Chapter 1: Introduction to Algorithms

Part IV

Sorting in linear time

Sorting algorithms that run in linear time

- Counting sort
- Radix sort
- Bucket sort

Readings

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Chapter 8: Sorting in Linear Time. *In Introduction to algorithms*. MIT press.

Andersson, A., Hagerup, T., Nilsson, S., & Raman, R. (1998). Sorting in linear time?. *Journal of Computer and System Sciences*, 57(1), 74-93.

Comparison sorts

Sorting algorithms studied so far are **comparison sorts**, i.e. they are based on comparisons of input elements.

Sorting algorithm	Time complexity
Insertion sort	$O(n^2)$
Selection sort	$O(n^2)$
Merge sort	$O(n \lg n)$
Heap sort	$O(n \lg n)$
Quick sort	$O(n^2)$

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

Counting sort

- Assumes that each of the n input elements is an integer in the range from 0 to k, for some integer k
- When $k = O(n)$, the sort runs in $\Theta(n)$ time
- Determines, for each input element x, the number of elements less than x.
  - This is how it positions x in its place in the output array
  - If there are 17 elements smaller than x, then x will be assigned position 18
- To sort an array A[1..n], we need two additional arrays
  - B[1..n] holds the sorted output
  - C[1..k] stores the number of repetitions of number in A

Counting sort

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

(a)

Counting sort

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

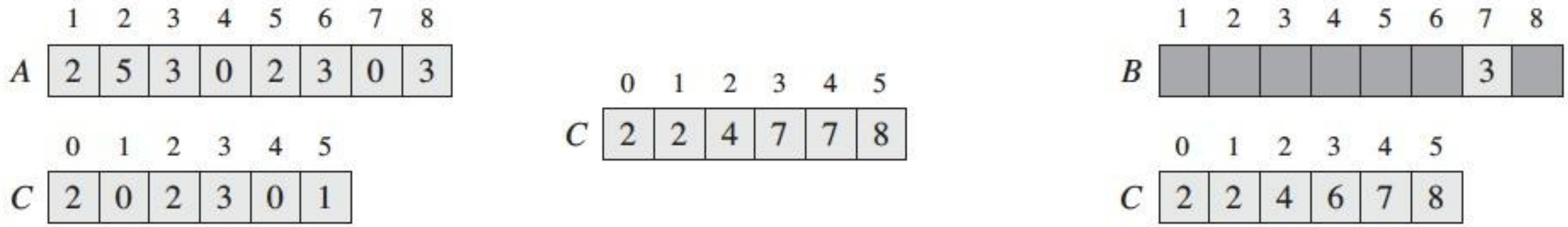
	0	1	2	3	4	5
C	2	0	2	3	0	1

(a)

	0	1	2	3	4	5
C	2	2	4	7	7	8

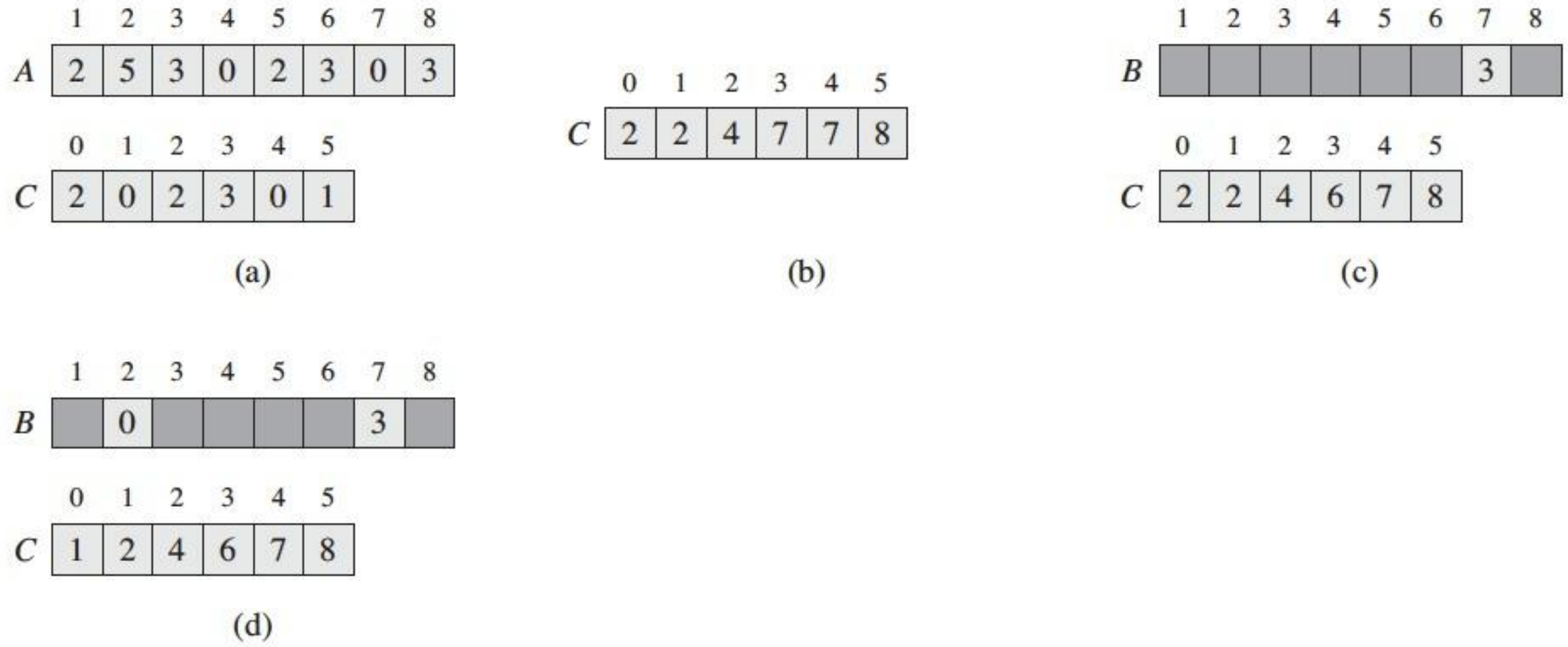
(b)

Counting sort

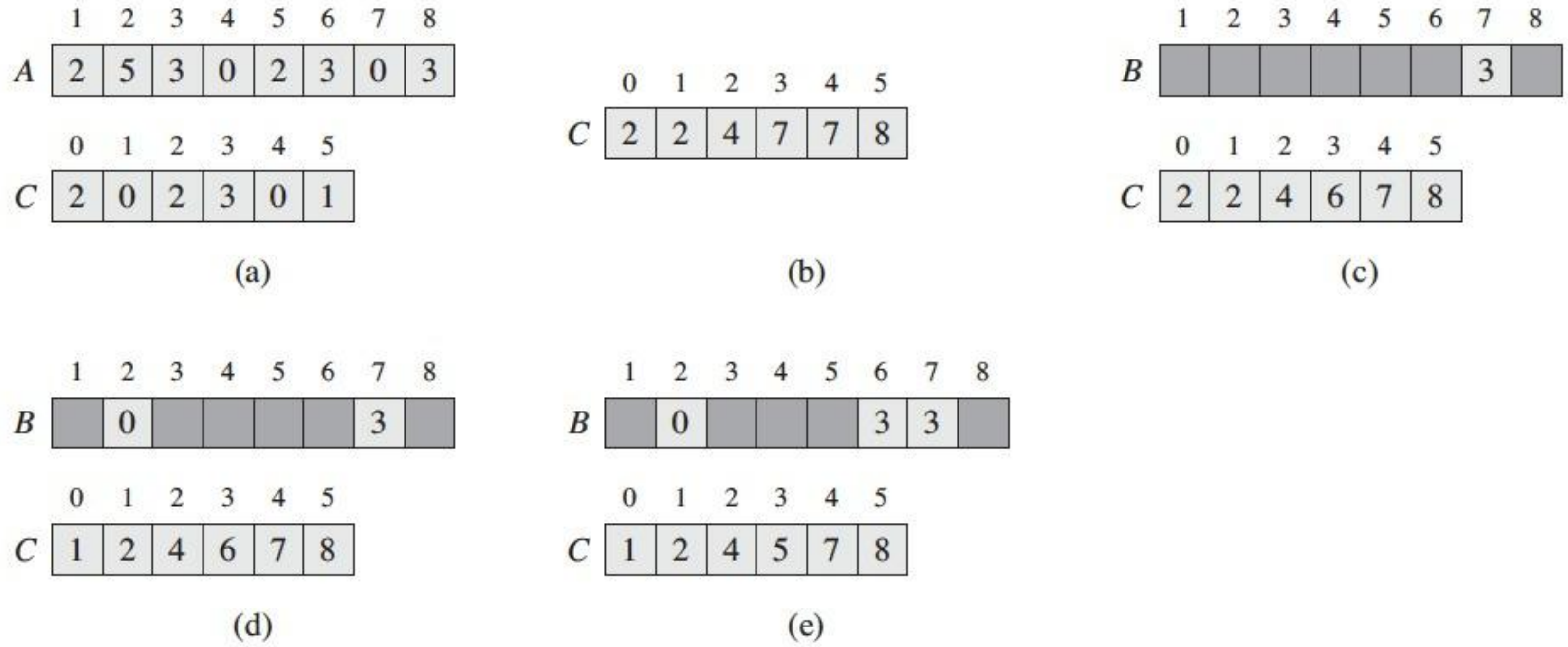


(a) (b) (c)

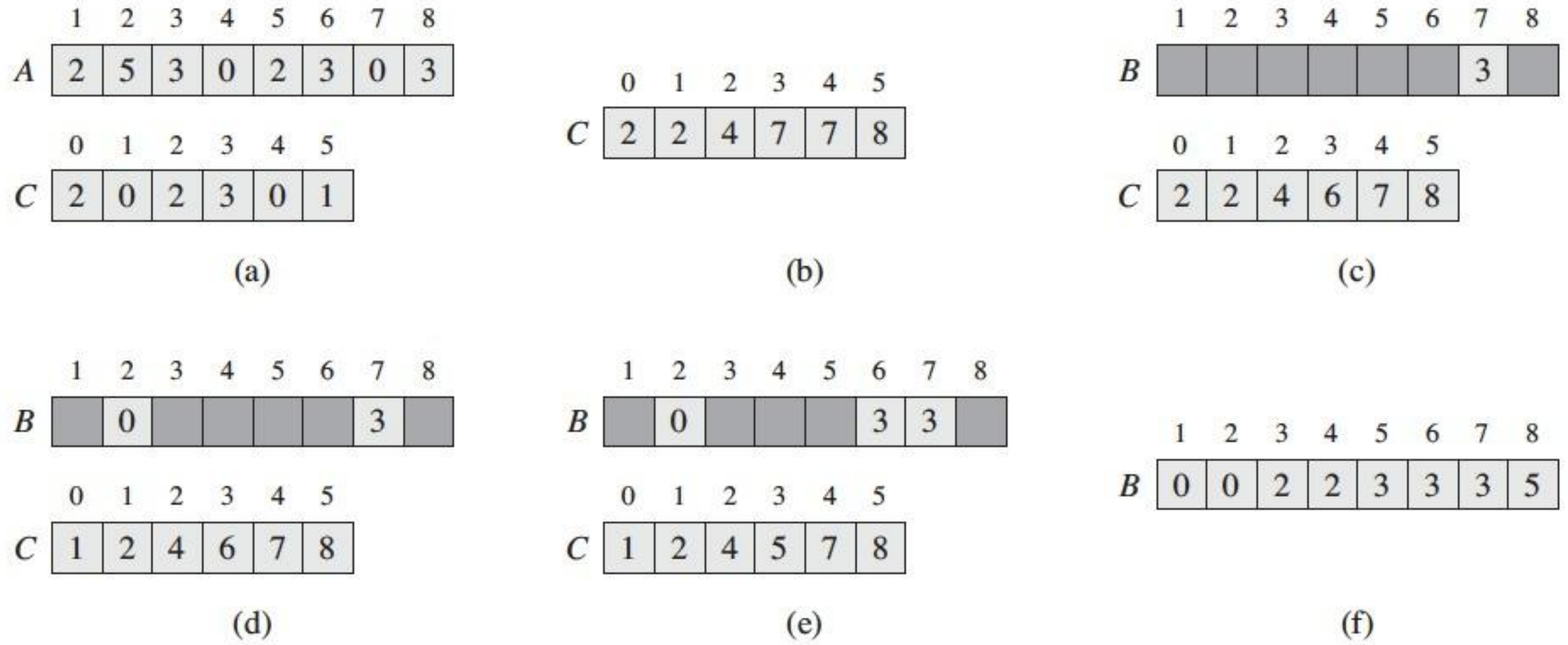
Counting sort



Counting sort



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
```

Line 2 - 3 → $\Theta(k)$

Line 4 - 5 → $\Theta(n)$

Line 7 - 8 → $\Theta(k)$

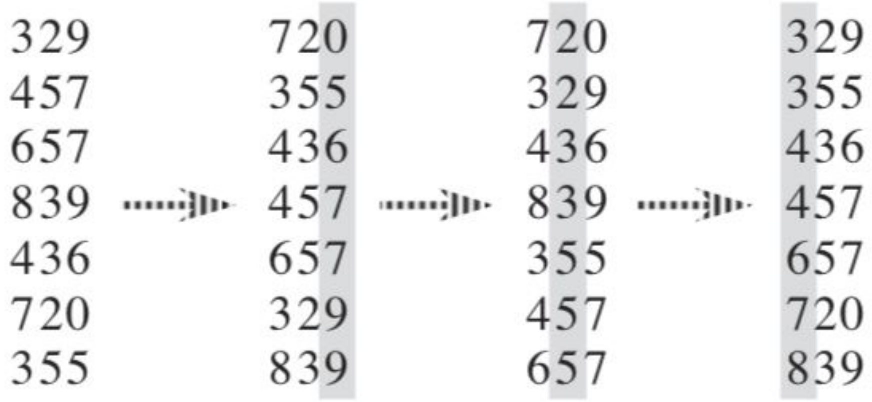
Line 10 - 12 → $\Theta(n)$

Counting sort

- Is **not** a comparison sort
- Beats the lower bound of $\Omega(n \lg n)$
- Is **stable**
- Is often used as a subroutine in radix sort

Radix sort

- The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit.
- In order for radix sort to work correctly, the digit sorts must be stable.
- Radix sort uses counting sort as a subroutine to sort.



Radix sort

RADIX-SORT(A, d)

for $i = 1$ **to** d
    use a stable sort to sort array A on digit i

Each element in the n-element array A has d digits, where digit 1 is the lowest-order digit and digit d is the highest-order digit.

Radix sort

RADIX-SORT(A, d)

```
for i = 1 to d
    use a stable sort to sort array A on digit i
```

When each digit is in the range 0 to k - 1 (so that it can take on k possible values), and k is not too large, counting sort is the obvious choice.

Each step for a digit takes $\Theta(n+k)$.

For d digits $\Theta(dn+dk) \Rightarrow$ The total time for radix sort is $\Theta(d(n+k))$

When d is constant and $k = O(n)$, we can make radix sort run in linear time.

Bucket sort

- Assumes that the input is drawn from a uniform distribution over the interval [0, 1)
- Divides the interval [0, 1) into n equal-sized subintervals, or **buckets**, and then distributes the n input numbers into the buckets
- To produce the output, we sort numbers in each bucket, then go through buckets in order

Bucket sort

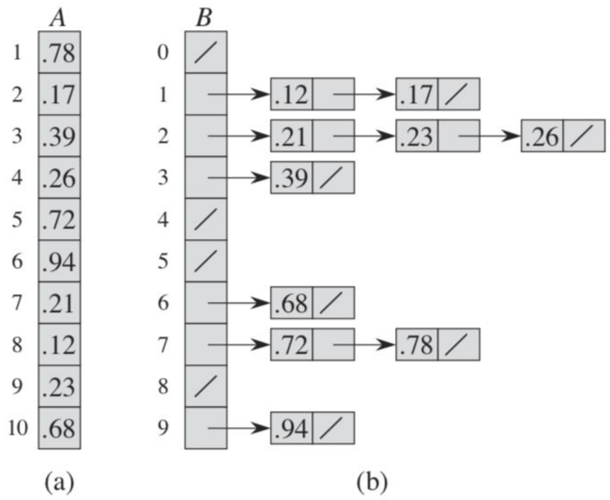


Figure 8.4 The operation of BUCKET-SORT for $n = 10$. **(a)** The input array $A[1..10]$. **(b)** The array $B[0..9]$ of sorted lists (buckets) after line 8 of the algorithm. Bucket i holds values in the half-open interval $[i/10, (i+1)/10)$. The sorted output consists of a concatenation in order of the lists $B[0], B[1], \ldots, B[9]$.

Bucket sort

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