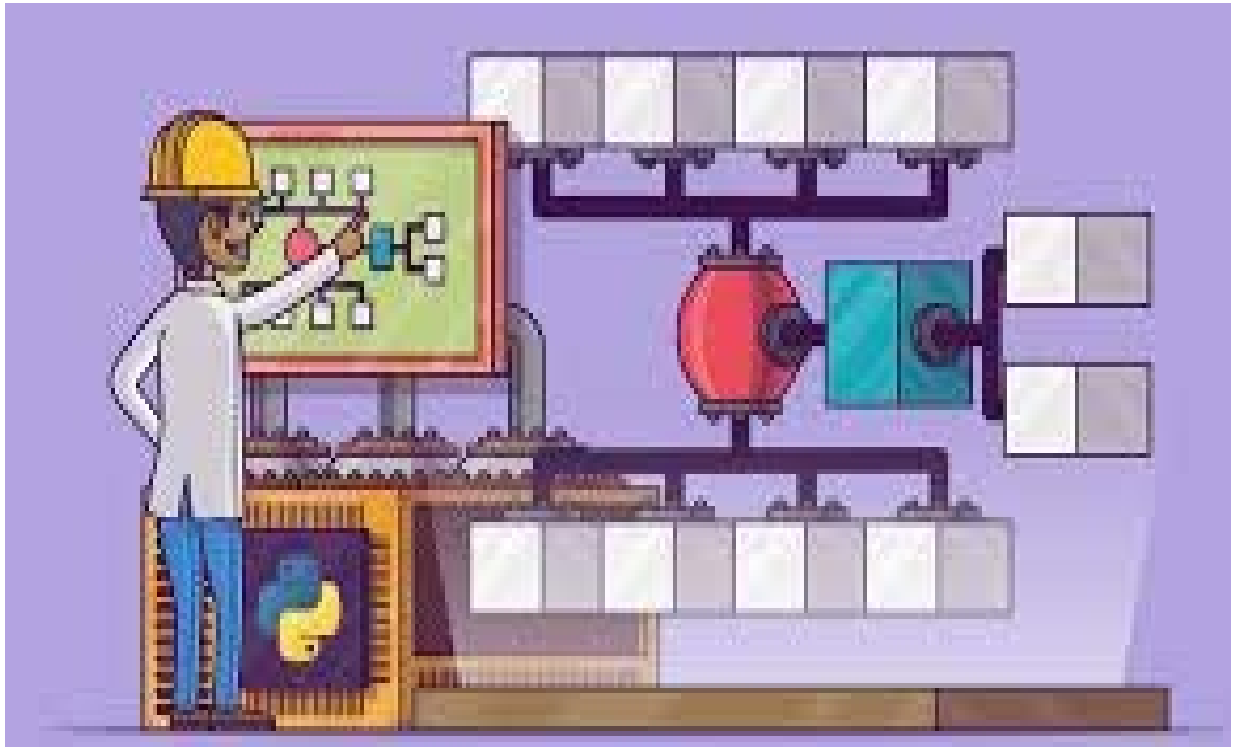
ساختمان داده ها



مدرس:
سمانه حسینی سمنانی

دانشگاه صنعتی اصفهان- دانشکده برق و کامپیوتر

مرتب سازی – sorting

- Comparison algorithms
  - Insertion sort
  - Merge sort
  - Heap sort
  - Quick sort
- Non-Comparison algorithms
  - Counting sort
  - Radix sort
  - Bucket sort

سمانه حسینی سمنانی
هیات علمی دانشکده برق و کامپیوتر- دانشگاه صنعتی اصفهان

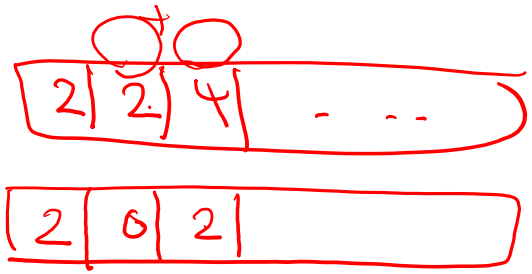
Counting sort

assumes that each of the n input elements is an integer in the range 0 to k, for some integer k.

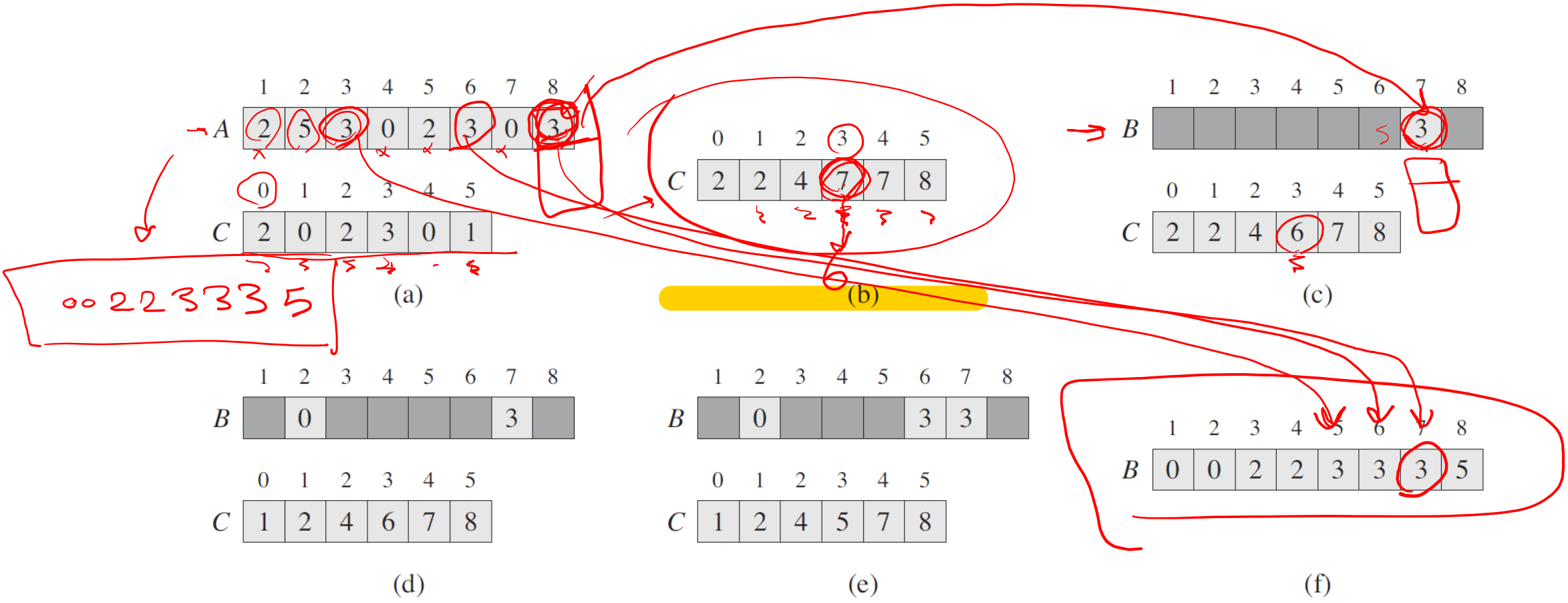
سمانه حسینی سمنانی
هیات علمی دانشکده برق و کامپیوتر- دانشگاه صنعتی اصفهان

Counting sort

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



Counting sort



Counting sort

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

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

$$k = O(n),$$

$$\Theta(n).$$

stable: numbers with the same value appear in the output array in the same order as they do in the input array.

Counting sort

- The property of stability is important:

1. when satellite data are carried around with the element being sorted.

2. Counting sort is often used as a subroutine in radix sort.

- In order for radix sort to work correctly, counting sort must be stable.

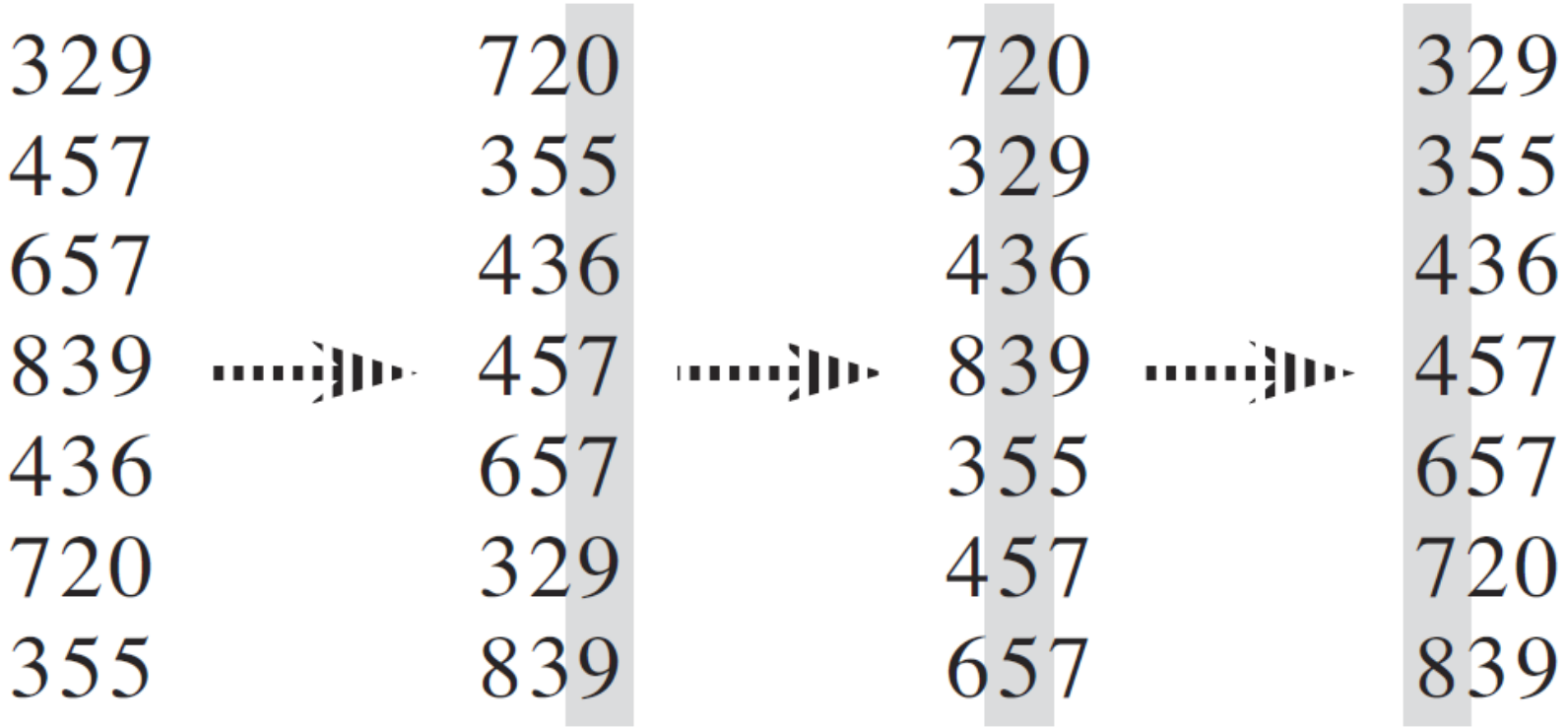
Radix sort

329
457
657
839
436
720
355

Radix sort

- Intuitively, you might sort numbers on their most significant digit
- Radix sort solves the problem of card sorting—counterintuitively—by sorting on the least significant digit first.

Radix sort



سمانه حسینی سمنانی
هیات علمی دانشکده برق و کامپیوتر- دانشگاه صنعتی اصفهان

Radix sort

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


سمانه حسینی سمنانی
هیات علمی دانشکده برق و کامپیوتر- دانشگاه صنعتی اصفهان

Radix sort

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 if the stable sort it uses takes $\Theta(n+k)$ time.

سمانه حسینی سمنانی
هیات علمی دانشکده برق و کامپیوتر- دانشگاه صنعتی اصفهان

Radix sort

1. Number of children
2. Age
3. Years of paid working

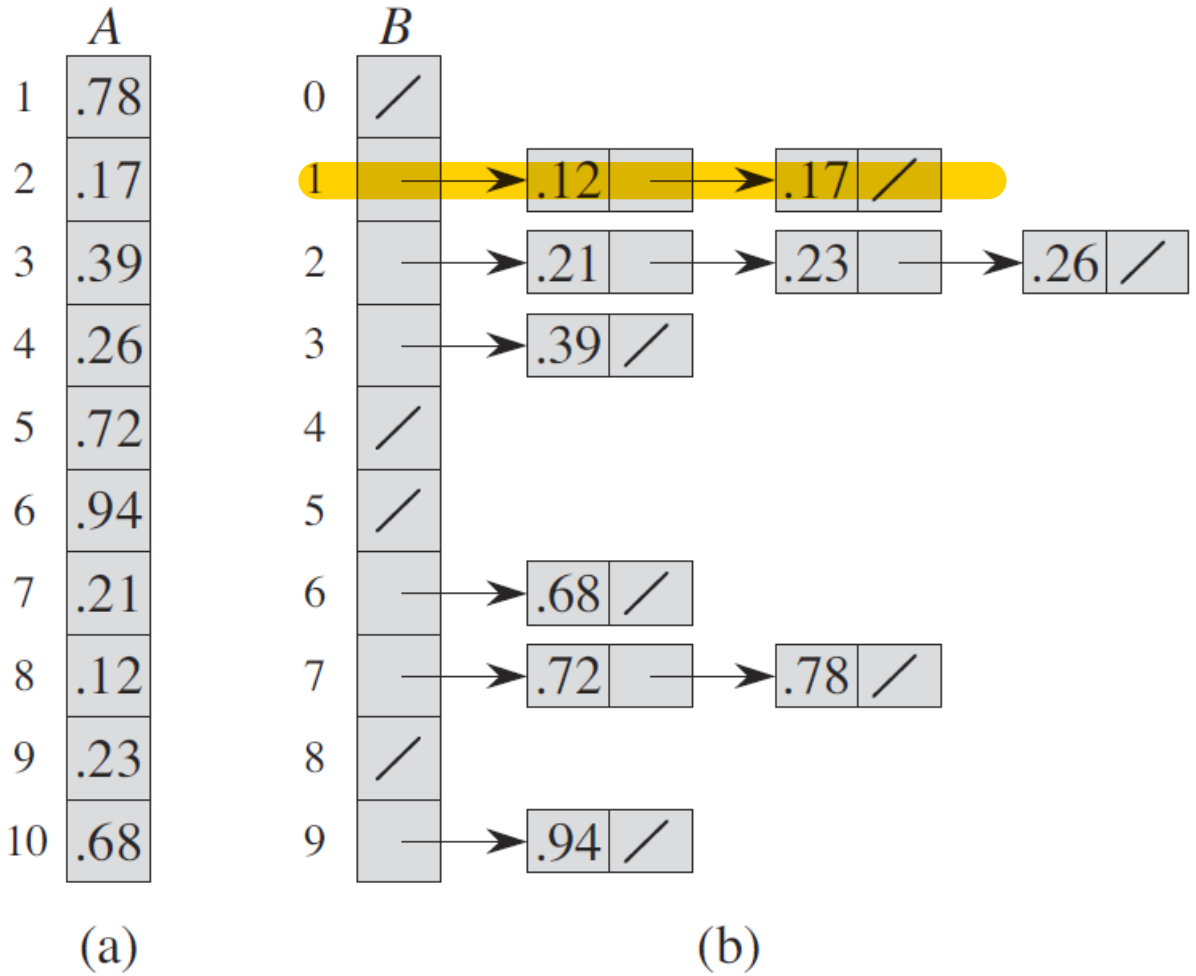
Bucket sort

- Like counting sort, bucket sort is fast because it assumes something about the input.
- counting sort assumes that the input consists of integers in a small range
- bucket sort assumes that the input is generated by a random process that distributes elements uniformly and independently over the interval [0, 1).
- Bucket sort divides the interval [0, 1) into n equal-sized subintervals or buckets
- Distributes the n input numbers into the buckets

Bucket sort

- we simply sort the numbers in each bucket and then go through the buckets in order, listing the elements in each
- bucket sort assumes that the input is an n-element array A and that each element A[i] in the array satisfies $0 < A[i] < 1$.
- We need an auxiliary array B[0…n-1] of linked lists (buckets) and assumes that there is a mechanism for maintaining such lists.

Bucket sort


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

Bucket sort

```
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[⌊n A[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
```

Bucket sort

To see that this algorithm works, consider two elements $A[i]$ and $A[j]$. Assume without loss of generality that $A[i] \leq A[j]$. Since $\lfloor n\,A[i] \rfloor \leq \lfloor n\,A[j] \rfloor$, either element $A[i]$ goes into the same bucket as $A[j]$ or it goes into a bucket with a lower index. If $A[i]$ and $A[j]$ go into the same bucket, then the **for** loop of lines 7–8 puts them into the proper order. If $A[i]$ and $A[j]$ go into different buckets, then line 9 puts them into the proper order. Therefore, bucket sort works correctly.

Bucket sort: analyze

- Since insertion sort runs in quadratic time

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

سمانه حسینی سمنانی
هیات علمی دانشکده برق و کامپیوتر- دانشگاه صنعتی اصفهان

Bucket sort

Algorithm	Worst-case running time	Average-case/expected running time
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Heapsort	$O(n \lg n)$	—
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (expected)
Counting sort	$\Theta(k + n)$	$\Theta(k + n)$
Radix sort	$\Theta(d(n + k))$	$\Theta(d(n + k))$
Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (average-case)