CISC 320 Introduction to Algorithms Fall 2005

Lecture 6

Sorting in linear time

A job interview question:

Given integers 1 to 100, but in an arbitrarily unsorted order. How fast can these 100 integers be sorted into the increasing order?

Given 10 integers valued between 1 and 100, and in an arbitrarily unsorted order. How fast can these 10 integers be sorted into the increasing order?

Given 10,000 integers valued between 1 and 100,and in an arbitrarily unsorted order. How fast can they be sorted into the increasing order?

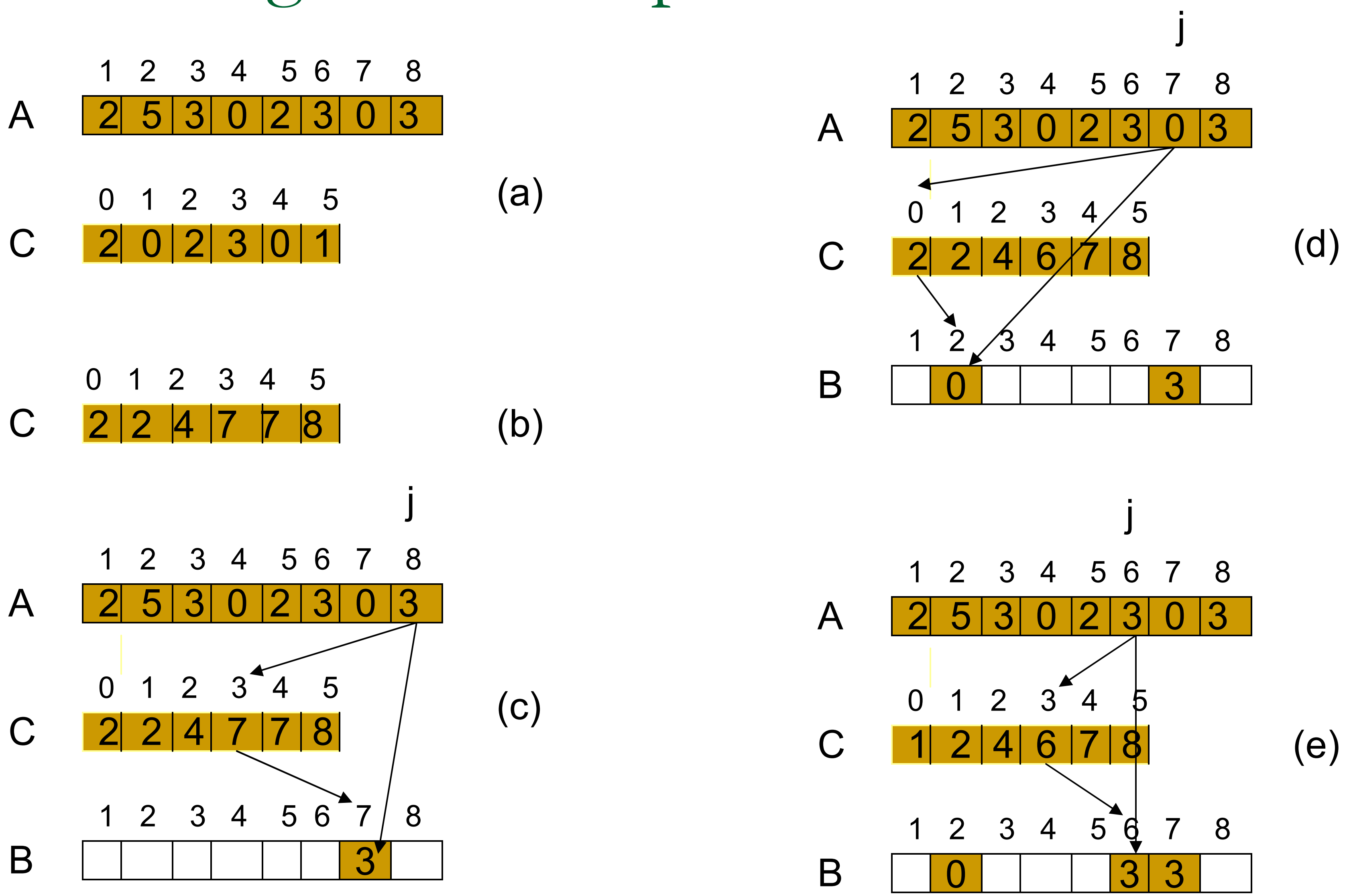
Counting sort: ideas

- Problem: to sort an array A of integers
- Extra info: each of the n input elements of A is an integer in the range [0, k].
- One scan through A will find how many times each integer i $\in$ [0,k] appears in A. The counts are stored in an auxiliary array C[0..k].
- Thus element C[i] is the number of times that integer i appears in A. $\sum_{j=0 \text{ to } i}$ C[i] gives the number of elements of A that are less than i, and this tells where to put i in a sorted array.

Counting sort: algorithm

```
Counting-sort (A, B, k)
    for (i=0; i<k; i++)
        C[i] = 0; // initialize array C to store counts.
    for (j=0; j<n; j++)
        C[A[j]] = C[A[j]] + 1; // scan A and count.
    for (I = 1, i<k, i++)
        C[i] = C[i] + C[i-1]; // transform to cumulative counts
    for (j = n; j > 1; j--)
        B[C[A[j]]] = A[j];  // sort to the right place
        C[A[j]] = C[A[j]] – 1; // update the count
```

Counting sort: example



Counting sort: complexity analysis

- Time

Initialize array C — Scan A and sort

$$\Theta(k) + \Theta(n) + \Theta(k) + \Theta(n) \in \Theta(k + n)$$

Transform C

Scan A and Count

- Space: $\Theta(k+n)$ not in-place

Note:

1. Counting sort is not comparison based sorting. Instead, it uses the actual values of the elements to index into an array. Thus, its linear performance breaks the $\Omega(n \log n)$ lower bound for comparison based sorting problems.
2. Stable sort

Radix sort: ideas

- Problem: to sort A, an array of *n* *d*-digit numbers
- Strategy: sort numbers based on their least significant digit, then on their second least significant digit, so on and so forth, until sort on their most significant digit is done.

Radix sort: an example

unsorted		ones		tens		hundreds
329		720		720		329
457		355		329		355
657		436		436		436
839	→	457	→	839	→	457
436		657		355		657
720		329		457		720
355		839		657		839

Correctness: Induction on digit position

Assume numbers are sorted up to the i-th digit. When sort on (i+1) –th digit:

1) Two numbers that differ in the (i+1)-th digit are correctly sorted.

2) Two numbers having same (i+1)-th digit remain the same order up to i-th digit, because sorting on each digit is stable.

Radix sort:

- Why not sort from most significant digits to less significant digits?

1 3	→	4 2
4 2		1 3

Radix sort: analysis

Time = (number of digits) x (time for sorting on each digit)

= d x $\Theta(10+n) \in O(n)$

note: counting sort is utilized for sorting on each digit.

Exercise: Show how to sort n integers in the range 1 to $n^2 - 1$ in $O(n)$ time.

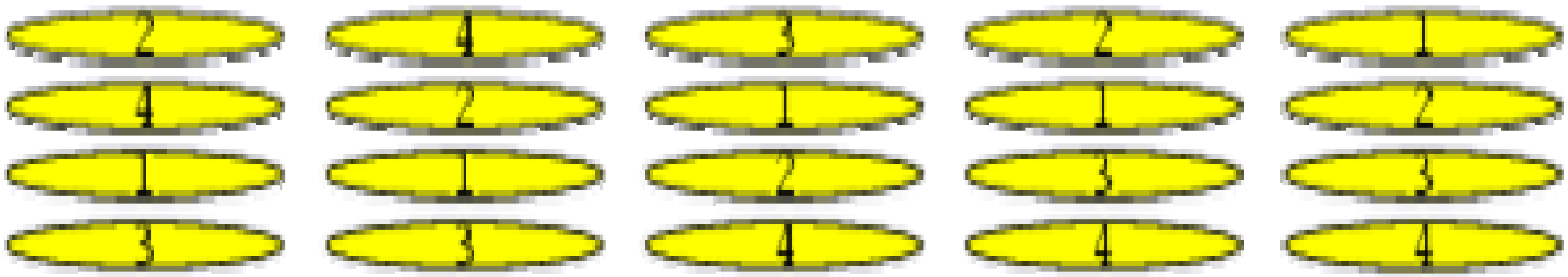
Pancake sorting?

Bounds for sorting by prefix reversal.

Gates, William H. and Christos H. Papadimitriou.

Discrete Mathematics 27, 47--57, 1979

The authors study the problem of sorting a sequence of distinct numbers by prefix reversal -- reversing a subsequence of adjacent numbers which always contains the first number of the current sequence. Let f(n) denote the smallest number of prefix reversals which is sufficient to sort n numbers in any ordering. The authors prove that f(n) <= (5n+5)/3 by demonstrating an algorithm which never needs more prefix reversals. They also prove that f(n) >= 17n/16 whenever n is a multiple of 16. The sequences which achieve this bound are periodic extensions of the basic sequence 1, 7, 5, 3, 6, 4, 2, 8, 16, 10, 12, 14, 11, 13, 15, 9. If, furthermore, each integer is required to participate in an even number of prefix reversals, the corresponding function g(n) is shown to satisfy 3n/2 - 1 <= g(n) <= 2n+3.


2 4 3 2 1
4 2 1 1 2
1 1 2 3 3
3 3 4 4 4