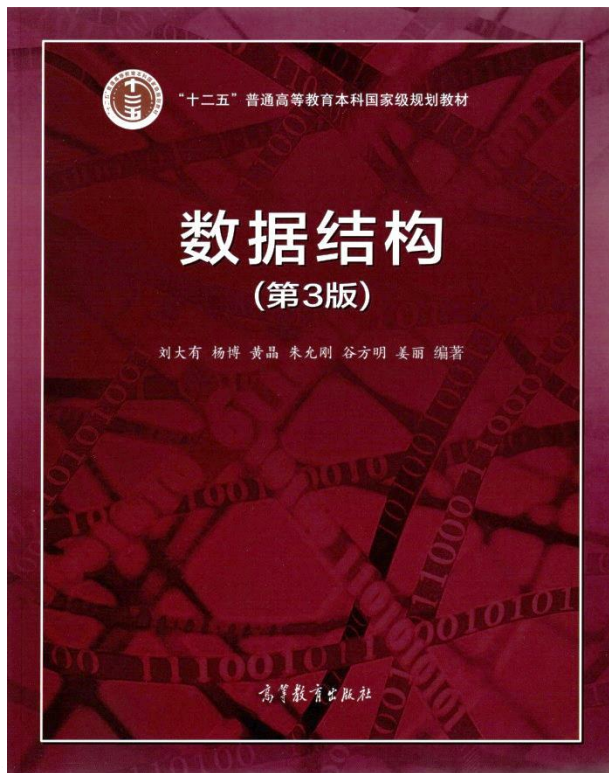




# 分布排序

- 桶排序
- 计数排序
- 基数排序



数据之法  
结构之美  
算法之道



邓明扬

麻省理工学院21级本科生

2020年NOI全国中学生信息学奥赛决赛金牌

2021年IOI世界中学生信息学奥赛冠军

2022年ICPC国际大学生程序设计竞赛全球总决赛冠军

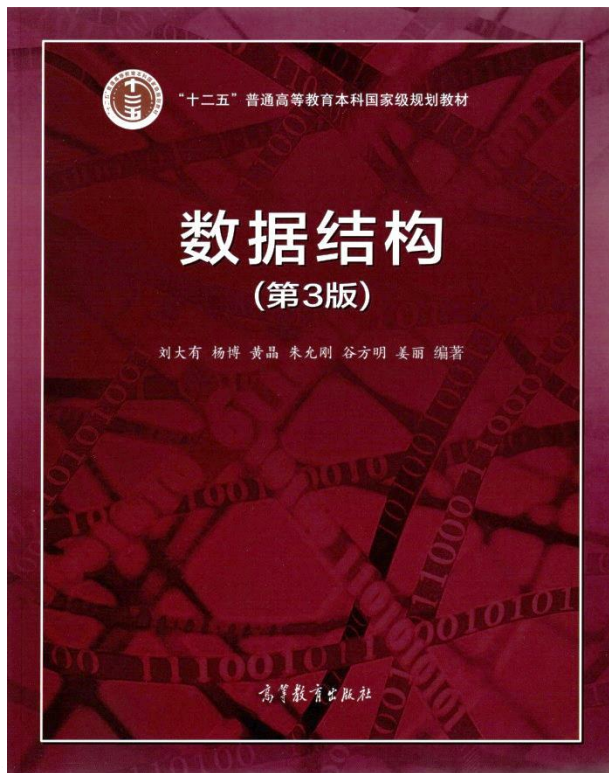
如果你是一个初学者的话，  
要尽量自己调代码，我知道很多  
小伙伴都喜欢找别人帮你调代码，  
但这个东西最后是需要自己练的。  
要多练习，不仅锻炼打代码  
的速度和准确性，也锻炼思考问  
题的能力。





# 分布排序

- 桶排序
- 计数排序
- 基数排序

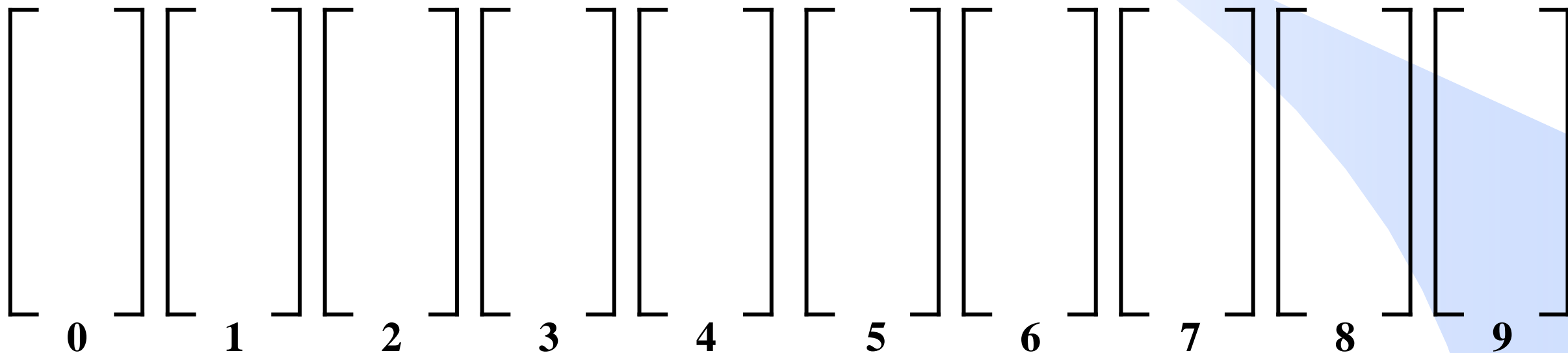


数据之法  
结构之美  
算法之道

# 桶排序 (Bucket Sort)

待排序的文件 $R$ 包含 $n$ 个整数，每个整数的值域为 $[0, m)$

9 6 3 5 2 0 1 5\*

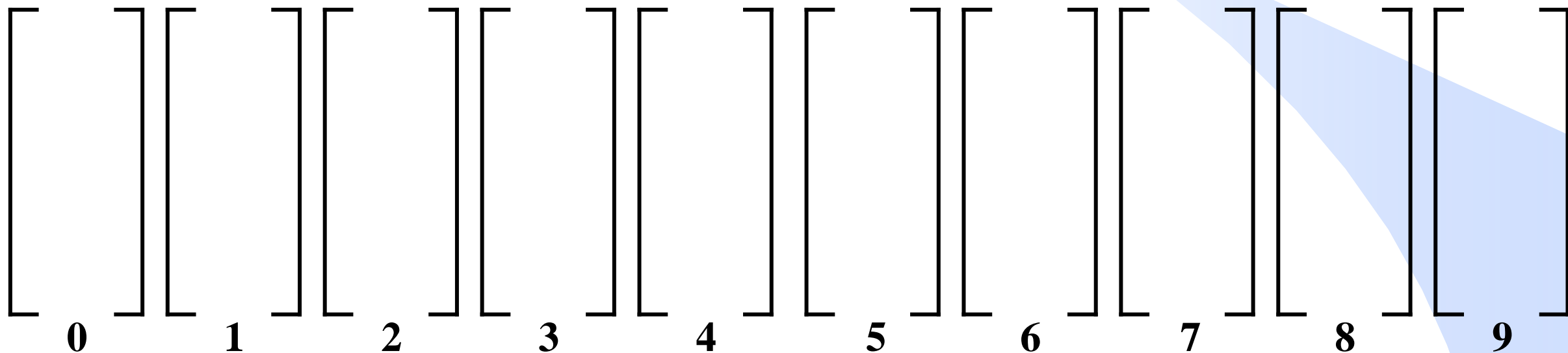


设置  $m$  个桶，对应元素的值域  $0 \dots m-1$

# 桶排序

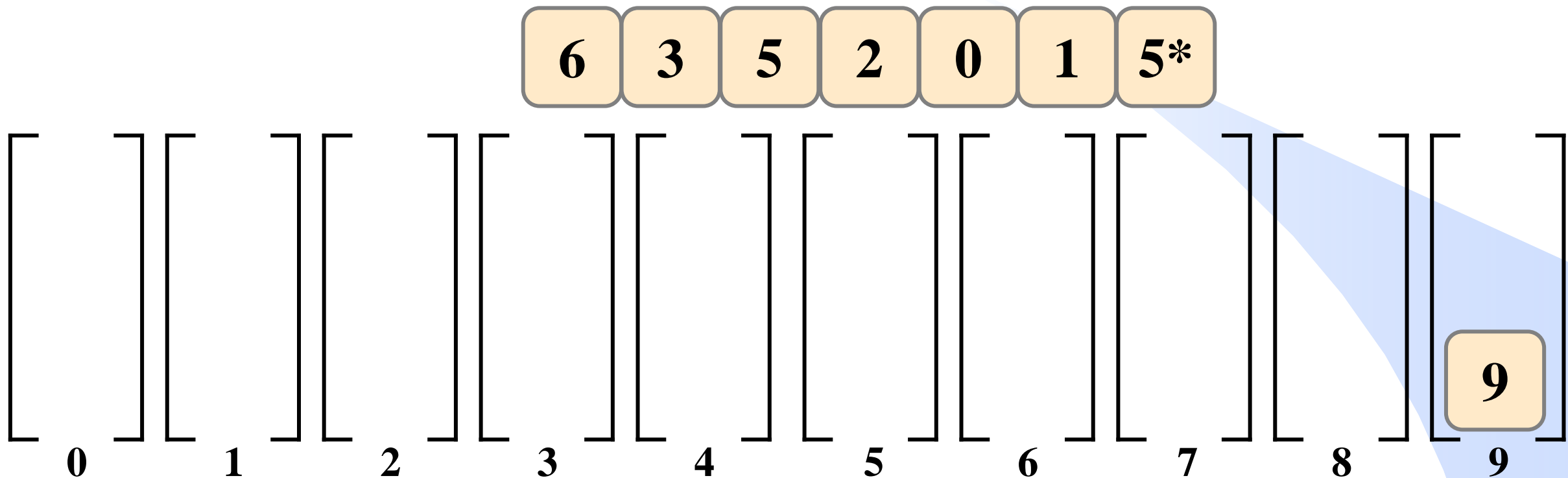
① 分配：扫描 $n$ 个元素，按关键词放入对应的桶中

9 6 3 5 2 0 1 5\*



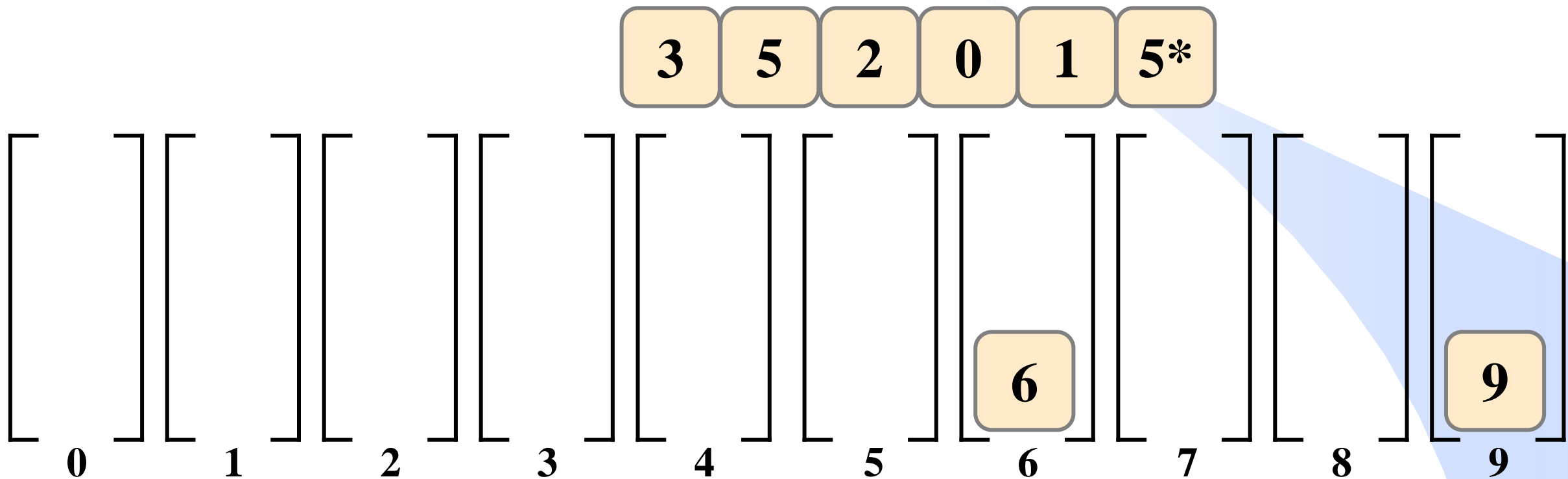
# 桶排序

① 分配：扫描 $n$ 个元素，按关键词放入对应的桶中



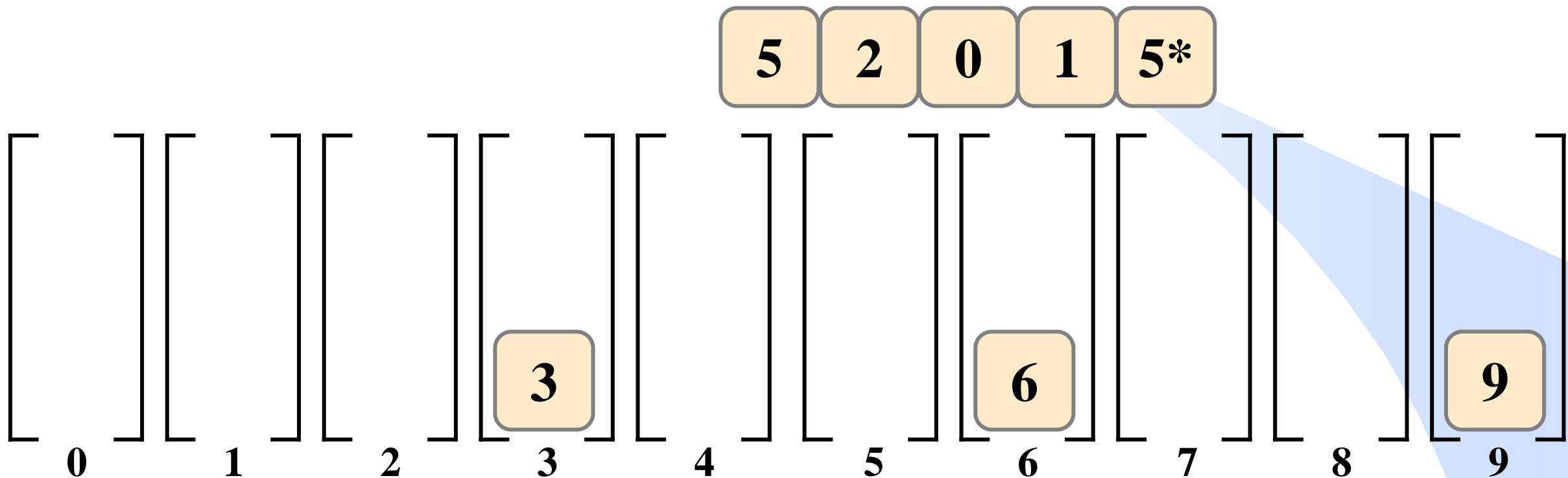
# 桶排序

① 分配：扫描 $n$ 个元素，按关键词放入对应的桶中



# 桶排序

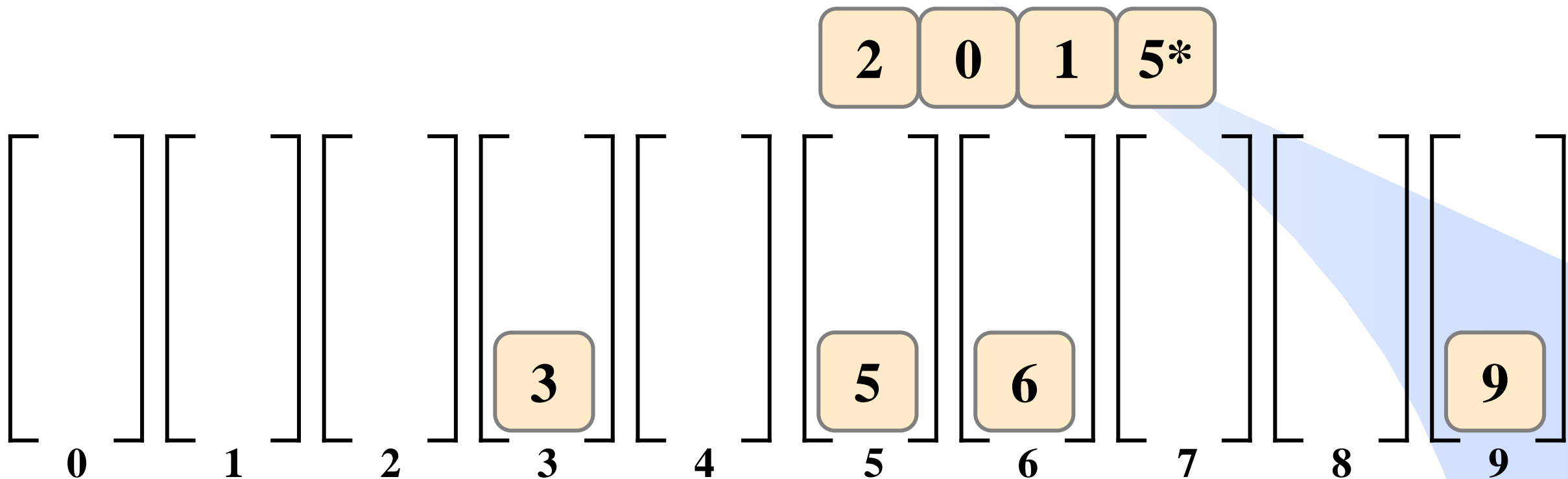
① 分配：扫描 $n$ 个元素，按关键词放入对应的桶中





# 桶排序

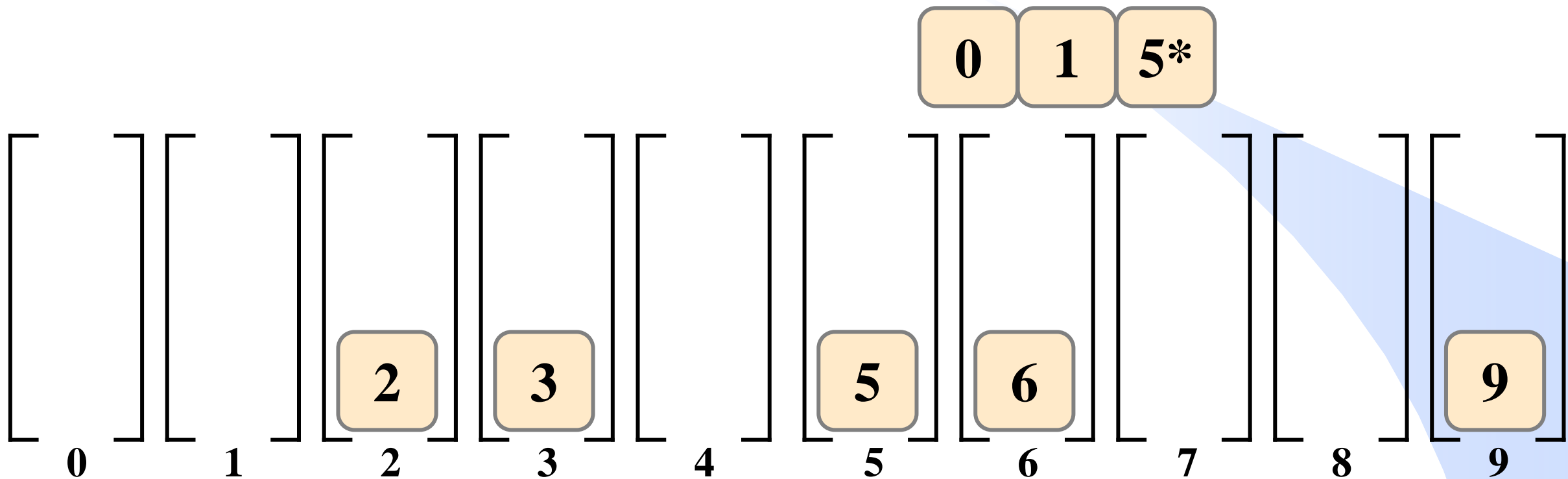
① 分配：扫描 $n$ 个元素，按关键词放入对应的桶中



# 桶排序

A

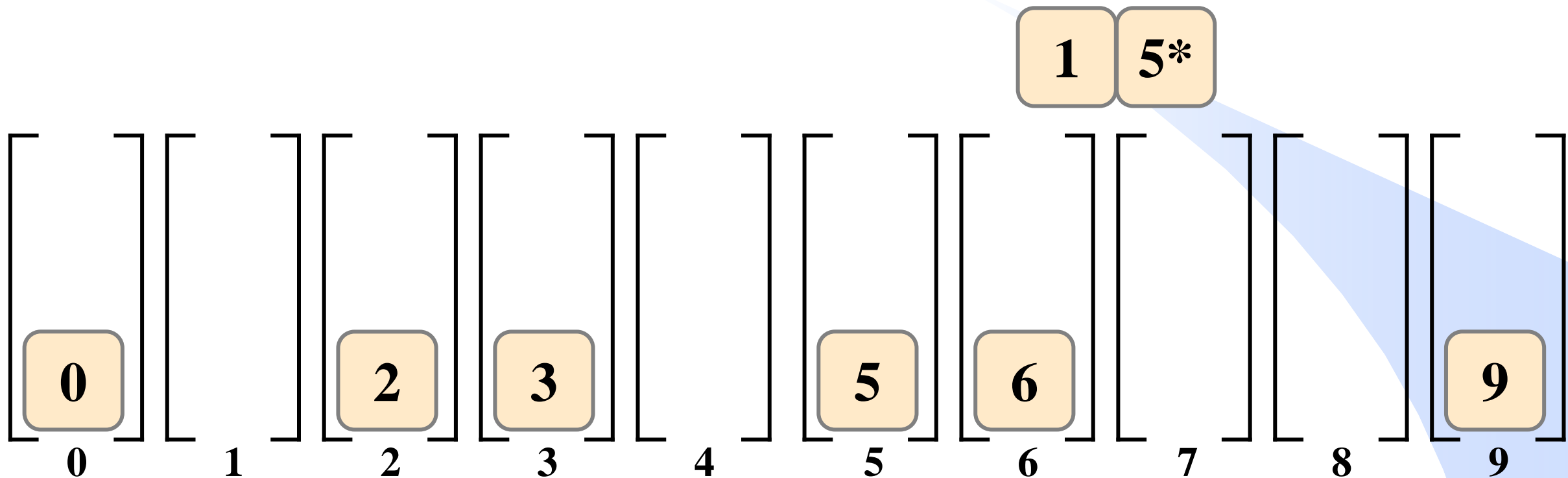
① 分配：扫描 $n$ 个元素，按关键词放入对应的桶中



# 桶排序

A

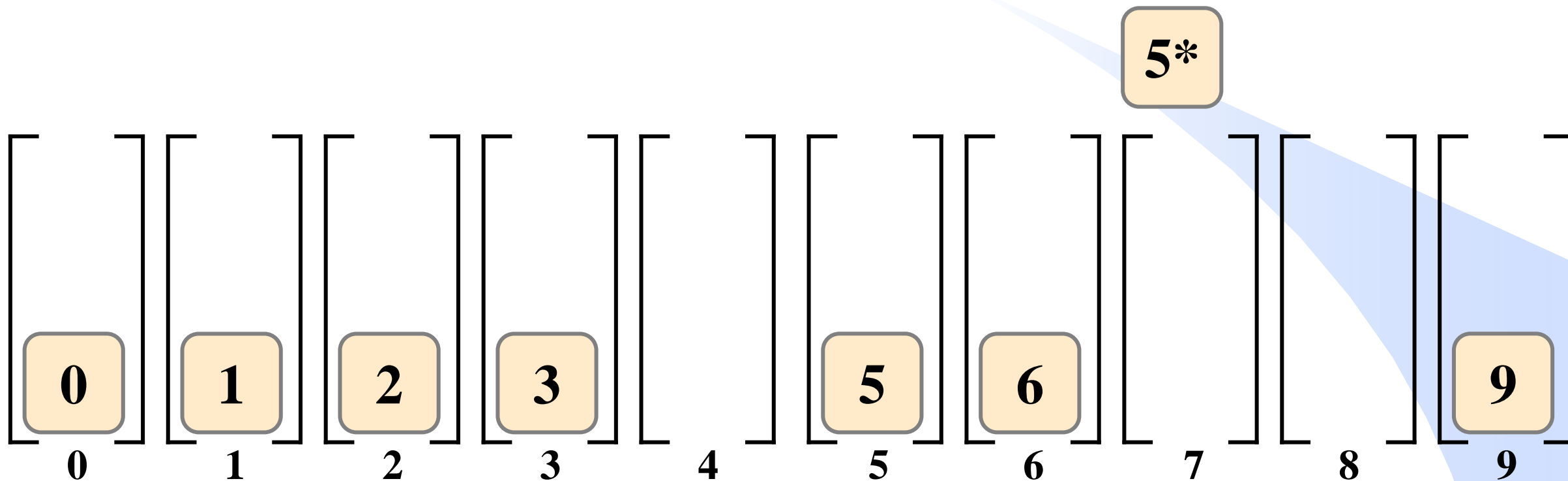
① 分配：扫描 $n$ 个元素，按关键词放入对应的桶中



# 桶排序

A

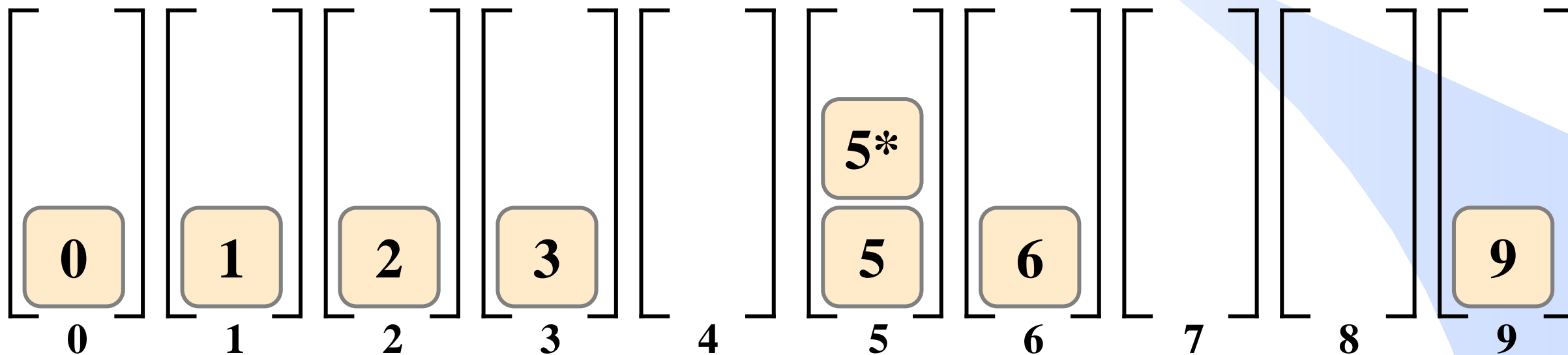
① 分配：扫描 $n$ 个元素，按关键词放入对应的桶中



# 桶排序

A

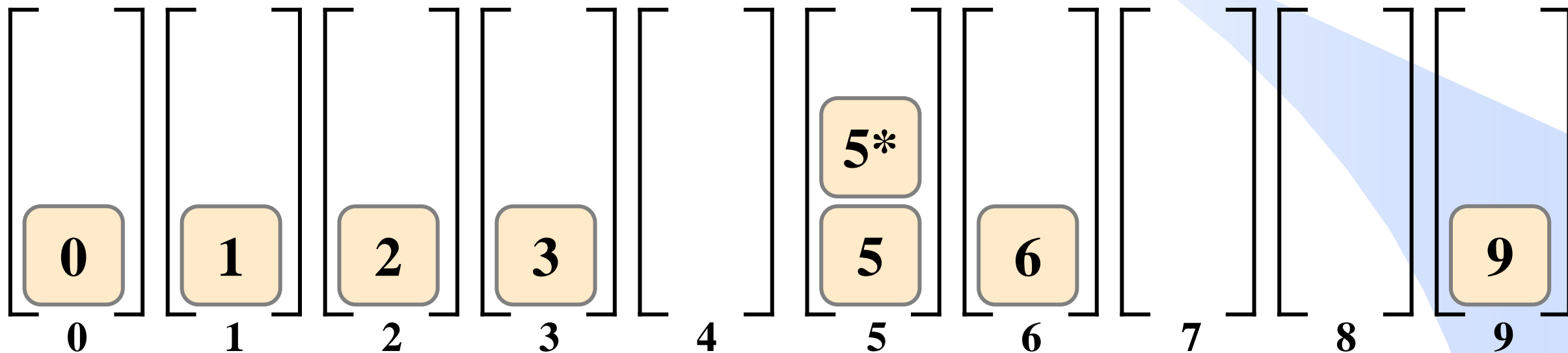
① 分配：扫描 $n$ 个元素，按关键词放入对应的桶中



# 桶排序

A

② 收集：依次将桶中元素取出

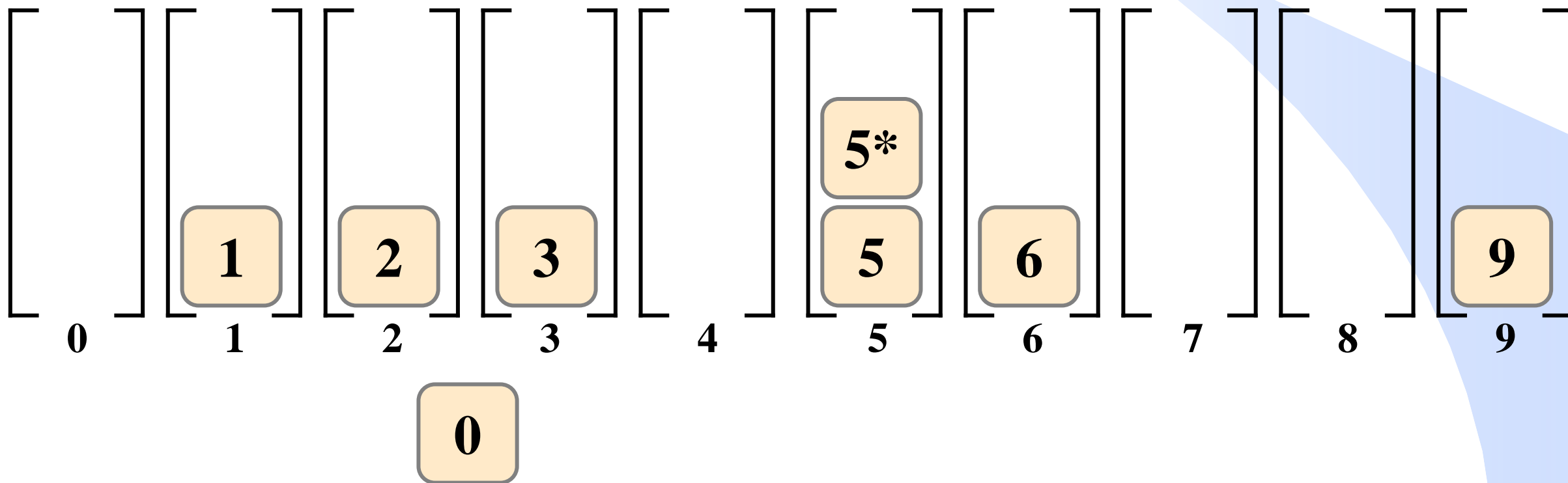




# 桶排序

A

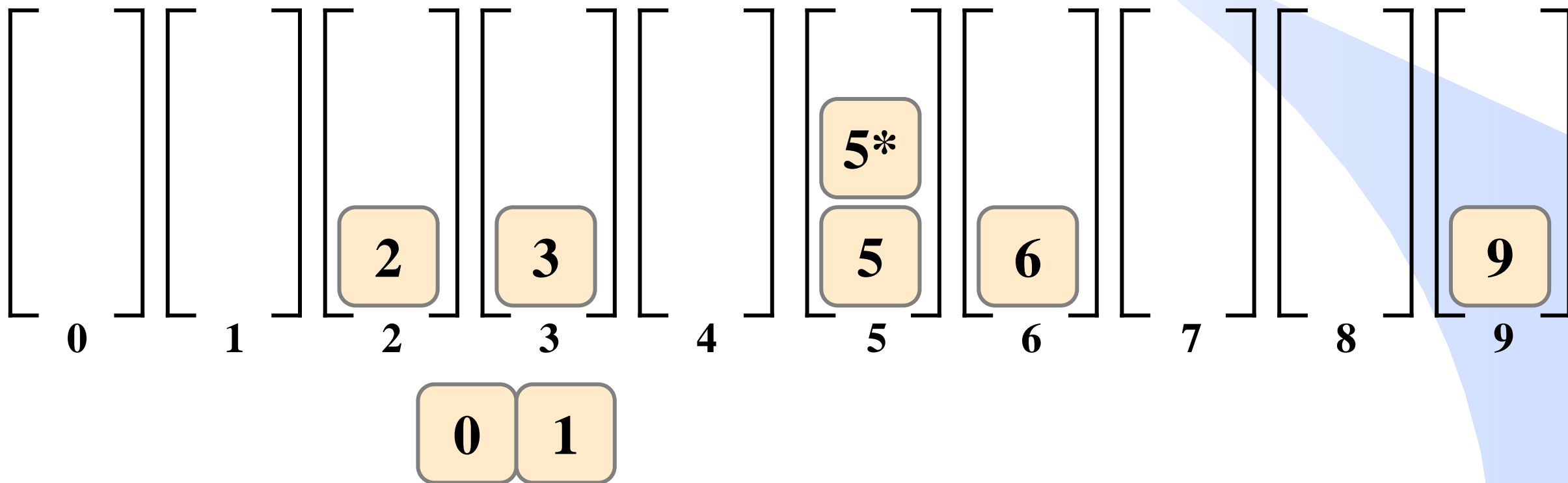
② 收集：依次将桶中元素取出



# 桶排序

A

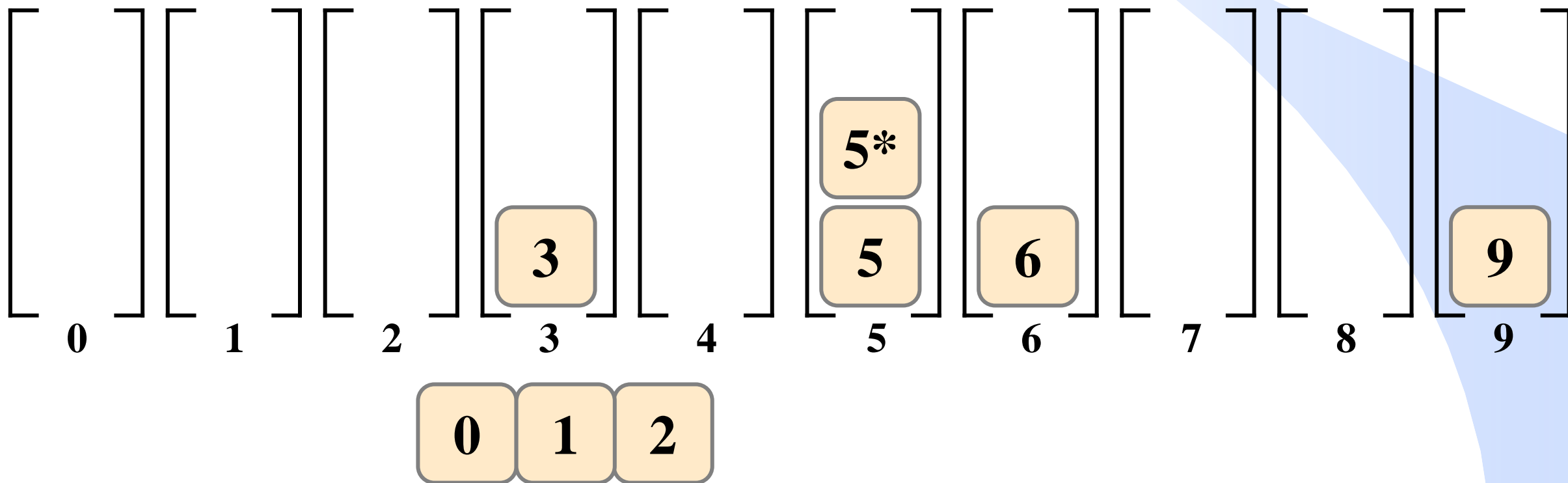
② 收集：依次将桶中元素取出



# 桶排序

A

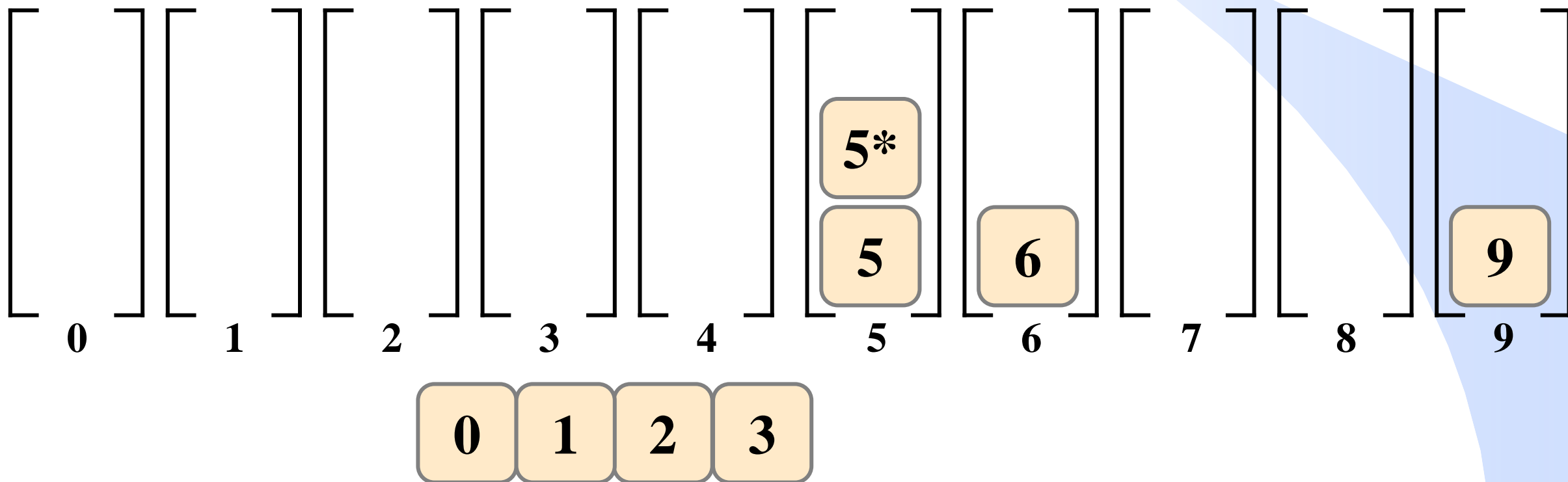
② 收集：依次将桶中元素取出



# 桶排序

A

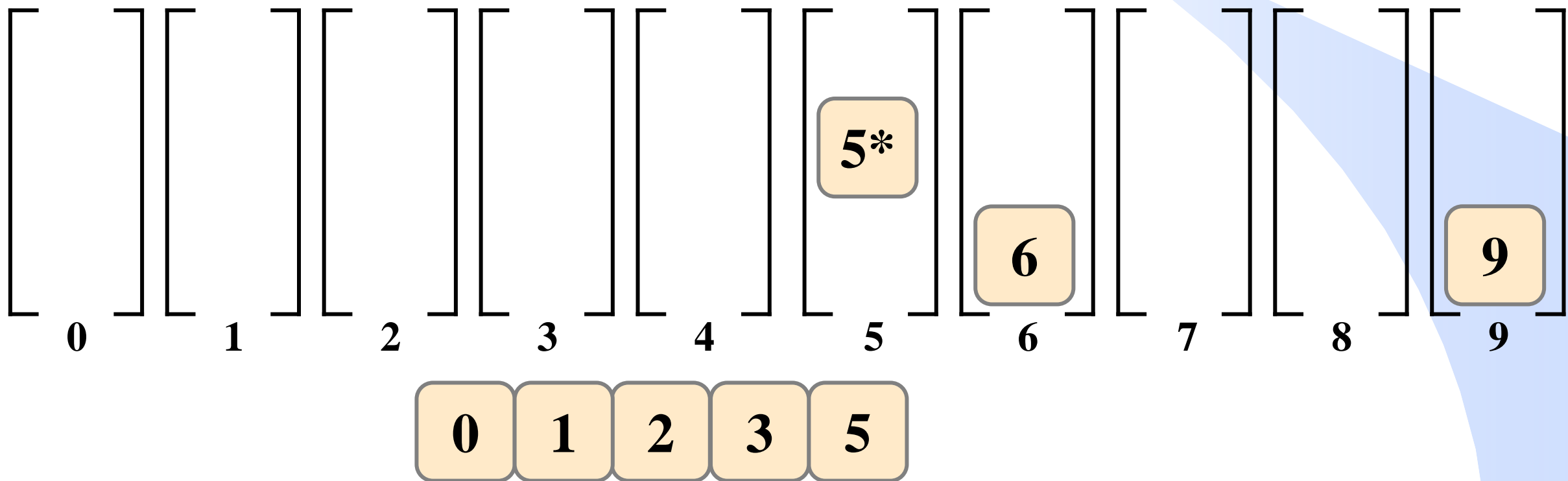
② 收集：依次将桶中元素取出



# 桶排序

A

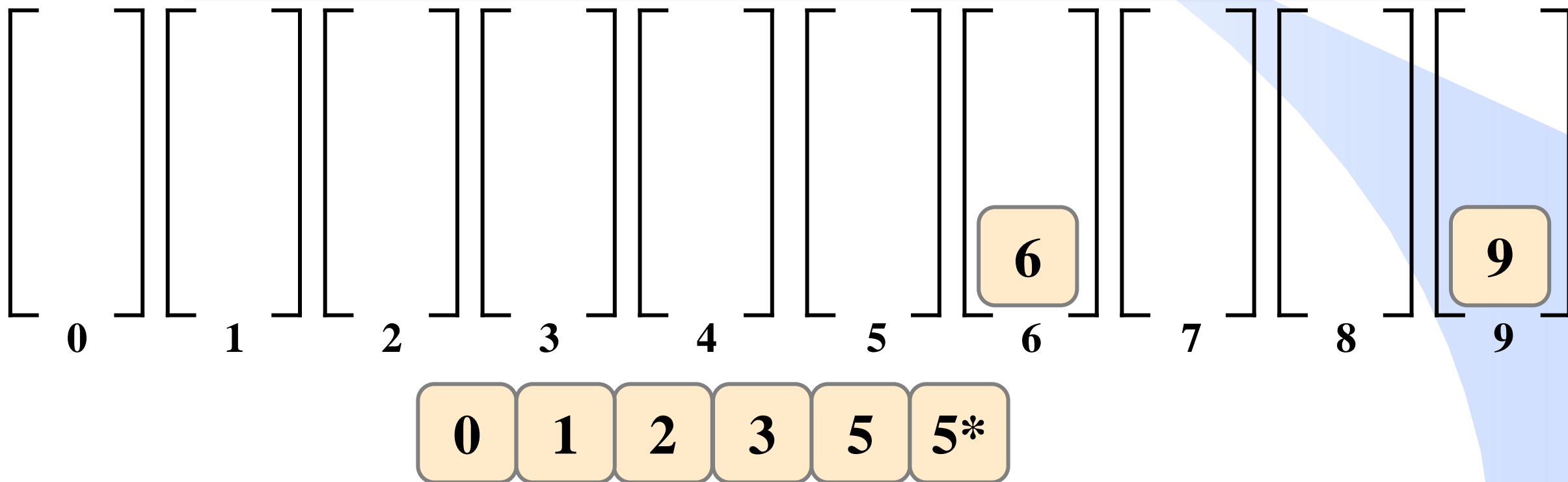
② 收集：依次将桶中元素取出



# 桶排序

## ② 收集：依次将桶中元素取出

桶排序是稳定的：关键词相同的元素在同一桶里，排在前面的元素在分配时先入桶，收集时也先出桶

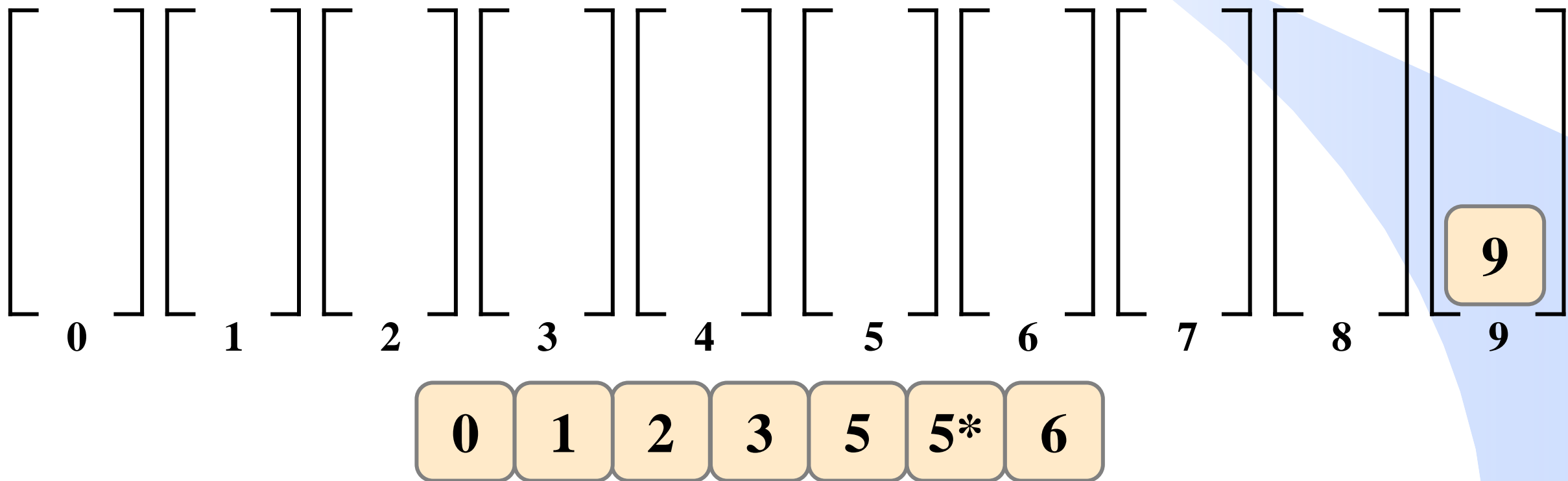




# 桶排序

A

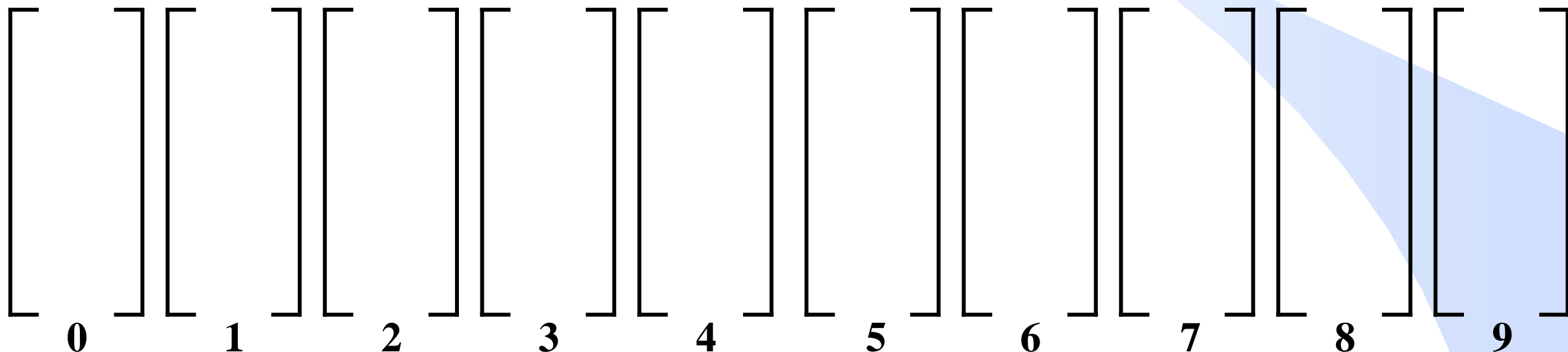
② 收集：依次将桶中元素取出



# 桶排序

- ① 分配：将 $n$ 个元素依次放入桶中
- ② 收集：依次将 $m$ 个桶中的元素取出

时空复杂度  
 $O(n+m)$



桶的实现  
链式队列



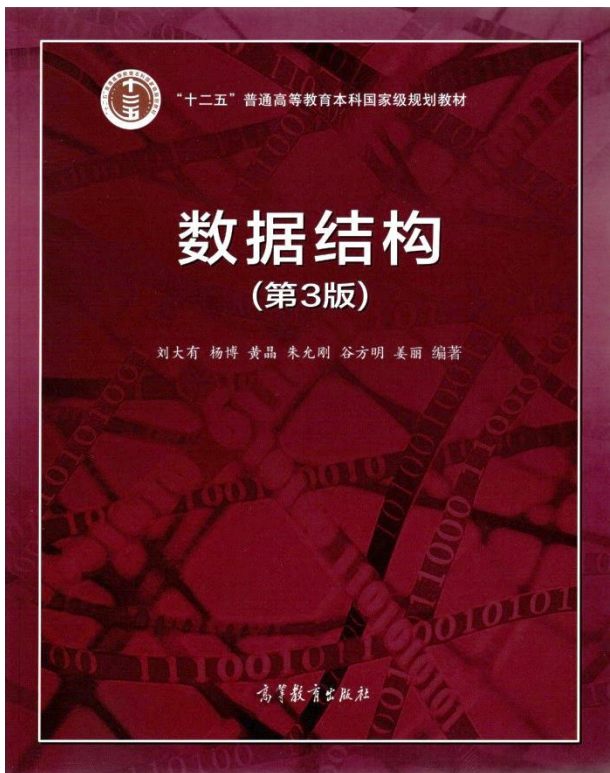
稳定

# 桶排序总结

排序算法	时间复杂度			空间复杂度	稳定性
	最好	平均	最坏		
桶排序	$O(n+m)$	$O(n+m)$	$O(n+m)$	$O(n+m)$	稳定

每个元素的值域为 $[0, m)$ , 即 $m$ 为桶的个数

若 $m=O(n)$ , 则时空复杂度为线性



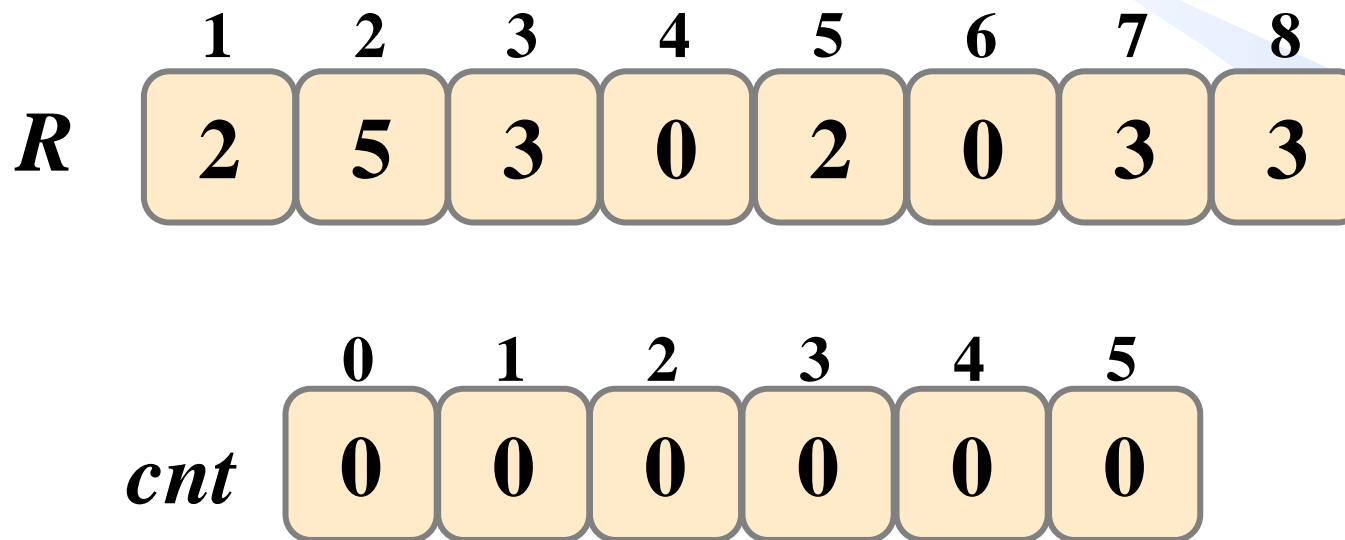
# 分布排序

- 桶排序
- **计数排序**
- 基数排序

数据之法  
结构之美  
算法之道

# 计数排序 (Counting Sort)

待排序的数组  $R$  包含  $n$  个整数，每个整数的值域为  $[0, m)$

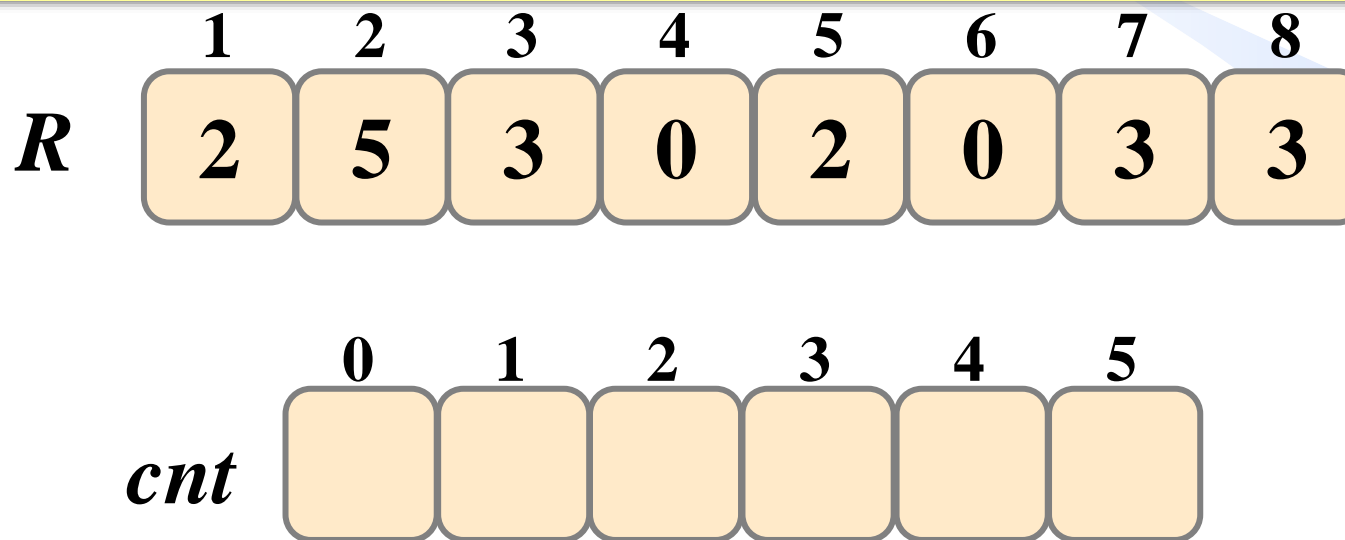


设置一个长度为  $m$  的数组  $cnt[]$ ，初值为 0

# 计数排序 (Counting Sort)

**B**

① 扫描数组 $R$ 计算每个元素出现的次数存入 $cnt$ 数组，即 $cnt[K]$ 为数组 $R$ 中有多少个 $K$ （等于 $K$ 的元素个数）。





# 计数排序 (Counting Sort)

**B**

① 扫描数组 $R$ 计算每个元素出现的次数存入 $cnt$ 数组，即 $cnt[K]$ 为数组 $R$ 中有多少个 $K$ （等于 $K$ 的元素个数）。

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

	0	1	2	3	4	5
$cnt$	2	0	2	3	0	1

```
for(int i=1; i<=n; i++){  
    int K = R[i]; //在 $R$ 中发现了一个元素 $K$   
    cnt[K]++;    // $K$ 在 $R$ 中的个数加1  
}
```

# 计数排序 (Counting Sort)

① 扫描数组 $R$ 计算每个元素出现的次数存入 $cnt$ 数组, 即 $cnt[K]$ 为数组 $R$ 中有多少个 $K$  (等于 $K$ 的元素个数)。

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

	0	1	2	3	4	5
$cnt$	2	0	2	3	0	1

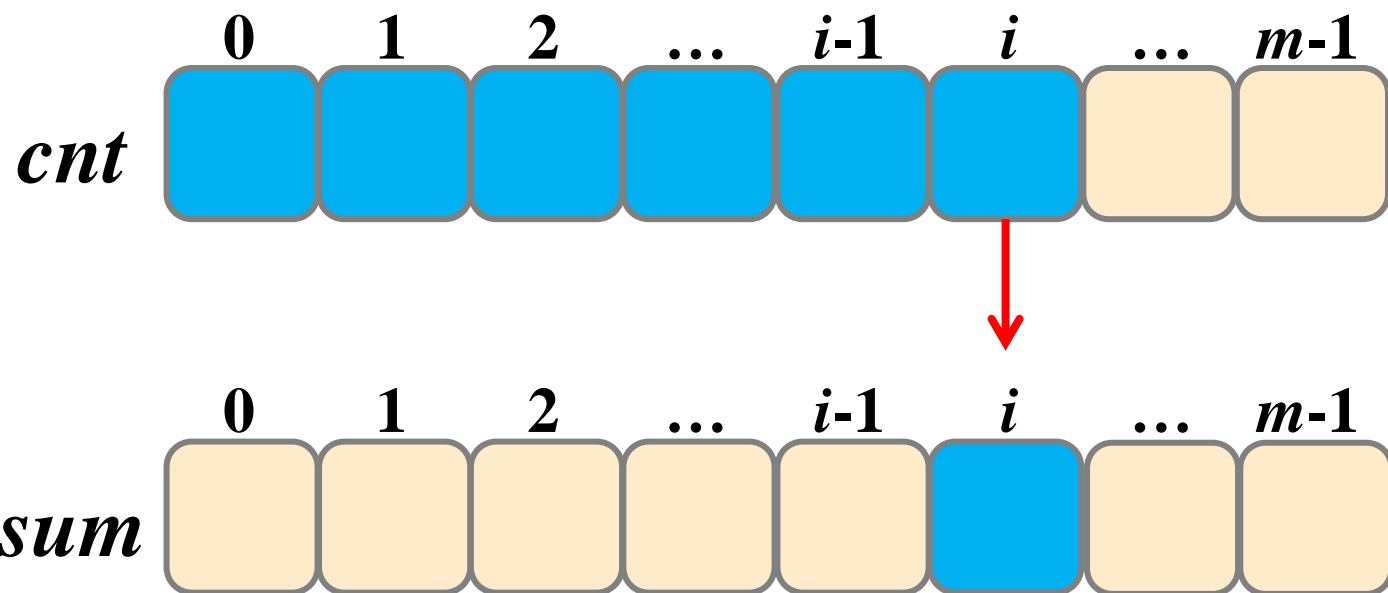
```
for(int i=1; i<=n; i++)  
    cnt[R[i]]++;
```

# 计数排序 (Counting Sort)

**B**

② 对 $cnt$ 数组求前缀和 $sum$ , 使 $sum[i]$ 表示数组 $R$ 中 $\leq i$ 的元素个数

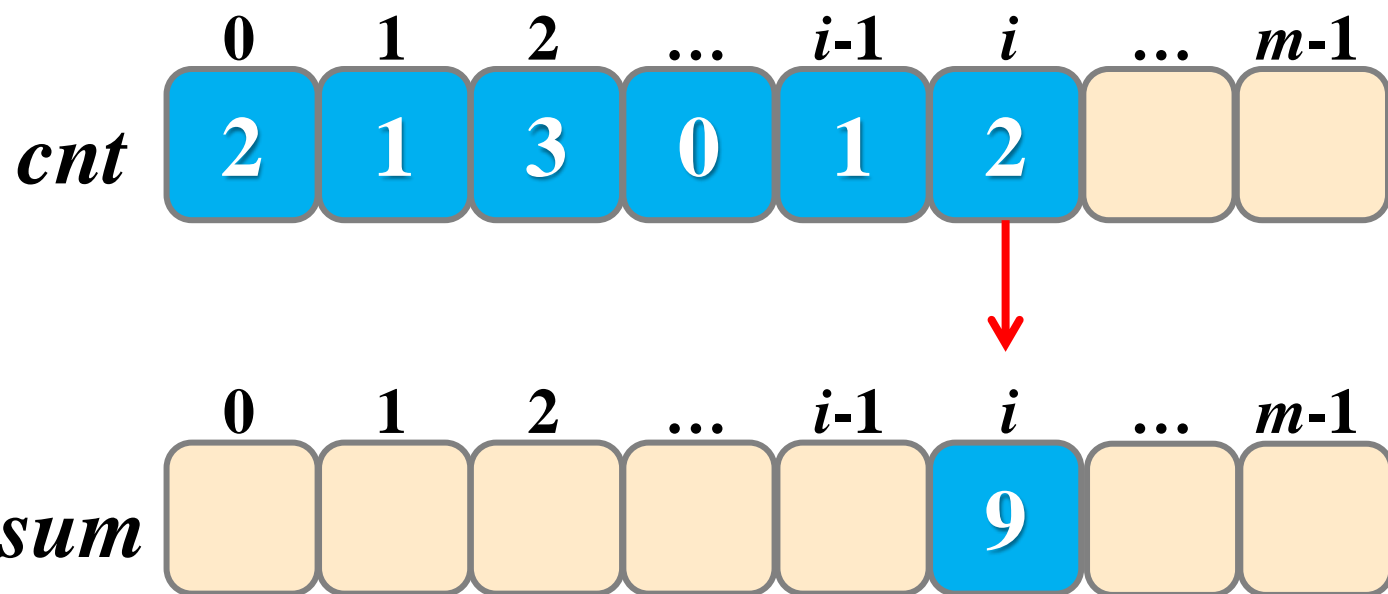
$$\text{令 } sum_i = cnt_0 + \dots + cnt_{i-1} + cnt_i$$



# 计数排序 (Counting Sort)

B

② 对 $cnt$ 数组求前缀和 $sum$ , 使 $sum[i]$ 表示数组 $R$ 中 $\leq i$ 的元素个数



$$\begin{aligned} \text{令 } sum_i &= cnt_0 + \dots + cnt_{i-1} + cnt_i \\ &= sum_{i-1} + cnt_i \end{aligned}$$

```
sum[0] = cnt[0];  
for (int i = 1; i < m; i++)  
    sum[i] = sum[i-1] + cnt[i];
```

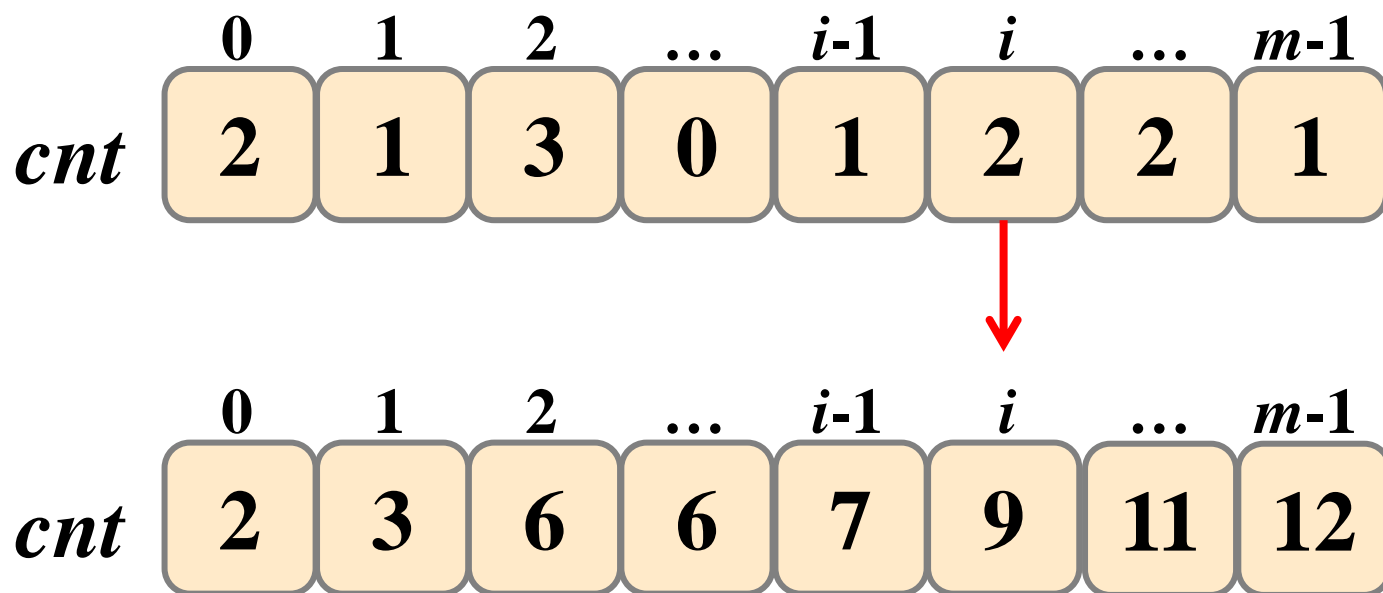
$sum[i]$ 就反映了元素 $i$ 的排名, 或者说元素 $i$ 在排序后的数组中的位置

# 计数排序 (Counting Sort)

B

② 对 $cnt$ 数组求前缀和 $sum$ , 使 $sum[i]$ 表示数组 $R$ 中 $\leq i$ 的元素个数

技巧: 不开 $sum$ 数组, 而是将其值仍存入 $cnt$ 数组



```
for(int i=1; i<m; i++)  
    sum[i]=sum[i-1]+cnt[i];
```

```
for(int i=1; i<m; i++)  
    cnt[i]=cnt[i-1]+cnt[i];
```

```
for(int i=1; i<m; i++)  
    cnt[i]+=cnt[i-1];
```

# 计数排序 (Counting Sort)

**B**

③从右往左扫描 $R$ 的每个元素 $K$ , 把 $K$ 放入排序后数组的 $cnt[K]$ 位置

	1	2	3	4	5	6	7	8
$R$	2	5	3	0	2	0	3*	3 <sup>#</sup>

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

	1	2	3	4	5	6	7	8
$B$								



# 计数排序 (Counting Sort)

*B*

③从右往左扫描*R*的每个元素*K*, 把*K*放入排序后数组的*cnt*[*K*]位置

	1	2	3	4	5	6	7	8
<i>R</i>	2	5	3	0	2	0	3*	3 <sup>#</sup>

	0	1	2	3	4	5
<i>cnt</i>	2	2	4	7	7	8

```
for(int i=n; i>=1; i--){  
    int K = R[i];  
    B[cnt[K]] = K;  
}
```

	1	2	3	4	5	6	7	8
<i>B</i>							3 <sup>#</sup>	

# 计数排序 (Counting Sort)

**B**

③从右往左扫描 $R$ 的每个元素 $K$ , 把 $K$ 放入排序后数组的 $cnt[K]$ 位置

	1	2	3	4	5	6	7	8
$R$	2	5	3	0	2	0	3*	3 <sup>#</sup>

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

	1	2	3	4	5	6	7	8
$B$							3 <sup>#</sup>	

```
for(int i=n; i>=1; i--){  
    int K = R[i];  
    B[cnt[K]] = K;  
    cnt[K]--;  
}
```

# 计数排序 (Counting Sort)

*B*

③从右往左扫描*R*的每个元素*K*, 把*K*放入排序后数组的*cnt*[*K*]位置

	1	2	3	4	5	6	7	8
<i>R</i>	2	5	3	0	2	0	3*	3 <sup>#</sup>

	0	1	2	3	4	5
<i>cnt</i>	2	2	4	5	7	8

	1	2	3	4	5	6	7	8
<i>B</i>						3*	3 <sup>#</sup>	

```
for(int i=n; i>=1; i--){  
    int K = R[i];  
    B[cnt[K]] = K;  
    cnt[K]--;  
}
```

稳定

# 计数排序 (Counting Sort)

**B**

③从右往左扫描 $R$ 的每个元素 $K$ , 把 $K$ 放入排序后数组的 $cnt[K]$ 位置

	1	2	3	4	5	6	7	8
$R$	2	5	3	0	2	0	3*	3#

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

	1	2	3	4	5	6	7	8
$B$		0				3*	3#	

```
for(int i=n; i>=1; i--){  
    int K = R[i];  
    B[cnt[K]] = K;  
    cnt[K]--;  
}
```

# 计数排序 (Counting Sort)

**B**

③从右往左扫描 $R$ 的每个元素 $K$ , 把 $K$ 放入排序后数组的 $cnt[K]$ 位置

	1	2	3	4	5	6	7	8
$R$	2	5	3	0	2	0	3*	3#

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

	1	2	3	4	5	6	7	8
$B$		0		2		3*	3#	

```
for(int i=n; i>=1; i--){  
    int K = R[i];  
    B[cnt[K]] = K;  
    cnt[K]--;  
}
```

# 计数排序 (Counting Sort)

**B**

③从右往左扫描 $R$ 的每个元素 $K$ , 把 $K$ 放入排序后数组的 $cnt[K]$ 位置

	1	2	3	4	5	6	7	8
$R$	2	5	3	0	2	0	3*	3#

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

	1	2	3	4	5	6	7	8
$B$	0	0		2		3*	3#	

```
for(int i=n; i>=1; i--){  
    int K = R[i];  
    B[cnt[K]] = K;  
    cnt[K]--;  
}
```

# 计数排序 (Counting Sort)

**B**

③从右往左扫描 $R$ 的每个元素 $K$ , 把 $K$ 放入排序后数组的 $cnt[K]$ 位置

	1	2	3	4	5	6	7	8
$R$	2	5	3	0	2	0	3*	3#

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

	1	2	3	4	5	6	7	8
$B$	0	0		2	3	3*	3#	

```
for(int i=n; i>=1; i--){  
    int K = R[i];  
    B[cnt[K]] = K;  
    cnt[K]--;  
}
```

# 计数排序 (Counting Sort)

**B**

③从右往左扫描 $R$ 的每个元素 $K$ , 把 $K$ 放入排序后数组的 $cnt[K]$ 位置

	1	2	3	4	5	6	7	8
$R$	2	5	3	0	2	0	3*	3 <sup>#</sup>

	0	1	2	3	4	5
$cnt$	0	2	3	4	7	7

	1	2	3	4	5	6	7	8
$B$	0	0		2	3	3*	3 <sup>#</sup>	5

```
for(int i=n; i>=1; i--){  
    int K = R[i];  
    B[cnt[K]] = K;  
    cnt[K]--;  
}
```



# 计数排序 (Counting Sort)

**B**

③从右往左扫描 $R$ 的每个元素 $K$ , 把 $K$ 放入排序后数组的 $cnt[K]$ 位置

	1	2	3	4	5	6	7	8
$R$	2	5	3	0	2	0	3*	3 <sup>#</sup>

	0	1	2	3	4	5
$cnt$	0	2	2	4	7	7

```
for(int i=n; i>=1; i--){  
    int K = R[i];  
    B[cnt[K]] = K;  
    cnt[K]--;  
}
```

	1	2	3	4	5	6	7	8
$B$	0	0	2	2	3	3*	3 <sup>#</sup>	5

↓

```
for(int i=n; i>=1; i--)  
    B[cnt[R[i]]--]=R[i];
```

## 计数排序 (Counting Sort)

待排序的数组 $R$ 包含 $n$ 个整数，每个整数的值域为 $[0, m)$ 。设置一个长度为 $m$ 的数组 $cnt[]$ ，初值为0。

- ① 扫描数组 $R$ 求每个元素出现的次数存入 $cnt$ 数组；
- ② 对 $cnt$ 数组求前缀和，使 $cnt[i]$ 表示数组 $R$ 中 $\leq i$ 的元素个数；
- ③ 从右往左扫描 $R$ 的每个元素 $K$ ，基于 $cnt[K]$ 将 $K$ 放入排序后的数组。

```
void CountingSort(int R[], int n, int m, int cnt[], int B[]) {  
    for(int i=0; i<m; i++) cnt[i]=0;           //初始化  
    for(int i=1; i<=n; i++) cnt[R[i]]++;       // ①  
    for(int i=1; i<m; i++) cnt[i]+=cnt[i-1];   // ②  
    for(int i=n; i>=1; i--) B[cnt[R[i]]--]=R[i]; // ③  
}
```

时空复杂度  $O(n+m)$

# 计数排序总结

排序算法	时间复杂度			空间复杂度	稳定性
	最好	平均	最坏		
计数排序	$O(n+m)$	$O(n+m)$	$O(n+m)$	$O(n+m)$	稳定

每个元素的值域为 $[0, m)$ , 即 $m$ 为 $cnt$ 数组的长度

若 $m=O(n)$ , 则时空复杂度为线性

## 思考

**B**

- 对吉林大学22级全体同学按英语考试分数排序，哪种排序算法更好？
  - A. 快速排序
  - B. 计数排序或桶排序
- 对世界各国的人口进行排序，哪种排序算法更好？
  - A. 快速排序
  - B. 计数排序或桶排序

## 思考

➤ 对10个整数排序，每个整数的范围为 $[0, 1000)$ 。

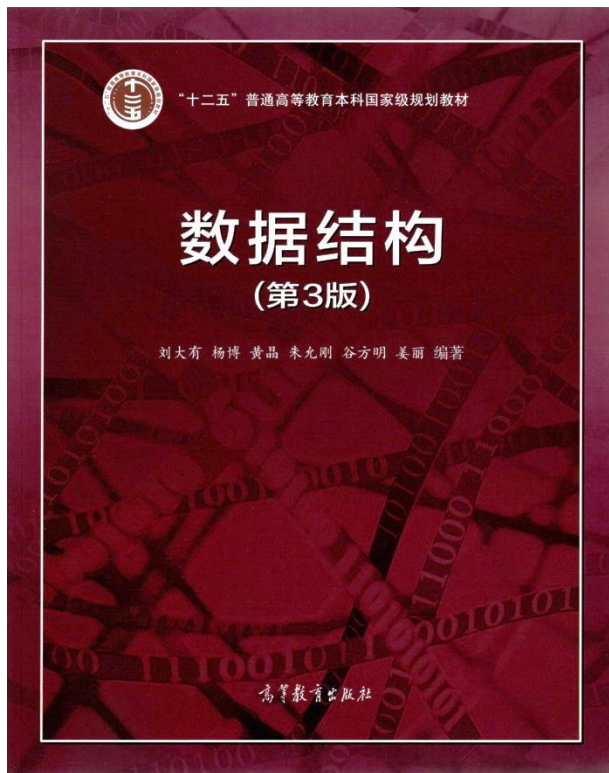
29 257 658 839 236 720 56 237 999 155

$n=10$   
 $m=1000$



# 分布排序

- 桶排序
- 计数排序
- **基数排序**



数据之法  
结构之美  
算法之道



# 基数排序



**Herman Hollerith**  
**(1860-1929)**  
发明打孔卡制表机  
IBM公司之父



# 基数排序 (Radix Sort)

- 元素的关键词由多个域构成，即  $K = K_d, K_{d-1}, \dots, K_2, K_1$ 
  - ✓ 若每个域为英文字母，则关键词即英文单词
  - ✓ 若每个域为1位十进制数字(0~9)，则关键词即  $d$  位十进制数
- 自  $K_1$  至  $K_d$  (自低位向高位)，依次以各域为序进行稳定排序

2	9	
2	5	7
6	5	8
8	3	9
2	3	6
7	2	0
1	5	5

按个位  
(最低位)  
稳定排序



7	2	0
1	5	5
2	3	6
2	5	7
6	5	8
	2	9
8	3	9

按十位  
(次低位)  
稳定排序



十位相等的  
元素个位小  
的靠前

7	2	0
	2	9
2	3	6
8	3	9
1	5	5
2	5	7
6	5	8

按百位  
(最高位)  
稳定排序



百位相等的  
元素十位小  
的靠前

	2	9
1	5	5
2	3	6
2	5	7
6	5	8
7	2	0
8	3	9



# 基数排序

➤ 对每一位采用哪种排序方法？

✓ 要求：高效且稳定

✓ 特点：每位关键词都是整数，且在 $[0, r)$ 范围内

✓ 方法：桶排序或计数排序

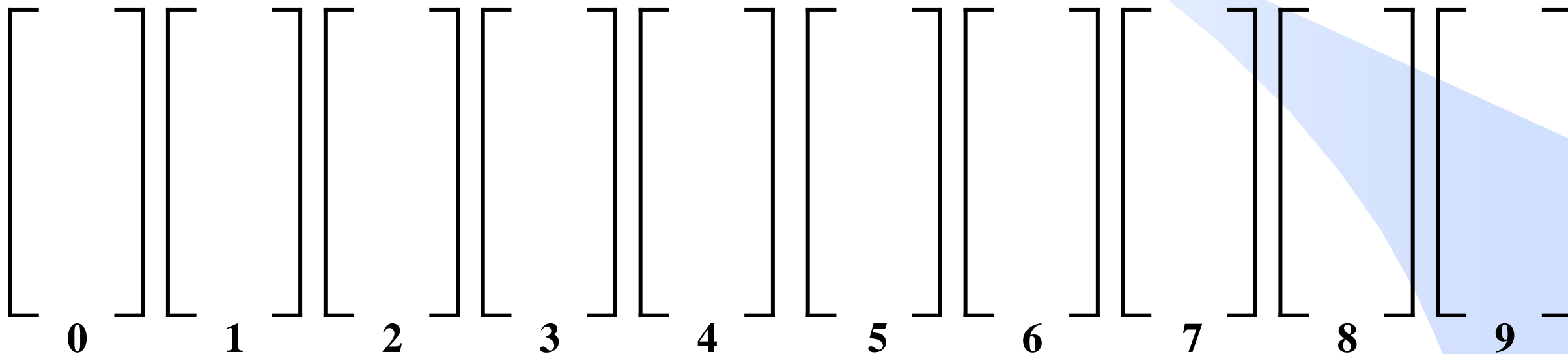
2	9	
2	5	7
6	5	8
8	3	9
2	3	6
7	2	0
1	5	5

# 基于桶排序的基数排序示例：① 按个位排序

A

29 257 658 839 236 720 155

按个位  
分配

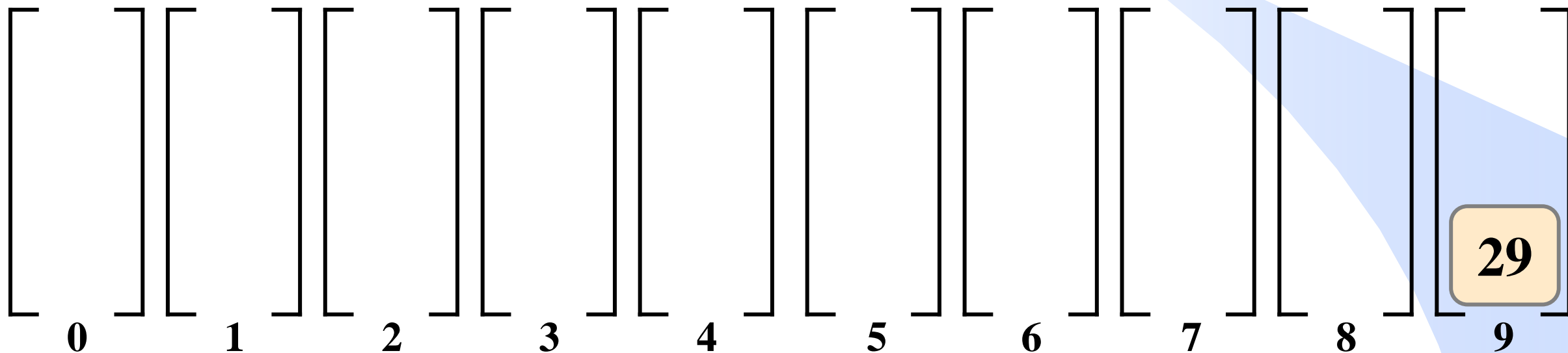


# 基于桶排序的基数排序示例：① 按个位排序

A

257 658 839 236 720 155

按个位  
分配

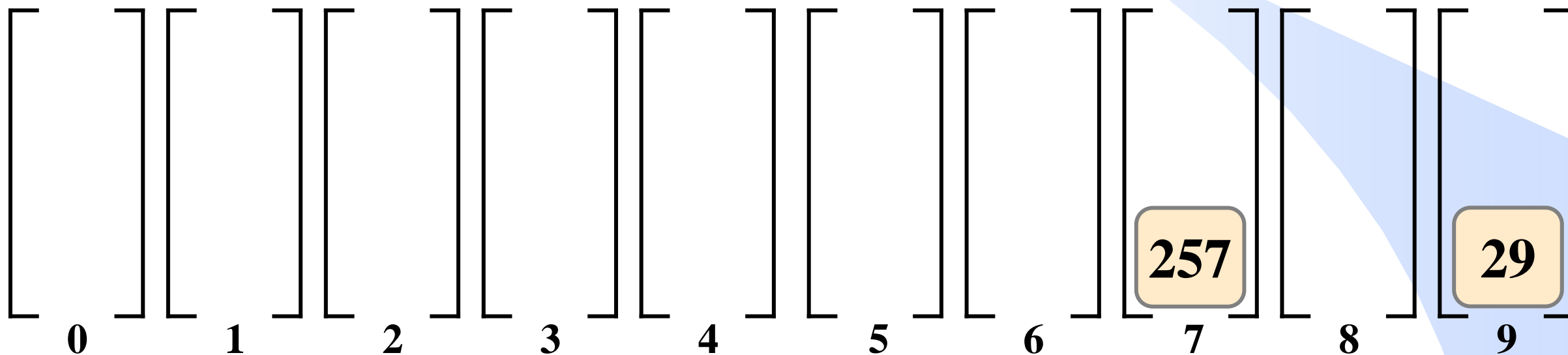


# 基于桶排序的基数排序示例：① 按个位排序

A

658 839 236 720 155

按个位  
分配



# 基于桶排序的基数排序示例：① 按个位排序

A

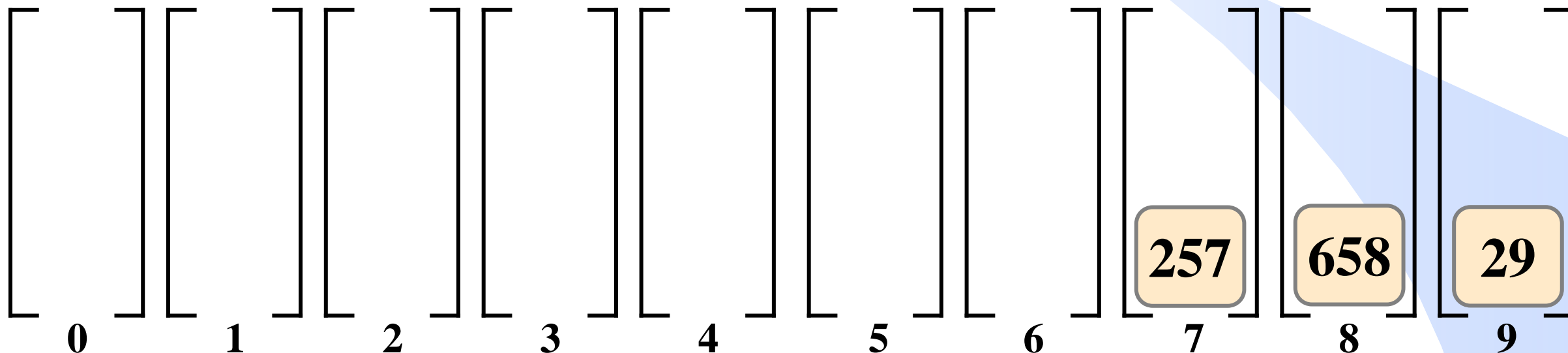
839

236

720

155

按个位  
分配



# 基于桶排序的基数排序示例：① 按个位排序

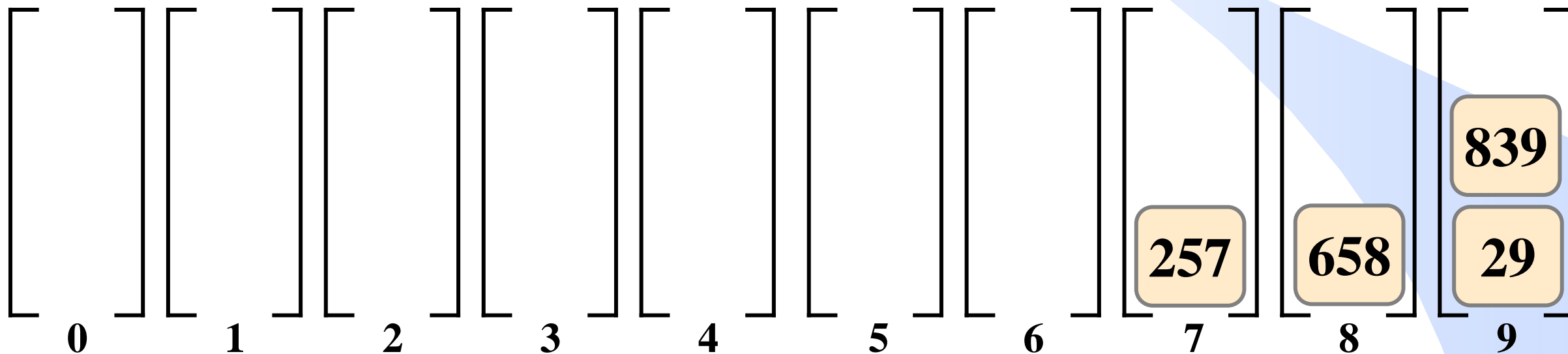
A

236

720

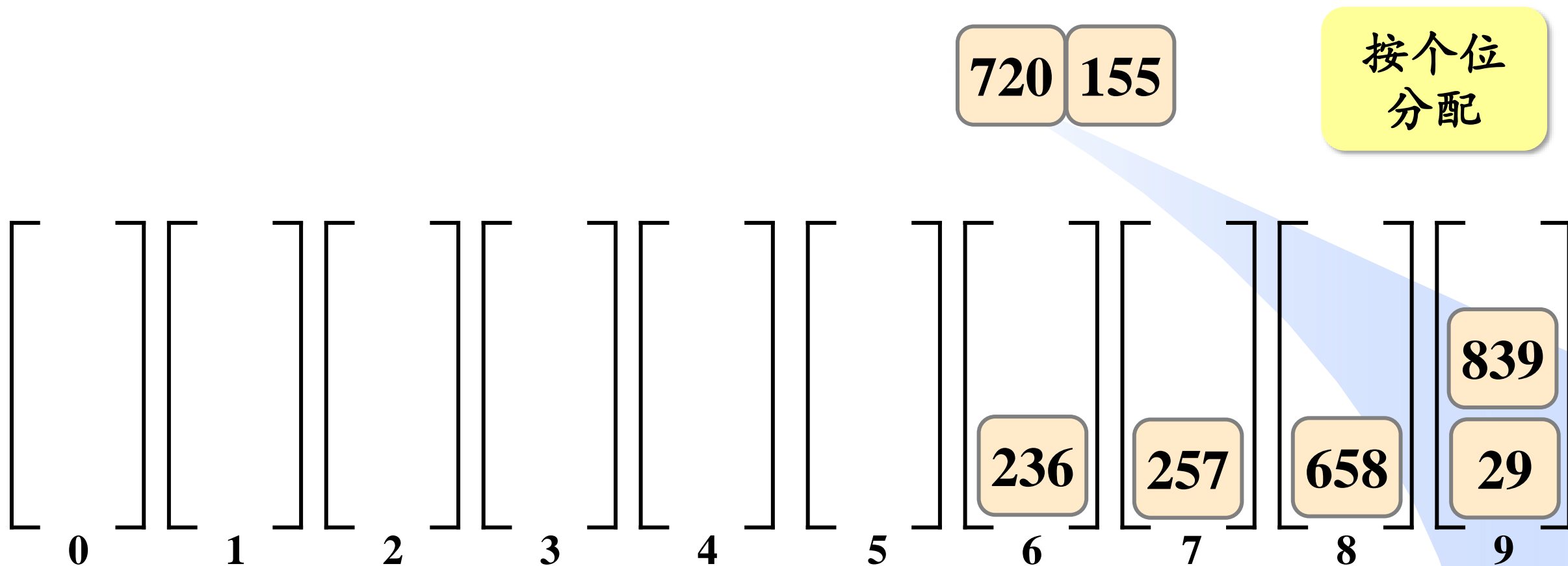
155

按个位  
分配



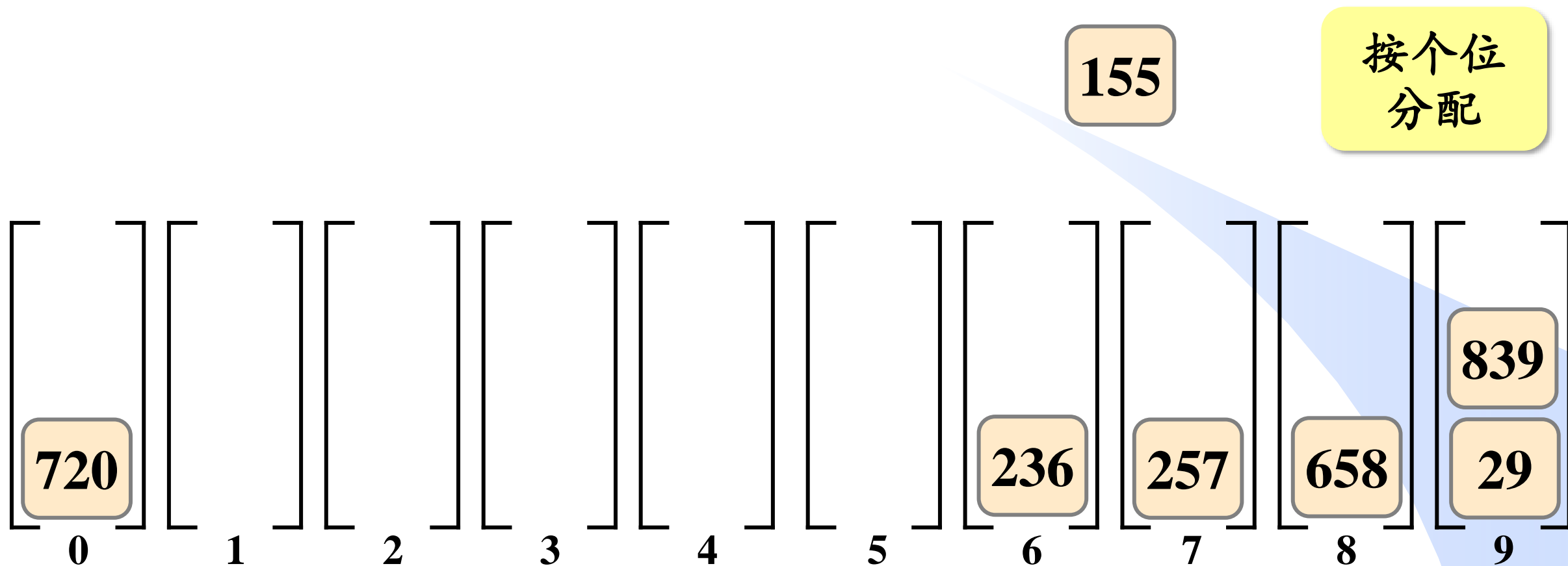
# 基于桶排序的基数排序示例：① 按个位排序

A



# 基于桶排序的基数排序示例：① 按个位排序

A

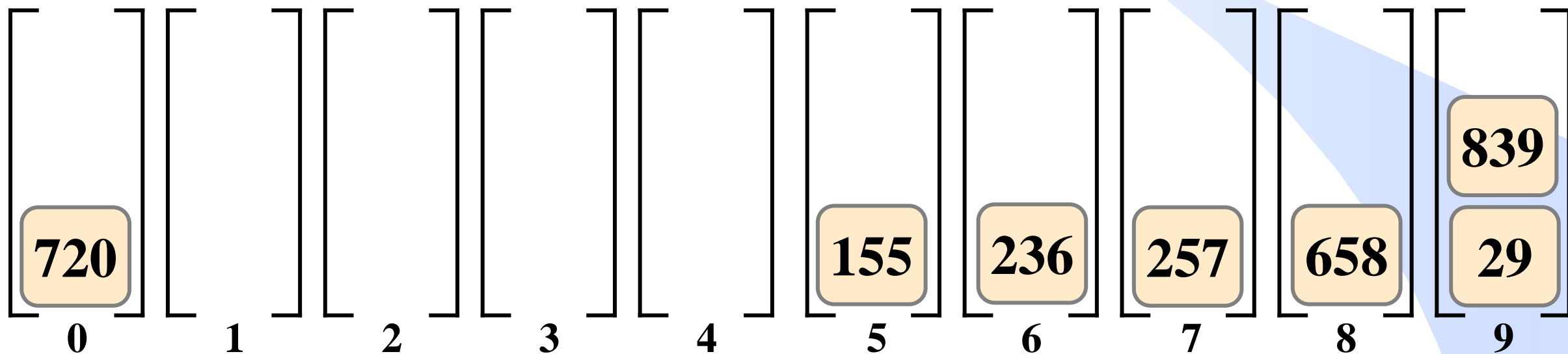




# 基于桶排序的基数排序示例：① 按个位排序

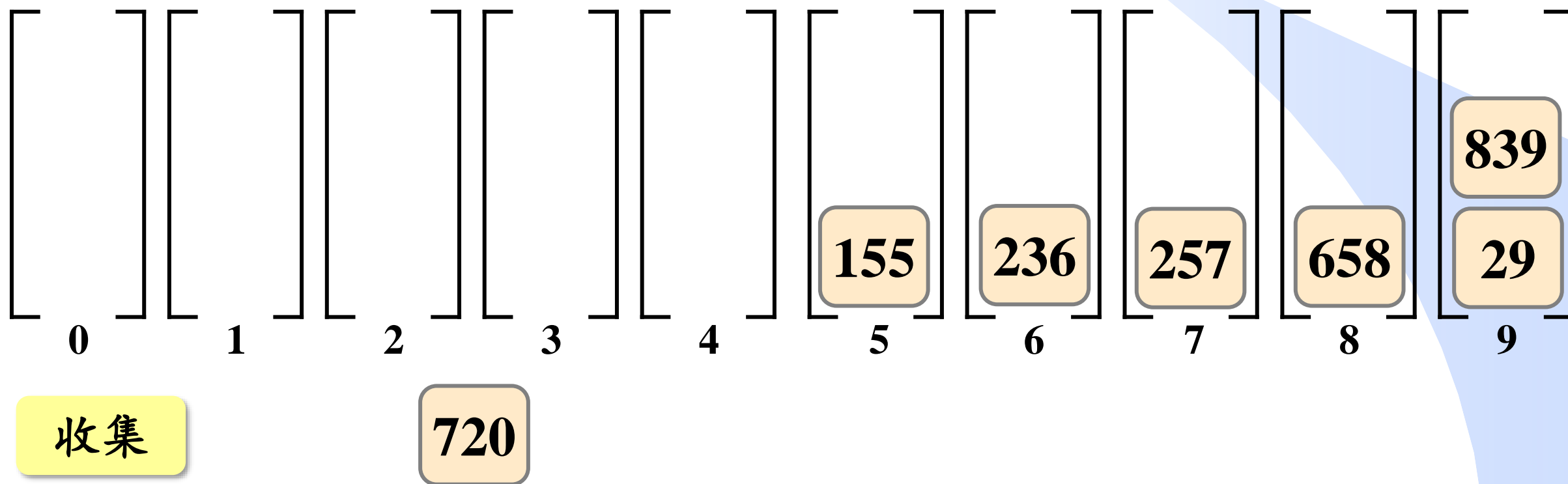
A

按个位  
分配



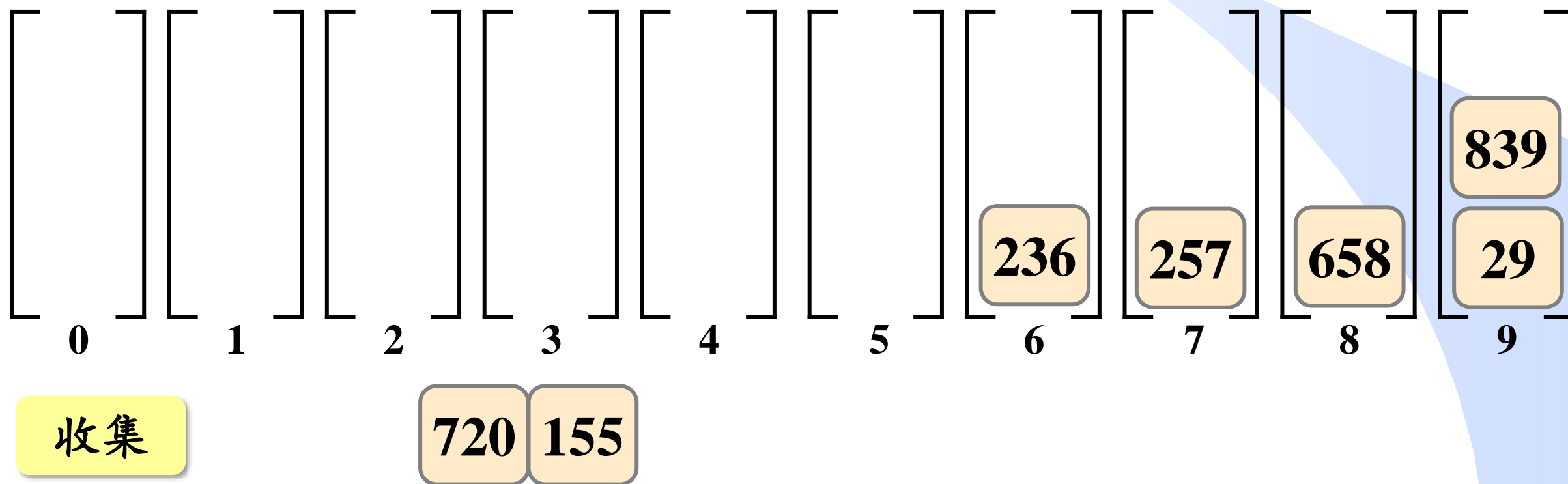
# 基于桶排序的基数排序示例：① 按个位排序

A



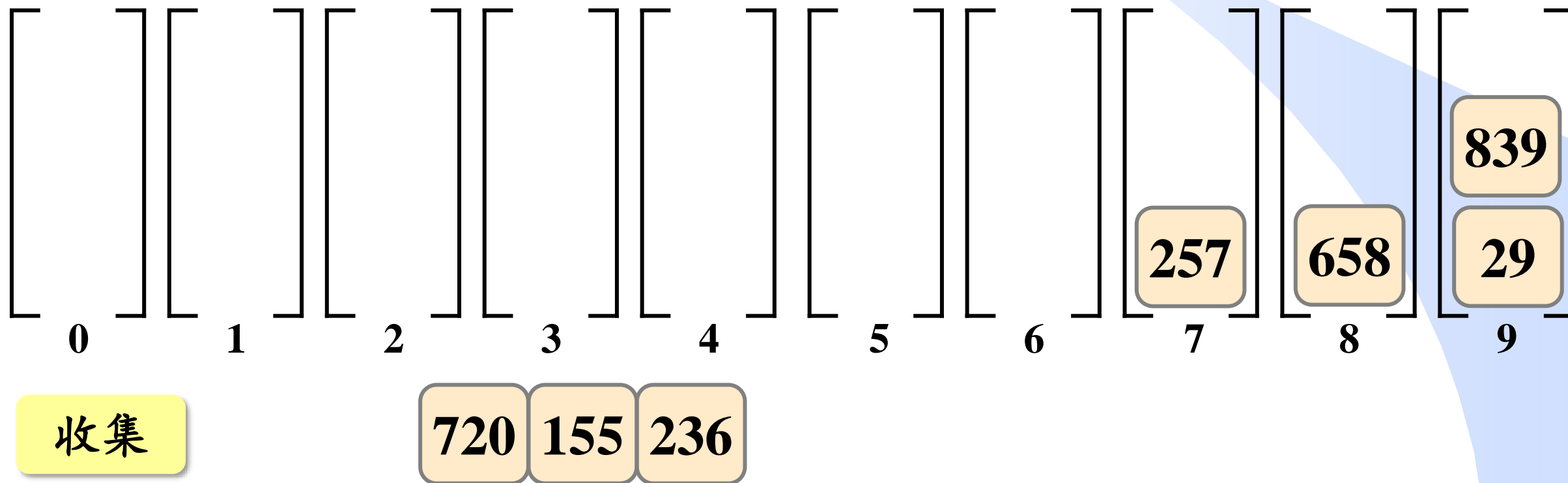
# 基于桶排序的基数排序示例：① 按个位排序

A



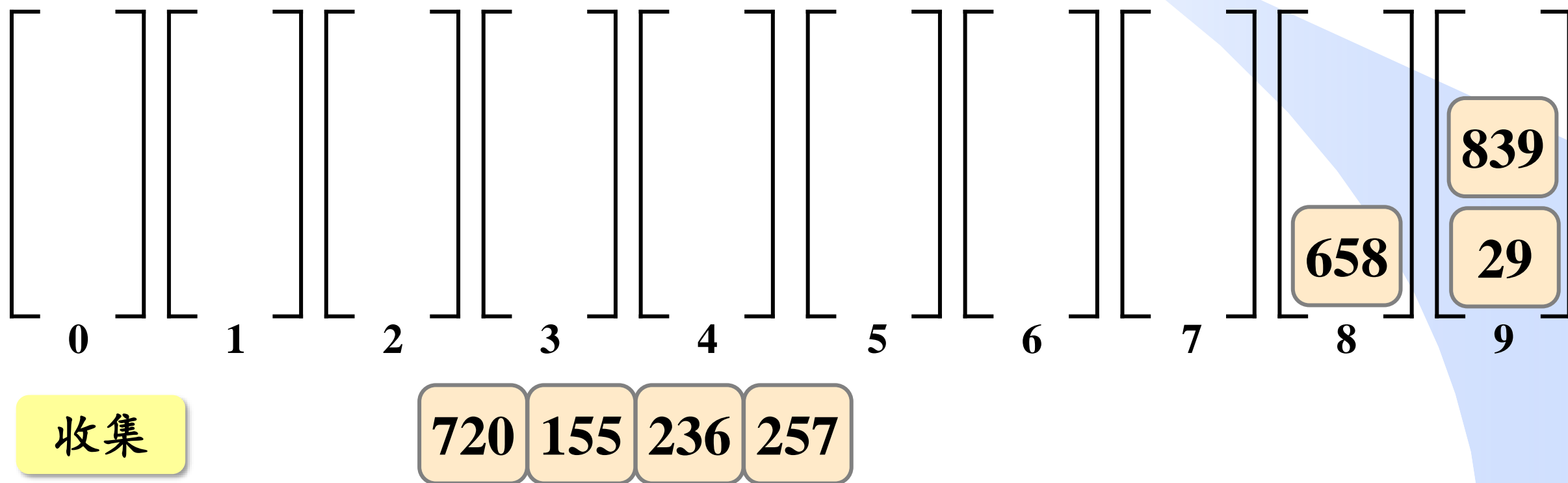
# 基于桶排序的基数排序示例：① 按个位排序

A



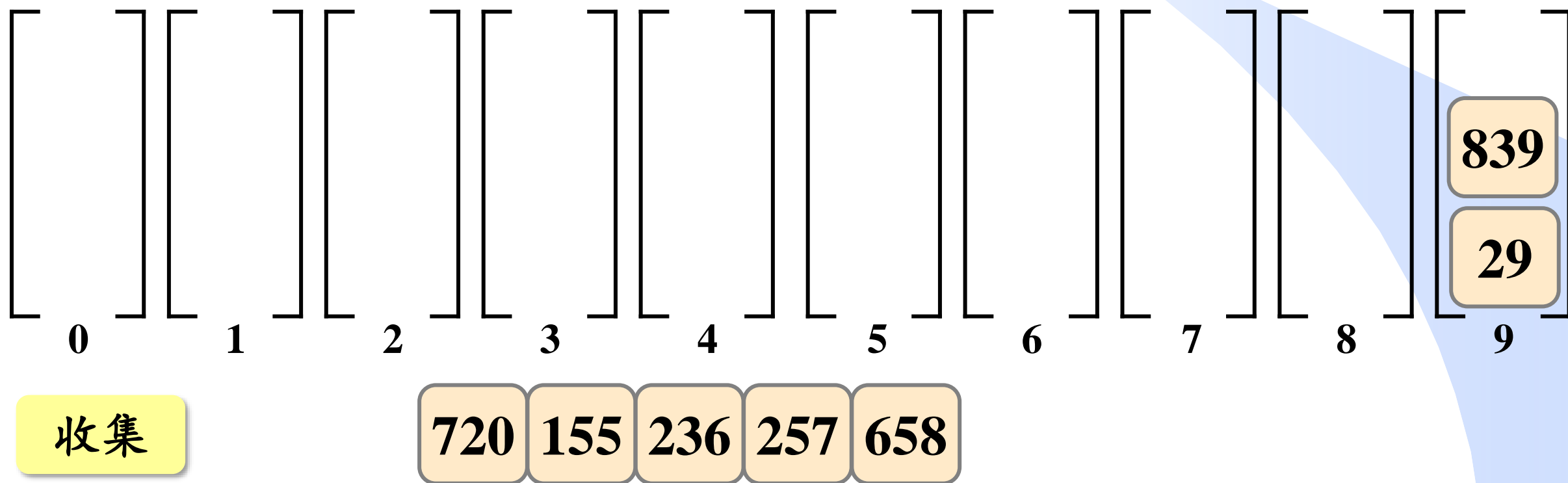
# 基于桶排序的基数排序示例：① 按个位排序

A



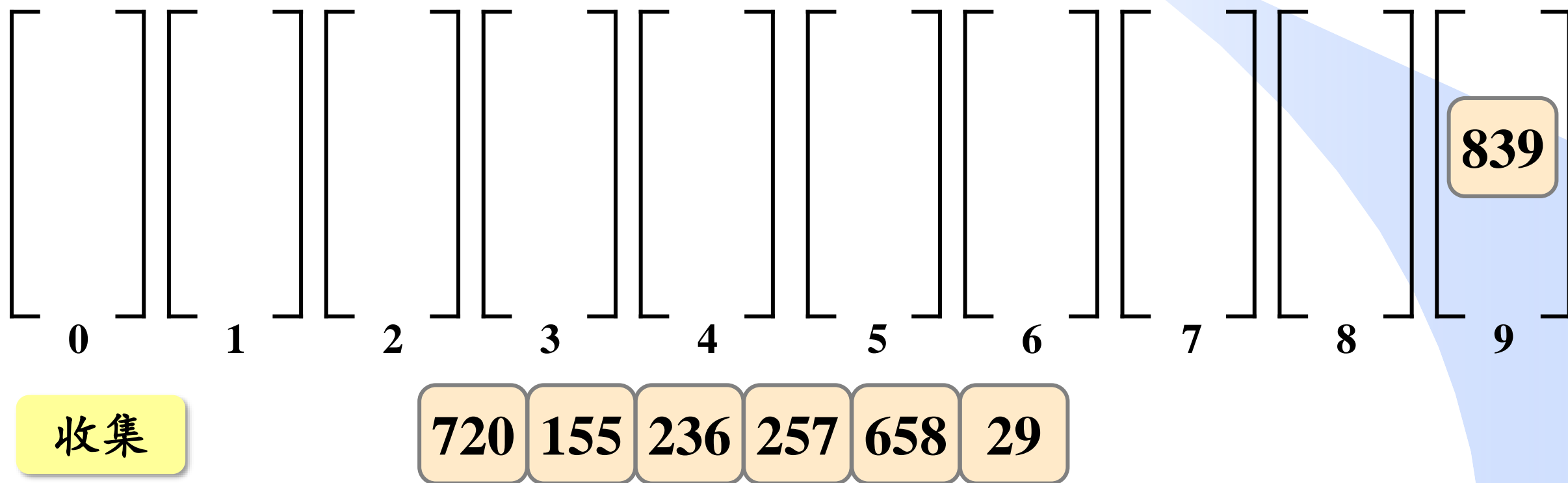
# 基于桶排序的基数排序示例：① 按个位排序

A



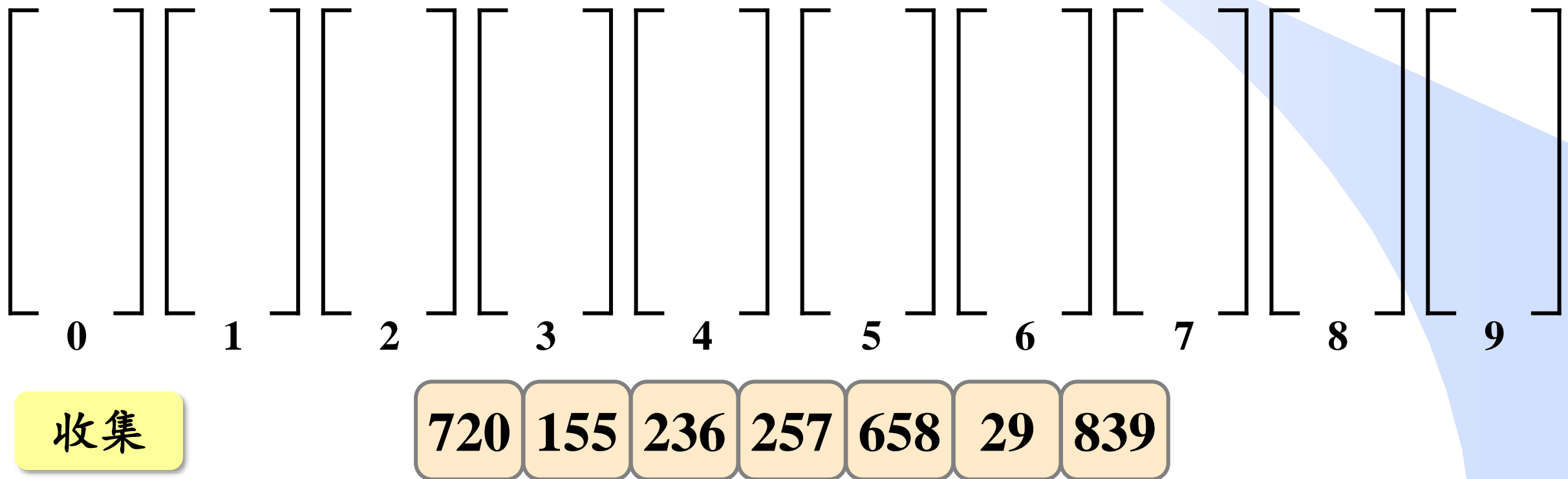
# 基于桶排序的基数排序示例：① 按个位排序

A



# 基于桶排序的基数排序示例：① 按个位排序

A



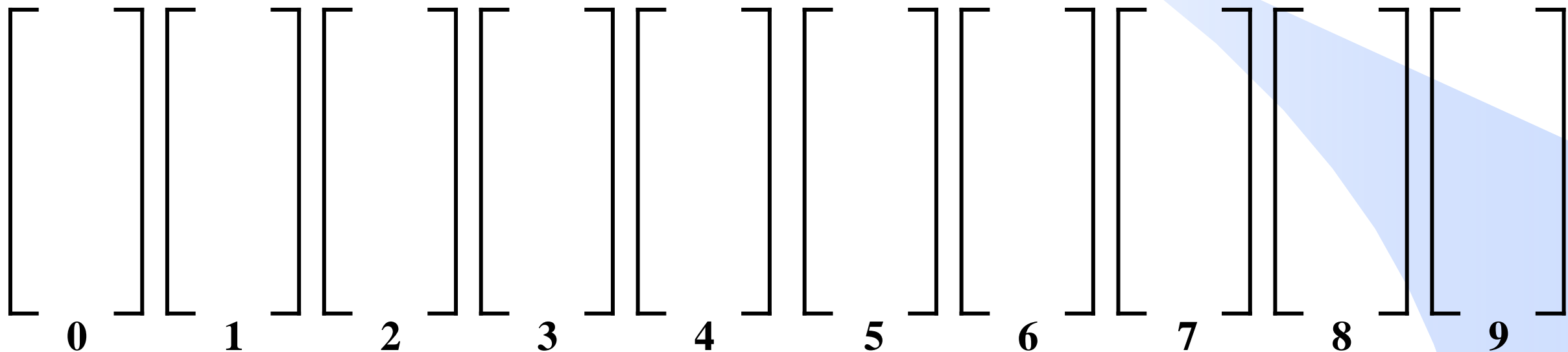


# 基于桶排序的基数排序示例：② 按十位排序

A

720 155 236 257 658 29 839

按十位  
分配

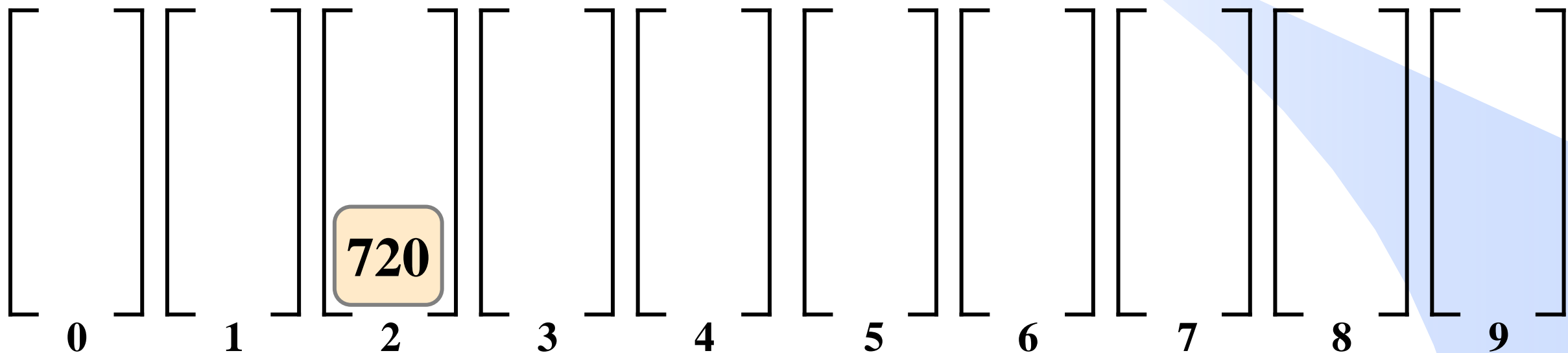


## 基于桶排序的基数排序示例：② 按十位排序

A

155 236 257 658 29 839

按十位  
分配



# 基于桶排序的基数排序示例：② 按十位排序

A

236

257

658

29

839

按十位  
分配

720

155

0

1

2

3

4

5

6

7

8

9

## 基于桶排序的基数排序示例：② 按十位排序

A

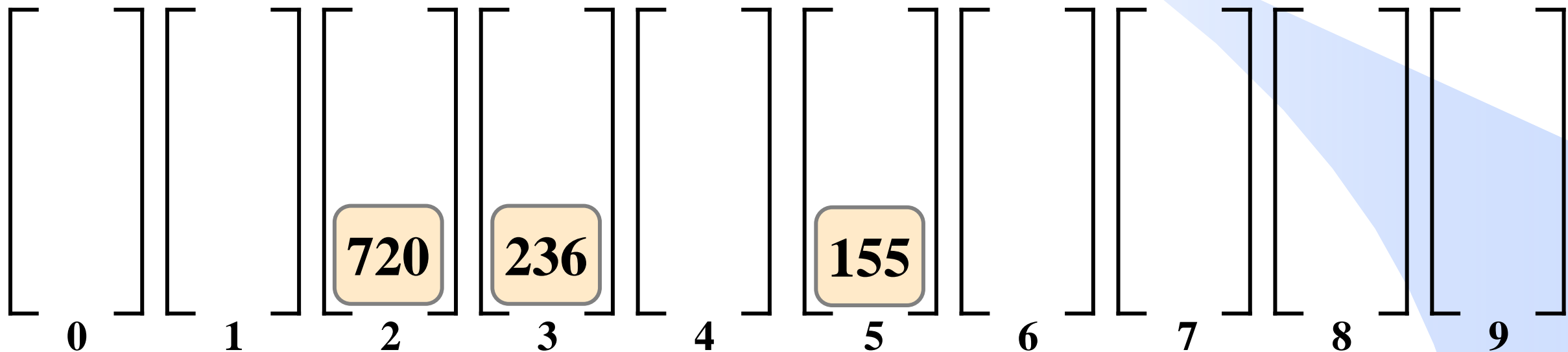
257

658

29

839

按十位  
分配



# 基于桶排序的基数排序示例：② 按十位排序

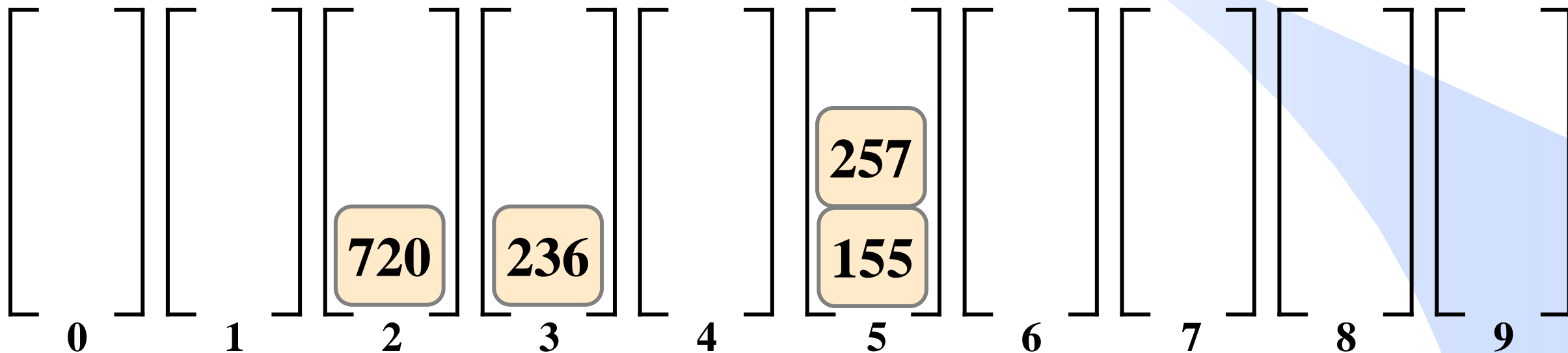
A

658

29

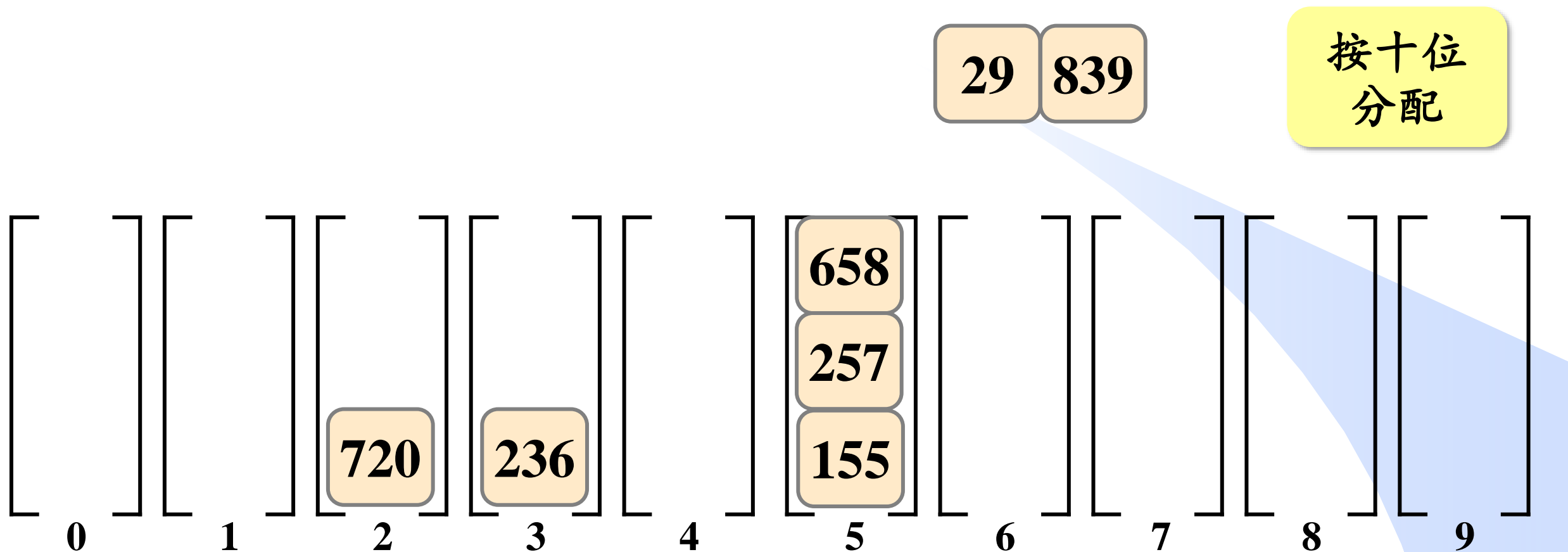
839

按十位  
分配



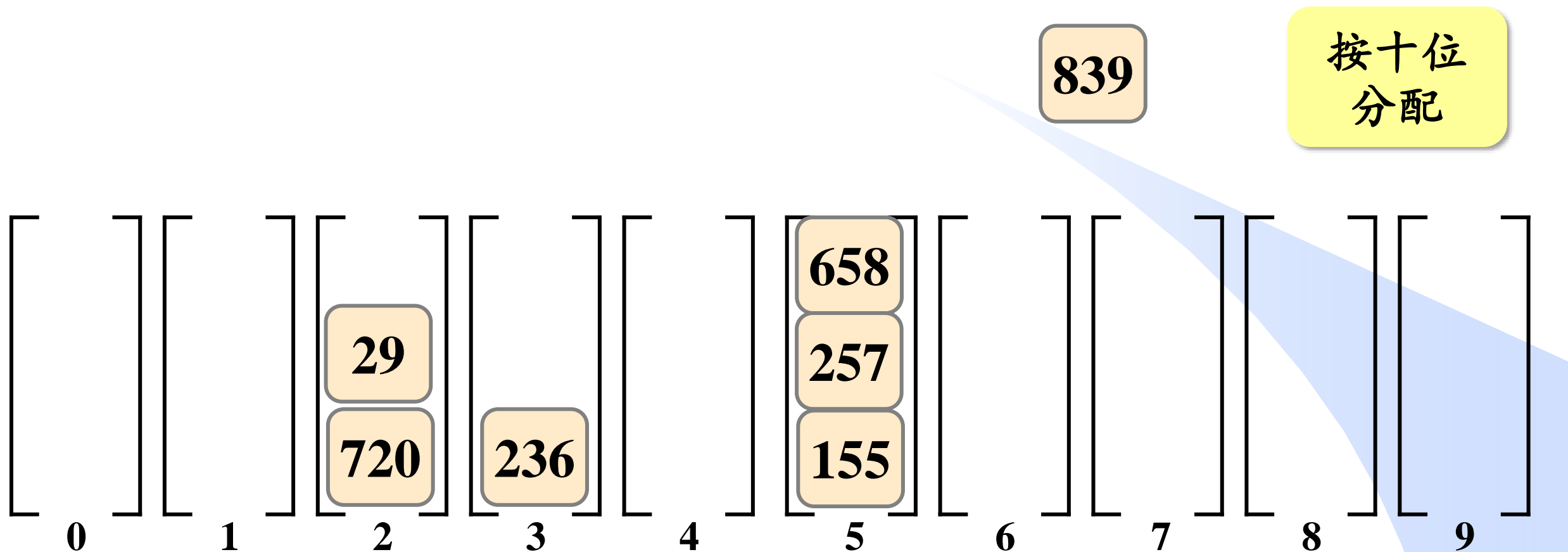
# 基于桶排序的基数排序示例：② 按十位排序

A



## 基于桶排序的基数排序示例：② 按十位排序

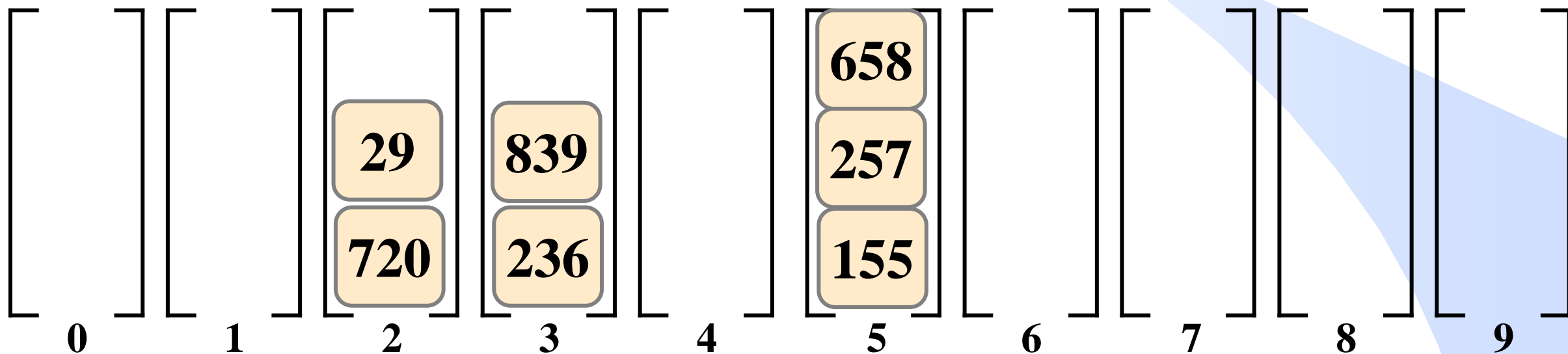
A



## 基于桶排序的基数排序示例：② 按十位排序

A

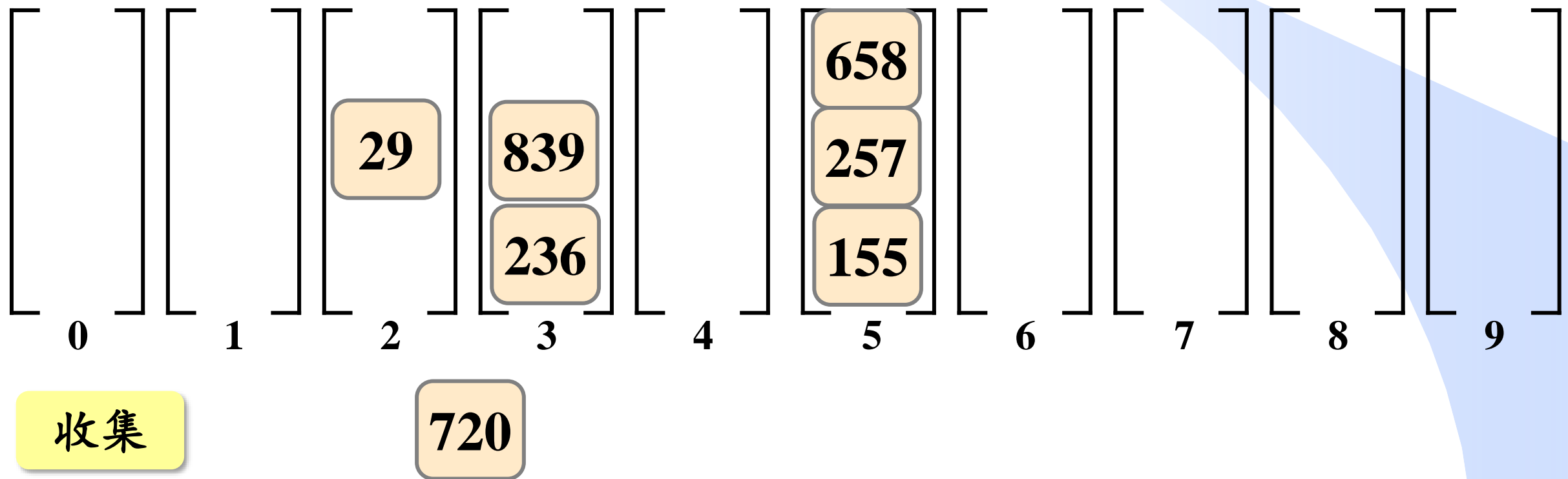
按十位  
分配





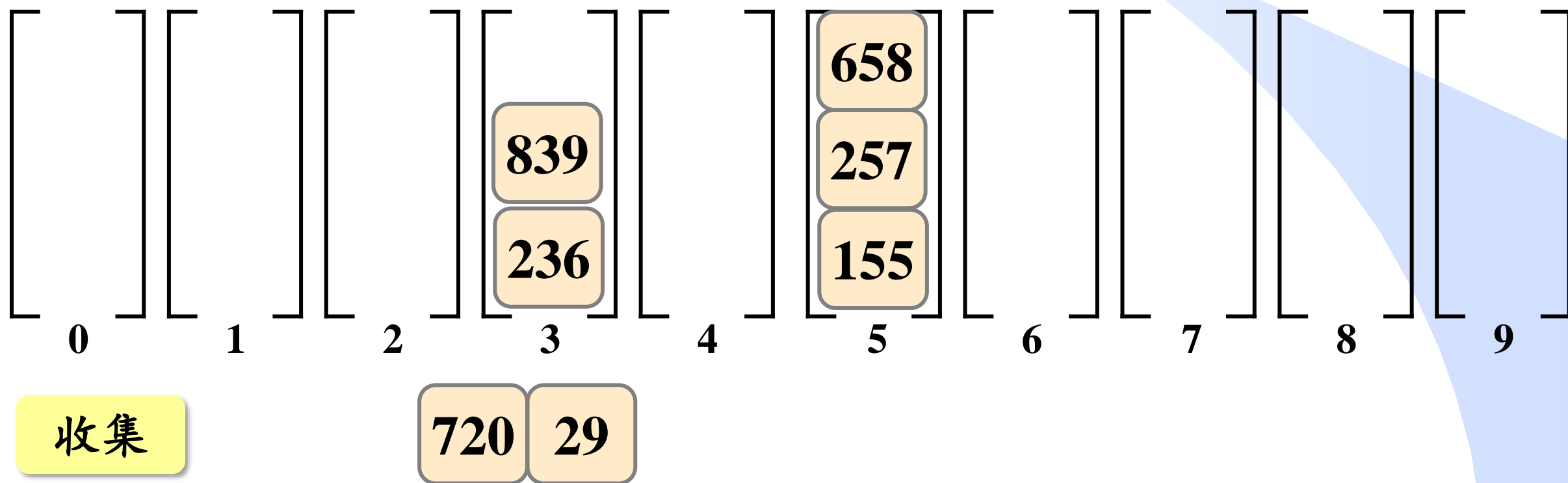
## 基于桶排序的基数排序示例：② 按十位排序

A



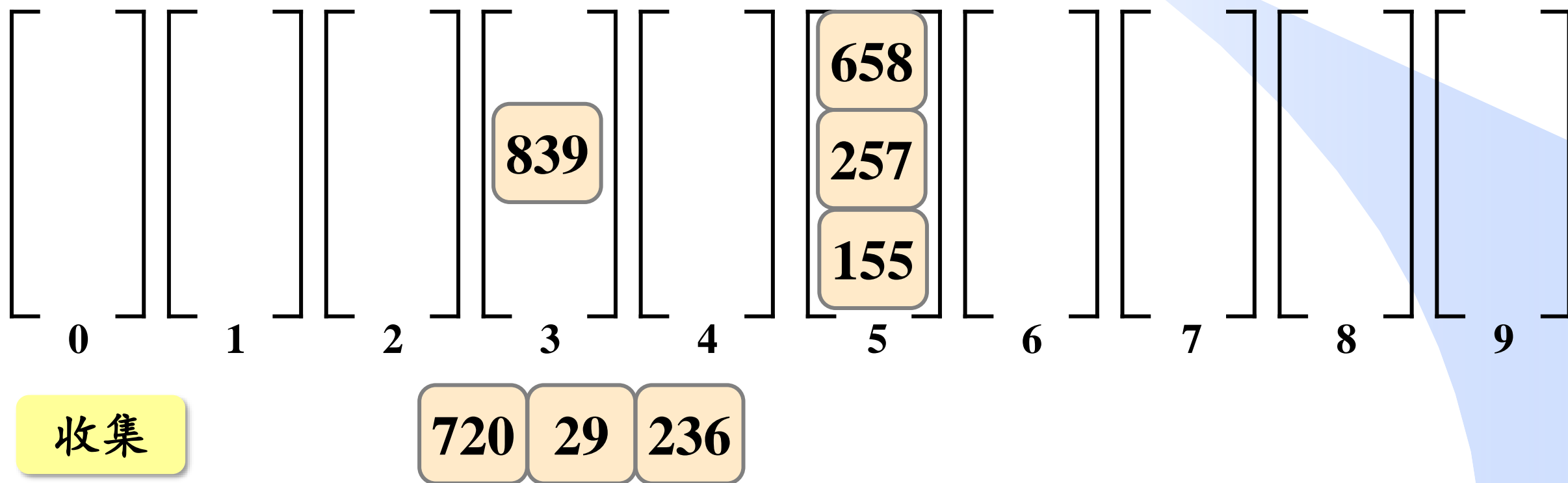
## 基于桶排序的基数排序示例：② 按十位排序

A



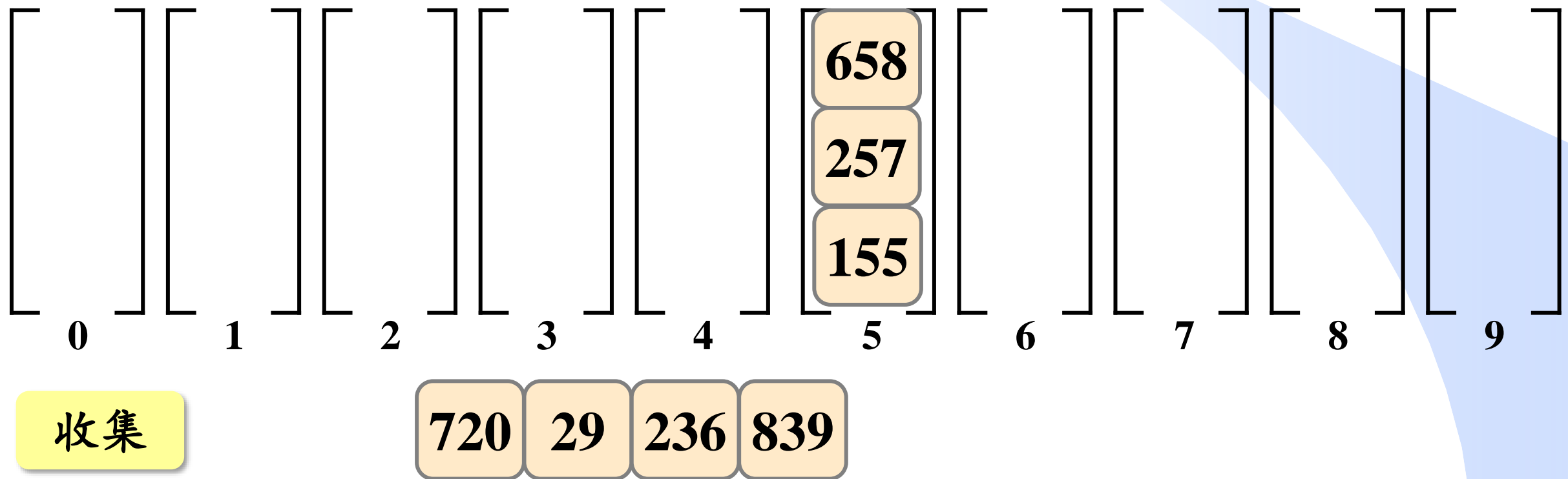
## 基于桶排序的基数排序示例：② 按十位排序

A



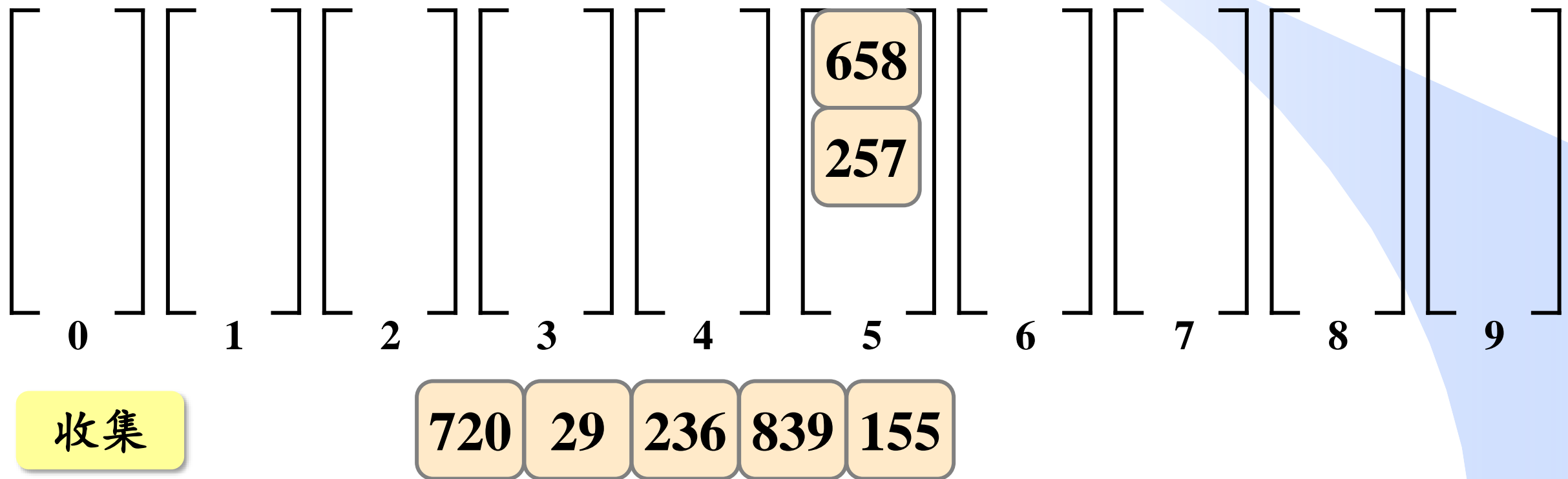
# 基于桶排序的基数排序示例：② 按十位排序

A



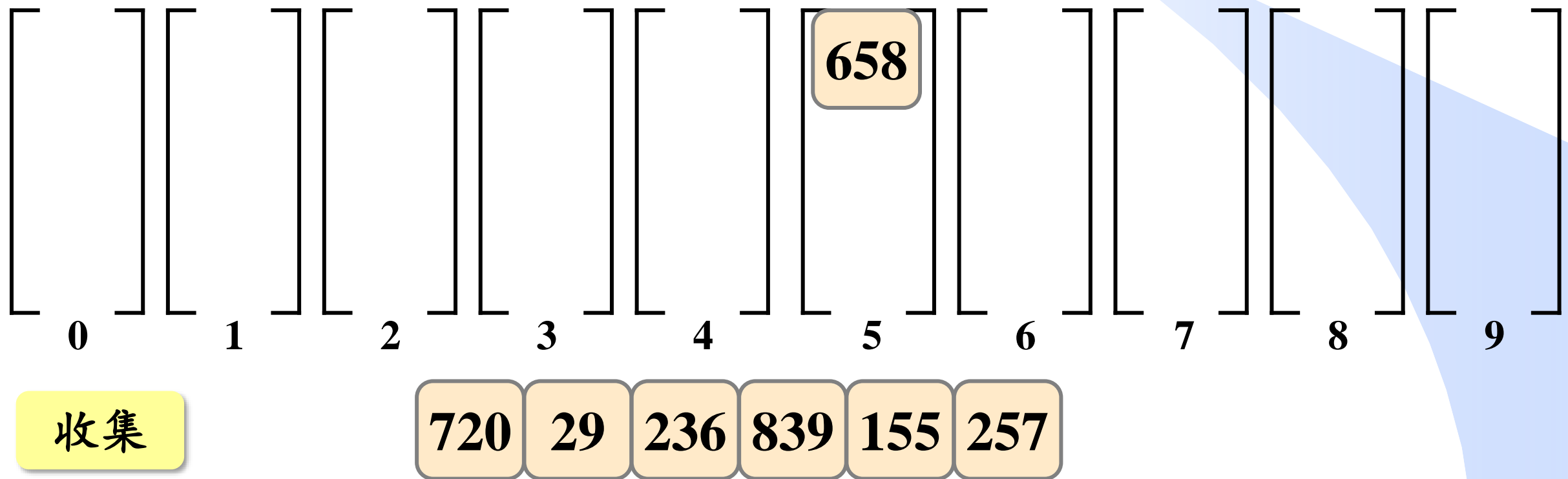
# 基于桶排序的基数排序示例：② 按十位排序

A



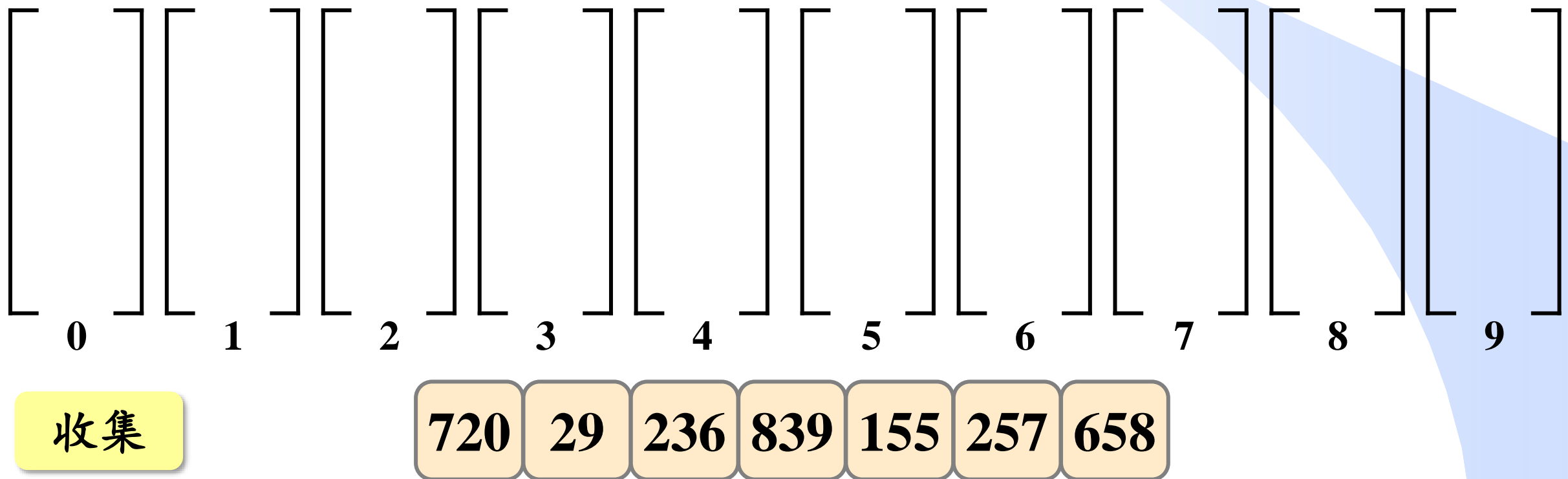
# 基于桶排序的基数排序示例：② 按十位排序

A



# 基于桶排序的基数排序示例：② 按十位排序

A

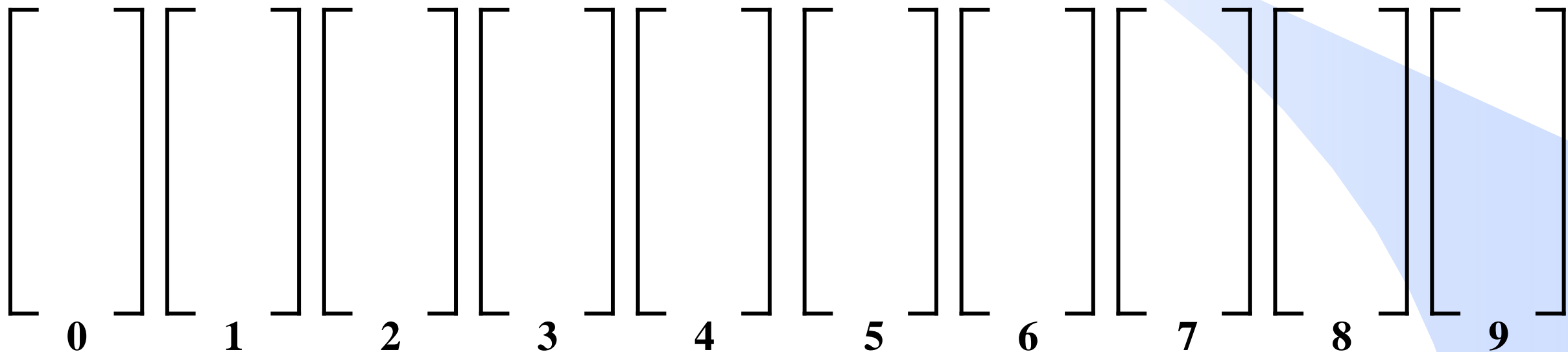


# 基于桶排序的基数排序示例：③ 按百位排序

A

720 29 236 839 155 257 658

按百位  
分配



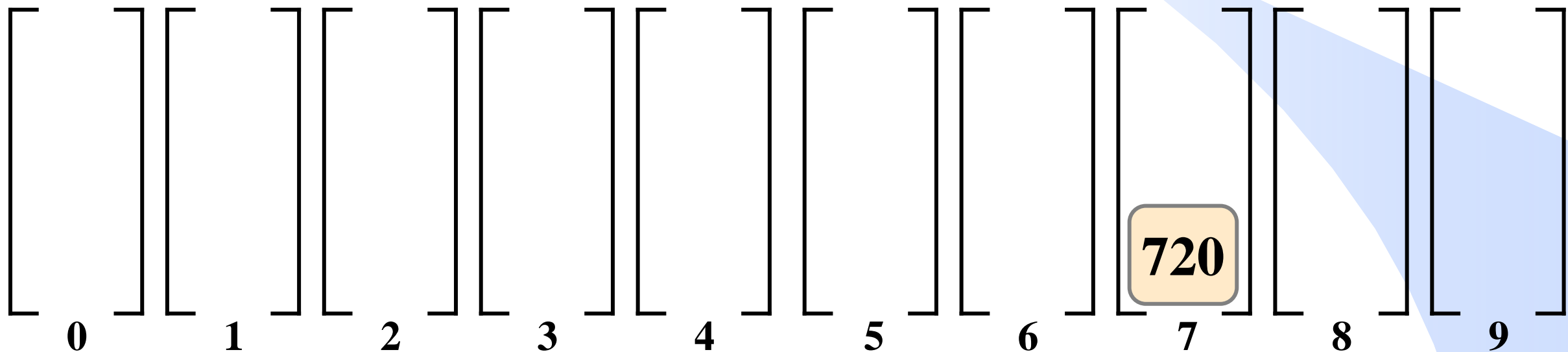


# 基于桶排序的基数排序示例：③ 按百位排序

A

29 236 839 155 257 658

按百位  
分配



# 基于桶排序的基数排序示例：③ 按百位排序

A

236

839

155

257

658

按百位  
分配

29

720

0

1

2

3

4

5

6

7

8

9

# 基于桶排序的基数排序示例：③ 按百位排序

A

839

155

257

658

按百位  
分配

29

236

720

0

1

2

3

4

5

6

7

8

9

# 基于桶排序的基数排序示例：③ 按百位排序

A

155

257

658

按百位  
分配

29

236

720

839

0

1

2

3

4

5

6

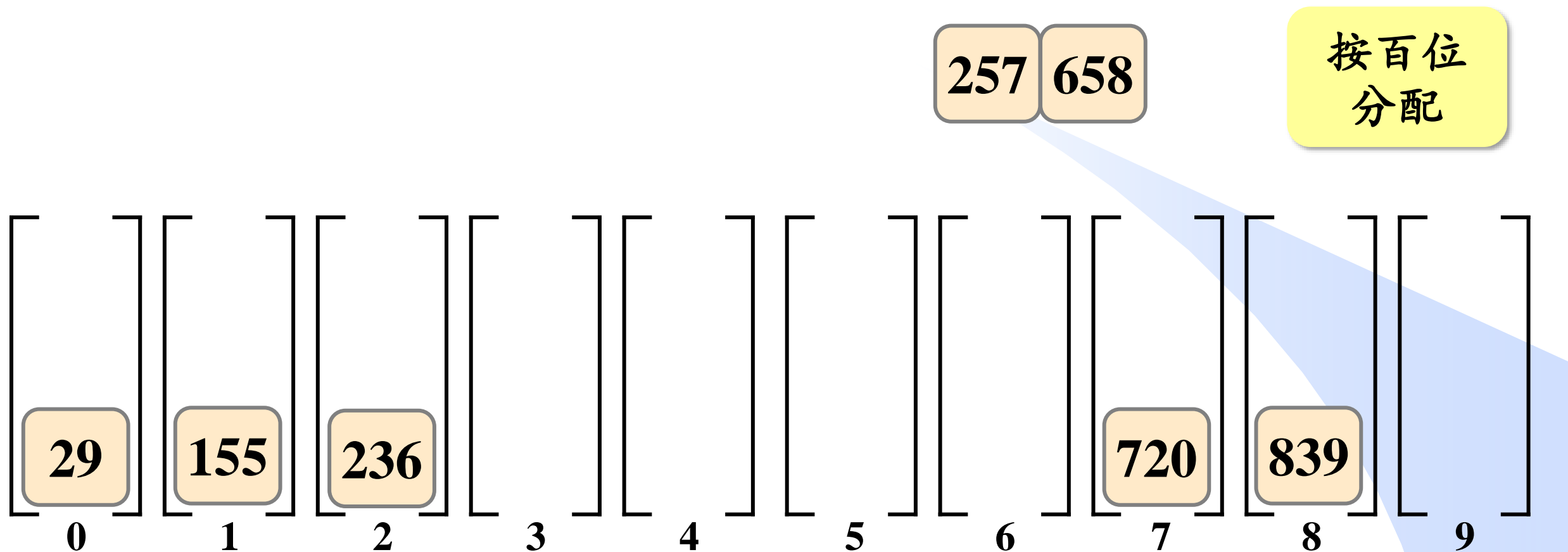
7

8

9

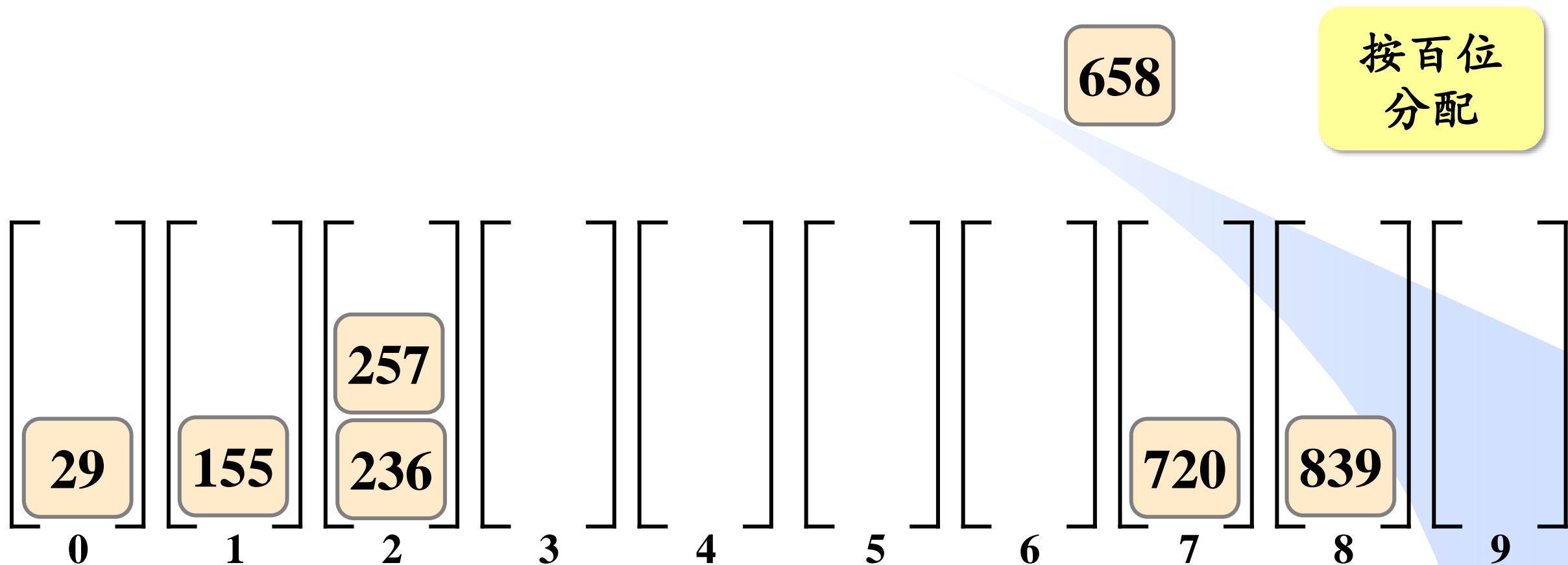
# 基于桶排序的基数排序示例：③ 按百位排序

A



# 基于桶排序的基数排序示例：③ 按百位排序

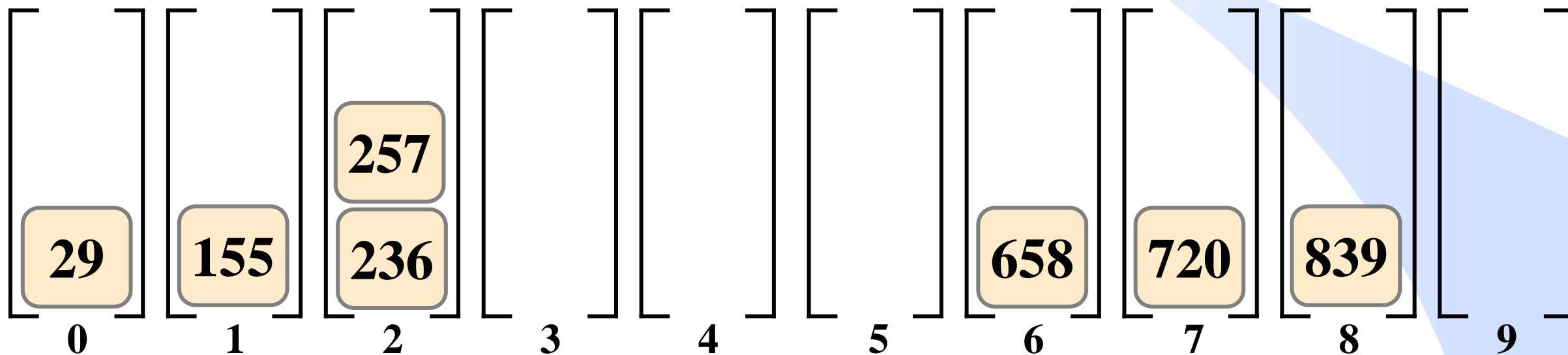
A



# 基于桶排序的基数排序示例：③ 按百位排序

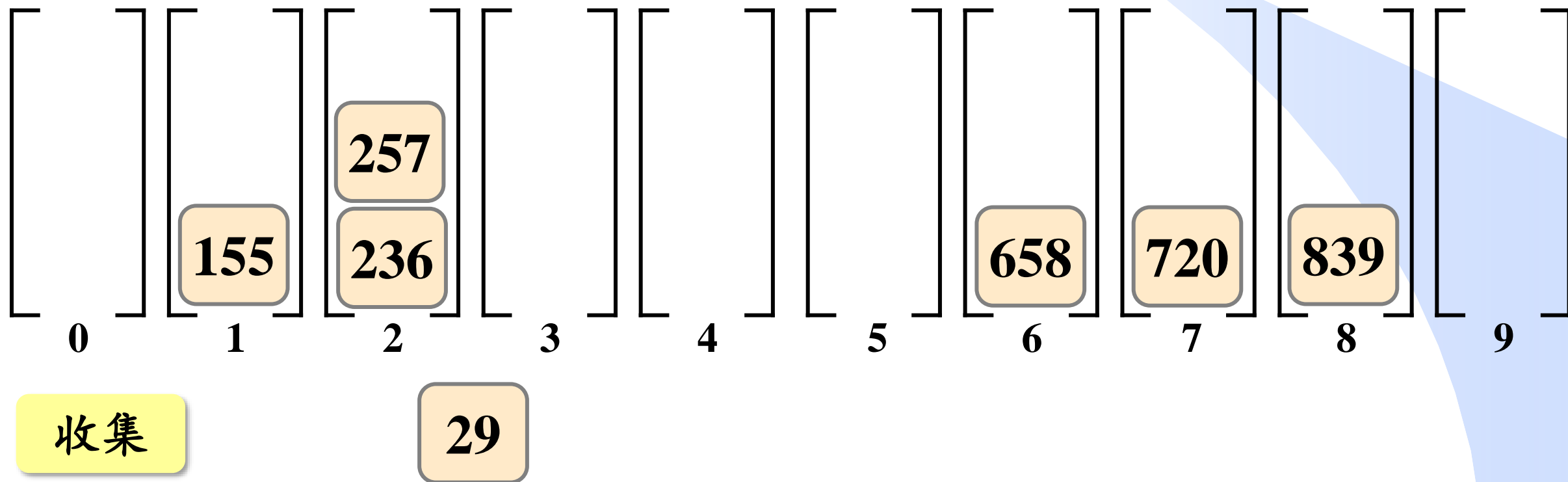
A

按百位  
分配



# 基于桶排序的基数排序示例：③ 按百位排序

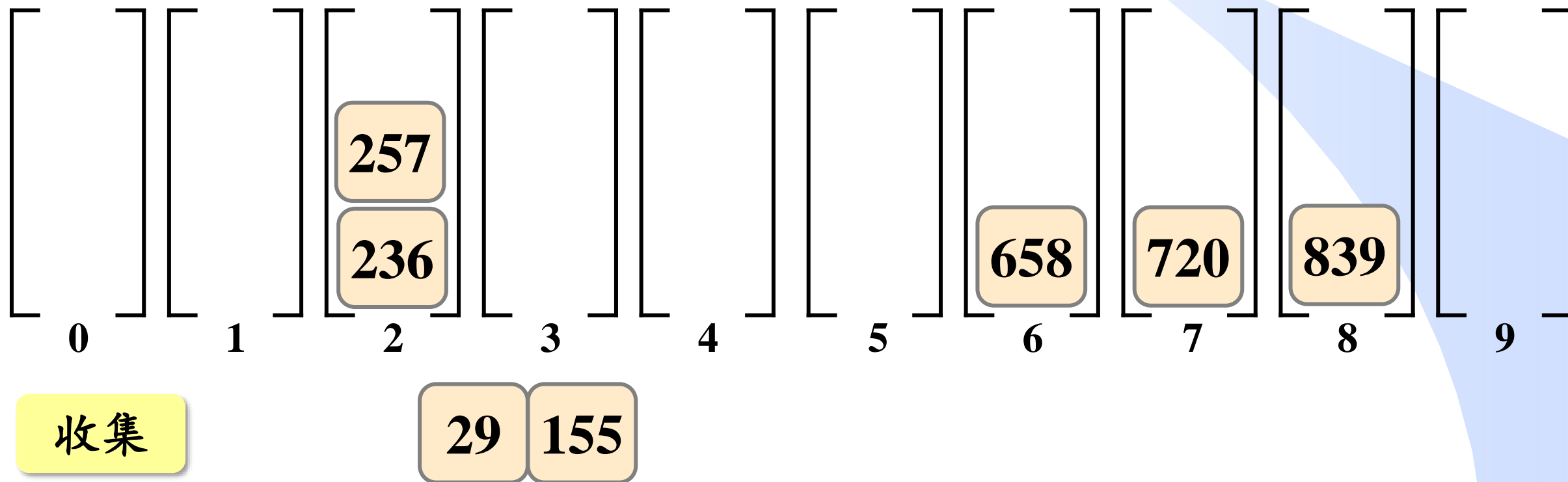
A





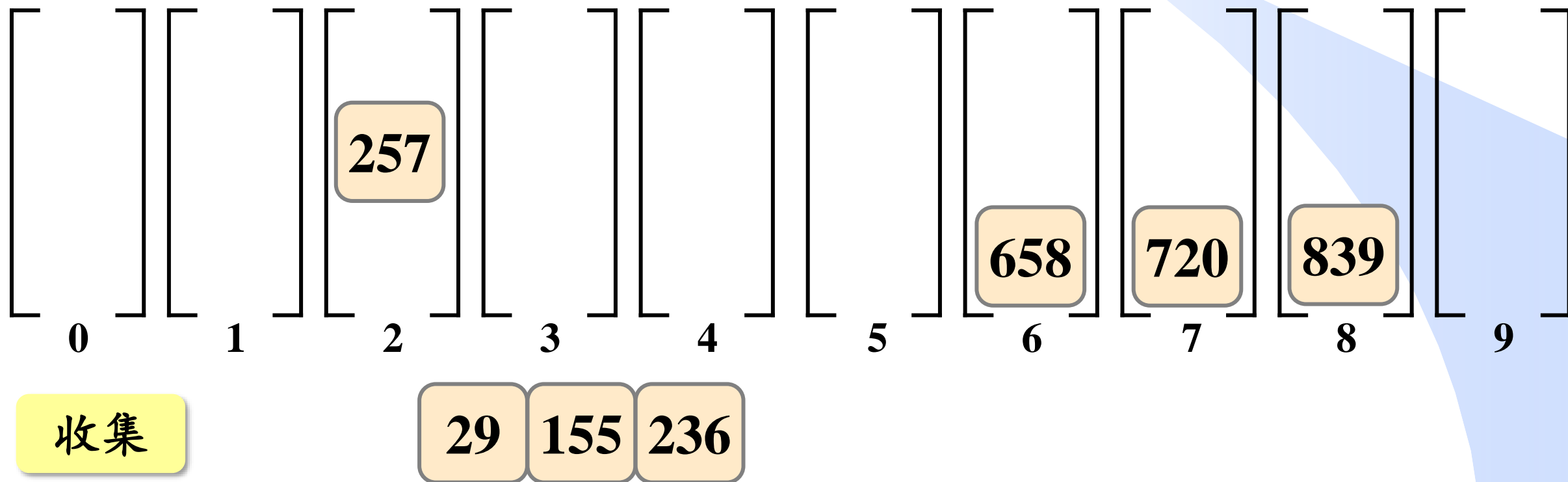
# 基于桶排序的基数排序示例：③ 按百位排序

A



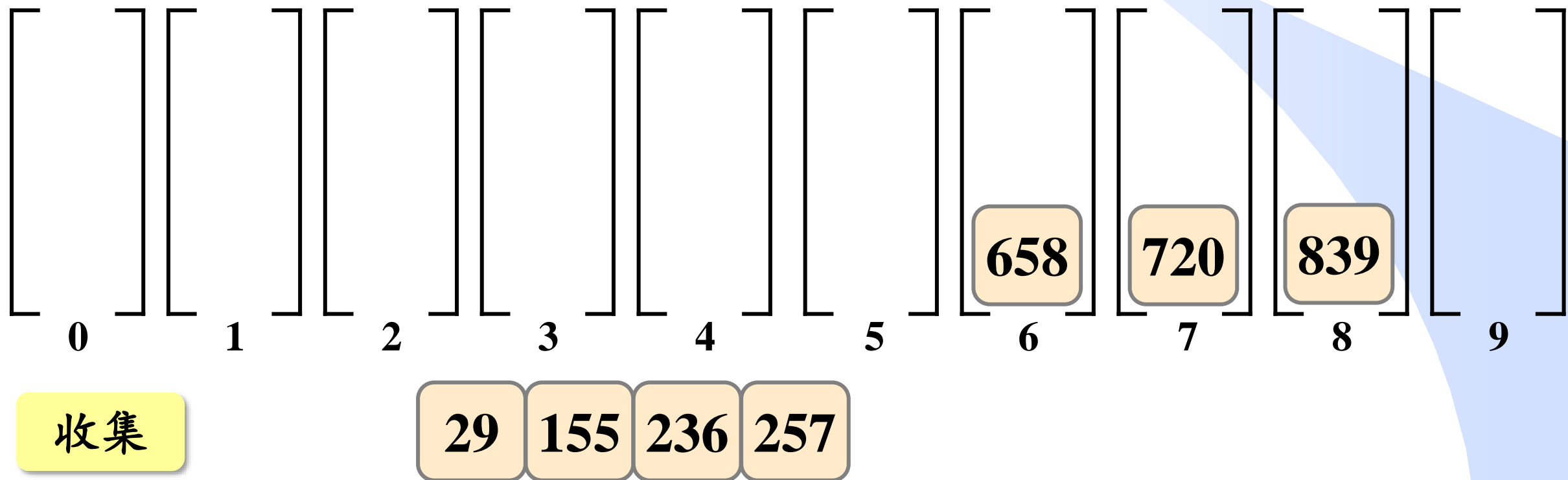
# 基于桶排序的基数排序示例：③ 按百位排序

A



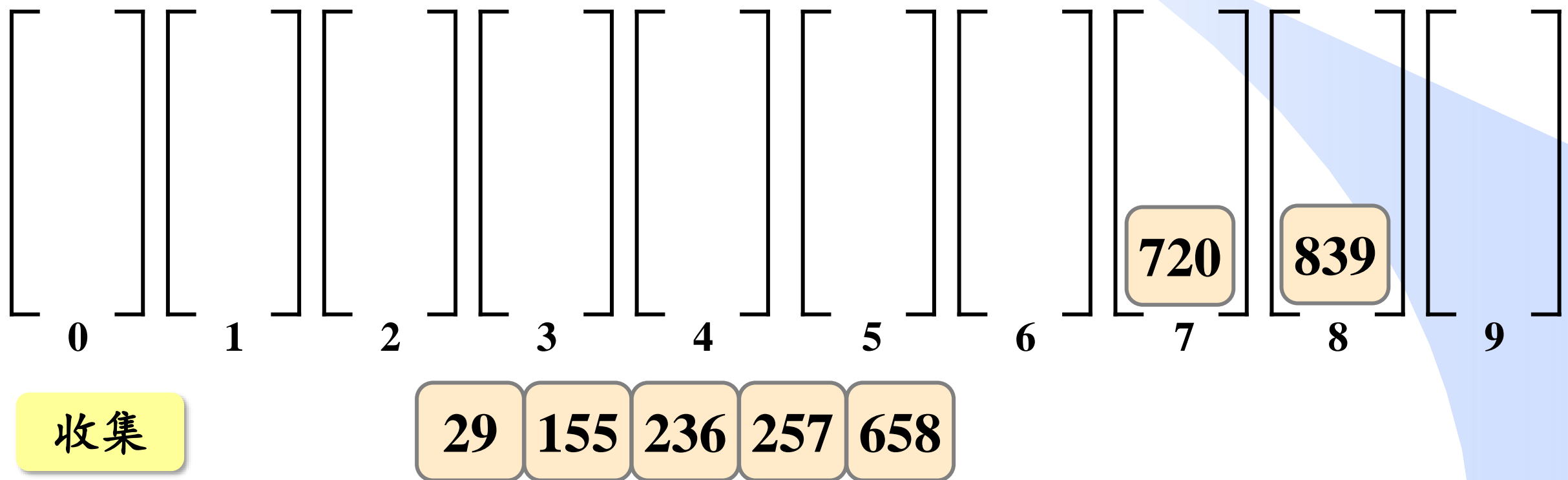
# 基于桶排序的基数排序示例：③ 按百位排序

A



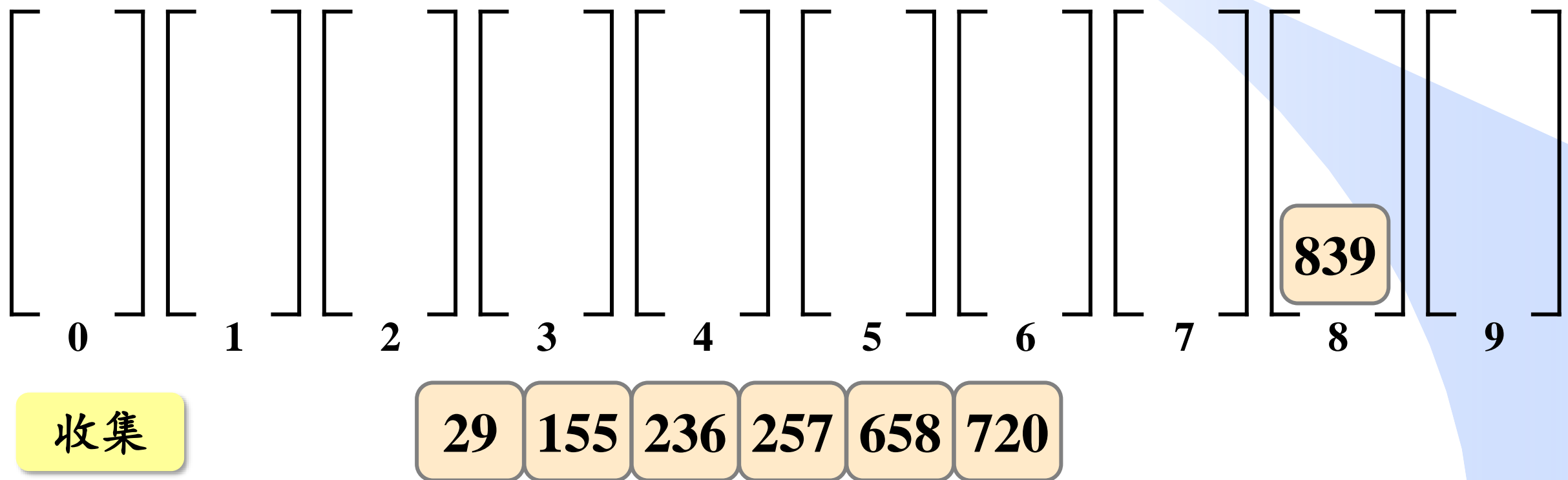
# 基于桶排序的基数排序示例：③ 按百位排序

A



# 基于桶排序的基数排序示例：③ 按百位排序

A



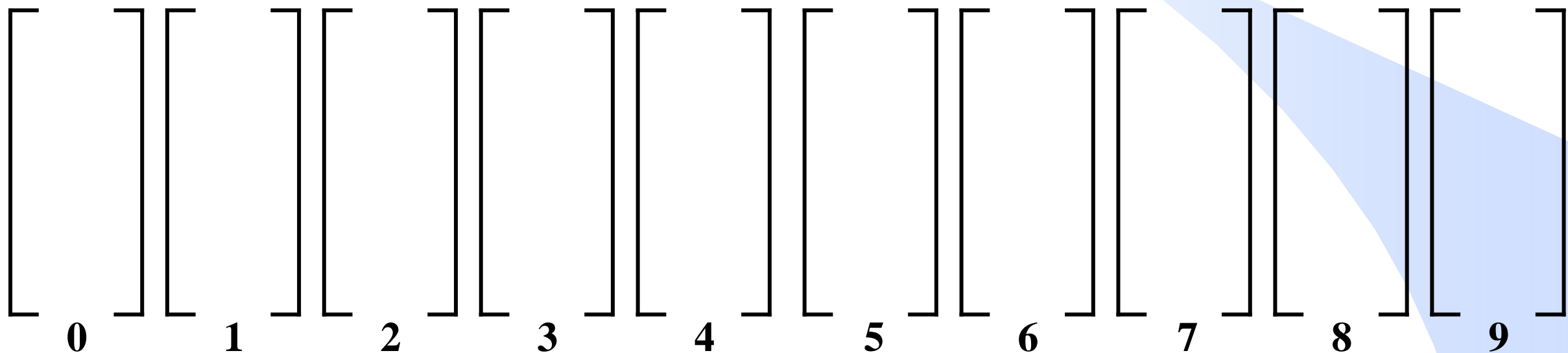
# 基于桶排序的基数排序示例：③ 按百位排序

A

稳定

时间复杂度  
 $O(d(n+r))$

空间复杂度  
 $O(n+r)$



收集

29 155 236 257 658 720 839

# 基于计数排序的基数排序

```

void RadixSort(int R[], int n, int d, int r) { // 每个元素含  $d$  位
    int *B = new int[n+1]; int *cnt = new int[r]; int *K = new int[n+1];
    for (int k=1, base=1; k<=d; k++, base*=r) { // 对第  $k$  位排序
        for (int i=0; i<r; i++) cnt[i]=0;
        for (int i=1; i<=n; i++) K[i]=(R[i]/base)%r;
        for (int i=1; i<=n; i++) cnt[K[i]]++;
        for (int i=1; i<r; i++) cnt[i] += cnt[i-1];
        for (int i=n; i>=1; i--) B[cnt[K[i]]--] = R[i];
        for (int i=1; i<=n; i++) R[i]=B[i];
    }
    delete B; delete cnt;
    delete K;
}

```

取  $R[i]$  的  
第  $k$  位

按第  $k$  位做  
计数排序

稳定

时间复杂度  
 $O(d(n+r))$

空间复杂度  
 $O(n+r)$

# 基数排序总结

排序算法	时间复杂度			空间复杂度	稳定性
	最好	平均	最坏		
基数排序	$O(d(n+r))$	$O(d(n+r))$	$O(d(n+r))$	$O(n+r)$	稳定

$d$ 为关键词的位数， $r$ 为基数  
若 $d$ 和 $r$ 为常数，则时空复杂度为线性

适合于：关键词包含固定的位数或可以拆分为多位



# 基数排序 vs 计数排序/桶排序

➤ 对10个整数排序，每个整数的范围为[0, 1000)。

29 257 658 839 236 720 56 237 999 155

排序算法	参数	特点	时间
桶排序	$n=10$ $m=1000$	需1000个桶	$O(n+m)=$ $O(10+1000)$
计数排序	$n=10$ $m=1000$	需长度为1000的 $cnt$ 数组	$O(n+m)=$ $O(10+1000)$
基数排序	$n=10$ $d=3, r=10$	需10个桶或长度为10的 $cnt$ 数组，但要做3轮	$O(d(n+r))=$ $O(3(10+10))$

# 分布排序总结

排序算法	时间复杂度			空间复杂度	稳定性
	最好	平均	最坏		
桶排序	$O(n+m)$	$O(n+m)$	$O(n+m)$	$O(n+m)$	稳定
计数排序	$O(n+m)$	$O(n+m)$	$O(n+m)$	$O(n+m)$	稳定
基数排序	$O(d(n+r))$	$O(d(n+r))$	$O(d(n+r))$	$O(n+r)$	稳定

每个元素的值域为 $[0, m)$   
 $d$ 为关键词的位数,  $r$ 为基数

并不基于关键词的比较, 需提前知道元素的分布规律