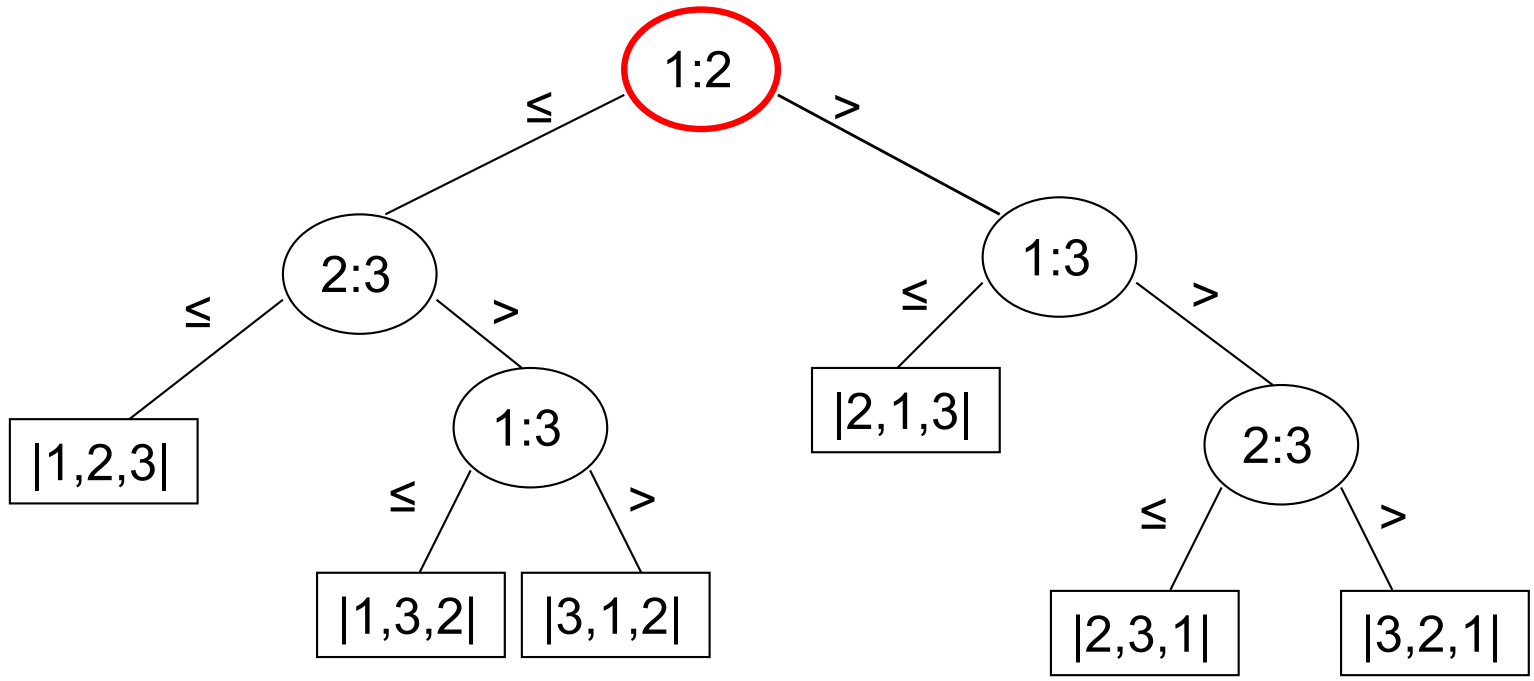


A decision tree model



[12, 7, 3]

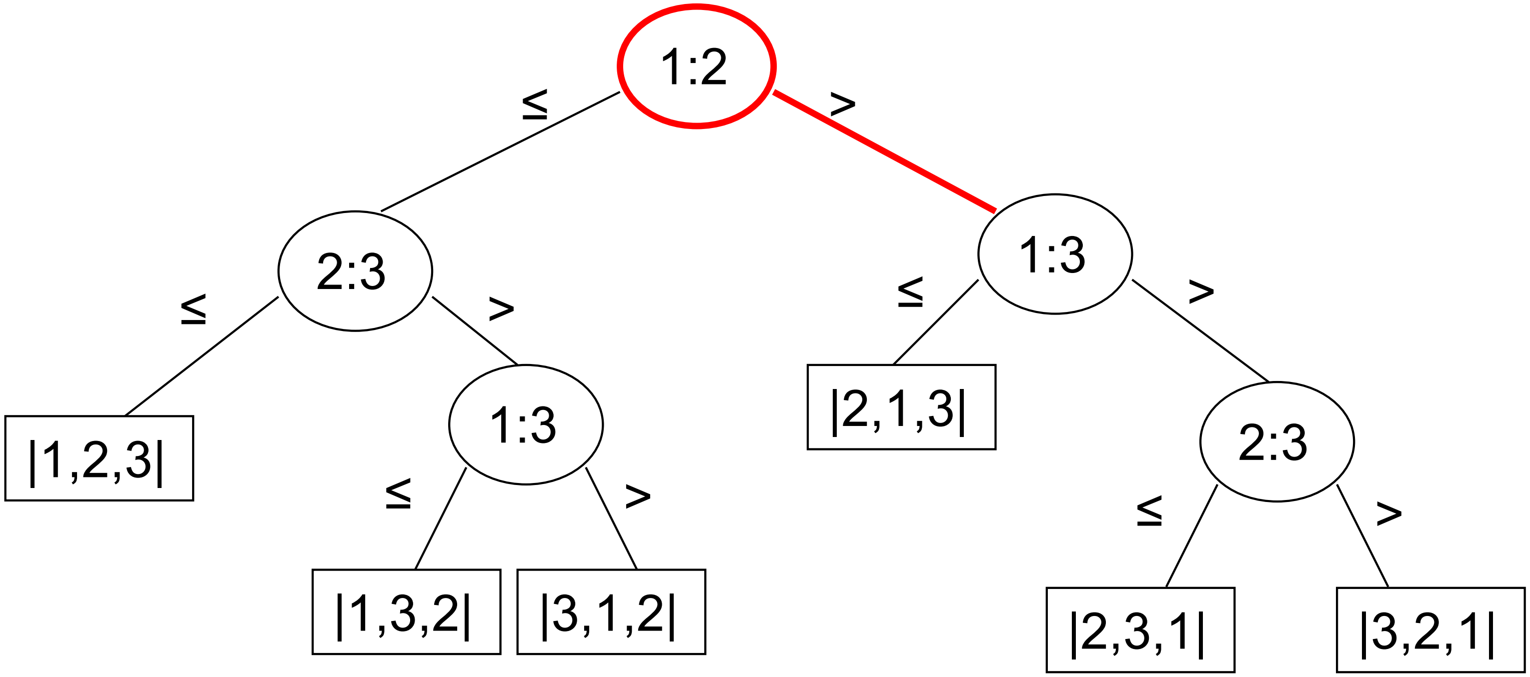
A decision tree model



[12, 7, 3]

Is 12 ≤ 7 or is 12 > 7?

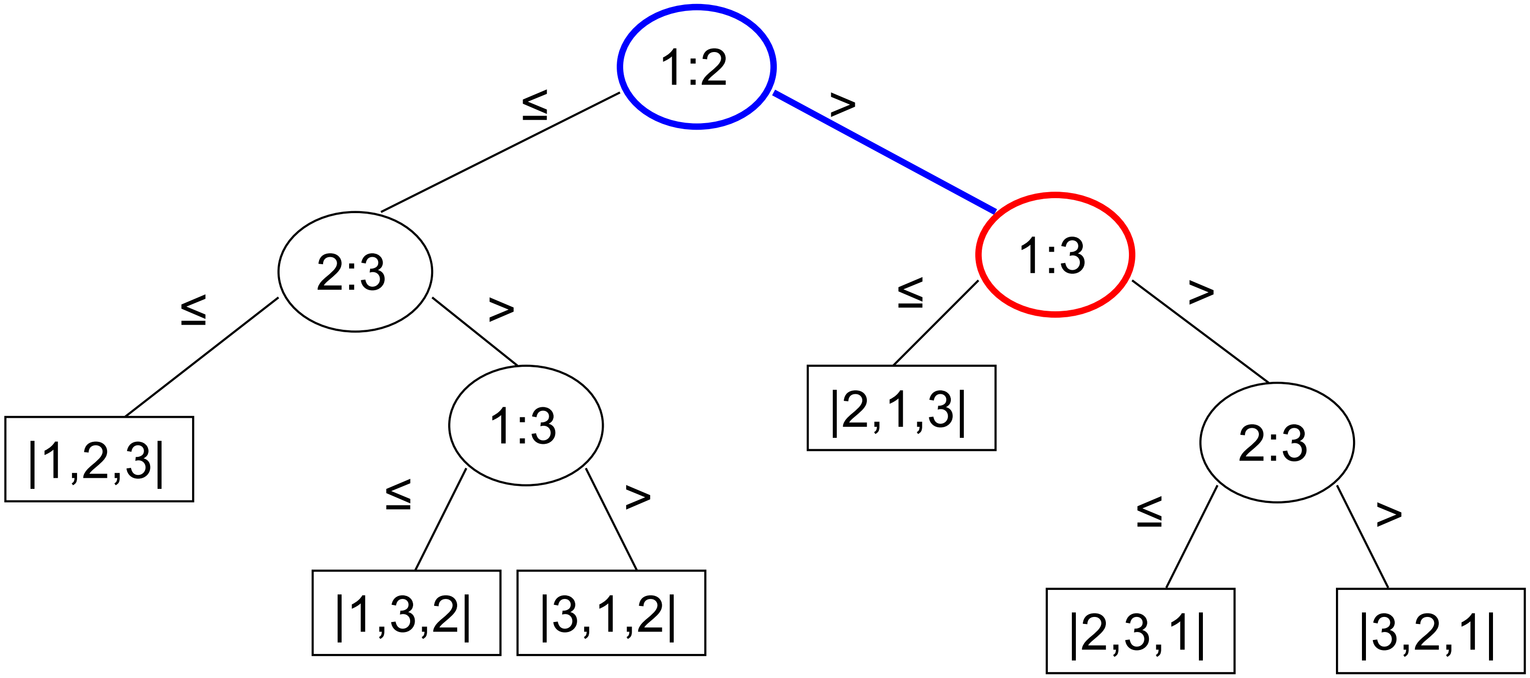
A decision tree model



[12, 7, 3]

Is 12 ≤ 7 or is 12 > 7?

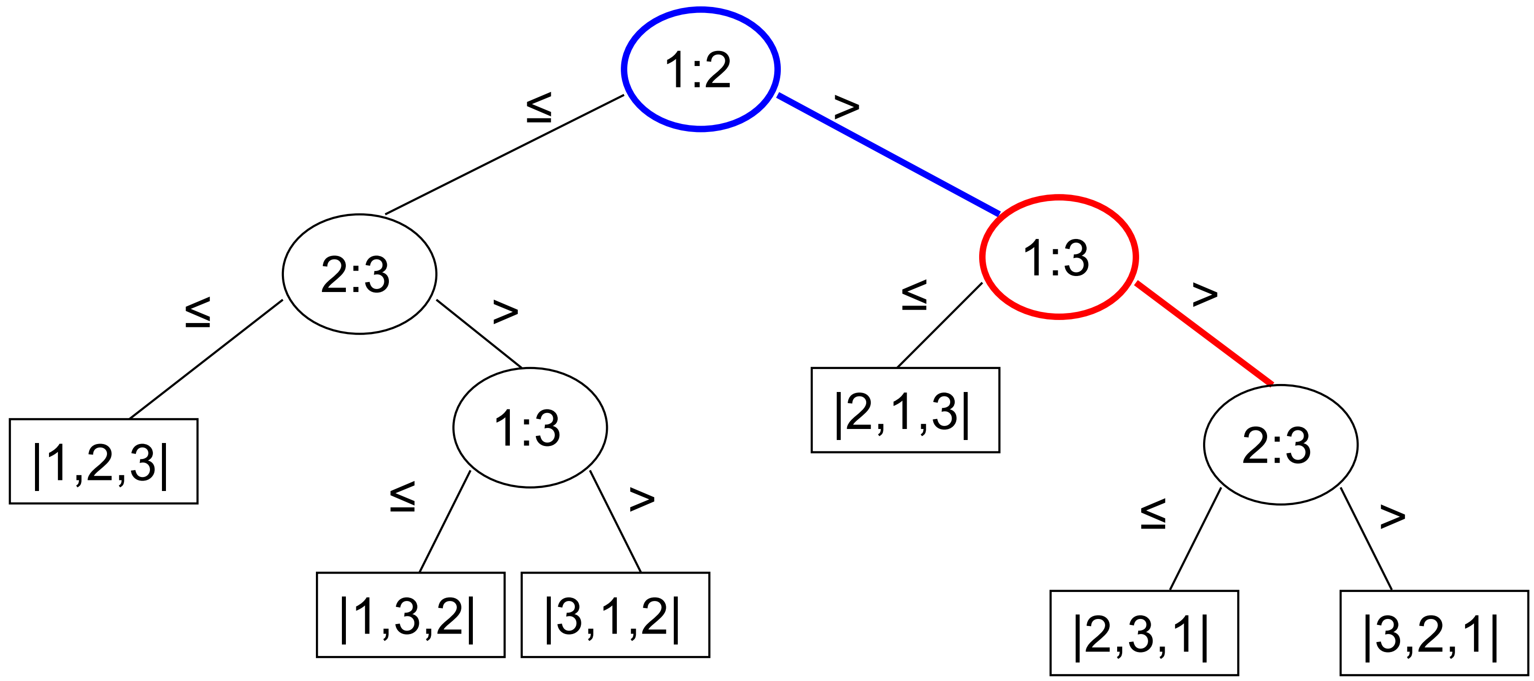
A decision tree model



[12, 7, 3]

Is 12 ≤ 3 or is 12 > 3?

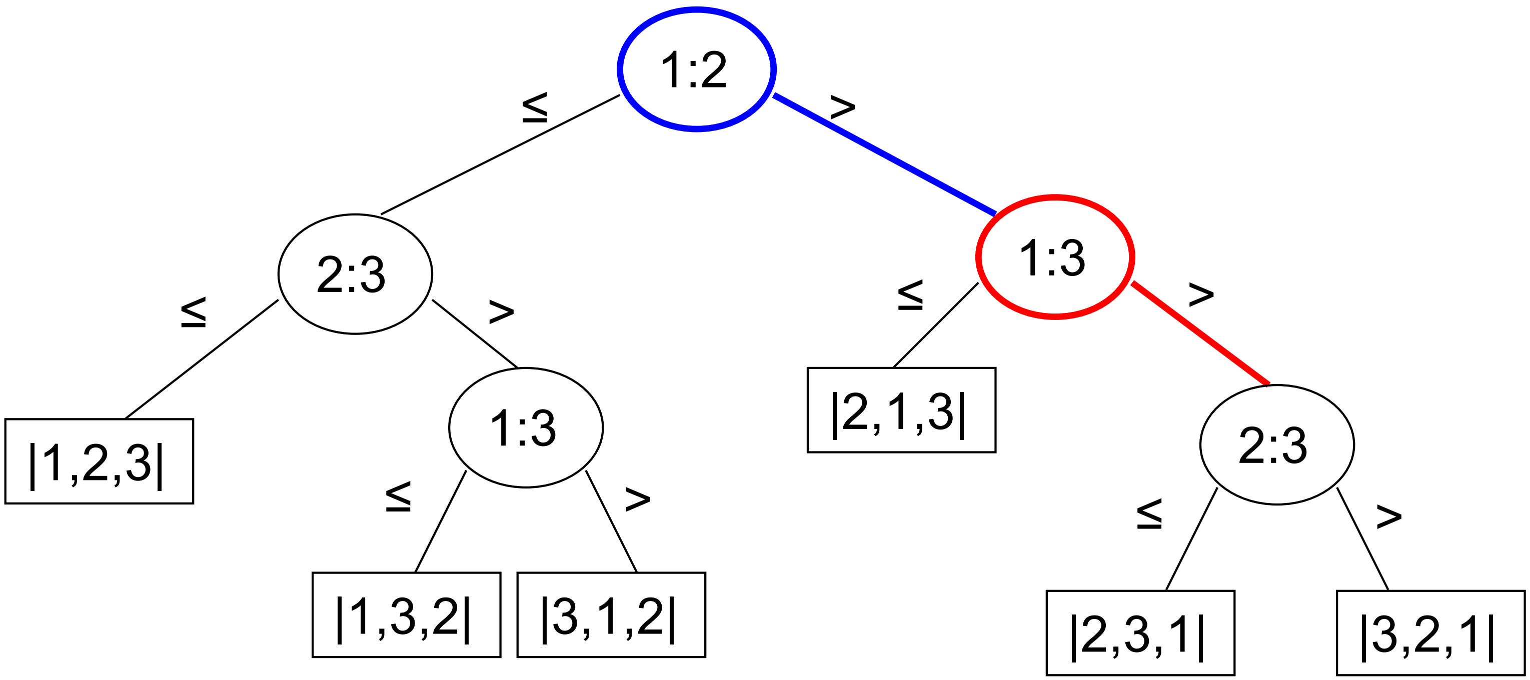
A decision tree model



[12, 7, 3]

Is 12 ≤ 3 or is 12 > 3?

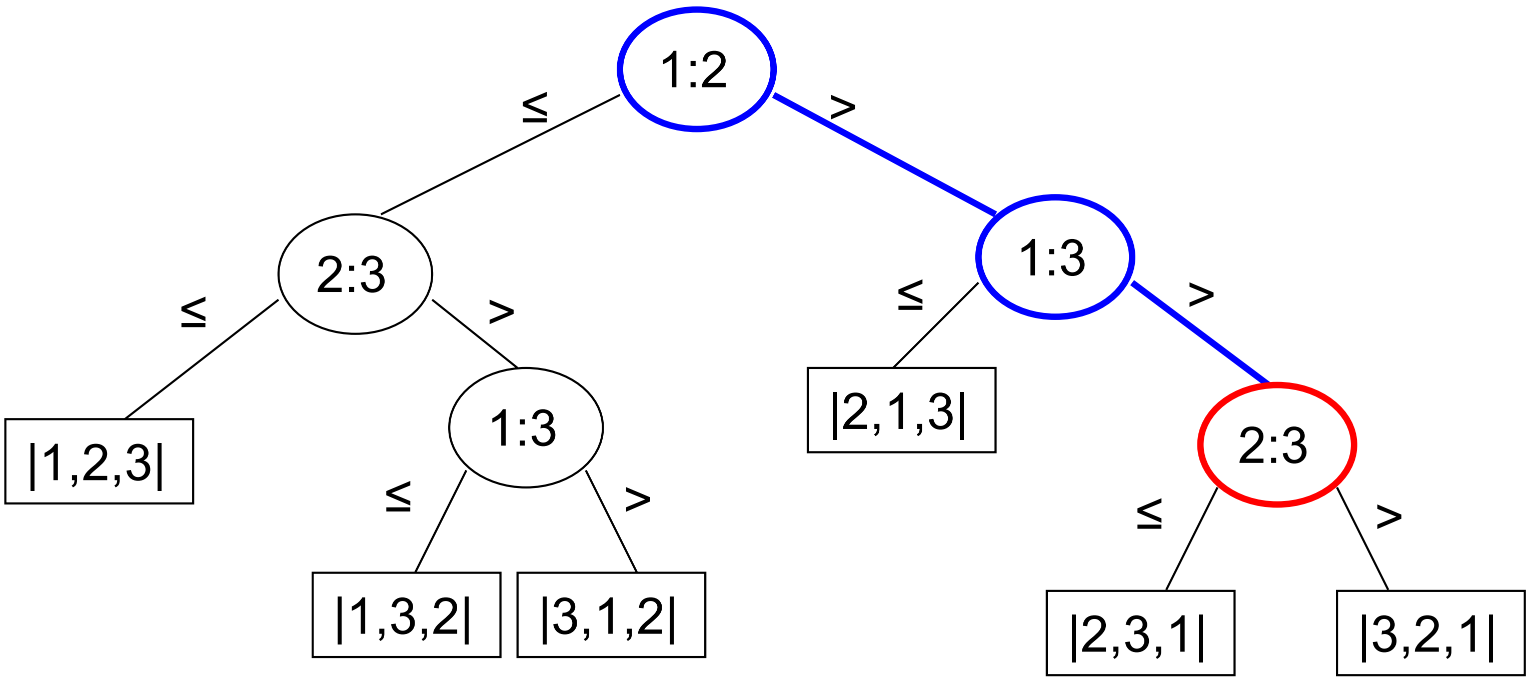
A decision tree model



[12, 7, 3]

Is 12 ≤ 3 or is 12 > 3?

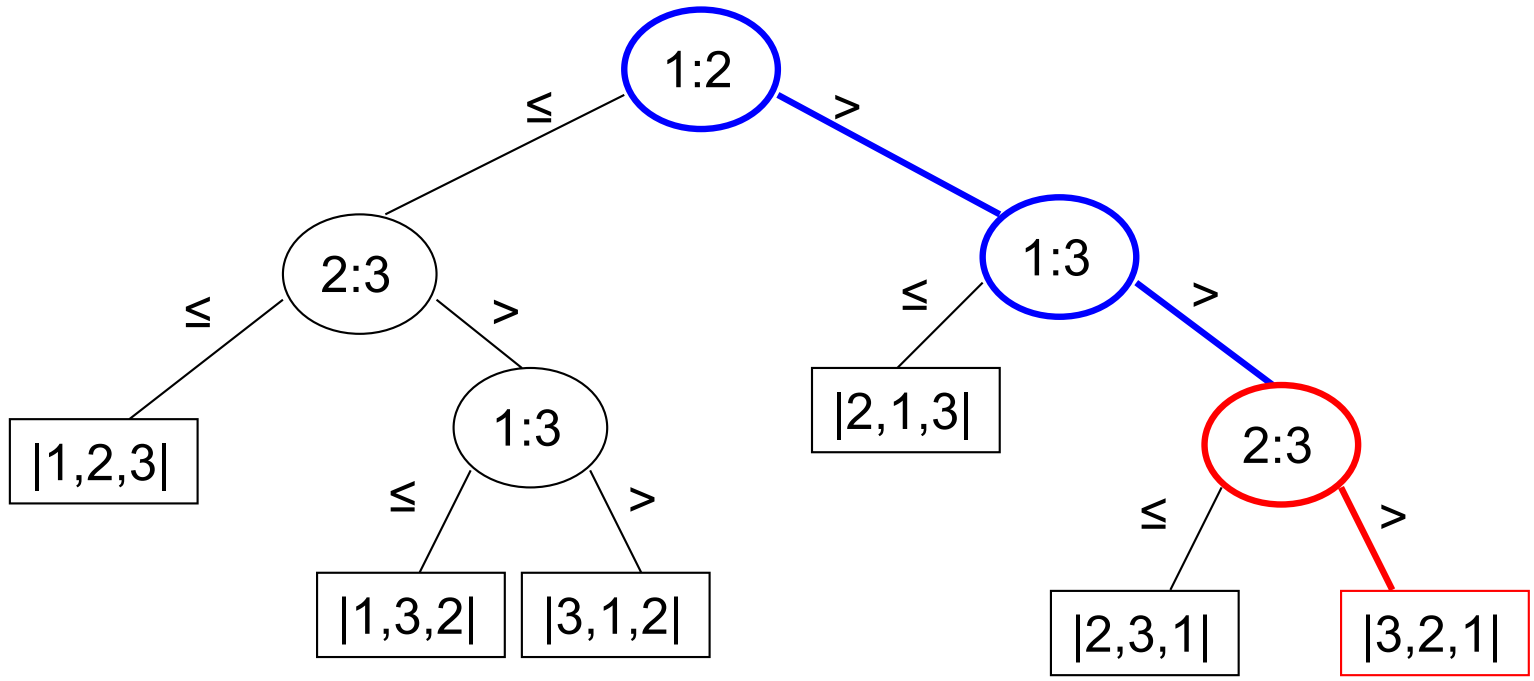
A decision tree model



[12, 7, 3]

Is 7 ≤ 3 or is 7 > 3?

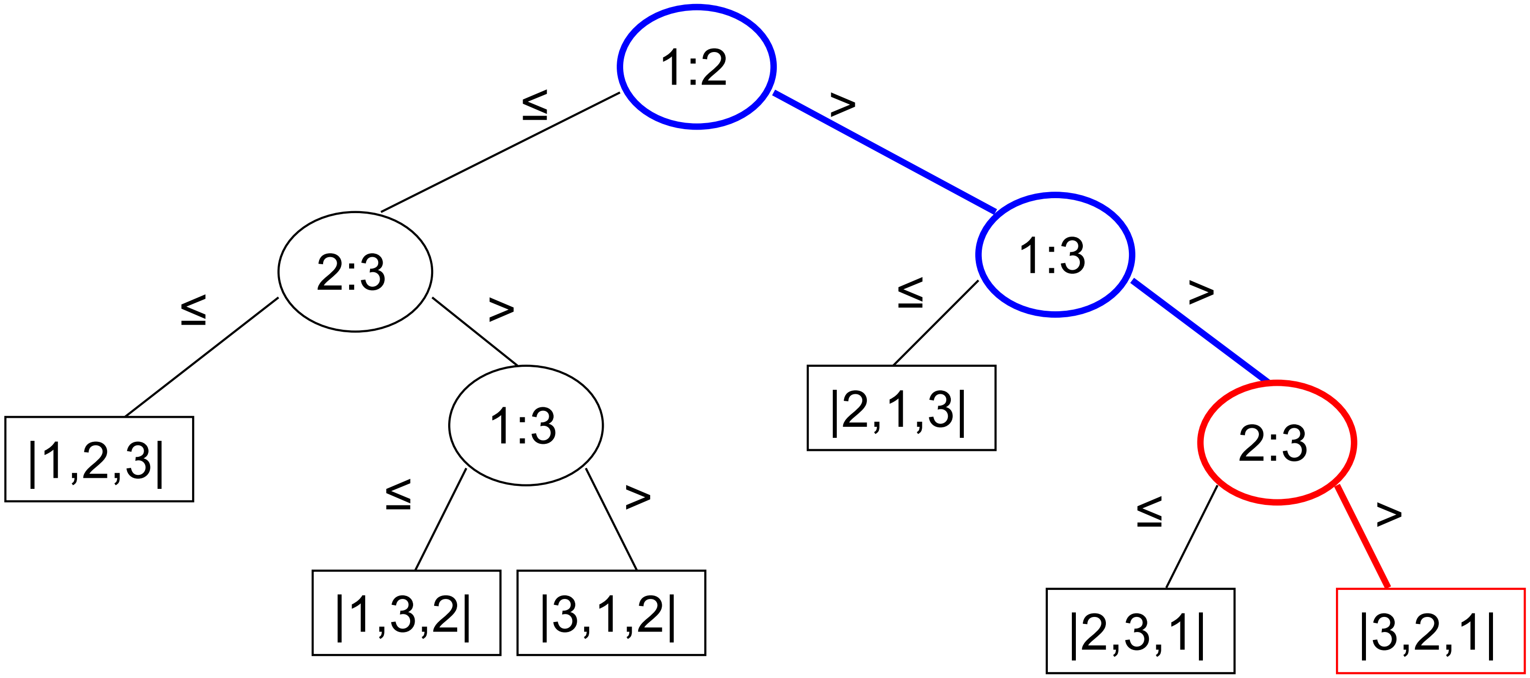
A decision tree model



[12, 7, 3]

Is 7 ≤ 3 or is 7 > 3?

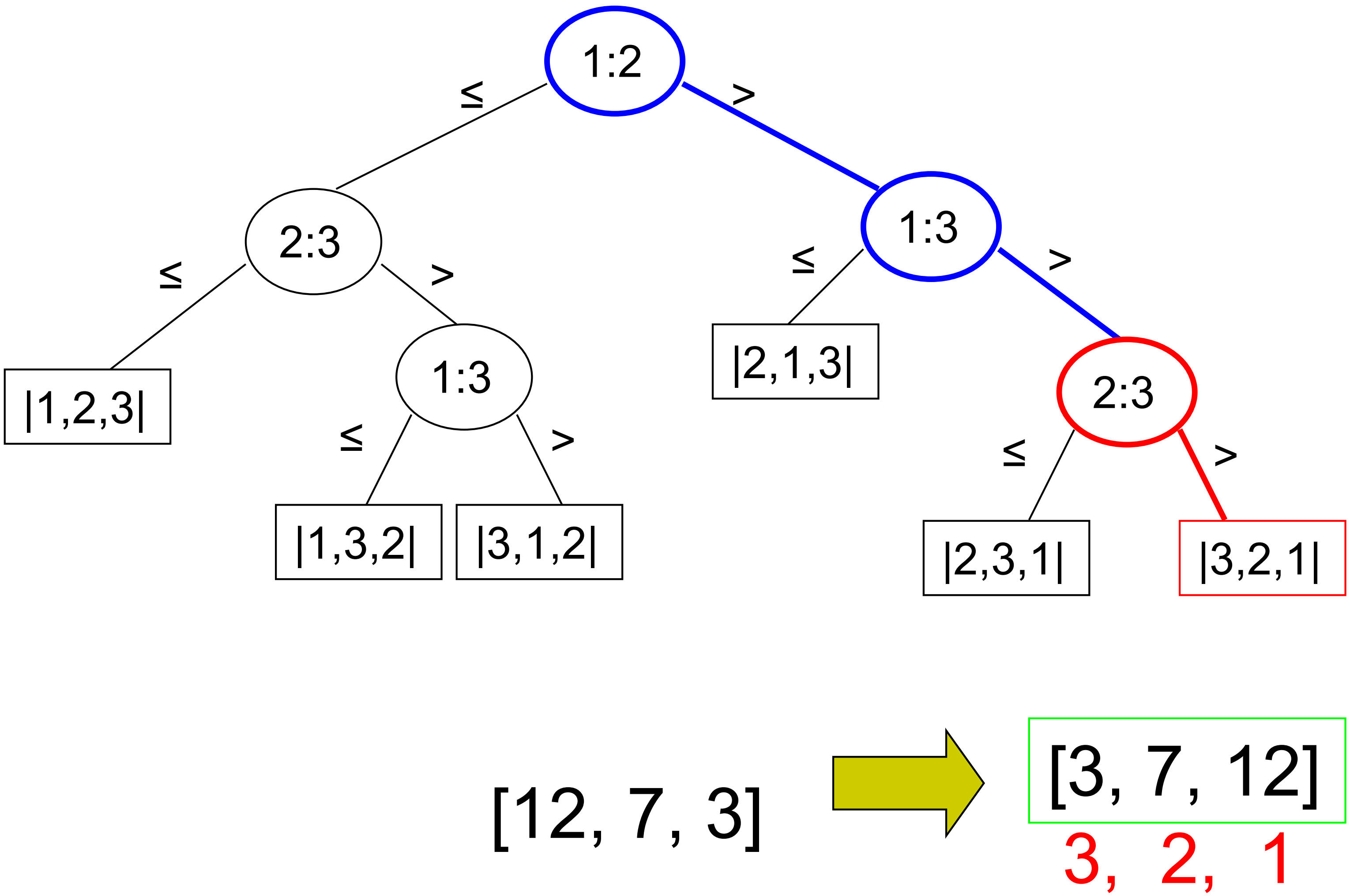
A decision tree model



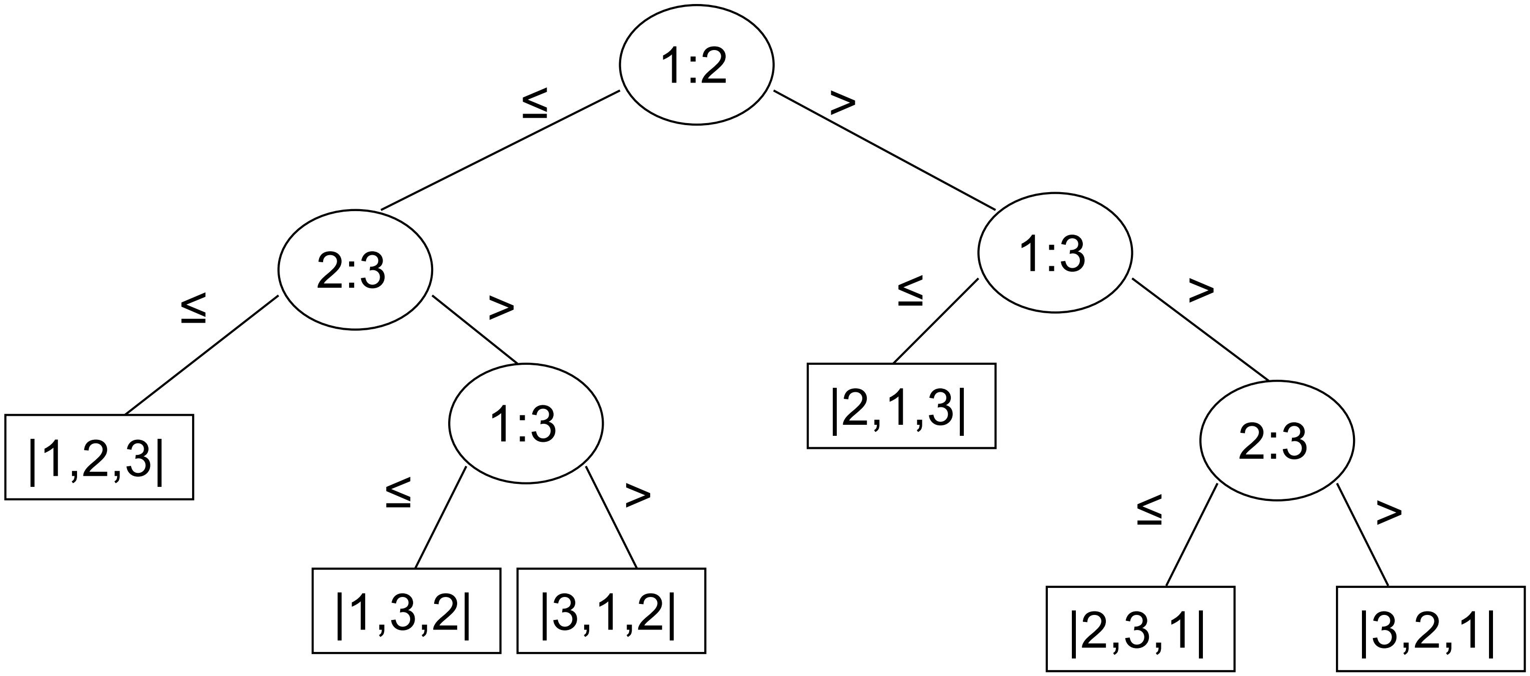
[12, 7, 3]

3, 2, 1

A decision tree model

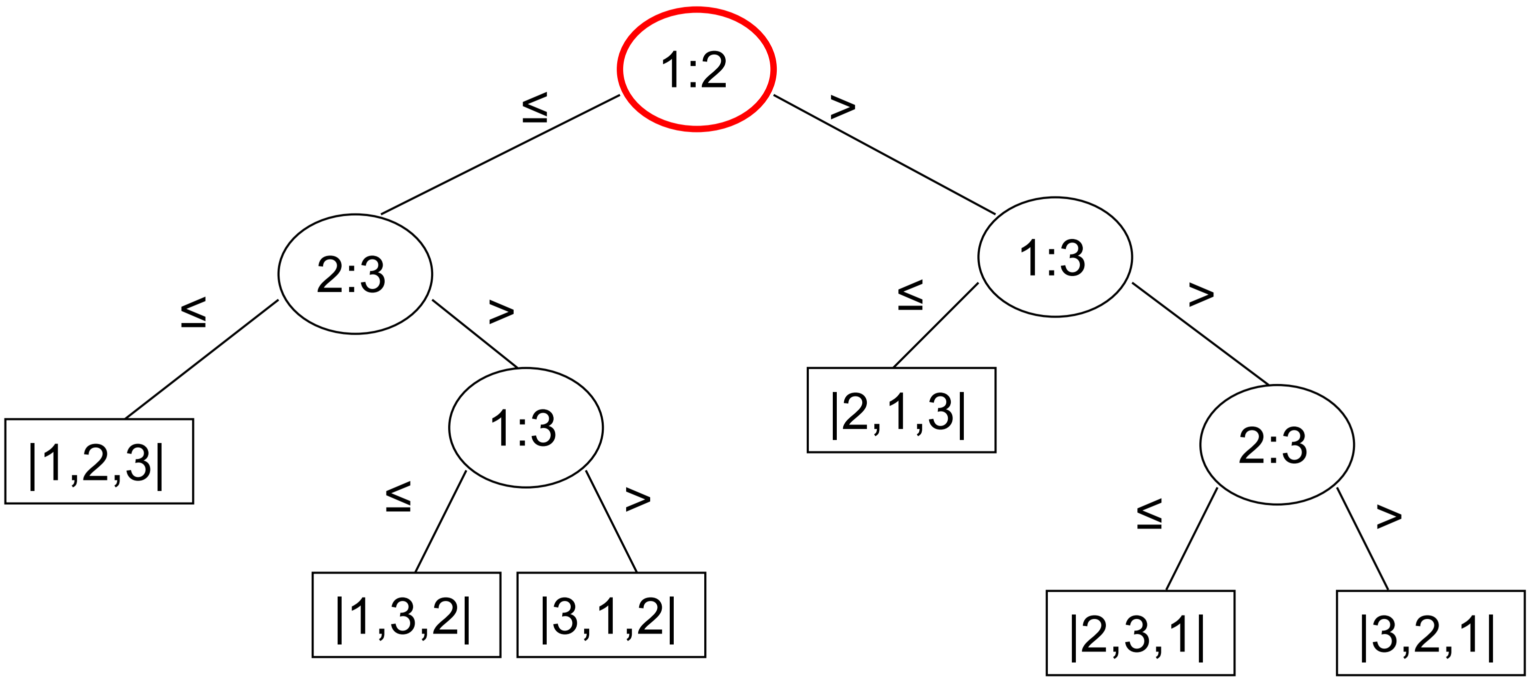


A decision tree model



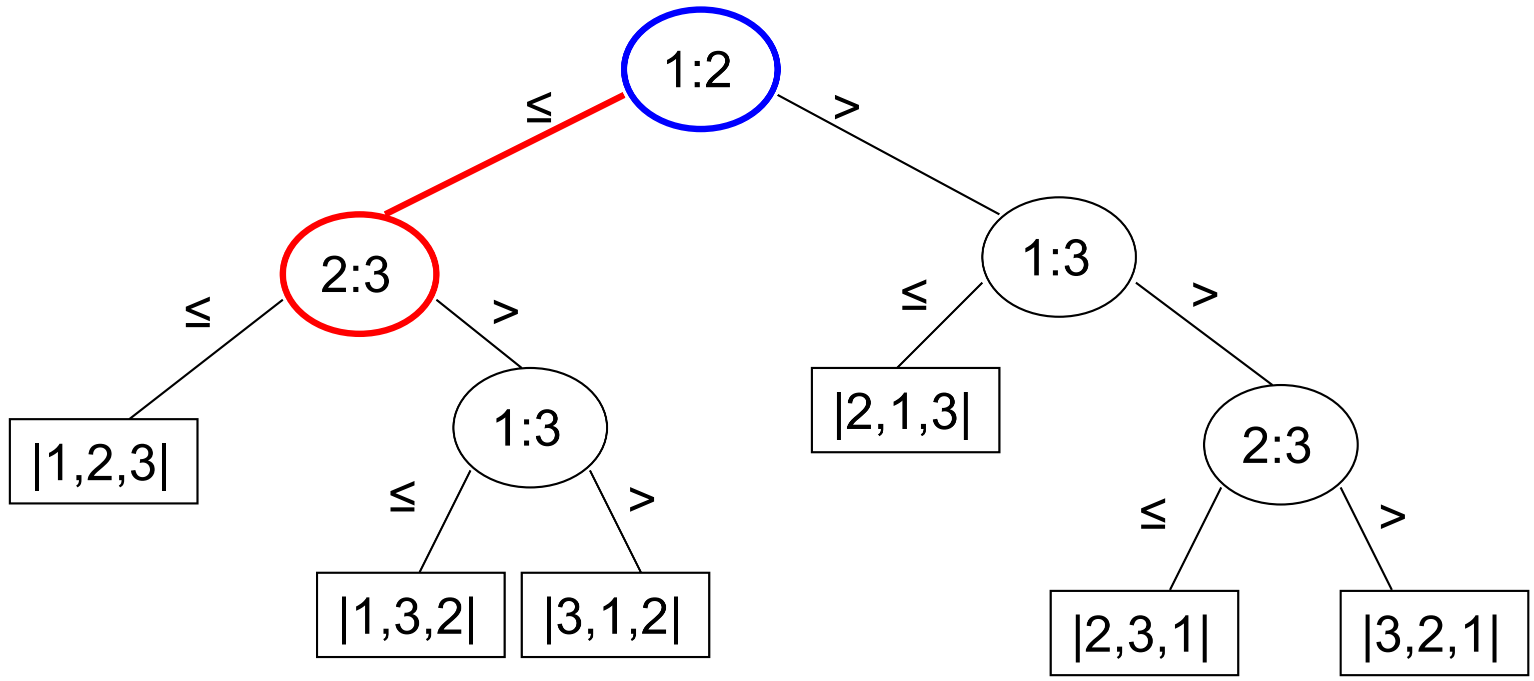
[7, 12, 3]

A decision tree model



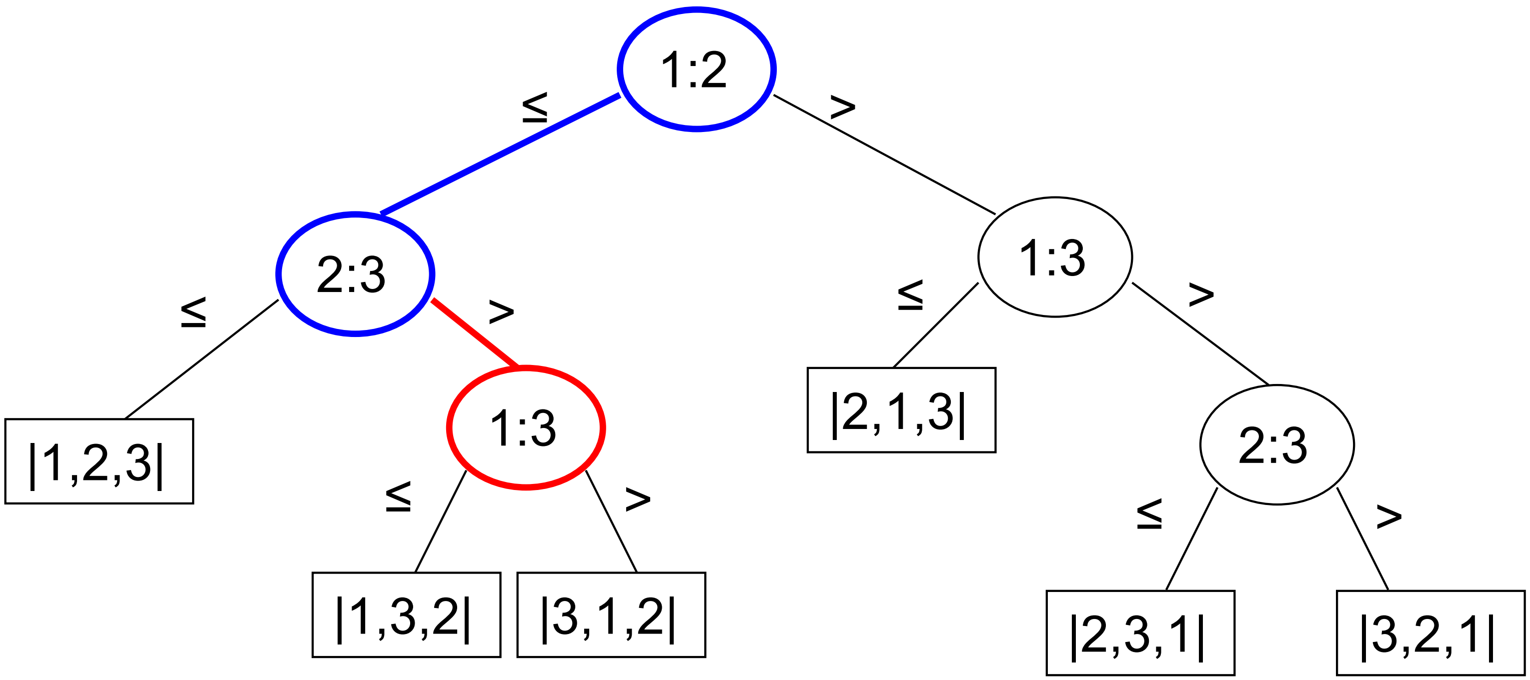
[7, 12, 3]

A decision tree model



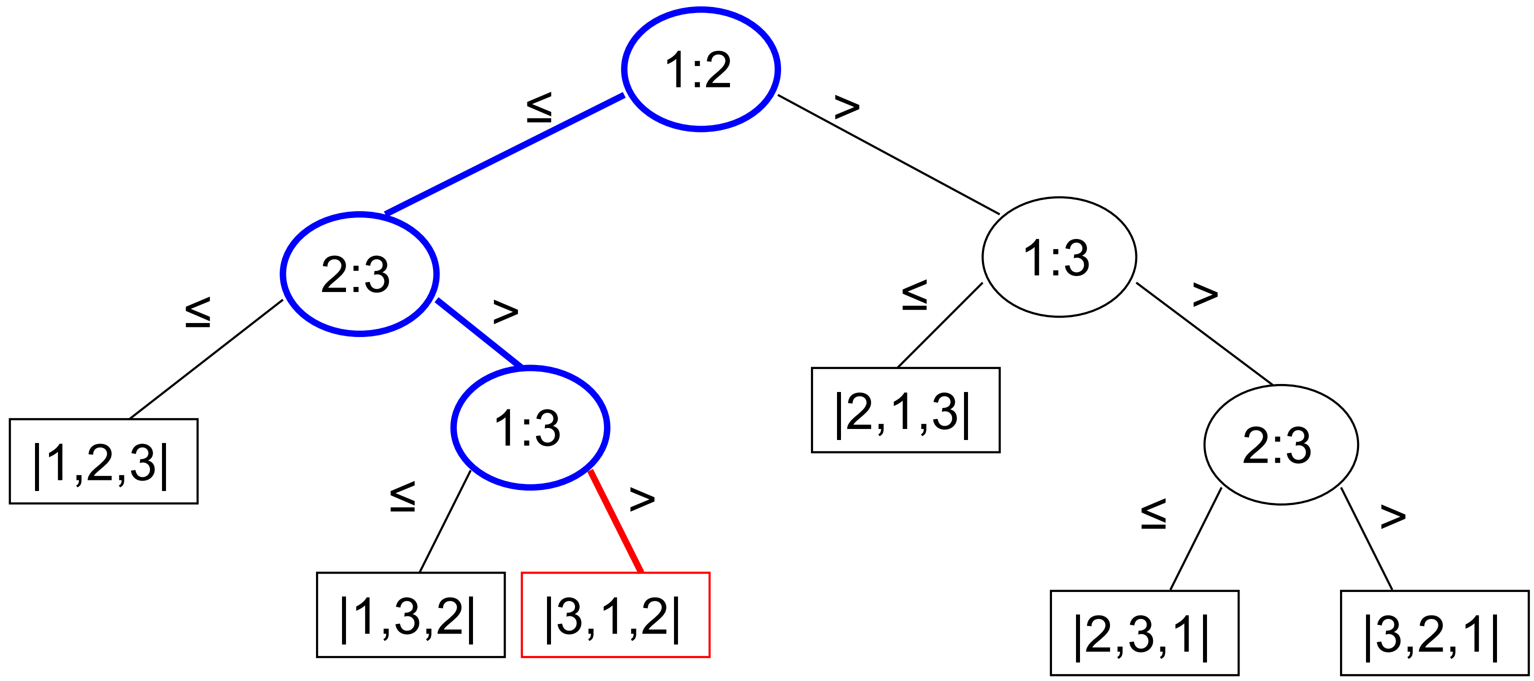
[7, 12, 3]

A decision tree model



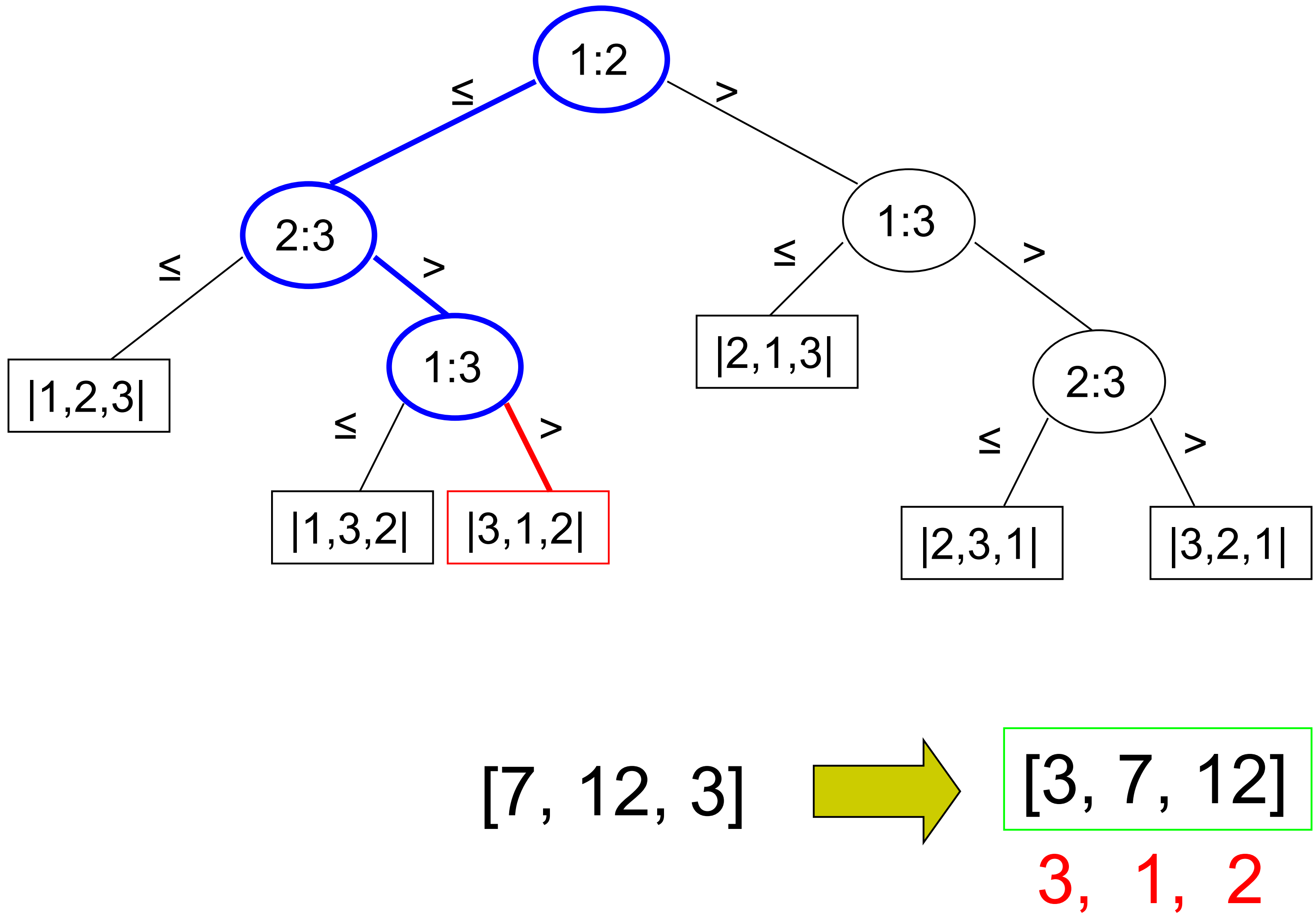
[7, 12, 3]

A decision tree model



[7, 12, 3]

A decision tree model



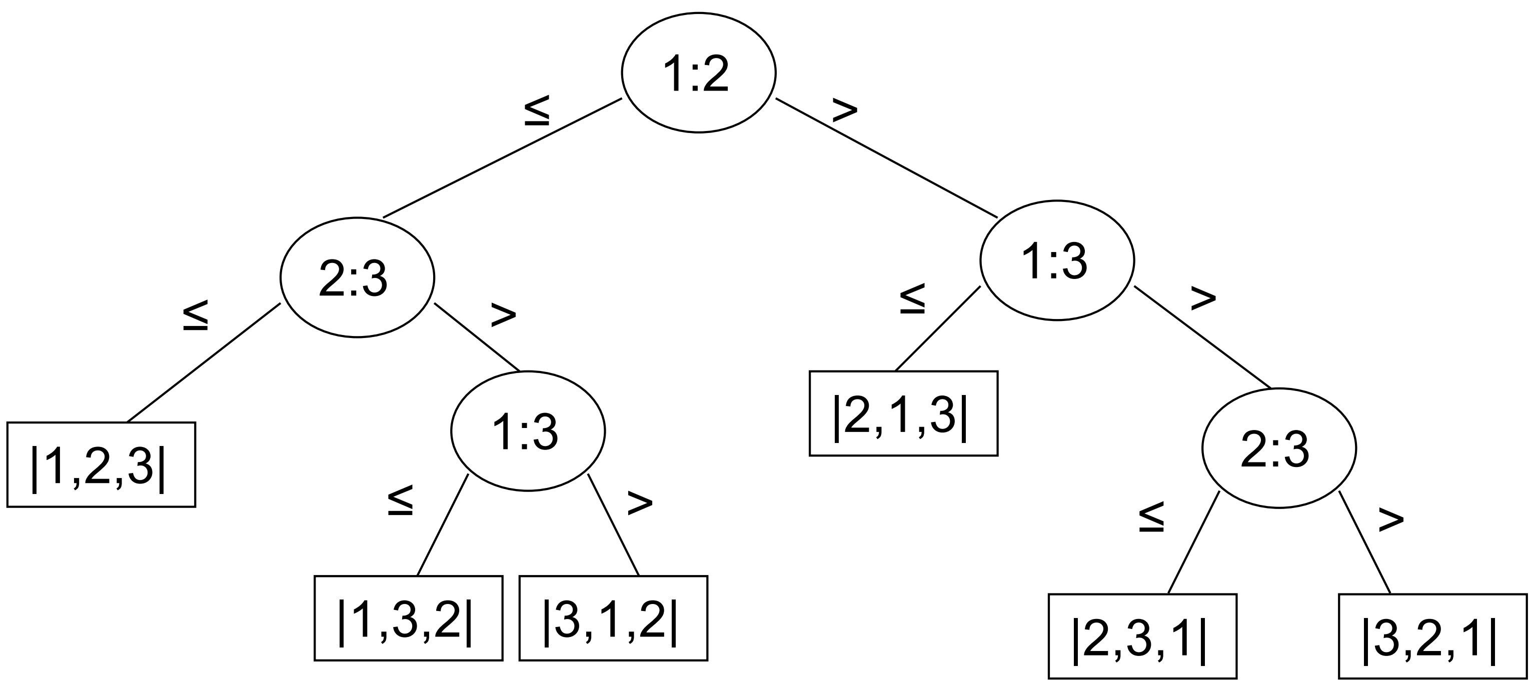
How many leaves are in a decision tree?

Leaves **must** have all possible permutations of the input

Input of size *n*, *n!* leaves

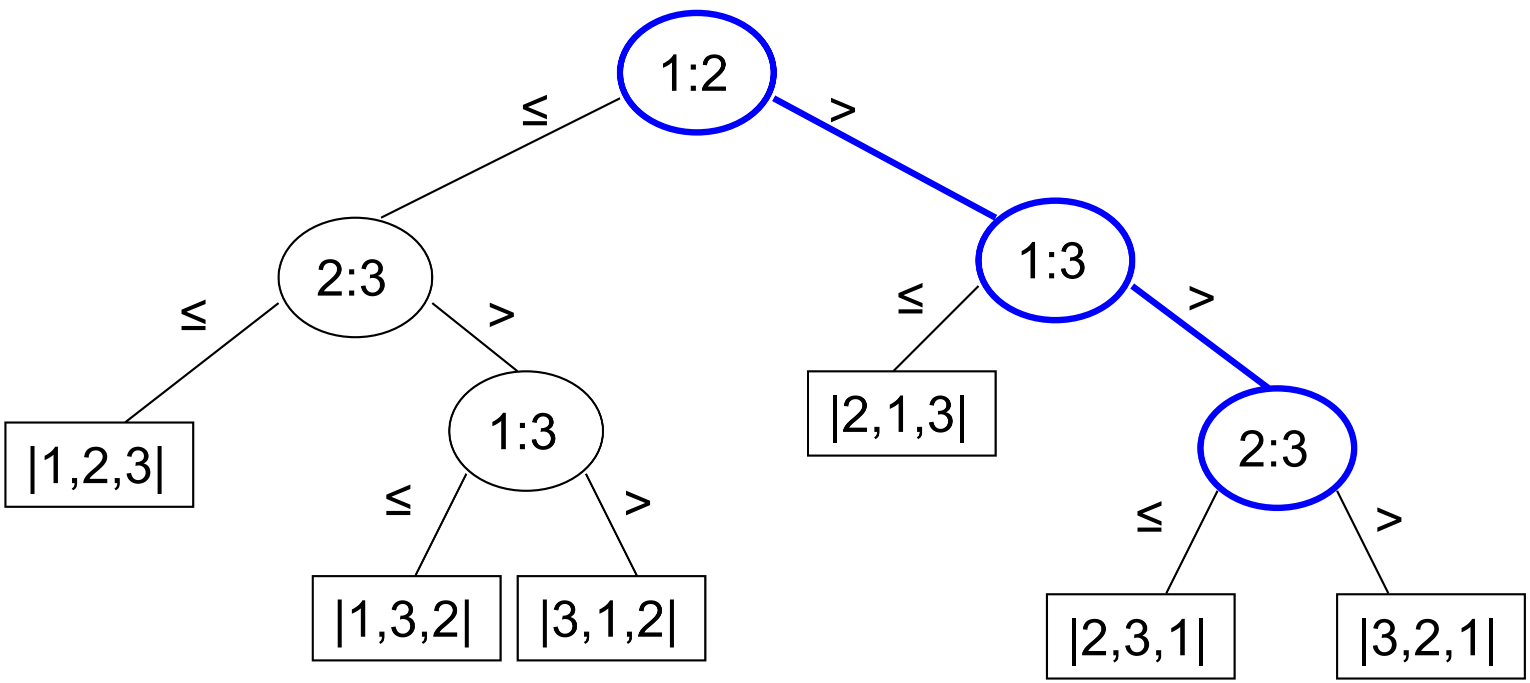
A lower bound

What is the worst-case number of comparisons for a tree?



A lower bound

The longest path in the tree, i.e. the height



A lower bound

What is the maximum number of leaves a binary tree of height *h* can have?

A full binary tree has 2^h leaves

$$2^h \geq n!$$

$$h \geq \log n!$$

Stirling's approximation:

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

Using Stirling's approximation,

$$h = \Omega(n \log n)$$

Can we do better?

Counting Sort

- ***Counting sort*** assumes that each of the n input elements is an integer in the range 0 to k, for some integer k = O(n).

- Runs in $\Theta(n)$ time.

Counting Sort

COUNTING-SORT(A, B, k)

let $C[0\,..\,k]$ be a new array
for $i = 0$ **to** k
    $C[i] = 0$
for $j = 1$ **to** $A.length$
    $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i.
for $i = 1$ **to** k
    $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i.
for $j = A.length$ **downto** 1
    $B[C[A[j]]] = A[j]$
    $C[A[j]] = C[A[j]] - 1$

In the code for counting sort, we assume that the input is an array $A[1\,..\,n]$, and thus $A.length = n$. We require two other arrays: the array $B[1\,..\,n]$ holds the sorted output, and the array $C[0\,..\,k]$ provides temporary working storage.

An Example

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

(a)

	0	1	2	3	4	5
C	2	2	4	7	7	8

(b)

	1	2	3	4	5	6	7	8
B							3	

	0	1	2	3	4	5
C	2	2	4	6	7	8

(c)

	1	2	3	4	5	6	7	8
B		0					3	

	0	1	2	3	4	5
C	1	2	4	6	7	8

(d)

	1	2	3	4	5	6	7	8
B		0				3	3	

	0	1	2	3	4	5
C	1	2	4	5	7	8

(e)

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

(f)

Counting sort is ***stable***: numbers with the same value appear in the output array in the same order as they do in the input array.

Problem

- Can you sort *n* numbers in O(n) time that are in range 0 to n^2-1?

Radix Sort

- Digit by digit sort
- Sort on least significant bit first using counting sort, so on.

329		720		720		329
457		355		329		355
657		436		436		436
839	→	457	→	839	→	457
436		657		355		657
720		329		457		720
355		839		657		839

Can we start sorting from most significant digit?

Radix Sort

- Each element in the n-element array A has d digits, where digit 1 is the lowest-order digit and d is the highest-order digit.

Pseudo code:

```
RADIX-SORT(A, d)
for i = 1 to d
    use a stable sort to sort array A on digit i
```

Complexity:

Using counting sort in each iteration takes O(n) time. Thus radix sort takes O(dn) time. When d is constant, the time is O(n).

Bucket Sort

- *A*ssumes that the input is drawn from a uniform distribution from a fixed interval.
- Divides into *n* equal sized buckets (subintervals)
  - Puts each value in a bucket
  - Sort each bucket
  - Concatenate the buckets.
- Average-case running time of O(n).
- Worst case time is O(n^2).