ILLINOIS INSTITUTE OF TECHNOLOGY

Transforming Lives.Inventing the Future. ***www.iit.edu***

CS430

Introduction to Algorithms

Lec 8

Lan Yao

Outlines

Lower Bound on Sorting

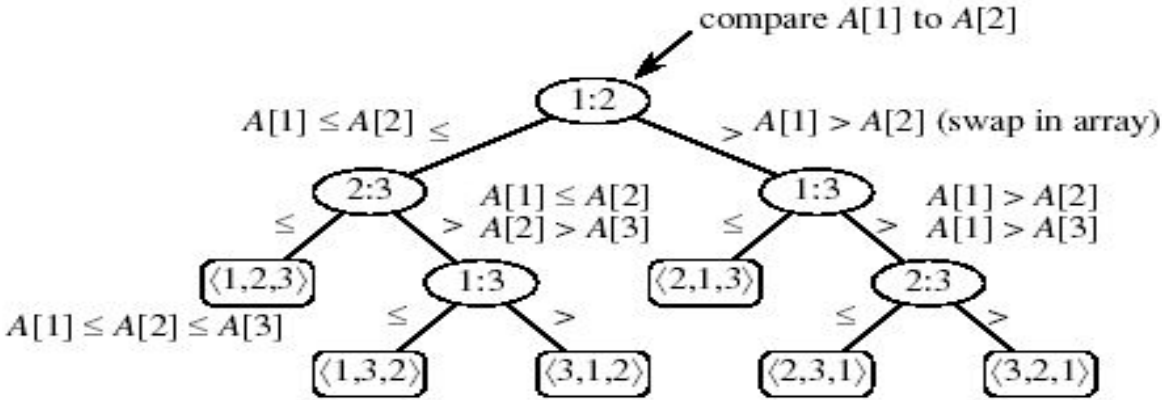
- Decision tree
- Counting sort
- Radix sort
- Bucket sort
- Order statistics

- Comparison sorts:
  - insertion sort, merge sort, heapsort, quicksort
  - the sort algorithm is determined based only on comparisons between the input elements. We call such sorting algorithms Comparison Sorts
    - the sorted order they determine is based only on comparisons between the input elements. That is, given two elements ai and aj, we perform one of the tests ai < aj, ai = aj, ai >= aj, ai <= aj, or ai > aj to determine their relative order.
    - Claim: We must make **at least Omega (n lgn)** comparisons in the general case

The decision-tree model

- Abstraction of any comparison sort.
- Represents comparisons made by a specific sorting algorithm on inputs of a given size.
- View the tree as if the algorithm splits in two at each node, based on the information it has determined up to that point; The tree models all possible execution traces

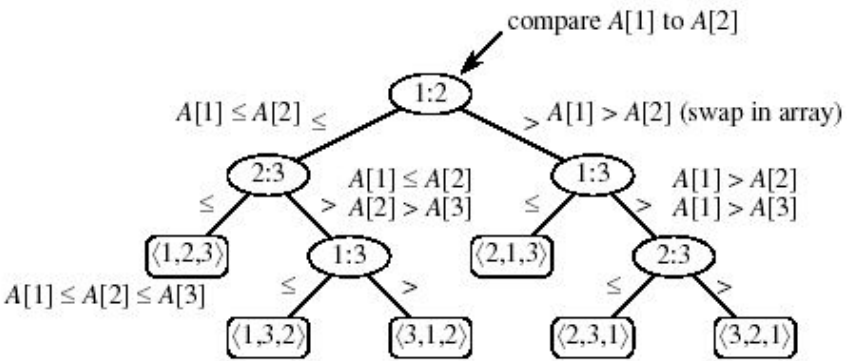
For insertion sort on 3 elements:



Each internal node is labeled by indices of array elements **from their original positions**. Each leaf is labeled by the permutation of orders that the algorithm determines.

Decision tree for Insertion sort

For insertion sort on 3 elements:



The number of leaves: n!=3!=6
The height of the tree: 3
The number of comparisons: 3

Lower bound on Comparison Sorts

- Any correct sorting algorithm must be able to produce each permutation of its input (n! total)
- Each of these leaves must be reachable from the root by a path corresponding to an actual execution of the comparison sort.
- Worst-case number of comparisons for a given comparison sort algorithm equals the height of its decision tree.

- Generalize the above properties

  h denotes the height of a decision tree, then

  The max number of leaves=2^h

  The permutation is $n! <= 2^h$ $(n>3)$ } h>=lgn!

  lnn!=nlnn-n+O(lnn) } h=Ω(nlgn)

  $$n! = \Theta(\sqrt{n}(\frac{n}{e})^n)$$

Decision Tree lower bound.

$$n! = \Theta\left(\sqrt{n}\left(\frac{n}{e}\right)^n\right) \geq c\cdot\left(\sqrt{n}\left(\frac{n}{e}\right)^n\right)$$

$$\Rightarrow \lg n! \geq \lg c\cdot\left[\sqrt{n}\left(\frac{n}{e}\right)^n\right] = \lg c + \lg\left[\sqrt{n}\cdot\left(\frac{n}{e}\right)^n\right]$$

$$= \lg c + \frac{1}{2}\lg n + n\lg n - n\cdot\lg e$$

suppose that there exists a constant c', such that $\lg c + \frac{1}{2}\lg n + n\lg n - n\cdot\lg e \geq c' n\lg n$

$$(1-c')n\lg n \geq \lg e\cdot n - \frac{1}{2}\lg n - \lg c$$

$$1-c' \geq \frac{\lg e}{\lg n} - \frac{\frac{1}{2}}{n} - \frac{\lg c}{n\lg n}$$

when $n\to\infty$, $1-c' \geq 0$. That is when $c'\leq 1$ and $c'>0$, it stands.

Such that: $\lg n! > c' n\lg n = \Omega(n\lg n)$

Other sort algorithms other than comparison

- Consider the following scenarios
  - grade ranking
    - 30 students
    - 10 of them get A
    - 18 of them get B
    - 2 of them get C
    - if your grade is A, you definitely know that you are in top 10 without conducting any comparison
  - order two grades
    - your grade is ending with 9 and Lanny's grade is ending with 1. Whose grade is better?

Linear Time Sorting
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 sort runs in T(n) time (best if k<<<n)
- For each input element “x”, count how many elements are less than “x”. This information can be used to place element x directly into its position in the output array.
- Does not sort in place, needs 2^{nd} array size “k” and 3^{rd} array size “n”
- Stable

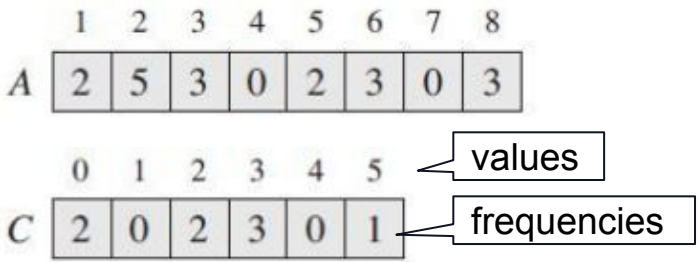
- **A stable algorithm**

A sorting algorithm is stable ,then it satisfies:

Numbers with the same value appear in the output array in the same order as they do in the input array and it breaks ties between two numbers by the rule that whichever number appears. First in the input array appears first in the output array.

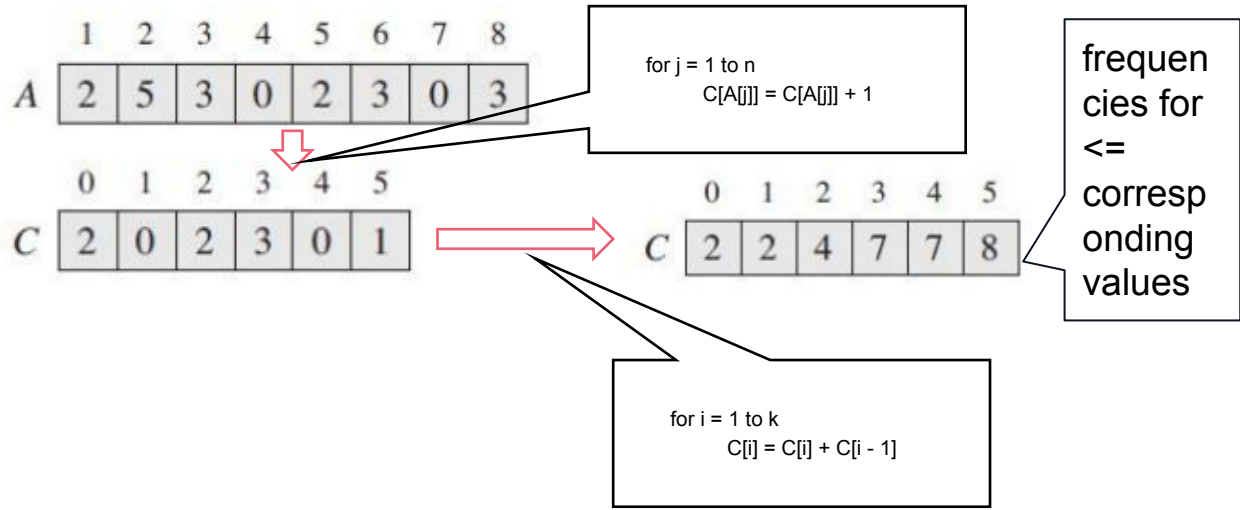
Counting Sort Demo

In array A, n=8 and k=5. Array C is the counting statistic of A. (C is the 2nd array)

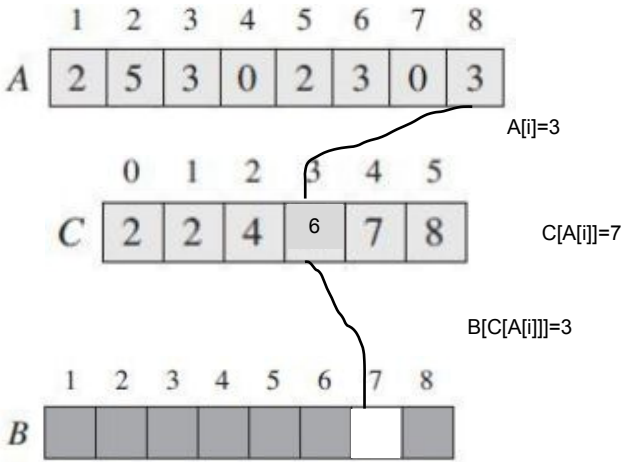


Counting Sort Demo (cont'd)

In array A, n=8 and k=5. Array C is the counting statistic of A. (C is the 2nd array)



B is an n-element array (the 3rd array) as the output




```
for j = n downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

Counting Sort Pseudocode

```
COUNTING-SORT(A, B, k)
    for i = 0 to k
        C[i] = 0
    for j = 1 to n
        C[A[j]] = C[A[j]] + 1   // count # of each value
    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 = n downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

- Complexity of Counting Sort

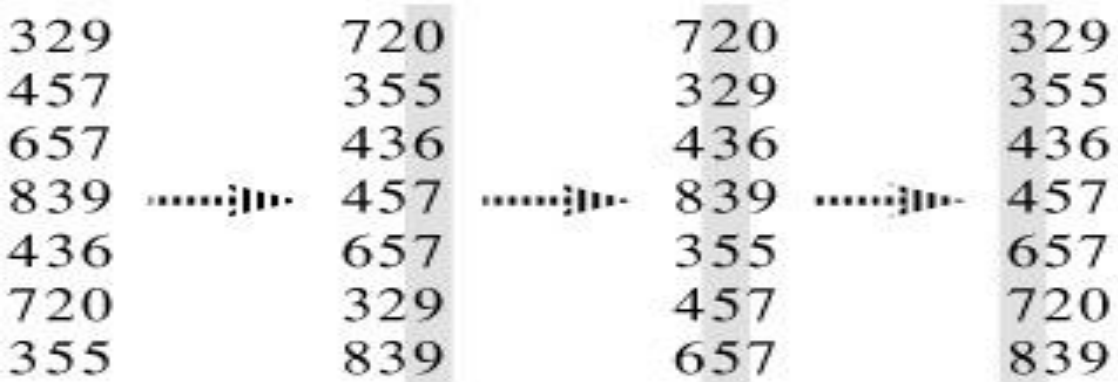
  $\Theta(n+k)$ and $k=O(n)$ in most cases then the entire complexity is $\Theta(n)$.

Linear Time Sorting
Radix Sort

- The procedure assumes that each element in the n-element array A has d digits, where digit no. 1 is the lowest-order digit and digit d is the highest-order digit.

- We use Counting Sort to sort on each digit.
- Sort from digit no. 1 to d.

Radix Sort Demo



Radix Sort Algorithm

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

Complexity

- Given n d-digit numbers in which each digit can take on up to k possible values, Radix Sort sorts these numbers in Θ (d*(k+n)) time if the stable sort it used takes Θ(n+k) time.