Sorting algorithms

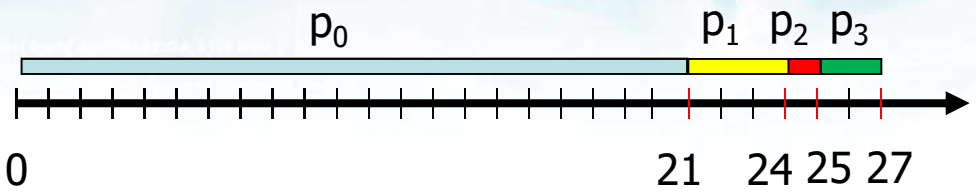
Classification

Paolo Camurati and Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

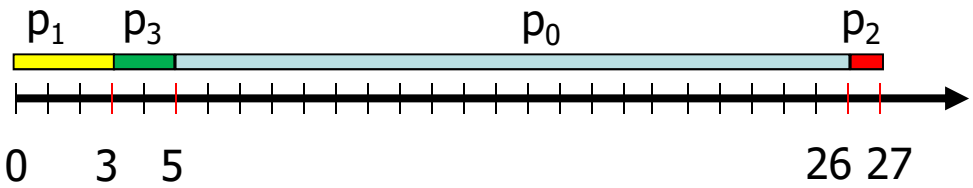
On the importance of sorting

- On an average application 30% of CPU time is spent on sorting data
- Example
  - CPU scheduling
    - Processes pi with duration
      - p_0 21
      - p_1 3
      - p_2 1
      - p_3 2
    - Impact of sorting on average wait time

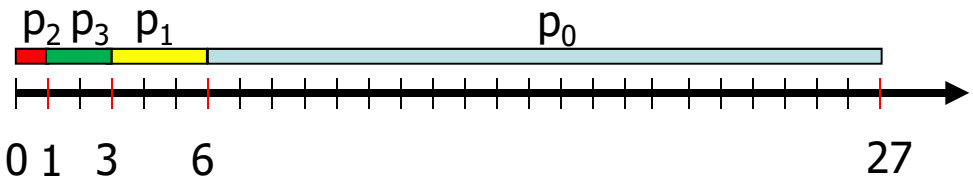
On the importance of sorting



- p_0-p_1-p_2-p_3
- average wait time (0+21+24+25)/4 =17.5



- p_1-p_3-p_0-p_2
- average wait time (0+3+5+26)/4 =8.5



- p_2-p_3-p_1-p_0

Sorted

- average wait time (0+1+3+6)/4 =2.5

Sorting applications

- Trivial applications
  - Sorting a list of names, organizing an MP3 library, displaying Google PageRank results, etc.
- Simple problems if data are sorted
  - Find the median, binary search in a database, find duplicates in a mailing list, etc.
- Non trivial applications
  - Data compression, computer graphics (e.g., convex hull), computational biology, etc.

Definitions

- Sorting
  - Input
    - Symbols belonging to a set having an order relation
    - $a_1, a_2, \ldots, a_n$
  - Output
    - Permutation of the input symbols
      - $a_1', a_2', \ldots, a_n'$
    - Such that the order relation
      - $a_1' \leq a_2' \leq \ldots \leq a_n'$

      holds

Definitions

- Order relation $\leq$
  - Binary relation between elements of a set A satisfying the following properties
    - Reflexivity
      - $\forall\ x \in A \rightarrow x \leq x$
    - Antisymmetry
      - $\forall\ x, y \in A \rightarrow x \leq y \wedge y \leq x \Rightarrow x = y$
    - Transitivity
      - $\forall\ x, y, z \in A \rightarrow x \leq y \wedge y \leq z \Rightarrow x \leq z$
- A is a partially ordered set (poset)
- If relation $\leq$ holds $\forall\ x, y \in A$, A is totally ordered set

Classification

- Internal sorting
  - Data are in main memory
  - Direct access to elements
- External sorting
  - Data are on mass memory
    - Completely or at least partially
  - Sequential access to elements

Classification

- In place sorting
  - n data in array plus a constant number of auxiliary memory locations
- Stable sorting
  - For data with duplicated keys the relative ordering is unchanged
  - Example
    - Record with 2 keys
      - Name (key is a string)
      - Group (key is an integer)

Stability: An example

Unsorted data

Chiara	3
Barbara	4
Andrea	3
Roberto	2
Giada	4
Franco	1
Lucia	3
Fabio	3

First sorting according to string name

Andrea	3
Barbara	4
Chiara	3
Fabio	3
Franco	1
Giada	4
Lucia	3
Roberto	2

Second sorting according to group **NON stable** algorithm

Franco	1
Roberto	2
Chiara	3
Fabio	3
Andrea	3
Lucia	3
Giada	4
Barbara	4

Second sorting according to group **stable** algorithm

Franco	1
Roberto	2
Andrea	3
Chiara	3
Fabio	3
Lucia	3
Barbara	4
Giada	4

Classification

- Complexity
  - Complexity can be computed in terms of overall number of steps, each one with a constant cost
  - A more detailed analysis is also possible, computing the total number of
    - Comparisons
    - Exchange operations
  - When data is large, exchanging them may be expensive and may be better to have more comparisons and less exchange operations
  - Asymptotic complexity however does not change

Classification

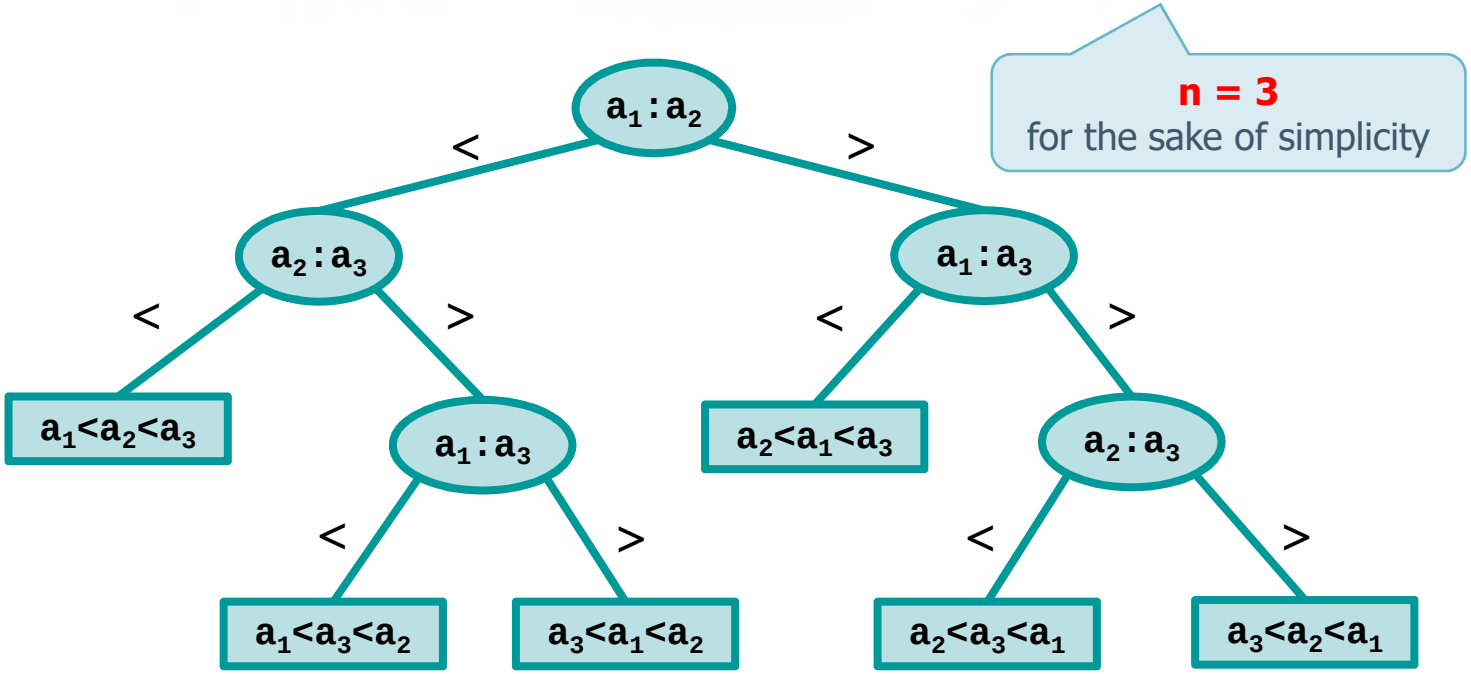
- $O(n^2)$
  - Simple, iterative, based on comparison
  - **Insertion** sort, **Selection** sort, **Exchange** (Bubble) sort
- $O(n^{3/2})$
  - **Shellsort** (with certain sequences)
- $O(n \cdot \log n)$
  - More complex, recursive, based on comparison
  - **Merge** sort, **Quicksort**, **Heapsort**
- $O(n)$
  - Applicable with **restrictions** on data, based on computation
  - **Counting** sort, Radix sort, Bin/Bucket sort

A lower bound for the complexity

- Algorithms **based on comparison**
  - Elementary operation
    - Comparison
      - $a_i : a_j$
  - Outcome
    - Decision
      - $a_i > a_j$ or $a_i \leq a_j$
    - Decisions organized as a decision tree

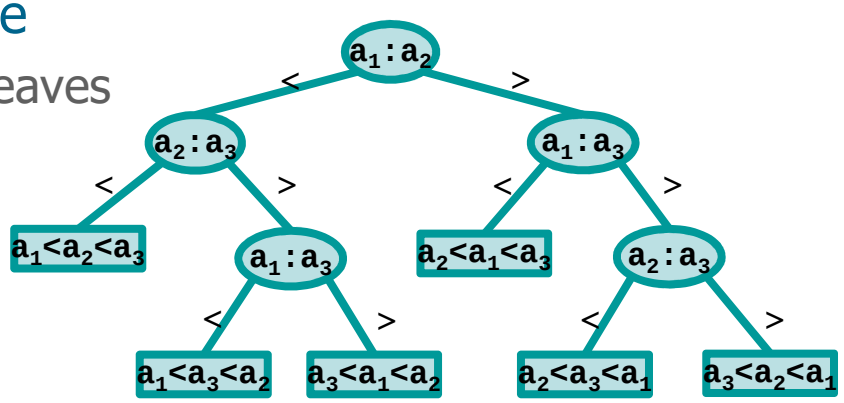
A lower bound for the complexity

- Sort array of 3 distinct elements a_1, a_2 , a_3



A lower bound for the complexity

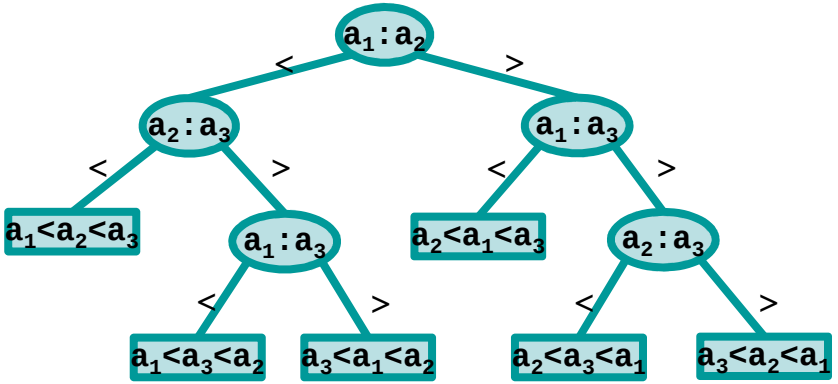
- For n distinct integers
  - The number of possible sortings equals the number of permutations, i.e., is n!
- Each solution
  - Sits on a tree leaf
- Complexity
  - Number h of comparisons, that is, the tree height h
- For a complete tree
  - The number of leaves is 2^h



A lower bound for the complexity

- Then we must have
  - $2^h \geq n!$

The tree has to include (generate) all possibile sorting on the leaves



A lower bound for the complexity

- Then we have
  - $2^h \geq n!$
  - $2^h \geq n! > (n/e)^n$
  - $2^h > (n/e)^n$
  - $\lg 2^h > \lg (n/e)^n$
  - $h > n \cdot \lg (n/e)$
  - $h > n (\lg n - \lg e) = \Omega(n \lg n)$

Stirling's approximation $n! > (n/e)n$

We compute the log of both members

Log property