ILLINOIS INSTITUTE OF TECHNOLOGY

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

CS430

Introduction to Algorithms

Lec 9

Lan Yao

Outlines

Lower Bound on Sorting

- Counting sort
- Radix sort
- Bucket sort
- Order statistics

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

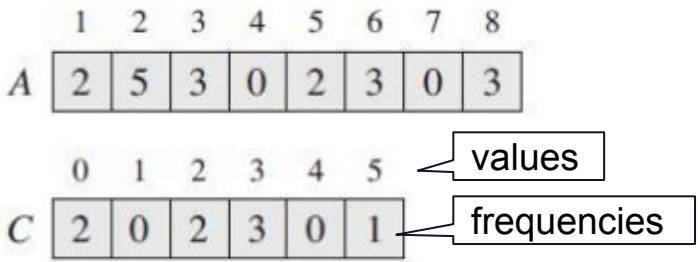
- **A stable algorithm**

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.

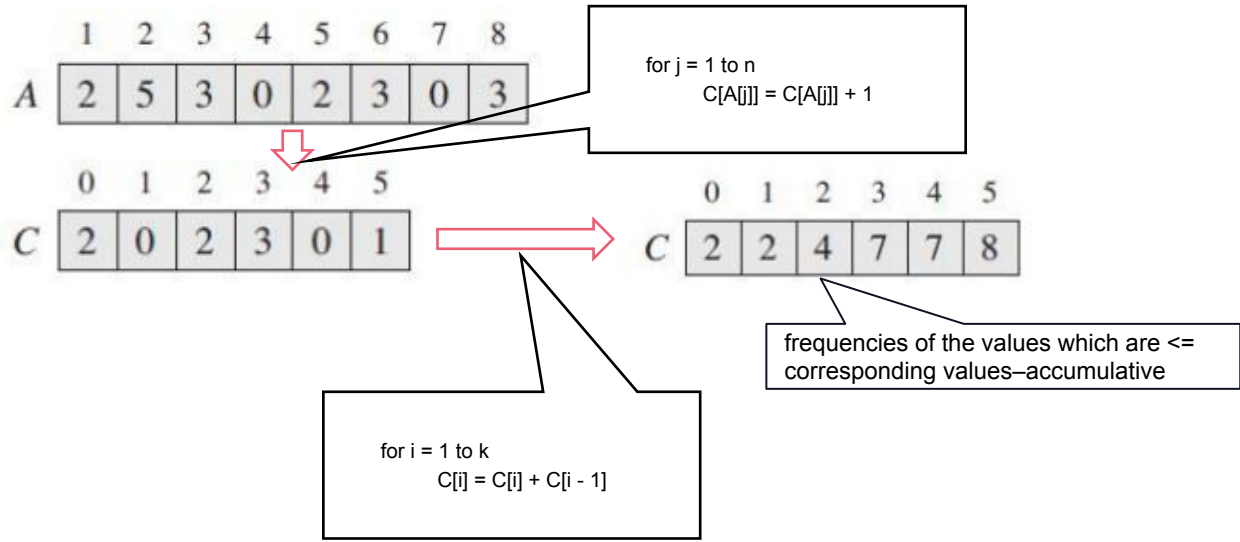
Counting Sort Demo

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

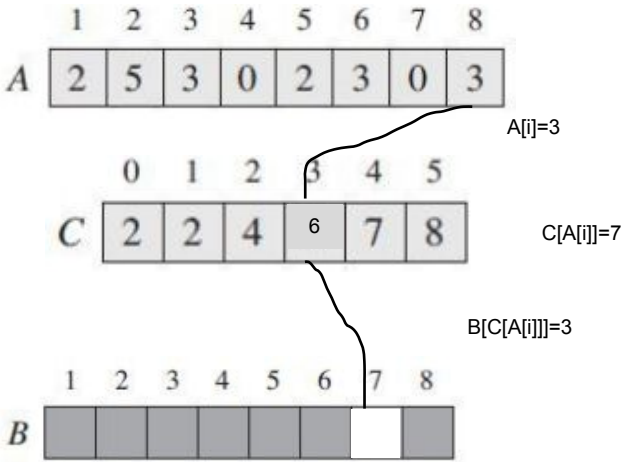


Counting Sort Demo (cont'd)

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

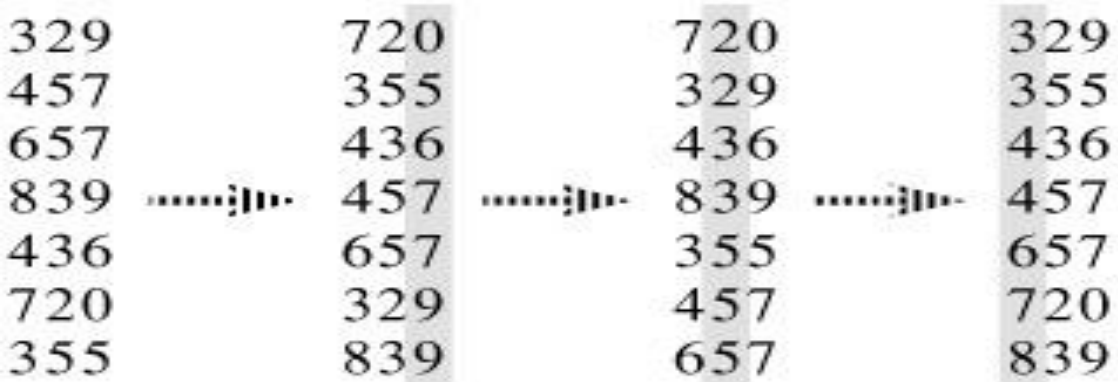
  $\Theta(n+k)$ and $k=O(n)$ in most cases then the entire complexity is $\Theta(n)$.

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.

Ex.

Given a sequence of n IP addresses, sort the sequence to future use for routing.

- We have n b-bit numbers. Sort these numbers. When we split each number into d sub-numbers of r bits, we have digits of r bits from each number.

$$d = \left\lceil \frac{b}{r} \right\rceil$$

  - 32/8=4
- Complexity
  - $\Theta(d*(k+n))$ and in our case, $k=2^r$, $b=O(\lg n)$ and $r=O(\lg n)$. Plug in, then $\Theta(d*(k+n)) = \Theta((b/r)(2^r+n)) = \Theta(\lg n*(n+n)/\lg n)$

    $=\Theta(n+n)=\Theta(n)$

- An example of IP address and its dot-decimal notition

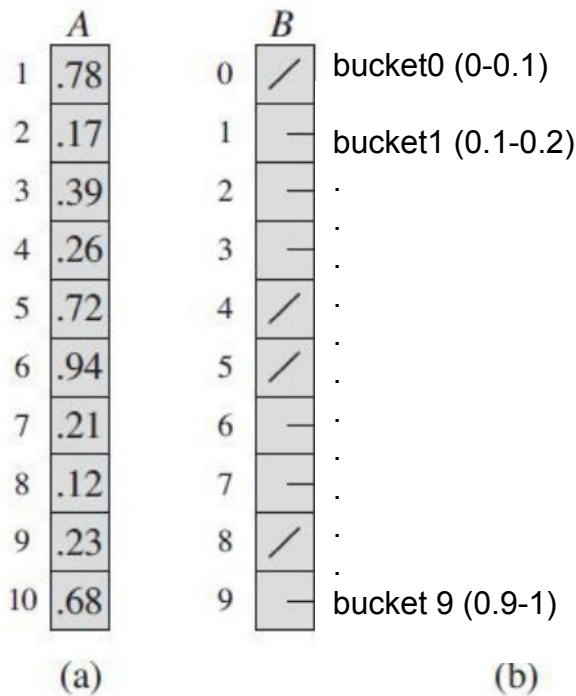
198	28	61	0

11000110	00011100	00111101	00000000

Linear Time Sorting
Bucket Sort

- The input is generated by a random process that distributes elements uniformly and independently over the interval from 0 to 1.
- The value of each input element is from 0 to 1.
- Bucket sorts divides the interval into **n** equal-size sub-interval buckets and then distribute the input elements into the buckets. Then we sort in each bucket and go through all buckets to list the elements as a sorted array.
- In each bucket, we use Insertion Sort.

Bucket Sort Demo



- We deploy n buckets, B[0] to B[n-1], where n is the number of elements.
- For each bucket, the data structure is a list.
- For a bucket i, it covers the domain of $(i \times \frac{1}{n},\ (i+1)\frac{1}{n})$

When an element's key fall in this domain, it belongs to this bucket.

- Suppose that a key of an element is a and its belongs to bucket i, how to find this i?

$$i \times \frac{1}{n} \le a \le (i+1)\frac{1}{n} \Rightarrow i \le a \times n \le i+1$$

$$\Rightarrow i = \lfloor a \times n \rfloor$$

Bucket Sort Algorithm

```
BUCKET-SORT(A)
1  let B[0..n − 1] be a new array
2  n = A.length
3  for i = 0 to n − 1
4       make B[i] an empty list
5  for i = 1 to n
6       insert A[i] into list B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
```

- Complexity of Bucket Sort
  Let n_i be the random value denoting the number of elements in bucket B(i), then

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

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

$E(n_i^2)$

Indicator X_{ij} is random valuable

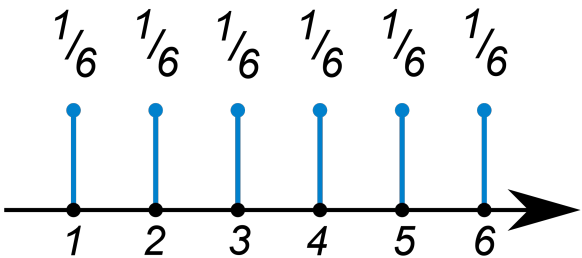
$$X_{ij} = \mathrm{I}\{A[j] \ \textit{falls in bucket } i\}$$

$$n_i = \sum_{j=1}^{n} X_{ij}$$

$$E(n_i^2) = E[(\sum_{j=1}^{n} X_{ij})^2] = E(\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik})$$

$$= E[\sum_{j=1}^{n} X^2{}_{ij} + \sum_{j=1}^{n}\sum_{k=1}^{n,k\neq j} X_{ij}X_{ik}] = \sum_{j=1}^{n} E(X^2{}_{ij}) + \sum_{j=1}^{n}\sum_{k=1}^{n,k\neq j} E(X_{ij}X_{ik})$$

- For continuous input, the uniform probability is determined by PDF.
  - for particular value of x, the probability is 0. Probability Density Function (PDF)
- For discrete values, the uniform probability is determined by PMF



  - for particular value of x, the probability is 1/a.

$$X_{ij} = \begin{cases} 0, \ \textit{with probability of } 1-\frac{1}{n} \\ 1, \ \textit{with probability of } \frac{1}{n} \end{cases}$$

$$E(X_{ij}^2) = 1^2 \bullet \frac{1}{n} + 0^2 \bullet (1-\frac{1}{n}) = \frac{1}{n}$$

$$E(X_{ij}X_{ik}) = E(X_{ij})E(X_{ik}) = \frac{1}{n} \bullet \frac{1}{n} = \frac{1}{n^2}$$

$$E(n_i^{\,2}) = \sum_{j=1}^{n} \frac{1}{n} + \sum_{j=1}^{n} \sum_{k=1, k \neq j}^{n} \frac{1}{n^2}$$

$$= \frac{n}{n} + n(n-1)\frac{1}{n^2} = 2 - \frac{1}{n}$$

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

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

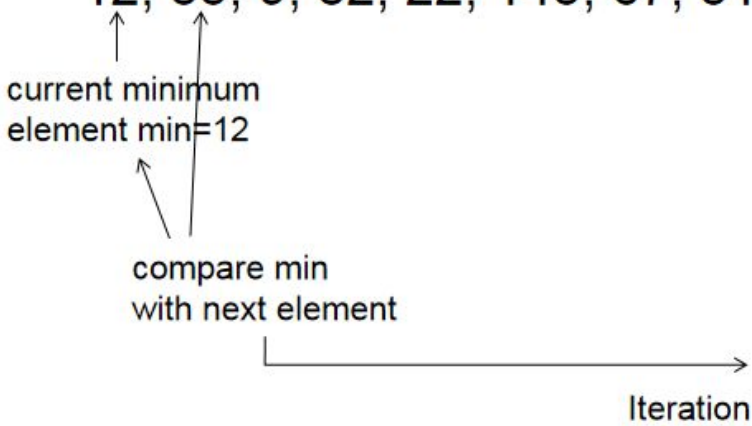
$$= \Theta(n) + nO(2 - \frac{1}{n})$$

$$= \Theta(n)$$

Order Statistics

- Statistics for an array: Maximum, Minimum and Median
- Potential method: Sort
- More efficient?

- Min/max
  - array A= {12, 35, 9, 82, 22, 145, 67, 51}, Minimum (A)?

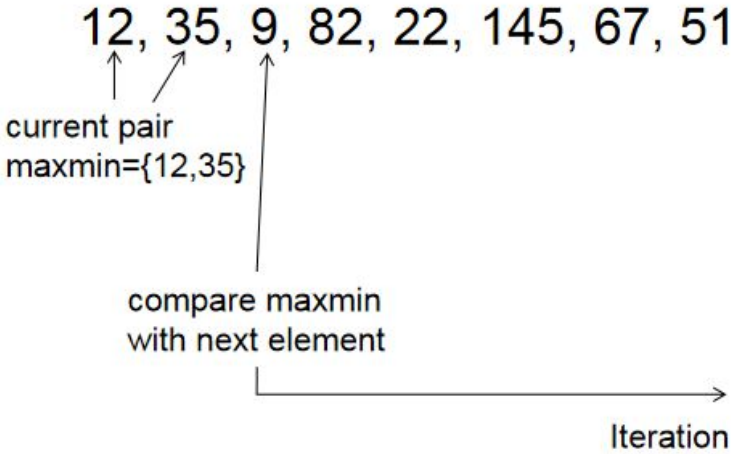


Minimum of the Array

```
MINIMUM(A)
1  min = A[1]
2  for i = 2 to A.length
3      if min > A[i]
4          min = A[i]
5  return min
```

Comparison times: (n-1)

- Min/max in pairs
  - array A= {12, 35, 9, 82, 22, 145, 67, 51}, max-min(A)?



Comparison times:

(n-2)*E(X)

X_i: PMF

In our case, probability p_i is equally likely as 1/2 .
E(X)=1*1/2+2*1/2=3/2

Select the ith smallest of n elements (the element with ***rank*** i).

- $i = 1$: ***minimum***;
- $i = n$: ***maximum***;
- $i = \lfloor (n+1)/2 \rfloor$ or $\lceil (n+1)/2 \rceil$: ***median***.

- Definitions:

The ith order statistic of a set of n elements is the ith smallest element. The minimum of a set of elements is the first order statistic (i=1). The maximum is the nth statistic (i=n). The median is the “Halfway point” of the set. If n is odd, the median is unique occurring at i=(n+1)/2. When n is even, there are two medians occurring at i=n/2 and i=(n/2)+1. The medians occur at

$$i = \left\lfloor \frac{n+1}{2} \right\rfloor$$, Lower median

$$i = \left\lceil \frac{n+1}{2} \right\rceil$$, Upper median

How fast can we solve the problem ?

- Min/max: O(n)
- General i : O(n log n) by sorting
- We will see how to do it in O(n) time

Input: a set A of n distinct numbers and an integer i, 1<=i<=n;
Output: the element x∈A that is larger than exactly i-1 other elements of A.

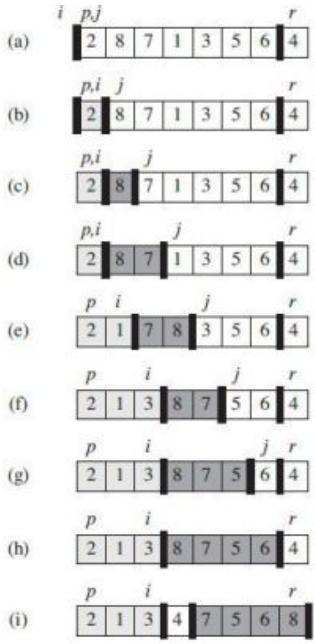
- Algorithm

*i*th Smallest Element of the Array

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2       return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k            // the pivot value is the answer
6       return A[q]
7  elseif i < k
8       return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4       if A[j] ≤ x
5            i = i + 1
6            exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```



- Complexity of order statistic

$E(T(n))=O(n)$.

  - The proof is not mandatory in this course.