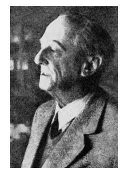
Comparison Sorting Lower Bounds

What is the theoretically fastest possible sort speed?

Comparison sorting algorithms sort arrays by comparing elements (with no extra information)

Examples: Insertion sort | Heapsort | Quicksort

H. Steinhaus considered a sorting puzzle - ranking all players in tennis tournaments (1939) 

L. Ford and S. Johnson introduced merge-insertion sort with decision tree analysis (1959)  

Runtime complexity results for comparison sorts

Lower bound for worst input: $\rightarrow \Omega(n \log n)$

Lower bound for average input: $\rightarrow \Omega(n \log n)$

$\Longrightarrow O(n \log n)$ sorts such as heapsort are asymptotically optimal

Note: **non-comparison** sorts (e.g. Radix sort) can do better

References/Notes/Image credits:
H. Steinhaus, "Mathematical Snapshots" (1939)
(Image of Steinhaus) https://en.wikipedia.org/wiki/Hugo_Steinhaus#/media/File:Hugo_Steinhaus.jpg
(Bumps photo) https://www.flickr.com/photos/stanbury/42283400832/in/photostream/

L. Ford and S. Johnson, "A tournament problem", The American Mathematical Monthly (1959)
(Image of Ford) https://en.wikipedia.org/wiki/Lester_R._Ford#/media/File:Lester_R._Ford.gif
(Image of Johnson) https://alchetron.com/Selmer-M-Johnson

Comparison Sorting Algorithms and Decision Trees

A comparison sort ingests an array $[a_0, \ldots, a_{n-1}]$

It only gains information by comparing pairs:

"is $a_i < a_j$"?

It outputs a permutation that orders the items

For our analysis, assume all elements are distinct

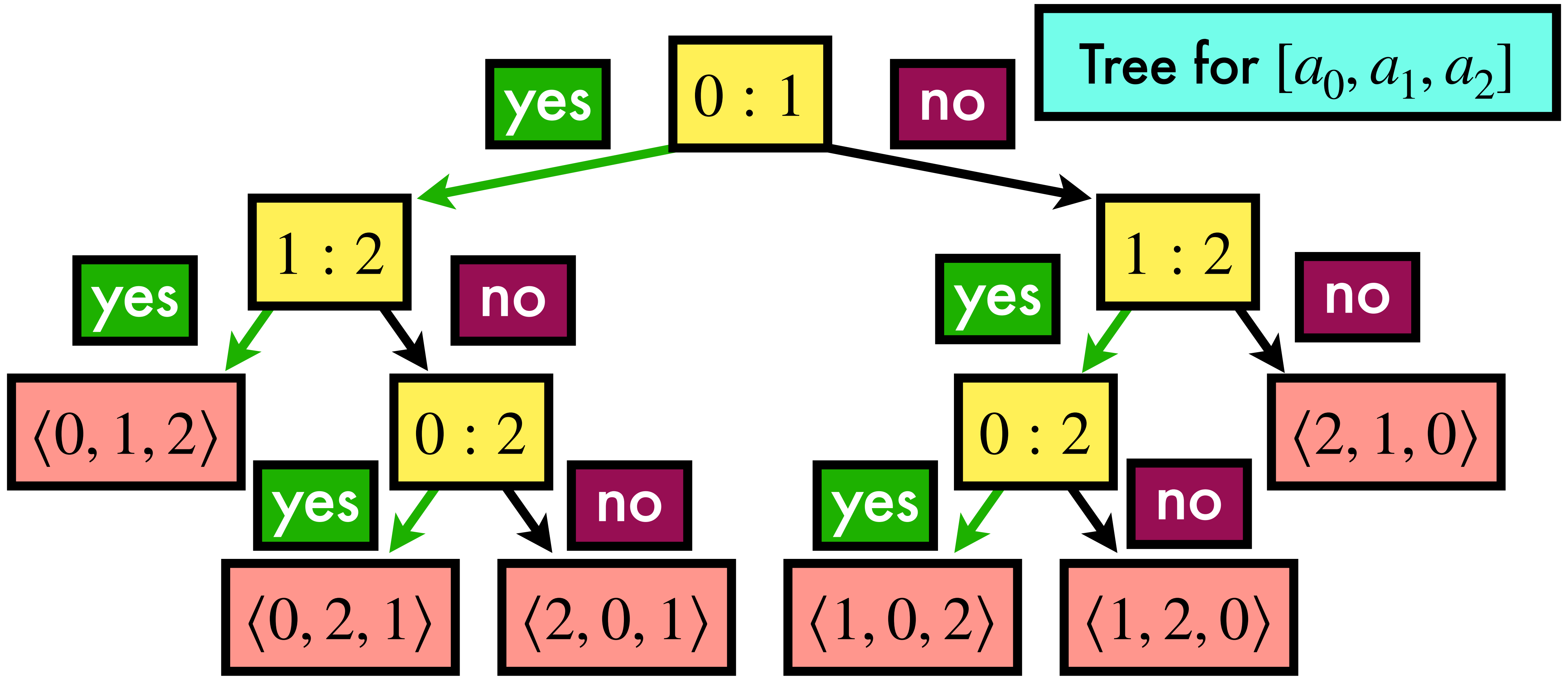
(repeated elements won't affect lower bounds)

Decision trees (or "Comparison trees")

Full binary trees where each internal node

$i : j$ represents a comparison $a_i < a_j$

Each leaf $\langle \pi(0), \ldots, \pi(n-1) \rangle$ is an array permutation



A comparison sort follows a path from root to leaf

References:
A. Blum and M. Blum, "Comparison-based Lower Bounds for Sorting", https://www.cs.cmu.edu/~avrim/451f11/lectures/lect0913.pdf
D. E. Knuth, "The art of computer programming, vol. 3: sorting and searching", Chap 5.3 (1998)
T. Cormen et al., "Introduction to algorithms", Chap 8, MIT press (2022)
J. Erickson, "Lower Bounds", https://jeffe.cs.illinois.edu/teaching/algorithms/notes/12-lowerbounds.pdf (2018)

Note: in a "full" binary tree, every node has either 0 or 2 children.

Lower Bound On The Worst Case

There are $n!$ possible permutations (without repetitions) i.e. ways to sort the input

A correct comparison sorting algorithm must be able to produce all of these

$\implies$ a comparison sort corresponds to a full binary tree with $\geq n!$ leaves

Worst case number of comparisons - length of longest simple path from root to leaf

What is the maximum height of a decision tree?

A binary tree with height h has at most 2^h leaves, so $n! \leq 2^h$

Stirling's approximation: $n! = \left(\frac{n}{e}\right)^n \sqrt{2\pi n}\left(1 + \Theta\left(\frac{1}{n}\right)\right) \geq \left(\frac{n}{e}\right)^n$

$$h \geq \log(n!) \geq \log\left(\frac{n}{e}\right)^n = n\log n - n\log e = \Omega(n\log n)$$

No comparison sort can be faster than this on its worst case input

References:
A. Blum and M. Blum, "Comparison-based Lower Bounds for Sorting", https://www.cs.cmu.edu/~avrim/451f11/lectures/lect0913.pdf
D. E. Knuth, "The art of computer programming, vol. 3: sorting and searching", Chap 5.3 (1998)
T. Cormen et al., "Introduction to algorithms", Chap 8, MIT press (2022)
J. Erickson, "Lower Bounds", https://jeffe.cs.illinois.edu/teaching/algorithms/notes/12-lowerbounds.pdf (2018)

Lower Bound On The Average Case

What is the smallest possible average height of a decision tree?

The average height is smallest when the tree is completely balanced

Completely balanced means no leaf heights differ by more than 1

If the tree is completely balanced, each leaf has depth $\lceil \log(n!) \rceil$ or $\lfloor \log(n!) \rfloor$

$\Longrightarrow$ smallest possible average height is $\Omega(n \log n)$

No comparison sort can be faster than this on average

References:
A. Blum and M. Blum, "Comparison-based Lower Bounds for Sorting", https://www.cs.cmu.edu/~avrim/451f11/lectures/lect0913.pdf
D. E. Knuth, "The art of computer programming, vol. 3: sorting and searching", Chap 5.3 (1998)