CSC 226

Algorithms and Data Structures: II
Sorting Lower Bound

Tianming Wei
twei@uvic.ca
ECS 466

Comparison-based Sorting

A Comparison of Sorting Algorithms

	Worst case	**Average Case**
Selection Sort	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n\ logn)$	$O(n\ logn)$
Quick Sort	$O(n^2)$	$O(n\ logn)$
Heap Sort	$O(n\ logn)$	$O(n\ logn)$

Comparison-based Sorting

Can we do better than $\Theta(n\ logn)$?

How fast can we sort?

Can we say that it is impossible to sort faster than $\Omega\,(n\,logn)$ in the worst case?

If we could prove it, then $\Omega\,(n\,logn)$ denotes the lower bound for comparison based sorting.

Lower Bound is $\Omega(n\ logn)$

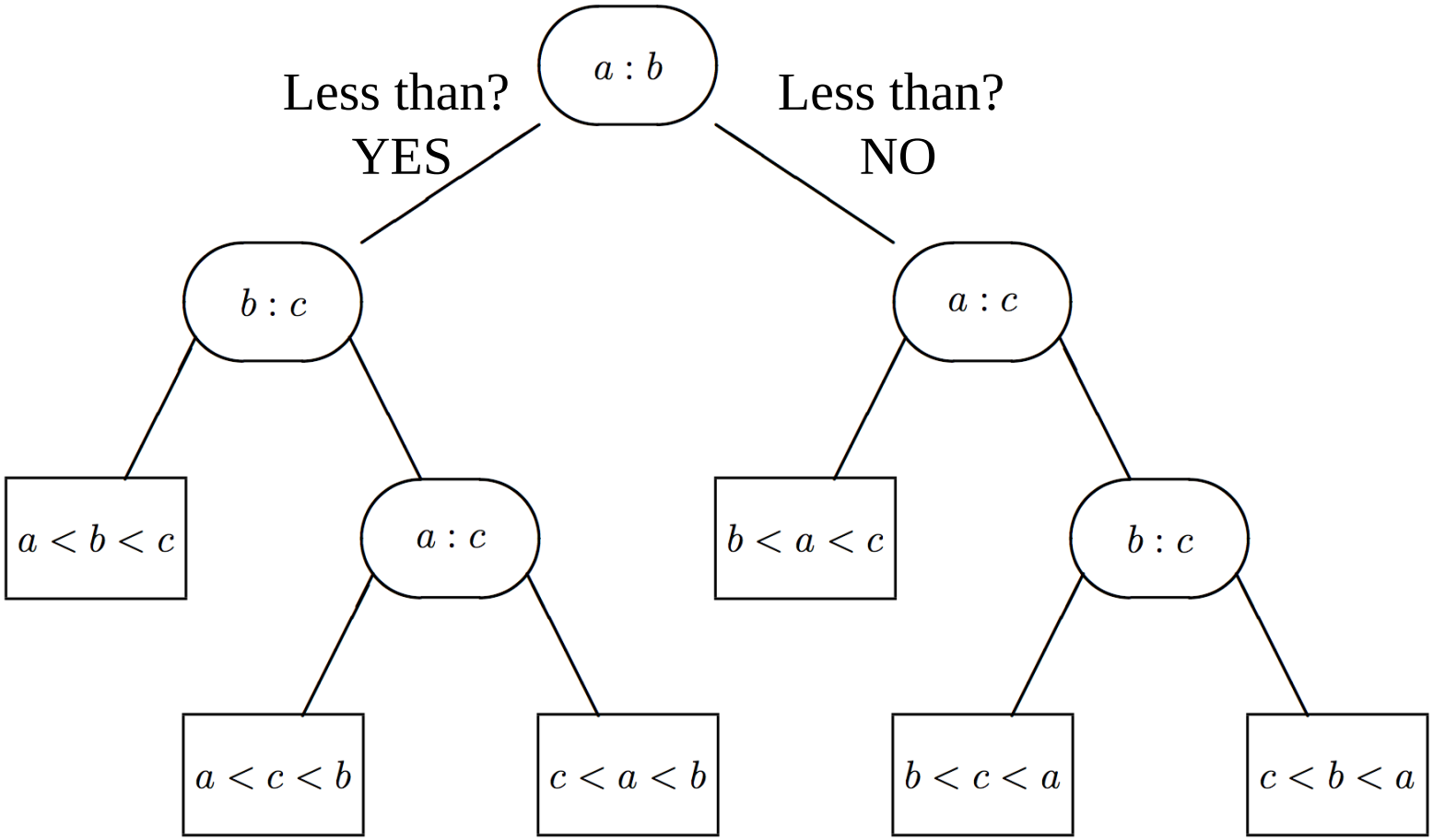
Let's start from an example with 3 elements.

Sort([a, b, c])

Then build a Tree with the

Comparison (or Decision Tree) Model

Lower Bound is $\Omega(n\ logn)$

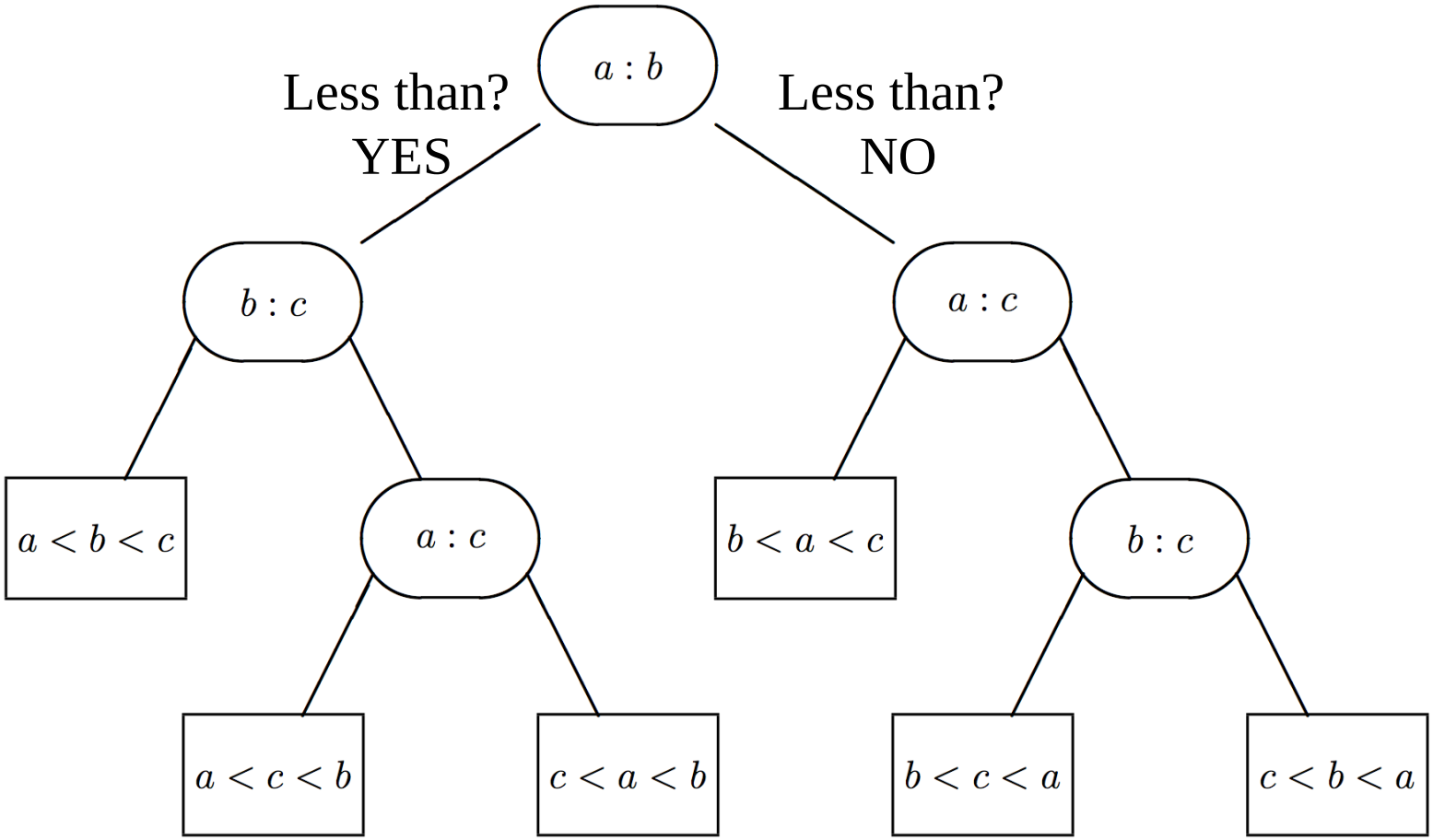


Lower Bound is $\Omega(n\ logn)$

Worst case number of comparisons performed corresponds to maximal height of tree.

Therefore, lower bound on height ⇒ lower bound on sorting.

Lower Bound is $\Omega(n\ logn)$

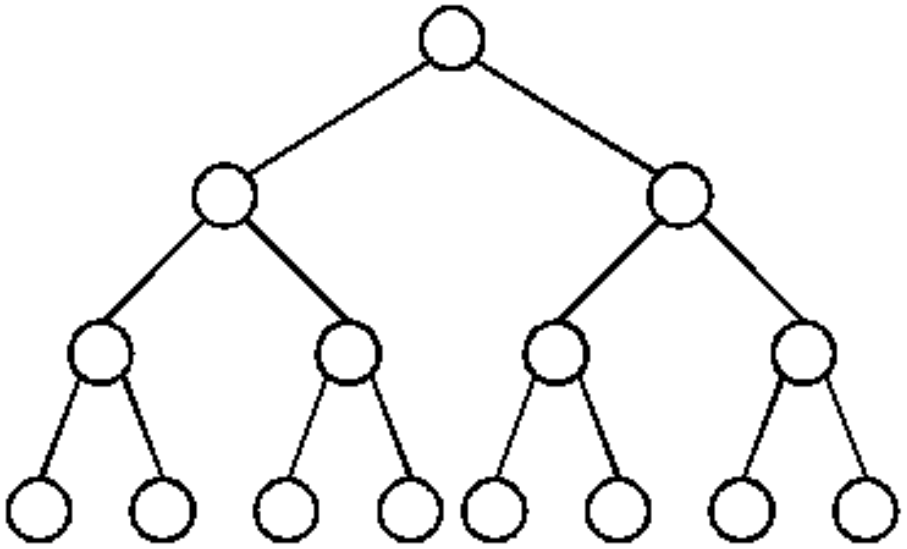


Lower Bound is $\Omega(n\ logn)$

- Assume elements are the (distinct) numbers 1 through n
- There must be $n!$ leaves
  - One for each of the $n!$ permutations of n elements
- And we know:
  - Binary Tree of height h has at most 2^h leaves

Lower Bound is $\Omega(n\ logn)$

Binary Tree of height h has at most 2^h leaves



2^0

2^1

2^2

2^3

Lower Bound is $\Omega(n\ logn)$

$$2^h \geq n! \Rightarrow h \geq \log_2(n!)$$

AND

$\log(n!)$ is $\Omega(n \log n)$

Lower Bound is $\Omega(n\ logn)$

$$\log(n!) = \log(n) + \log(n-1) + \cdots + \log(2)$$
$$= \sum_{i=2}^{n} \log(i)$$
$$\leq \sum_{i=2}^{n} \log(n)$$
$$= \Omega(n \log n)$$

Right?

Lower Bound is $\Omega(n\ logn)$

$$\log(n!) = \log(n) + \log(n-1) + \cdots + \log(2)$$

$$= \sum_{i=2}^{n} \log(i)$$

$$\leq \sum_{i=2}^{n} \log(n)$$

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

$$= O(n \log n)$$

Lower Bound is $\Omega(n\ logn)$

$$\log(n!) = \log(n) + \log(n-1) + \cdots + \log(2)$$

$$= \sum_{i=2}^{n} \log(i)$$

$$= \sum_{i=2}^{\frac{n}{2}-1} \log(i) + \sum_{i=n/2}^{n} \log(i)$$

$$\geq \quad 0 \quad + \sum_{i=n/2}^{n} \log(n/2)$$

$$= \frac{n}{2} \log\left(\frac{n}{2}\right)$$

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

Lower Bound is $\Omega(n\ logn)$

AND now you know

$\log(n!)$ is $\Theta(n \log n)$

More About Quicksort

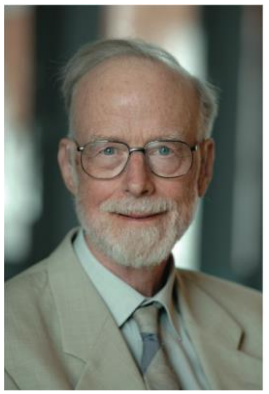
Tony Hoare

- Invented quicksort to translate Russian into English. (1959)

  [but couldn't explain his algorithm or implement it!]

- Learned Algol 60 (and recursion).
- Implemented quicksort. (1961)

" There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."



Tony Hoare
1980 Turing Award

More About Quicksort

Bob Sedgewick

- Refined and popularized quicksort.
- Analyzed many versions of quicksort.

Bob Sedgewick

Programming Techniques — S. L. Graham, R. L. Rivest Editors

Implementing Quicksort Programs

Robert Sedgewick
Brown University

This paper is a practical study of how to implement the Quicksort sorting algorithm and its best variants on real computers, including how to apply various code optimization techniques. A detailed implementation combining the most effective improvements to Quicksort is given, along with a discussion of how to implement it in assembly language. Analytic results describing the performance of the programs are summarized. A variety of special situations are considered from a practical standpoint to illustrate Quicksort's wide applicability as an internal sorting method which requires negligible extra storage.

Key Words and Phrases: Quicksort, analysis of algorithms, code optimization, sorting
CR Categories: 4.0, 4.6, 5.25, 5.31, 5.5

Acta Informatica 7, 327—355 (1977)

The Analysis of Quicksort Programs*

Robert Sedgewick

Received January 19, 1976

Summary. The Quicksort sorting algorithm and its best variants are presented and analyzed. Results are derived which make it possible to obtain exact formulas describing the total expected running time of particular implementations on real computers of Quicksort and an improvement called the median-of-three modification. Detailed analysis of the effect of an implementation technique called loop unwrapping is presented. The paper is intended not only to present results of direct practical utility, but also to illustrate the intriguing mathematics which arises in the complete analysis of this important algorithm.

Quicksort Variations

- Sedgewick 2-way partitioning
- Dijkstra 3-way partitioning
- Bentley-McIlroy 3-way partitioning
- Dual-pivot partitioning

  [very widely used. C, C++, Java 6, ….]

https://algs4.cs.princeton.edu/lectures/23DemoPartitioning.pdf

System Sort in Java 7

Arrays.sort()

- Has one method for objects that are Comparable.
- Has an overloaded method for each primitive type.
- Has an overloaded method for use with a Comparator.
- Has overloaded methods for sorting subarrays.

Algorithms

- Dual-pivot quicksort for primitive types.
- Timsort for reference types.

Bottom line: Use the system sort!