Design and Analysis of Algorithms

Lecture-13

Dharmendra Kumar (Associate Professor)
Department of Computer Science and Engineering
United College of Engineering and Research,
Prayagraj

Sorting in Linear Time

Comparison Sort

In a comparison based sort, we use only comparisons between elements to gain order information about an input sequence $<a_1, a_2, \ldots\ldots\ldots\ldots\ldots, a_n>$. That is, given two elements a_i and a_j , we perform one of the tests $a_i < a_j$, $a_i \leq a_j$, $a_i=a_j$, $a_i \geq a_j$, or $a_i > a_j$ to determine their relative order.

The decision-tree model

We can view comparison sorts abstractly in terms of decision trees.

A ***decision tree*** is a full binary tree that represents the comparisons between elements that are performed by a particular sorting algorithm operating on an input of a given size.

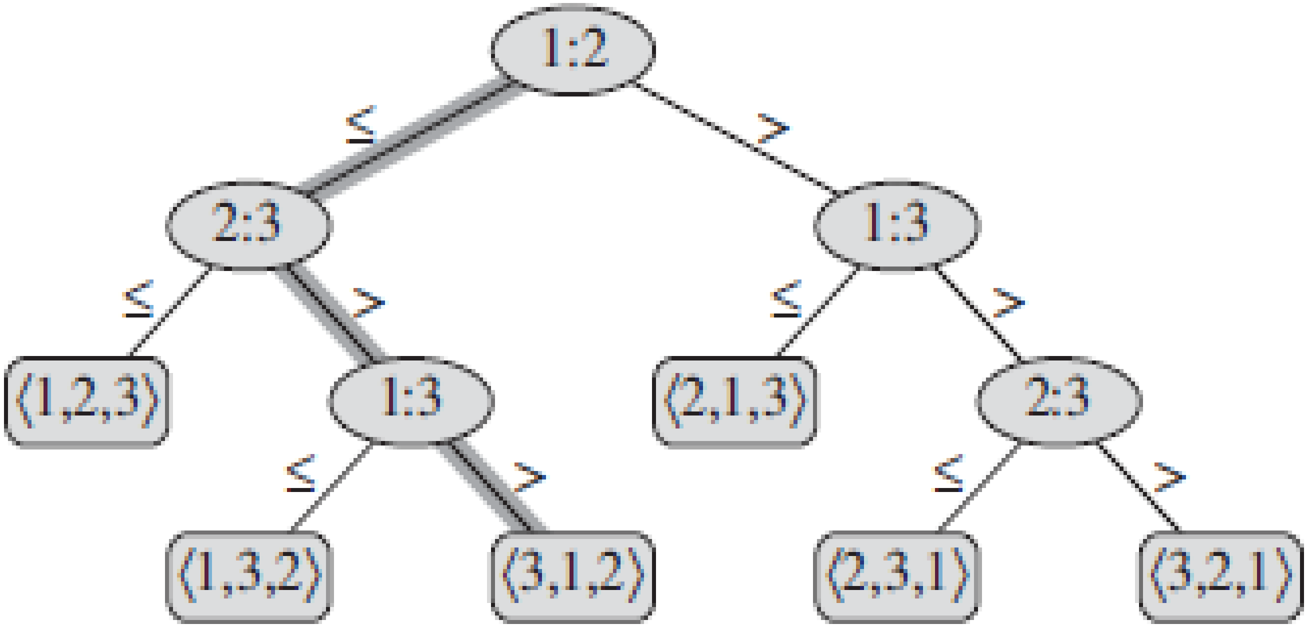
Sorting in Linear Time

In a decision tree, we annotate each internal node by **i:j** for some i and j in the range $1 \leq i, j \leq n$, where n is the number of elements in the input sequence. We also annotate each leaf by a permutation $< \boldsymbol{\pi}(1), \boldsymbol{\pi}(2), \boldsymbol{\pi}(3), \ldots\ldots\ldots\ldots, \boldsymbol{\pi}(n) >$.

- The execution of the sorting algorithm corresponds to tracing a simple path from the root of the decision tree down to a leaf.
- Each internal node indicates a comparison $a_i \leq a_j$. The left sub-tree then dictates subsequent comparisons once we know that $a_i \leq a_j$, and the right sub-tree dictates subsequent comparisons knowing that $a_i > a_j$.
- When we come to a leaf, the sorting algorithm has established the ordering $<a_{\boldsymbol{\pi}(1)}, a_{\boldsymbol{\pi}(2)}, \ldots\ldots\ldots\ldots, a_{\boldsymbol{\pi}(n)}>$.

Sorting in Linear Time

The decision tree operating on three elements is the following:-



Sorting in Linear Time

Theorem:

Any comparison sort algorithm requires Ω(nlg n) comparisons in the worst case.

Proof: The worst-case number of comparisons for a given comparison sort algorithm equals the height of its decision tree.

Consider a decision tree of height h with l reachable leaves corresponding to a comparison sort on n elements. Because each of the n! permutations of the input appears as some leaf, we have n! ≤ l . Since a binary tree of height h has no more than 2^h leaves, therefore

$$n! \leq l \leq 2h \;\Rightarrow\; h \geq \lg(n!)$$

$$\text{Therefore} \quad h = \Omega(n\lg n)$$

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.

- When k = O(n), the sorting algo. runs in $\theta(n)$ time.

- Counting sort determines, for each input element x, the number of elements less than x. It uses this information to place element x directly into its position in the output array.

Counting sort

Example: Sort the following elements using counting sort

2, 5, 3, 0, 2, 3, 0, 3

Solution:

	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)

Counting sort

	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

```
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
```

- Time complexity of counting sort is

  $T(n) = \theta(n+k)$
- If $k \le n$, then $\mathbf{T(n) = \theta(n)}$