Sorting in Linear Time

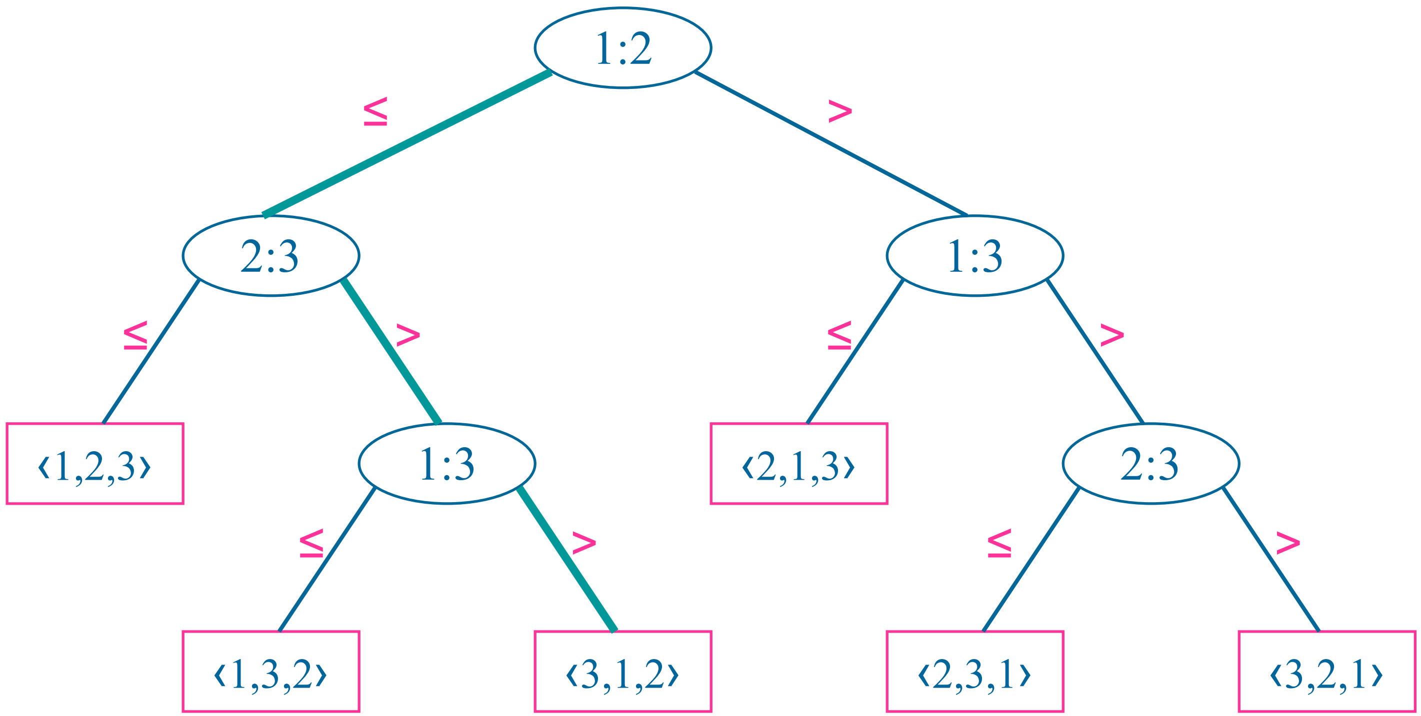
Sorting Algorithms

Sorting Algorithm	Worst Case Running Time	Average Case Running Time
Insertion sort	$O(n^2)$	$O(n) \leftrightarrow O(n^2)$
Merge sort	$O(n \lg n)$	$O(n \lg n)$
Heap sort	$O(n \lg n)$	$O(n \lg n)$
Quick sort	$O(n^2)$	$O(n \lg n)$

- What is common between previous presented sorting algorithms?
- All of them are comparison Sorting algorithms
  - The only operation used to gain ordering information about a sequence is the pair-wise comparison of two elements

- What is Lower bounds for comparison sorting algorithms?
- The decision-tree model
  - Decision-tree is a full binary tree that represents the comparison between elements that are performed by a particular sorting algorithms operating on an input of a given size
  - Control, data movement, and all other aspects of the algorithm are ignored.

Decision tree for comparison sort on three elements



- ✓Internal nodes label $i : j \Rightarrow$ comparing a_i and a_j
- ✓Leaf nodes label $\Rightarrow$ permutation $\langle\pi(1),\pi(2),\ldots.,\pi(n)\rangle$
- ✓A[3] = ‹ 6,8,5›

- Decision tree $\Rightarrow$ heigh h
- $n!$ permutations $\Rightarrow$ appears as some leaves
- Binary tree with height $h \Rightarrow$ no more than 2^h leaves
- $n! \leq 2^h$
- $h \geq \lg(n!) = \Theta(n \lg n) = \Omega(n \lg n)$
- Heap & Merge Sort asymptotically optimal comparison sort.

Counting Sorting

- No comparisons between elements
- Depends on assumption that the numbers being sorted are in the range 1.. k

Data Structure:

- ✓Input: A[1..n], where A[j] $\in$ {1, 2, 3, …, k}
- ✓Output: B[1..n], sorted (auxiliary)
- ✓Array C[1..k] $\Rightarrow$ auxiliary storage

Counting Sort Algorithm

```
CountingSort(A, B, k)
for i ← 1 to k
        C[i] ← 0
for j ← 1 to length[A]
        C[A[j]] ← C[A[j]] + 1

for i ← 1 to k
        C[i]  ← C[i] + C[i-1]

for j ← length[A] downto 1
        B[C[A[j]]] ← A[j]
        C[A[j]] ← C[A[j]] -1
```

Running Time of Counting Sort

```
CountingSort(A, B, k)
for i ← 1 to k
        C[i] ← 0                          Θ(k)
for j ← 1 to length[A]
        C[A[j]] ← C[A[j]] + 1             Θ(n)

for i ← 1 to k
        C[i]  ← C[i] + C[i-1]             Θ(k)

for j ← length[A] downto 1
        B[C[A[j]]] ← A[j]                 Θ(n)
        C[A[j]] ← C[A[j]] -1
```

- $\mathrm{T}(n) = O(\ k + n\ )$
- Counting sort is used when $k = O(n)$
- $\mathrm{T}(n) = O(\ n\ )$

Counting Sort is a *stable* sorting algorithm !!!

Radix Sort

- ✓Sort n-element array A
- ✓Each element in $\Rightarrow$ d digits
- ✓Digit 1 $\Rightarrow$ Lowest order digit
- ✓Digit d $\Rightarrow$highest order digit

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

- $T(n) = O(\, d\, (n + k)\,)$

Bucket Sort

- input is n real from [0, 1)
- Basic idea:
  - Create n linked lists (*buckets*) to divide interval [0,1) into subintervals of size $1/n$
  - Add each input element to appropriate bucket and sort buckets with insertion sort

Bucket Sort Algorithm

Bucket-Sort(A)

1- $n \leftarrow length[A]$

2- for $i \leftarrow 1$ to n

3- inset A[i] into list B[$\lfloor nA[i] \rfloor$]

4- for $i \leftarrow 0$ to n-1

5- sort list B[i] with insertion sort

6- concatenate the lists B[0], B[1], …., B[n-1] together in order

Running Time of Bucket Sort

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

$$\mathrm{E}[T(n)] = \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} O(\mathrm{E}(n_i^2))$$

$$\mathrm{E}(n_i^2) = 2 - 1/n$$

$$T(n) = \Theta(n)$$