Ch13: Sorting in Linear Time

305233, 305234

Algorithm Analysis and Design

Jiraporn Pooksook
Naresuan University

Comparison sorts

- The sorted order they determine is based only on comparisons between the input elements.
- Any comparison sort must make $\Omega(n \lg n)$ comparisons in the worst case to sort n elements.

Counting Sort(A,B,k)

```
for  i=0 to k
        do C[ i ] = 0
for  j=1 to length[A]
        do C[A[ j ]] = C[A[ j ]]+1
for i = 1 to k
        do C[i] = C[i] + C[i-1]
for j=length[A] downto 1
        do B[C[A[ j ]]] = A[ j ]
        C[A[ j ]] = C[A[ j ]] -1
```

Analyze Counting Sort

- Assume that each of the n input elements is an integer in the range 0 to k, for some integer k.
- Line 1-2, takes time $\Theta(k)$
- Line 3-4 takes time $\Theta(n)$
- Line 5-6 takes time $\Theta(k)$
- Line 7-9 takes time $\Theta(n)$
- Overall, the sort runs in $\Theta(k+n)$ time.
- When we have k = O(n) then the running time is $\Theta(n)$

Example: Counting sort

A

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

C

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

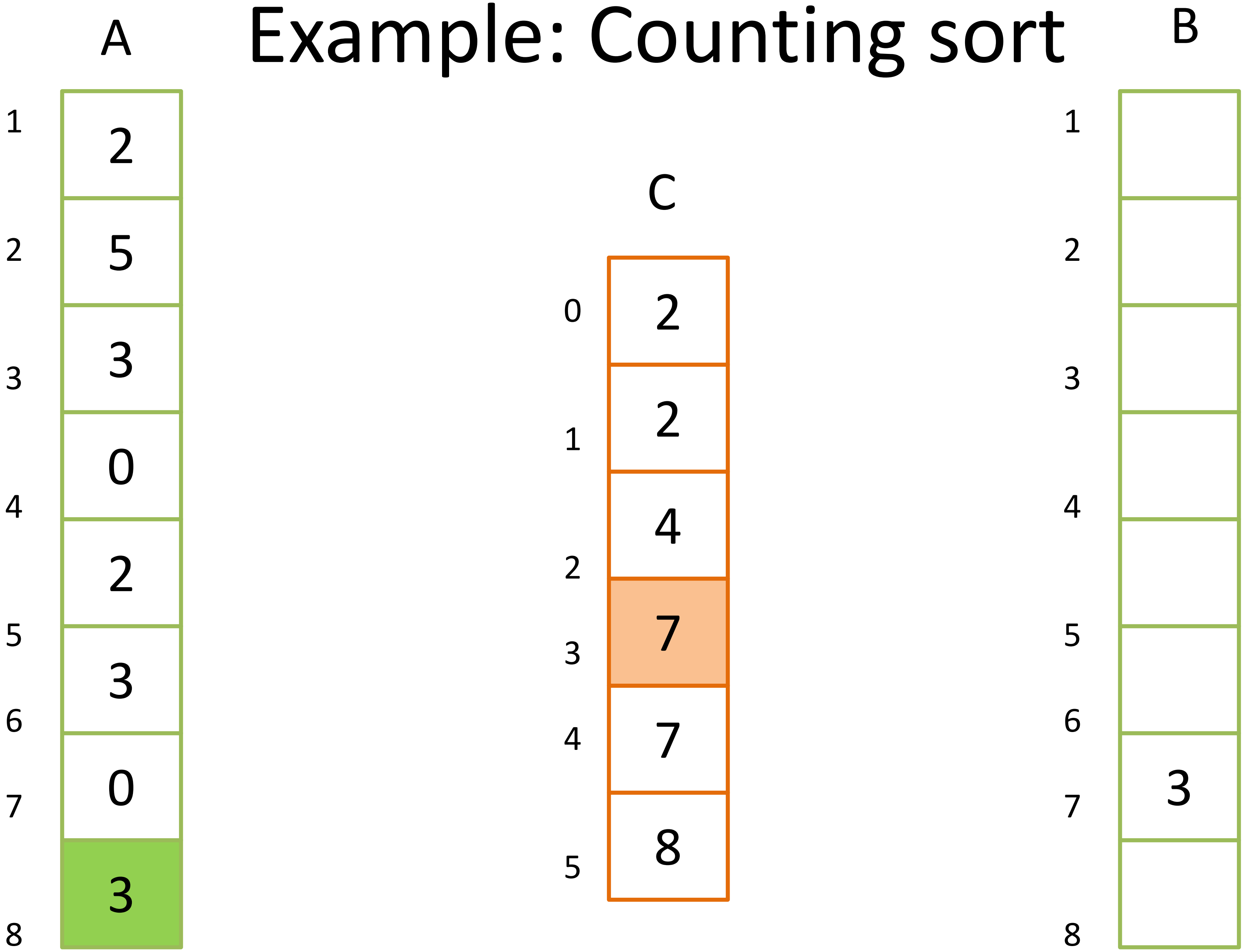
จำนวนเลข สำหรับแต่ละ index มีกี่ตัว

C

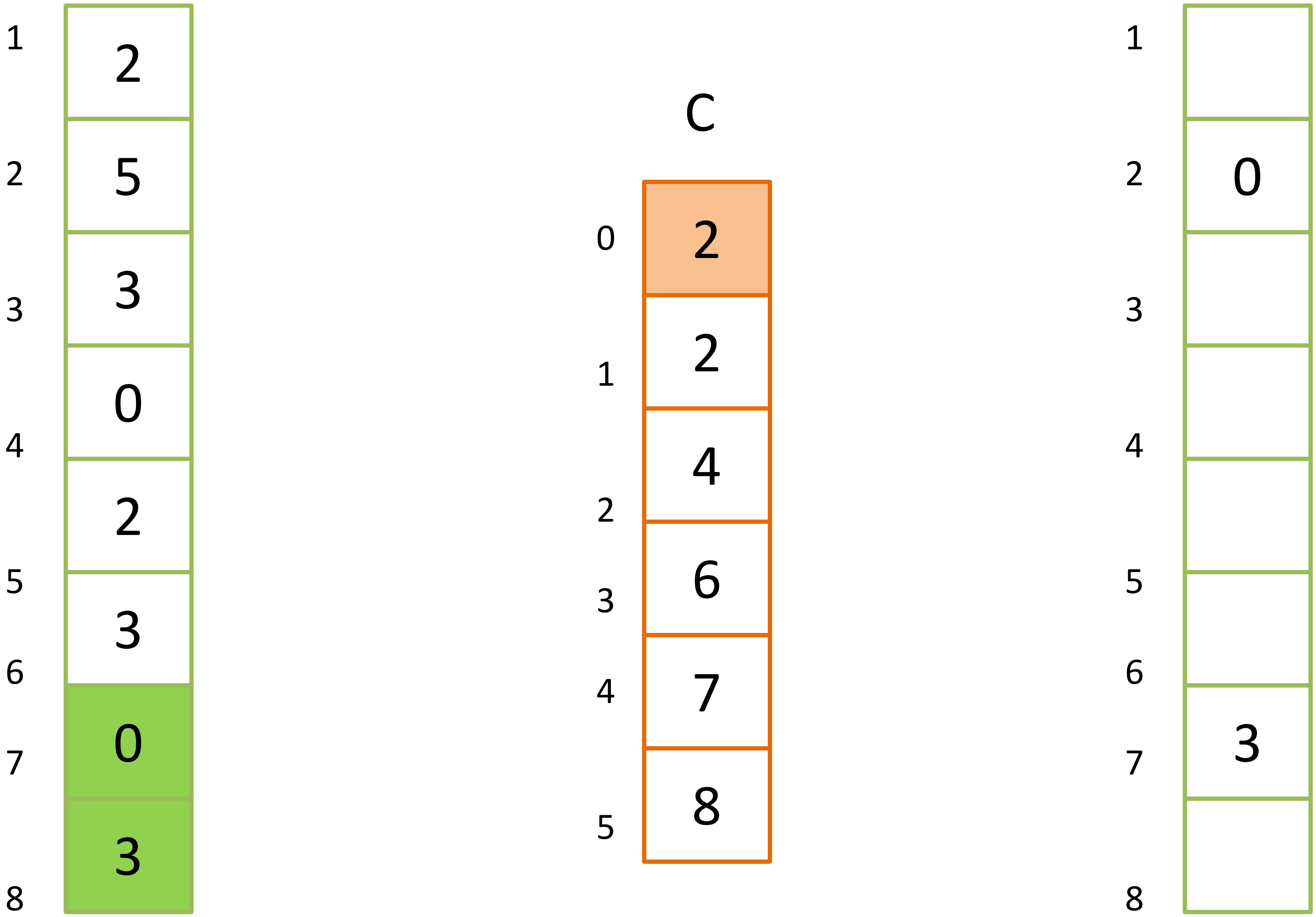
0	2
1	2
2	4
3	7
4	7
5	8

จำนวนเลข ที่น้อยกว่าหรือเท่ากับ สำหรับแต่ละ index มีกี่ตัว

Example: Counting sort



Example: Counting sort



Example: Counting sort

A

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

C

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

B

1	
2	0
3	
4	
5	
6	3
7	3
8	

Example: Counting sort

A

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

C

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

B

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

Example: Counting sort

A

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

C

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

B

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

Example: Counting sort

A

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

C

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

B

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

Example: Counting sort

A

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

C

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

B

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

Example: Counting sort

A

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

C

0	0
1	2
2	3
3	4
4	7
5	7

B

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

Counting sort

- Counting sort is stable.
  - numbers with the same value appear in the output array in th same order as they do in the input array

Radix Sort

- An algorithm used by the card-sorting machines.
- The digit sorts in this algorithm stable.
- Typically a sequential random-access machine sometimes uses radix sort to records of information that are keyed by multiple fields such as sorting dates by three keys: year, month and day.

Radix Sort

```
for  i = 1 to d // d is the highest-order digit
        do use a stable sort to sort array A on
        digit i
```

Example: Radix sort

3	2	9
4	5	7
6	5	7
8	3	9
4	3	6
7	2	0
3	5	5

7	2	0
3	5	5
4	3	6
4	5	7
6	5	7
3	2	9
8	3	9

Example: Radix sort

7	2	0
3	5	5
4	3	6
4	5	7
6	5	7
3	2	9
8	3	9

7	2	0
3	2	9
4	3	6
8	3	9
3	5	5
4	5	7
6	5	7

Example: Radix sort

7	2	0
3	2	9
4	3	6
8	3	9
3	5	5
4	5	7
6	5	7

3	2	9
3	5	5
4	3	6
4	5	7
6	5	7
7	2	0
8	3	9

Analyze Radix Sort

- When each digit is in the range 0 to k-1 and k is not too large, counting sort is an obvious choice.
- Each pass over n d-digit numbers then takes time

$$\Theta(n+k)$$

- There are d passes , then the total time of radix sort is $\Theta(d(n+k))$
- When d is a constant and k = O(n), radix sort runs in linear time.
- Given n b-bit number and any positive integer r ≤ b, radix sort sorts these numbers in

$$\Theta((b/r)(n+2^r))$$

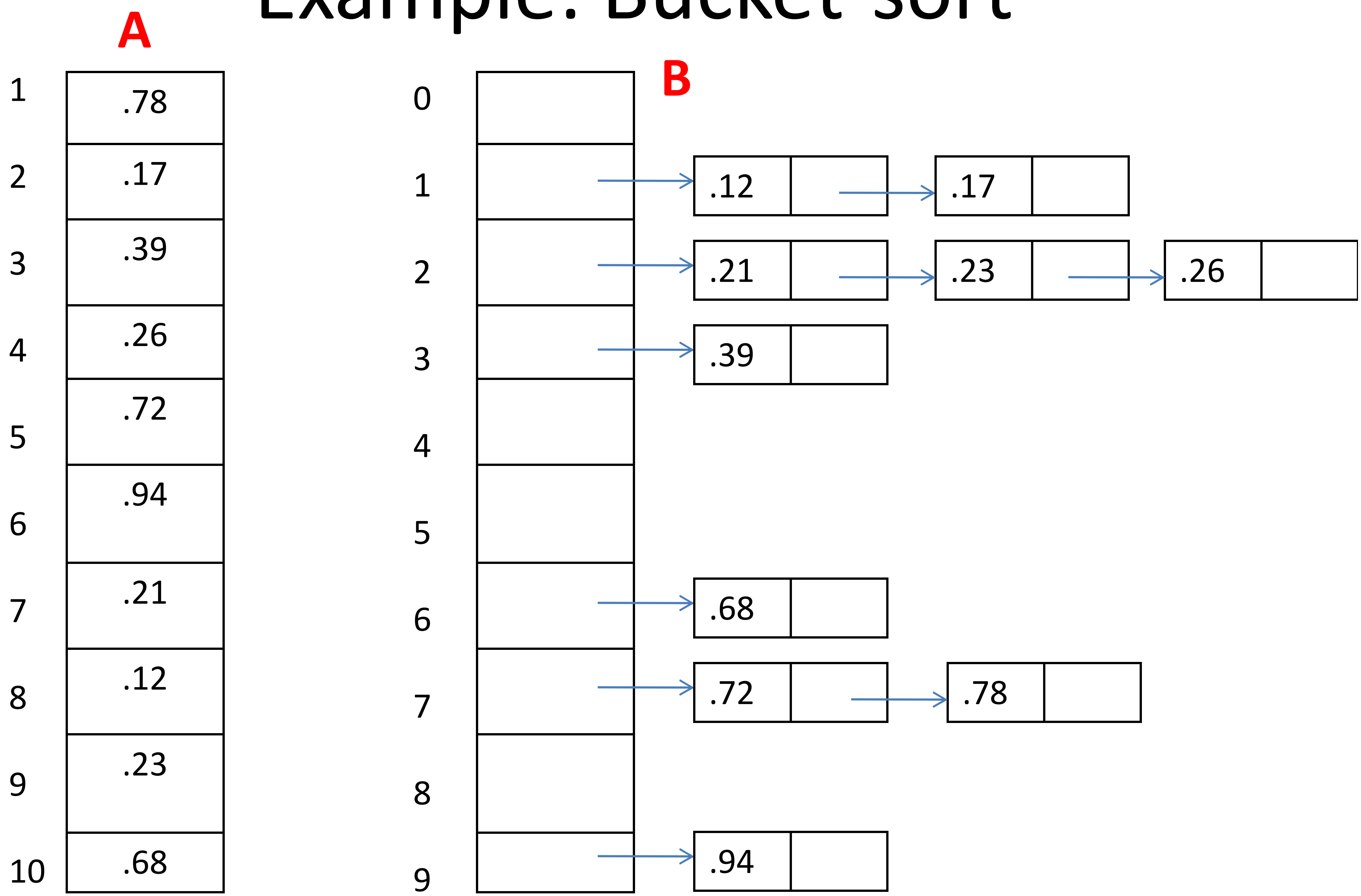
Bucket Sort

- Assume that the input is generated by a random process that distributes elements uniformly over the interval [0,1).
- Divide the interval [0,1) into n equal-sized subintervals, or buckets.
- Distribute the n input numbers into the buckets.
- Sort the numbers in each bucket and go through the buckets in order; listing the elements in each.

Bucket-Sort(A)

```
n = length[A]
for i = 1 to n
        do insert A[i] into list B[⌊nA[i]⌋]
for i = 0 to n-1
        do sort list B[i] with insertion sort
concatenate the lists B[0], B[1], ... , B[n-1]
together in order.
```

Example: Bucket-sort



Analyze Bucket-sort

- The running time depends on line 5.
- Analyze the cost of calling insertion sort in line 5 and the number of expected time we call insertion sort is 2 -1/n
- Hence the running time of bucket sort is

$$T(n) = \Theta(n) + n.O(2 - 1/n) = \Theta(n)$$

Practice : Counting sort

2
5
0
1
1
3
4
1
4
2