طراحی و تحلیل الگوریتم ها

دکتر امیر لکی زاده

استادیار گروه مهندسی کامپیوتر دانشگاه قم

Sorting in Linear Time

- We will study algorithms that do not depend only on comparing *whole* keys to be sorted.

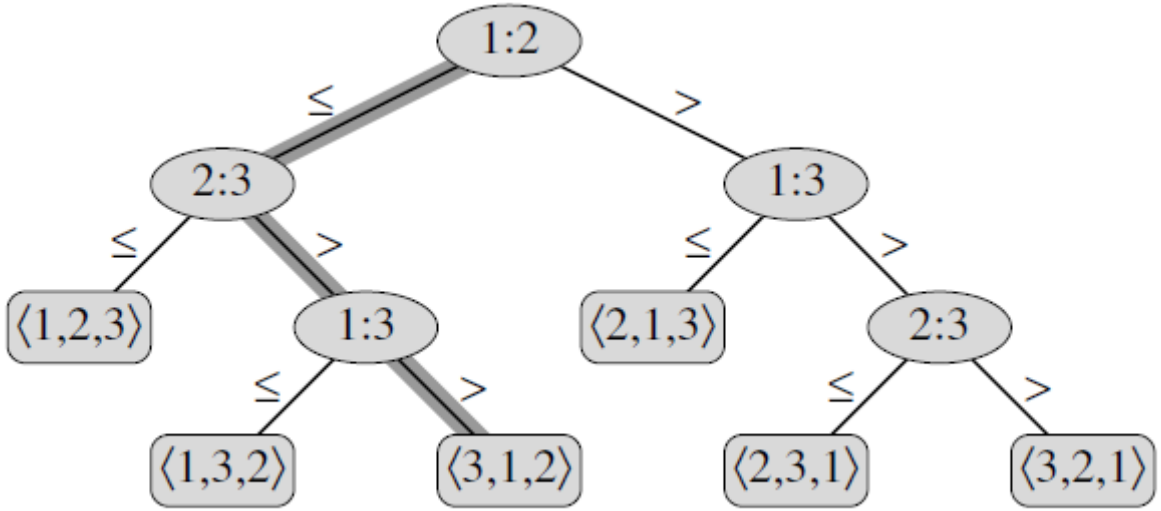
- Counting sort
- Bucket sort
- Radix sort

Lower Bounds for Comparison Sorting

Theorem 8.1

Any comparison sort algorithm requires $\Omega(n \lg n)$ comparisons in the worst case.

Lower Bounds for Comparison Sorting



Counting sort

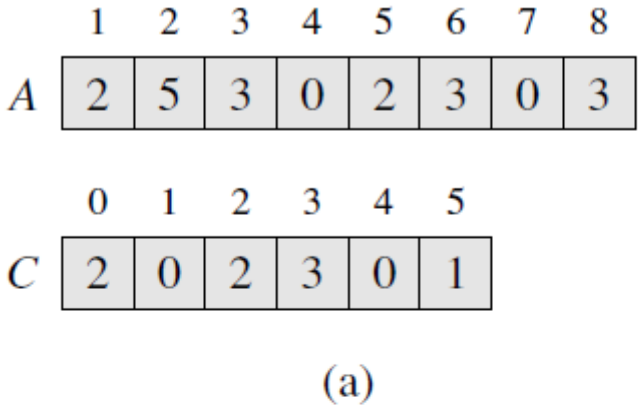
- **Assumptions:**
  - *n* records
  - Each record contains keys and data
  - All keys are in the range of 1 to k
- **Space**
  - The unsorted list is stored in A, the sorted list will be stored in an additional array B
  - Uses an additional array C of size k

Counting sort

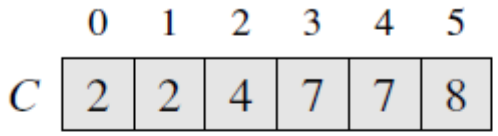
- **Main idea:**
  1. For each key value *i*, *i* = 1,…,*k,* count the number of times the keys occurs in the unsorted input array *A*. Store results in an auxiliary array, C
  2. Use these counts to compute the offset. $Offset_i$ is used to calculate the location where the record with key value *i* will be stored in the sorted output list *B*.
     The $offset_i$ value has the location where the last key_i .

- **When would you use counting sort?**
- **How much memory is needed?**

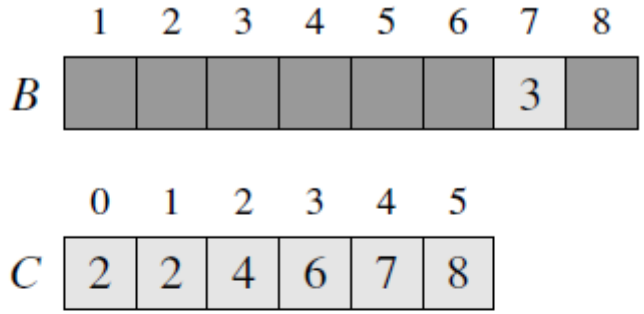
Counting Sort



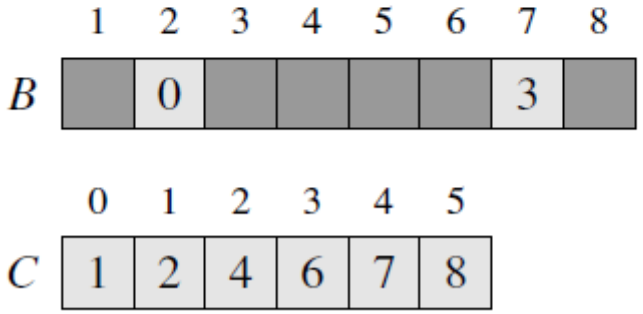
(a)



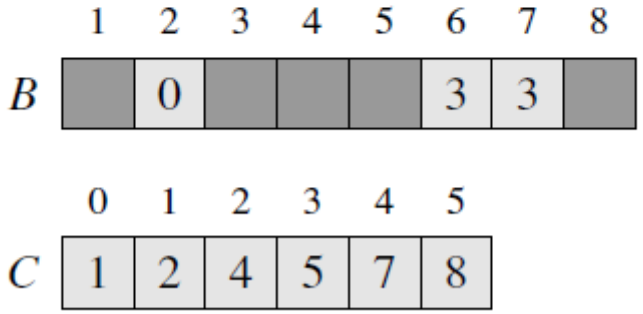
(b)



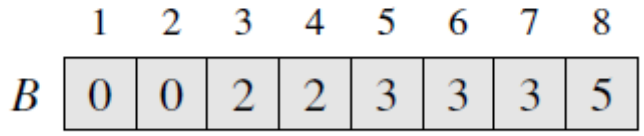
(c)



(d)



(e)

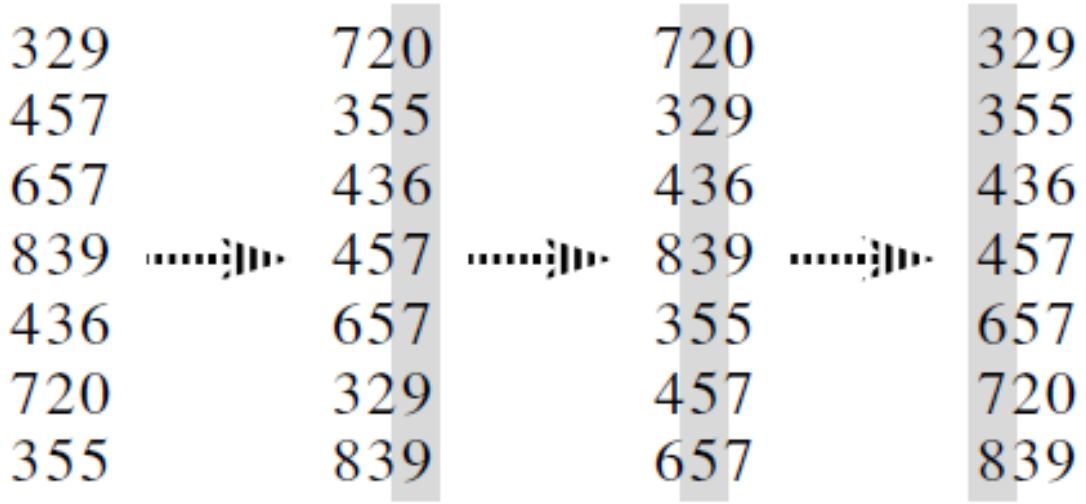


(f)

Counting Sort

```
COUNTING-SORT(A, B, k)
 1  for i ← 0 to k
 2       do C[i] ← 0
 3  for j ← 1 to length[A]
 4       do C[A[j]] ← C[A[j]] + 1
 5  ▷ C[i] now contains the number of elements equal to i.
 6  for i ← 1 to k
 7       do C[i] ← C[i] + C[i − 1]
 8  ▷ C[i] now contains the number of elements less than or equal to i.
 9  for j ← length[A] downto 1
10       do B[C[A[j]]] ← A[j]
11          C[A[j]] ← C[A[j]] − 1
```

Radix Sort


329
457
657
839
436
720
355
720
355
436
457
329
839
720
329
436
839
355
457
657
329
355
436
457
657
720
839

Radix Sort

RADIX-SORT(A, d)

1 **for** $i \leftarrow 1$ **to** d

2 **do** use a stable sort to sort array A on digit i

Lemma 8.3

Given n d-digit numbers in which each digit can take on up to k possible values, RADIX-SORT correctly sorts these numbers in $\Theta(d(n + k))$ time.

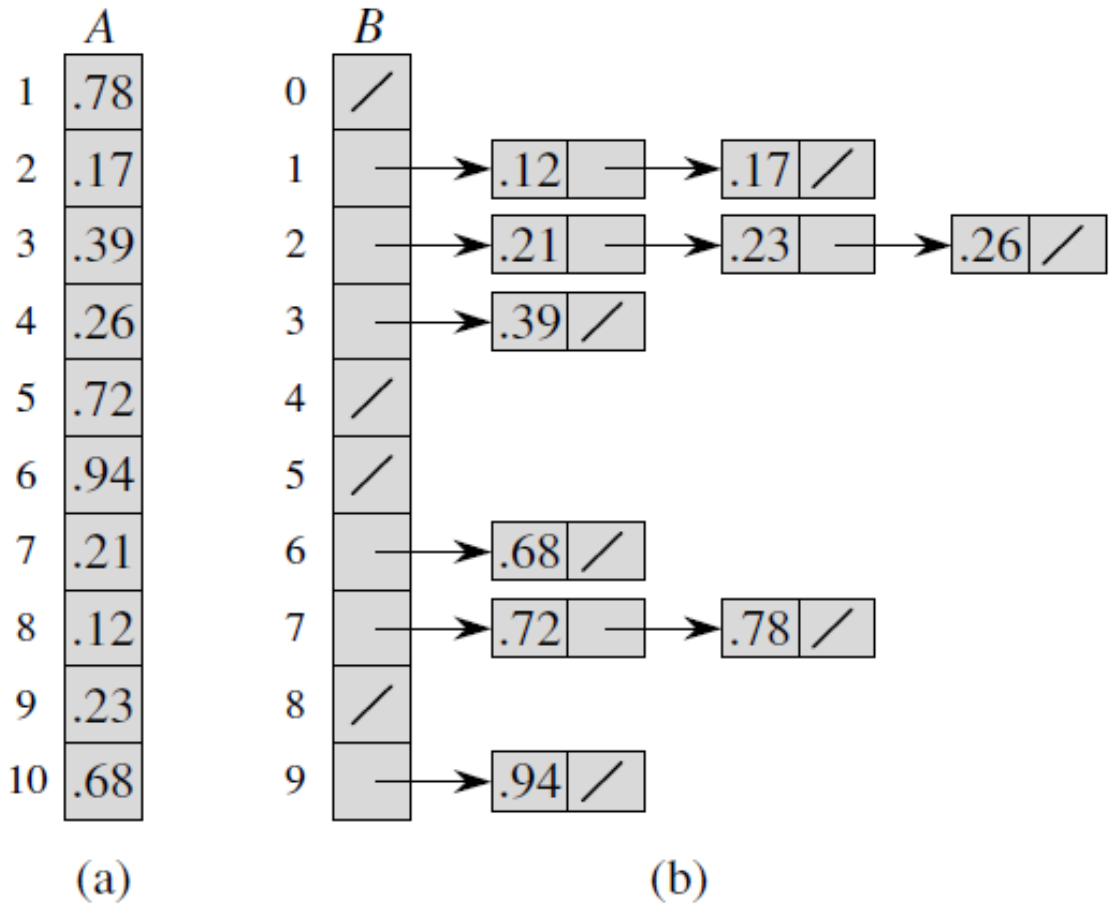
Radix Sort

Lemma 8.4

Given n b-bit numbers and any positive integer $r \leq b$, RADIX-SORT correctly sorts these numbers in $\Theta((b/r)(n + 2^r))$ time.

Proof For a value $r \leq b$, we view each key as having $d = \lceil b/r \rceil$ digits of r bits each. Each digit is an integer in the range 0 to $2^r - 1$, so that we can use counting sort with $k = 2^r - 1$. (For example, we can view a 32-bit word as having 4 8-bit digits, so that $b = 32$, $r = 8$, $k = 2^r - 1 = 255$, and $d = b/r = 4$.) Each pass of counting sort takes time $\Theta(n + k) = \Theta(n + 2^r)$ and there are d passes, for a total running time of $\Theta(d(n + 2^r)) = \Theta((b/r)(n + 2^r))$. ■

Bucket Sort


A
1 .78
2 .17
3 .39
4 .26
5 .72
6 .94
7 .21
8 .12
9 .23
10 .68
B
0
1 .12 .17
2 .21 .23 .26
3 .39
4
5
6 .68
7 .72 .78
8
9 .94
(a)
(b)

Bucket Sort

```
BUCKET-SORT(A)
1  n ← length[A]
2  for i ← 1 to n
3        do insert A[i] into list B[⌊n A[i]⌋]
4  for i ← 0 to n − 1
5        do sort list B[i] with insertion sort
6  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
```

Bucket Sort

```
BUCKET-SORT(A)
1  n ← length[A]
2  for i ← 1 to n
3        do insert A[i] into list B[⌊n A[i]⌋]
4  for i ← 0 to n − 1
5        do sort list B[i] with insertion sort
6  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
```

Sorting in Linear Time

8-3 Sorting variable-length items

a. You are given an array of integers, where different integers may have different numbers of digits, but the total number of digits over *all* the integers in the array is n. Show how to sort the array in $O(n)$ time.

b. You are given an array of strings, where different strings may have different numbers of characters, but the total number of characters over all the strings is n. Show how to sort the strings in $O(n)$ time.

(Note that the desired order here is the standard alphabetical order; for example, `a` < `ab` < `b`.)

Sorting in Linear Time

8-5 Average sorting

Suppose that, instead of sorting an array, we just require that the elements increase on average. More precisely, we call an n-element array A ***k-sorted*** if, for all $i = 1, 2, \ldots, n - k$, the following holds:

$$\frac{\sum_{j=i}^{i+k-1} A[j]}{k} \leq \frac{\sum_{j=i+1}^{i+k} A[j]}{k} .$$

a. What does it mean for an array to be 1-sorted?

b. Give a permutation of the numbers $1, 2, \ldots, 10$ that is 2-sorted, but not sorted.

c. Prove that an n-element array is k-sorted if and only if $A[i] \leq A[i + k]$ for all $i = 1, 2, \ldots, n - k$.

d. Give an algorithm that k-sorts an n-element array in $O(n \lg(n/k))$ time.

We can also show a lower bound on the time to produce a k-sorted array, when k is a constant.

e. Show that a k-sorted array of length n can be sorted in $O(n \lg k)$ time. (*Hint:* Use the solution to Exercise 6.5-8.)

f. Show that when k is a constant, it requires $\Omega(n \lg n)$ time to k-sort an n-element array. (*Hint:* Use the solution to the previous part along with the lower bound on comparison sorts.)