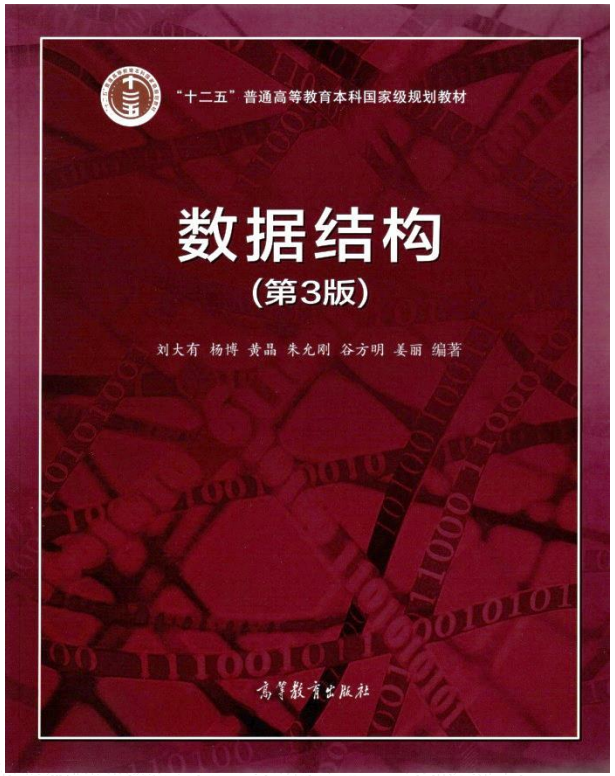


分布排序

- 桶排序
- 计数排序
- 基数排序

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

Last updated on 2023.12

邓明扬

麻省理工学院21级本科生

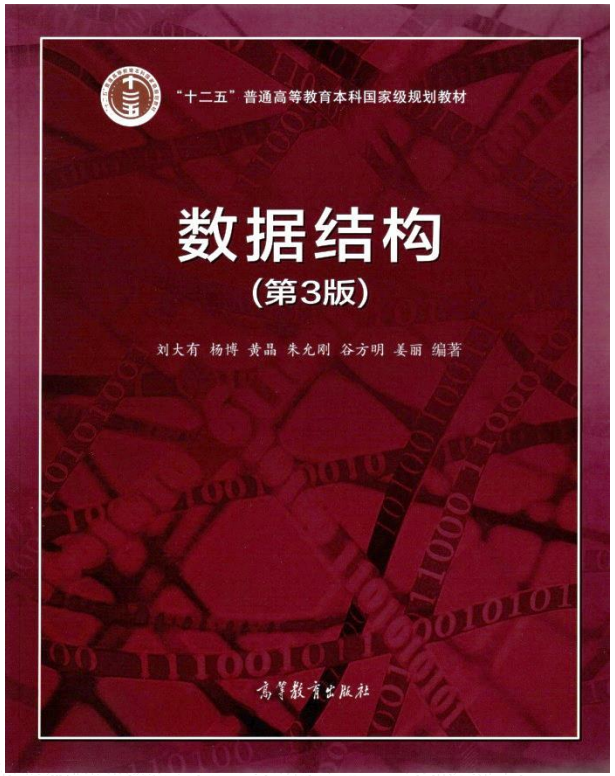
2020年NOI全国中学生信息学奥赛决赛金牌

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

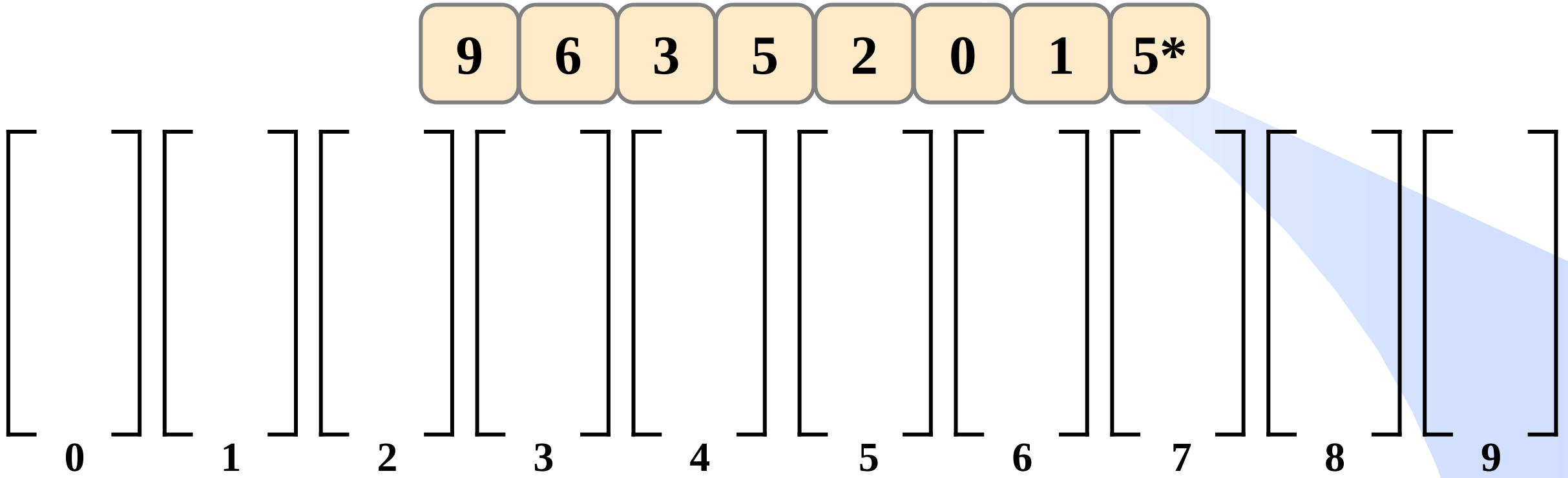
zhuyungang@jlu.edu.cn

Last updated on 2023.12

桶排序（Bucket Sort）

A

待排序的文件*R*包含*n*个整数，每个整数的值域为[0, *m*)

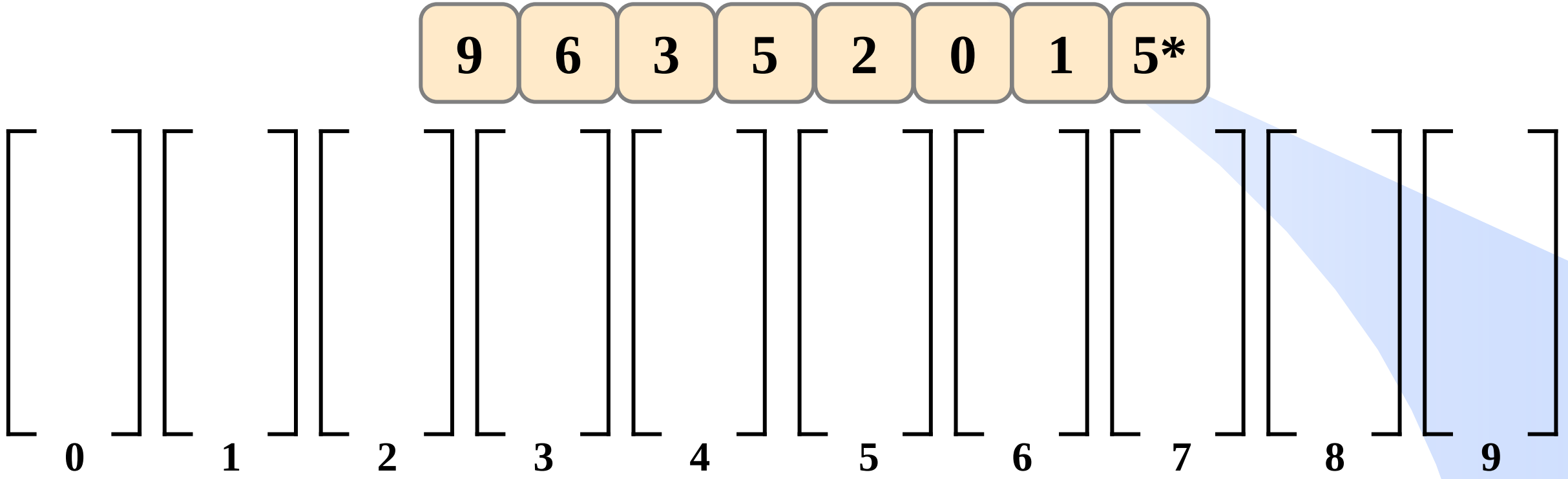


设置 *m* 个桶，对应元素的值域0...*m*-1

桶排序

A

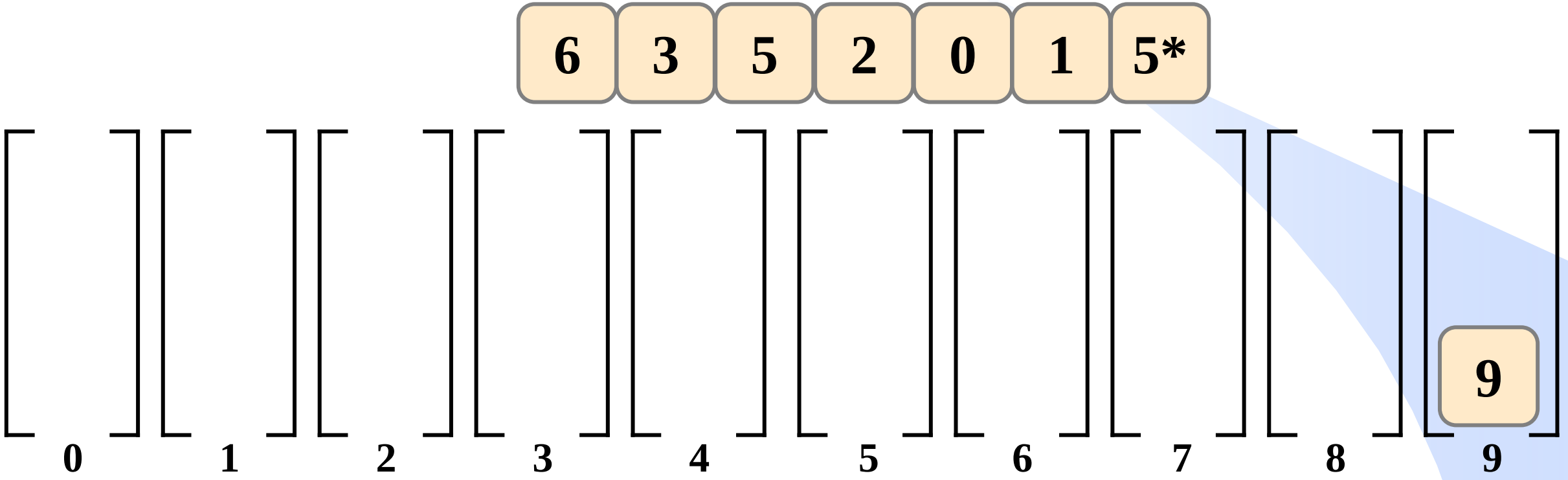
① 分配：扫描n个元素，按关键词放入对应的桶中



桶排序

A

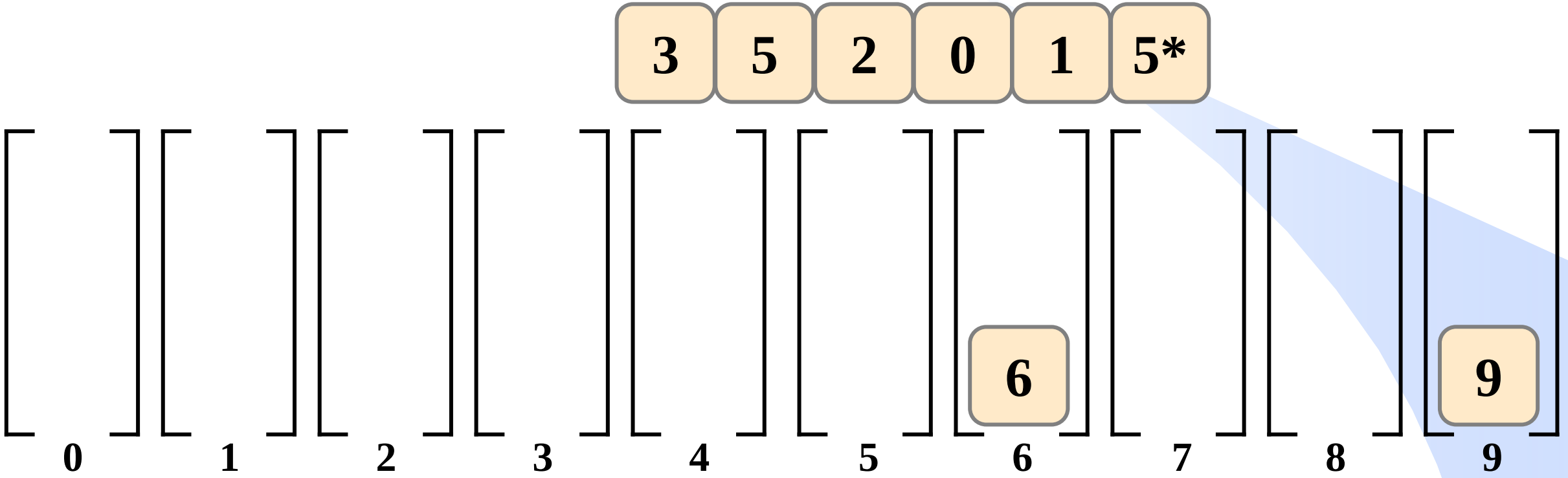
① 分配：扫描n个元素，按关键词放入对应的桶中



桶排序

A

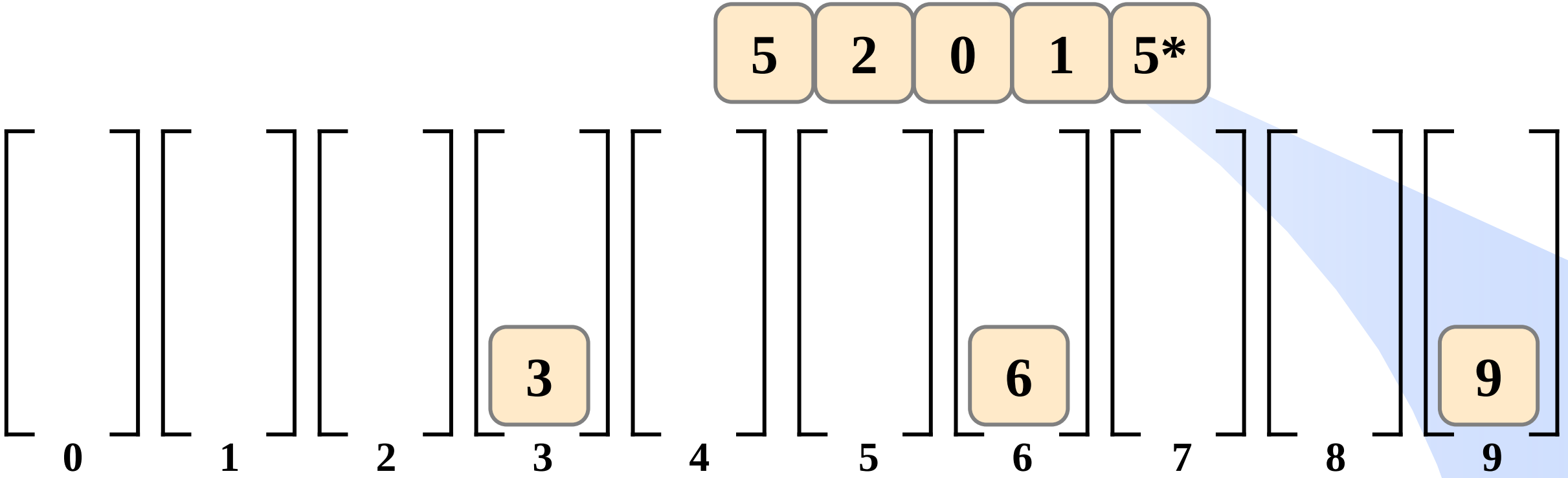
① 分配：扫描n个元素，按关键词放入对应的桶中



桶排序

A

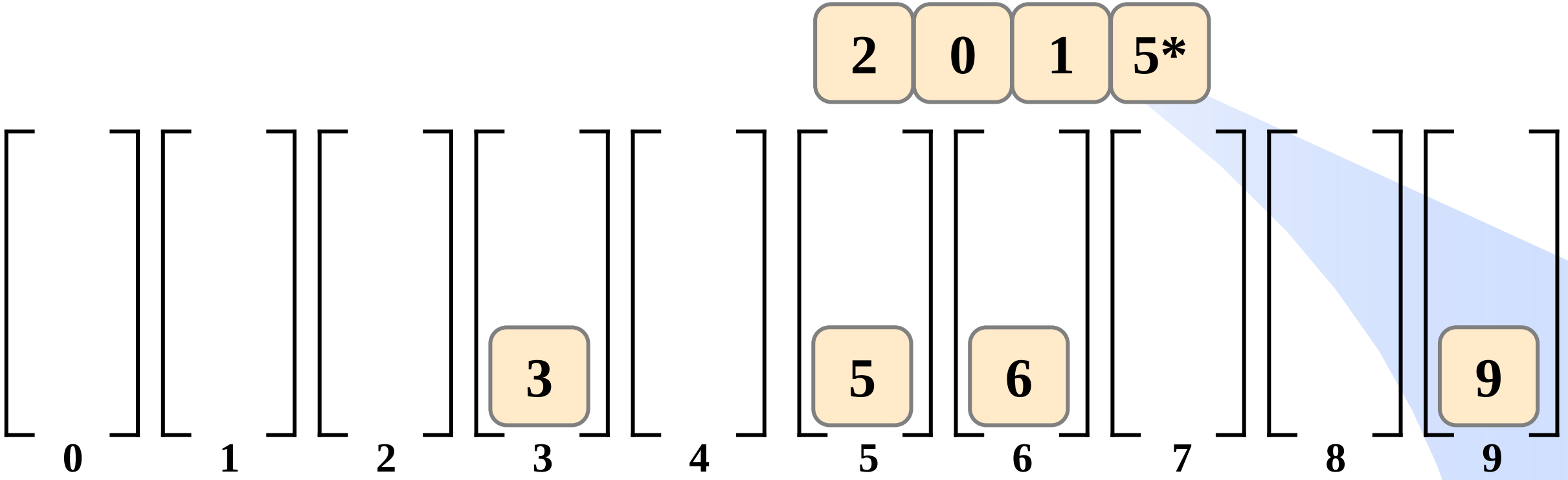
① 分配：扫描n个元素，按关键词放入对应的桶中



桶排序

A

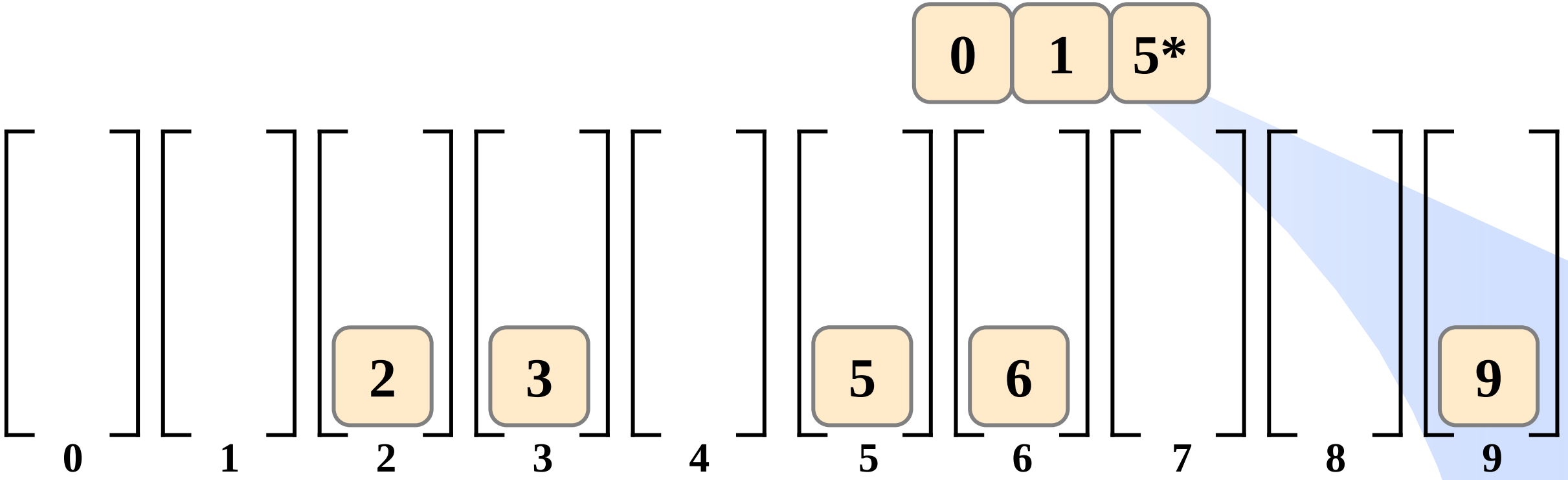
① 分配：扫描n个元素，按关键词放入对应的桶中



桶排序

A

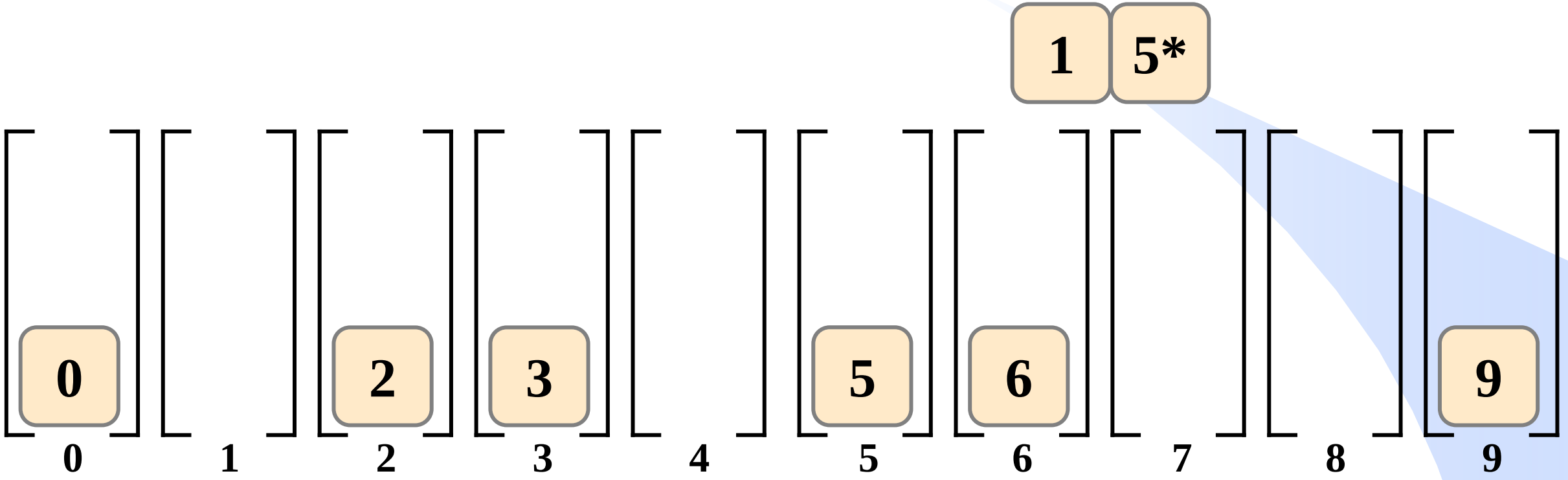
① 分配：扫描n个元素，按关键词放入对应的桶中



桶排序

A

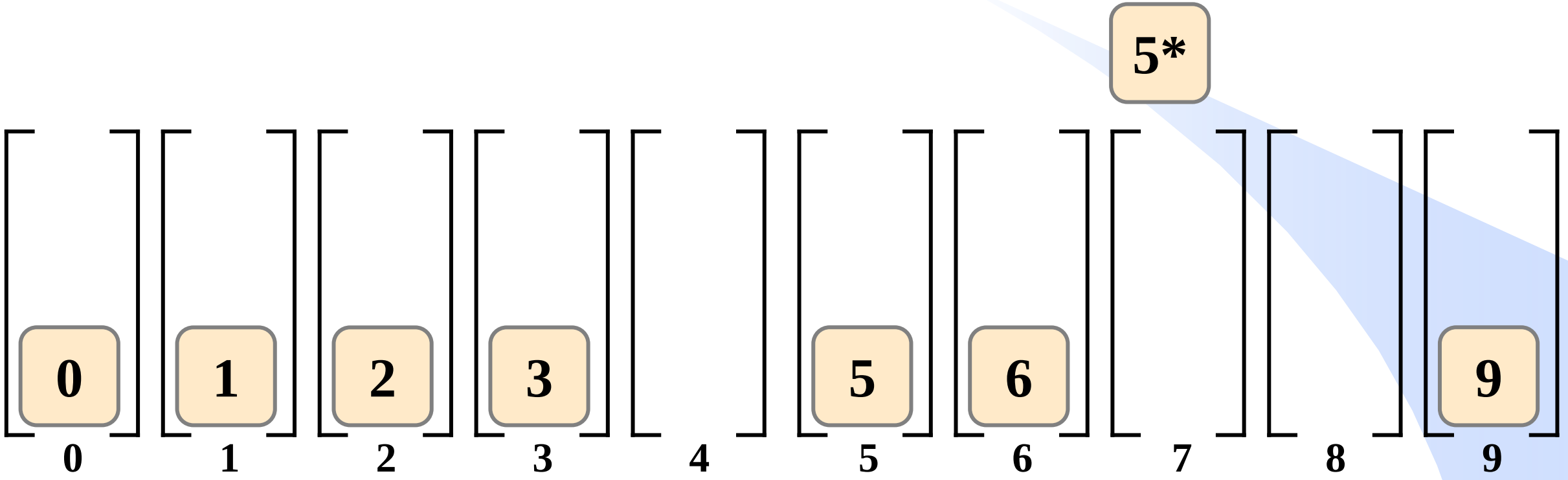
① 分配：扫描n个元素，按关键词放入对应的桶中



桶排序

A

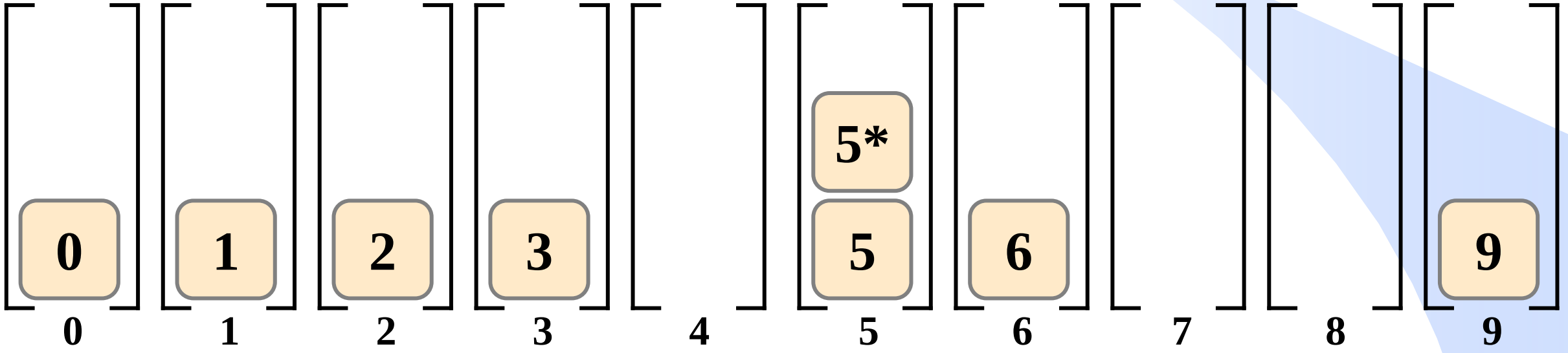
① 分配：扫描n个元素，按关键词放入对应的桶中



桶排序

A

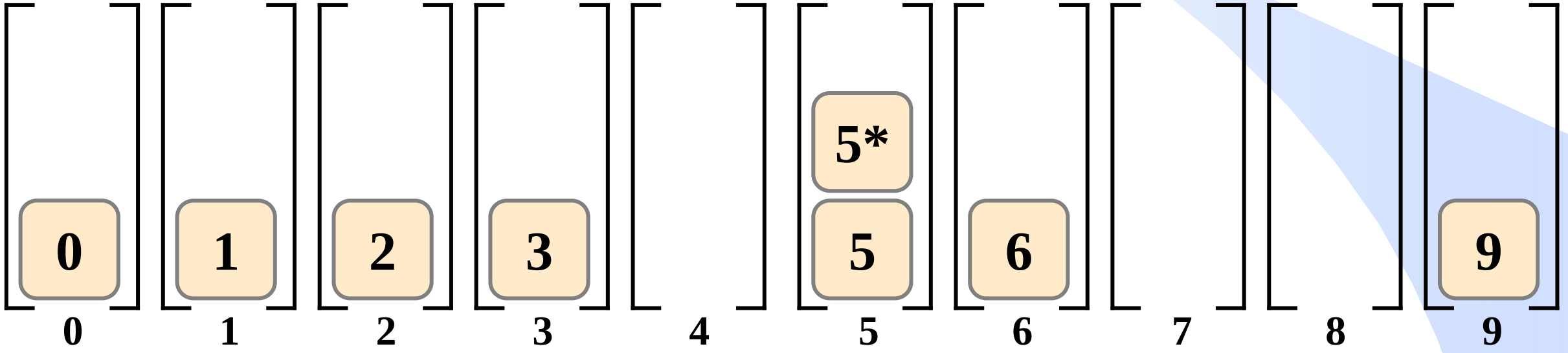
① 分配：扫描n个元素，按关键词放入对应的桶中



桶排序

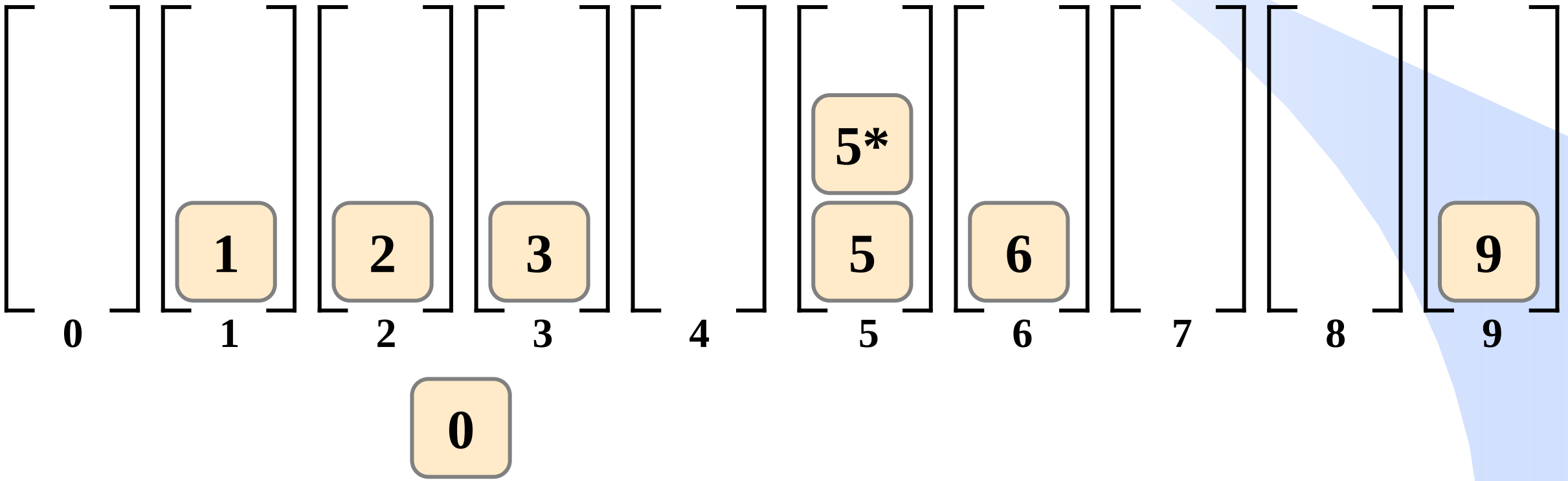
A

② 收集：依次将桶中元素取出



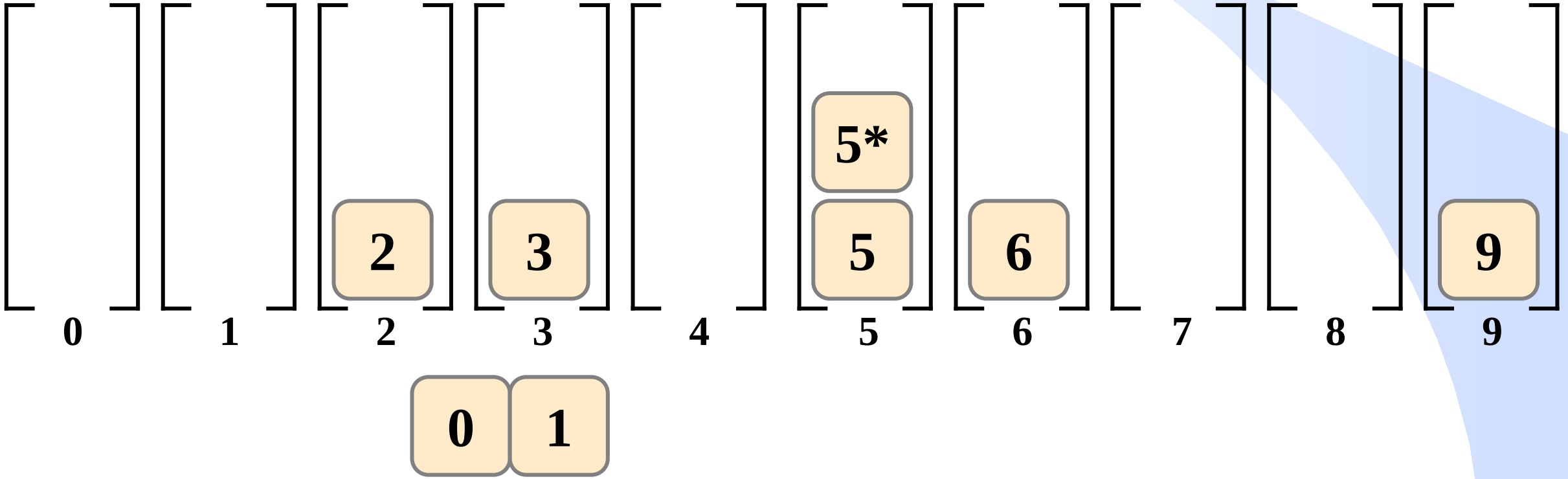
桶排序

② 收集：依次将桶中元素取出



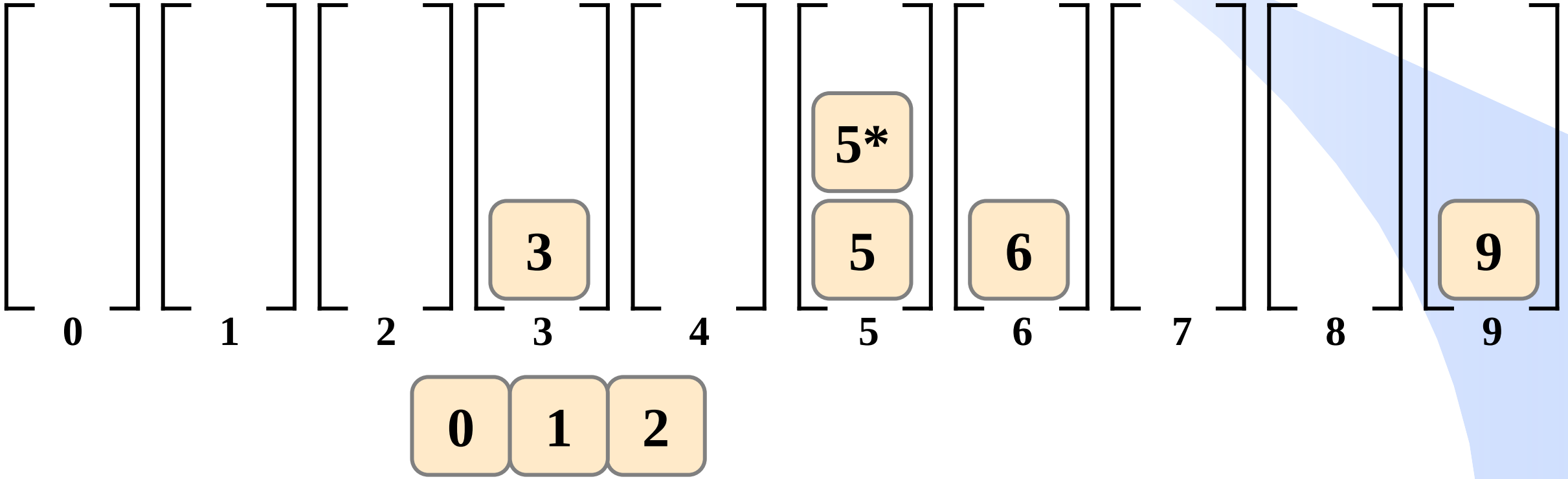
桶排序

② 收集：依次将桶中元素取出



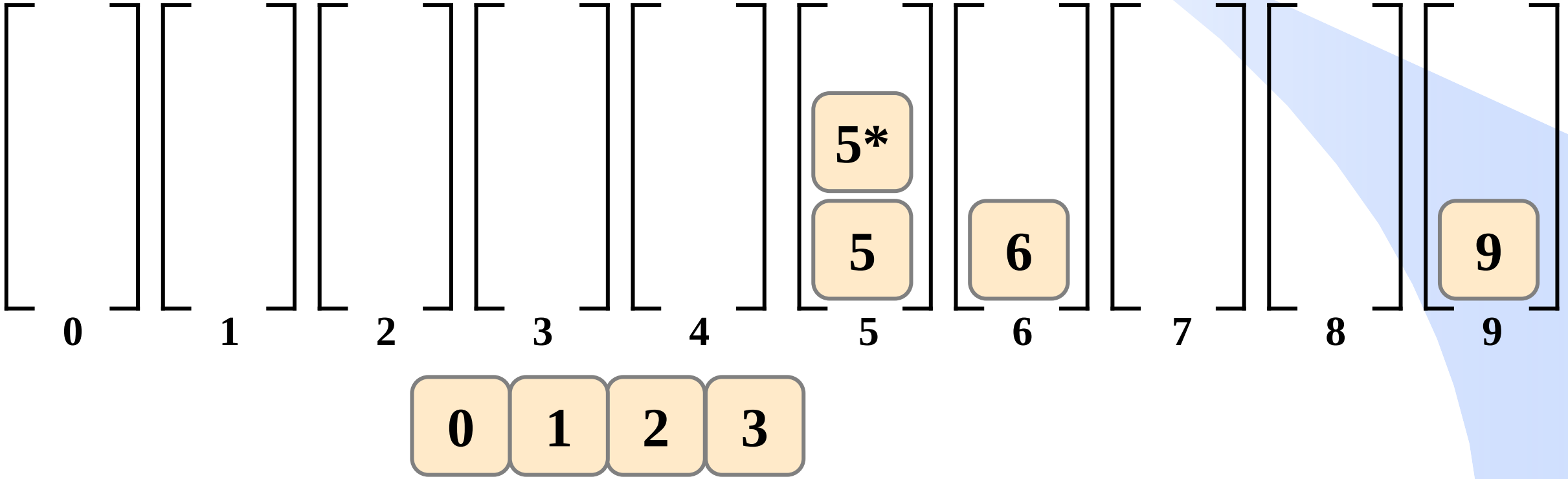
桶排序

② 收集：依次将桶中元素取出



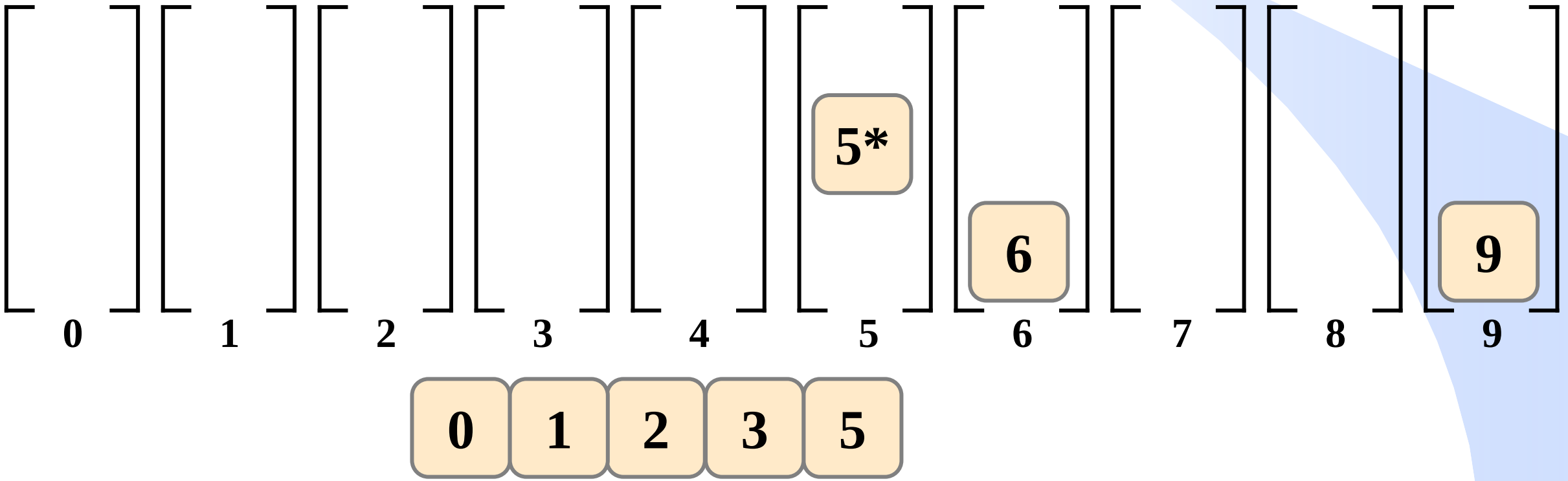
桶排序

② 收集：依次将桶中元素取出



桶排序

② 收集：依次将桶中元素取出

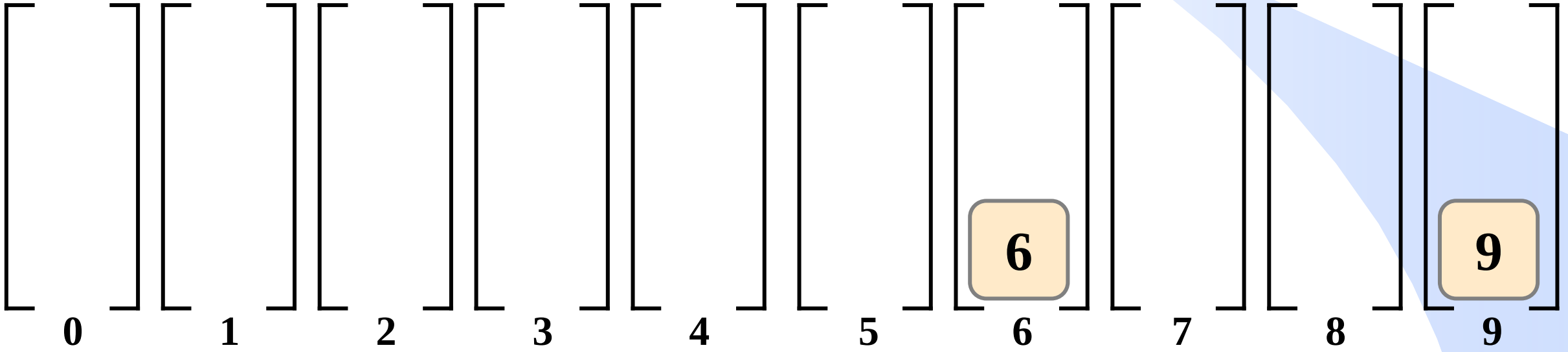


桶排序

A

② 收集：依次将桶中元素取出

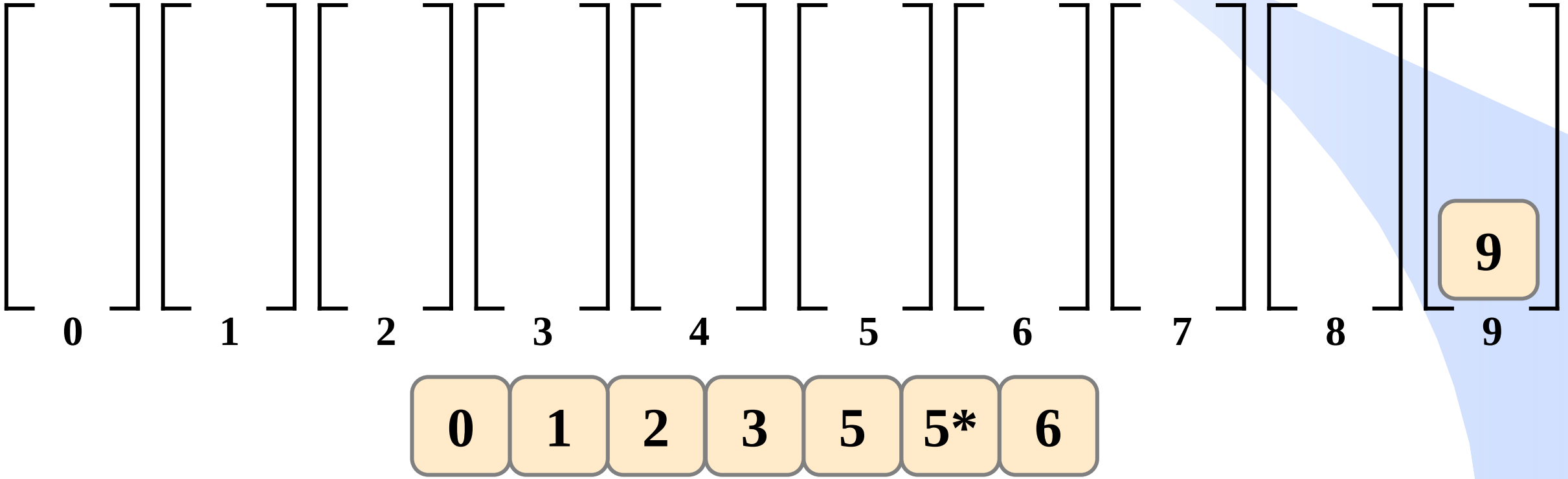
桶排序是稳定的：关键词相同的元素在同一桶里，排在前面的元素在分配时先入桶，收集时也先出桶



0 1 2 3 5 5*

桶排序

② 收集：依次将桶中元素取出

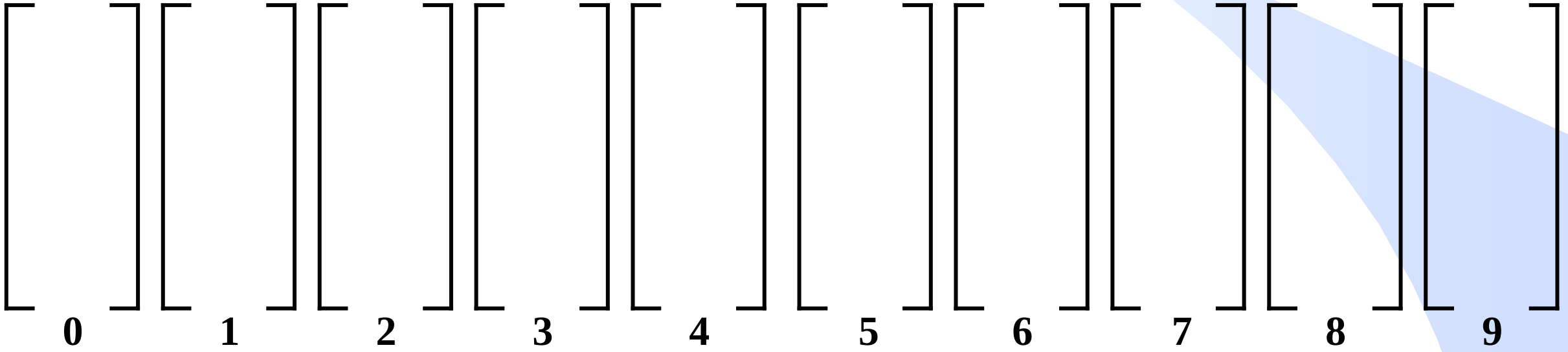


桶排序

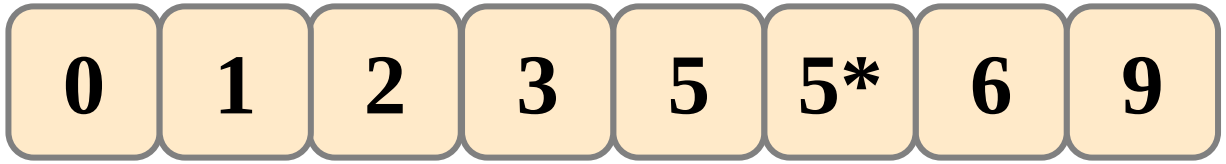
A

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

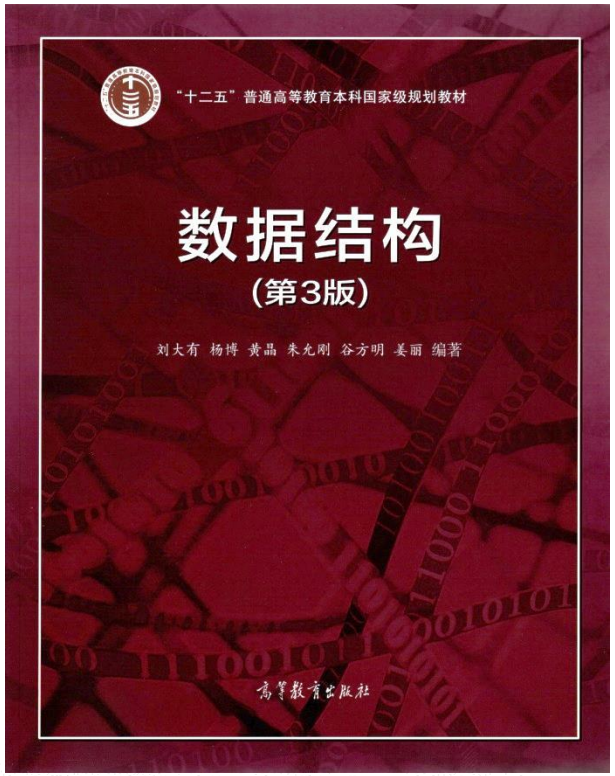
桶排序总结

A

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

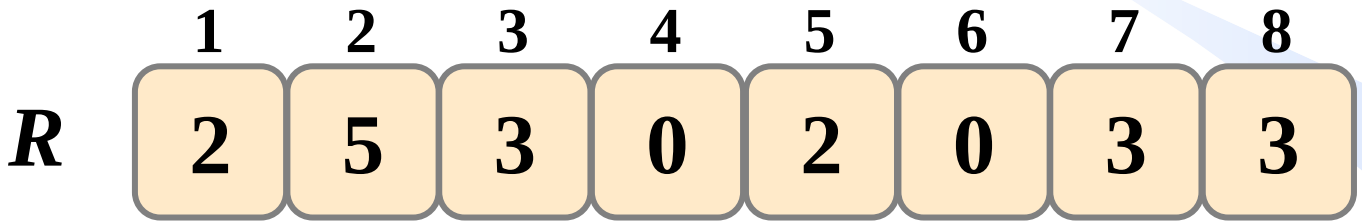
zhuyungang@jlu.edu.cn

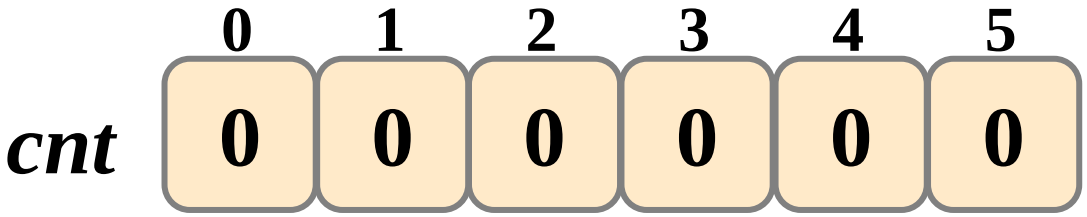
Last updated on 2023.12

计数排序（Counting Sort）

B

待排序的数组*R*包含*n*个整数，每个整数的值域为[0, *m*)



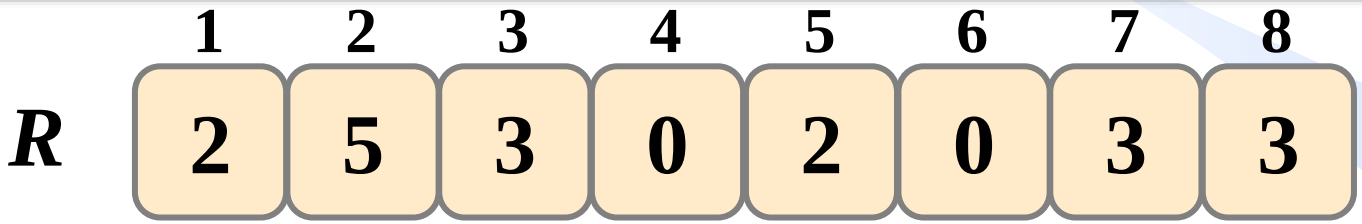


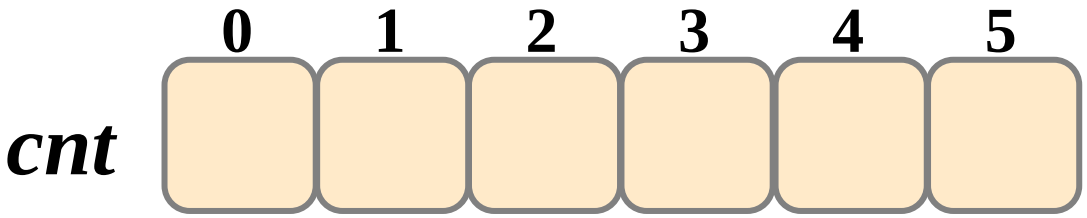
设置一个长度为 *m* 的数组*cnt*[ ]，初值为0

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*[*i*]为数组*R*中有多少个*i*。

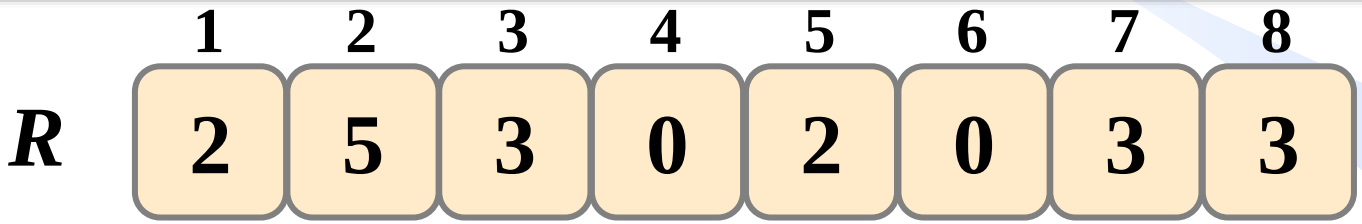


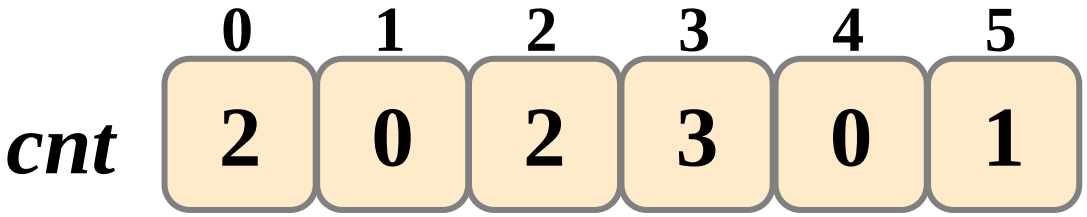


计数排序（Counting Sort）

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*[*i*]为数组*R*中有多少个*i*。



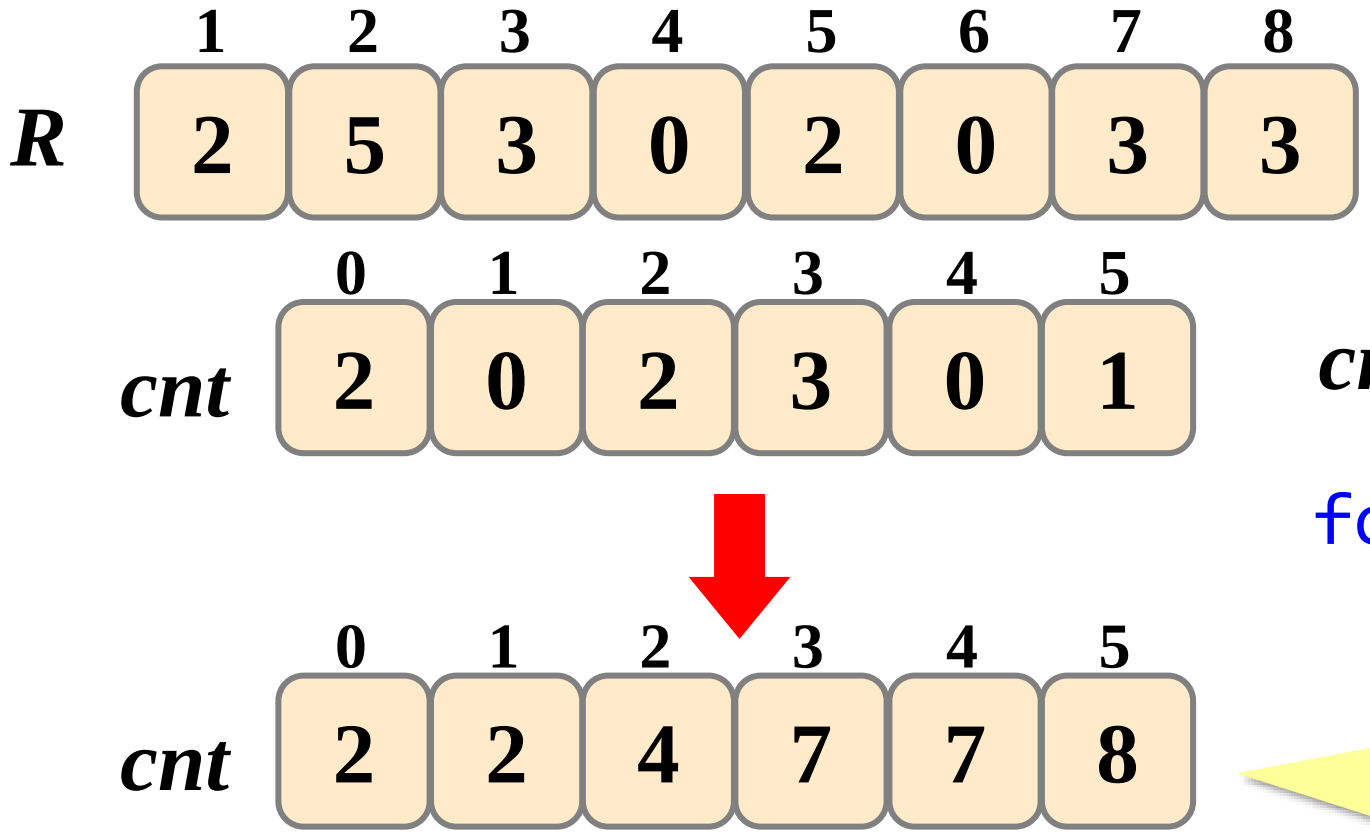



```
for(int i=1; i<=n; i++)
    cnt[R[i]]++;
```

计数排序（Counting Sort）

B

② 对*cnt*数组求前缀和，使*cnt*[*i*]表示数组*R*中≤*i*的元素有多少个



$cnt[i] \leftarrow cnt[i-1] + cnt[i]$

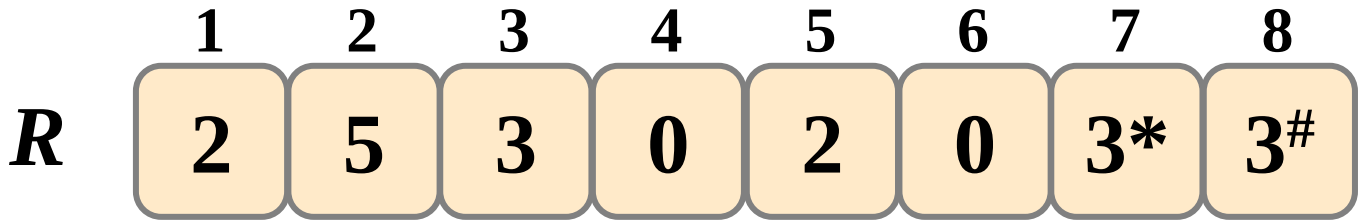
```
for(int i=1; i<m; i++)
    cnt[i]+=cnt[i-1];
```

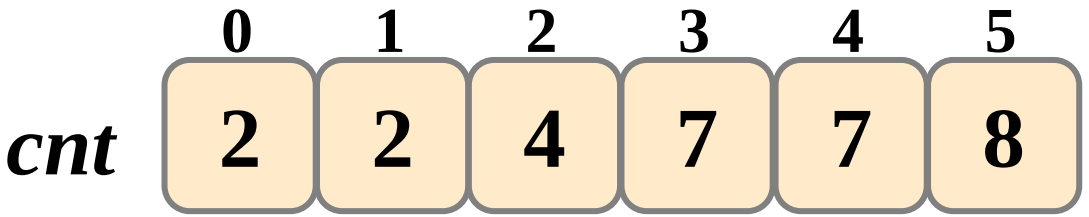
cnt[*i*]就反映了元素*i*的排名，或者说元素*i*在排序后的数组中的位置

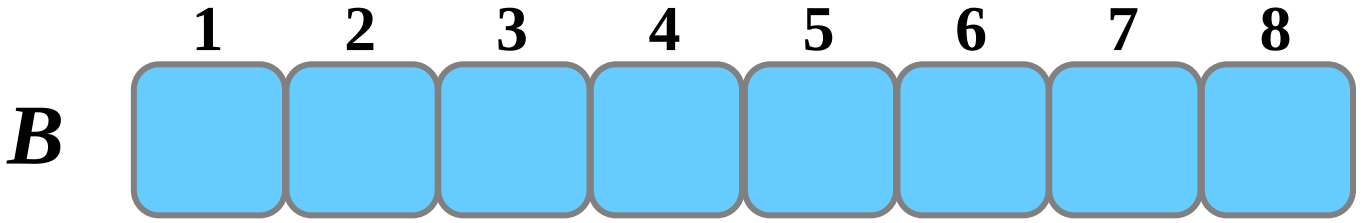
计数排序（Counting Sort）

B

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





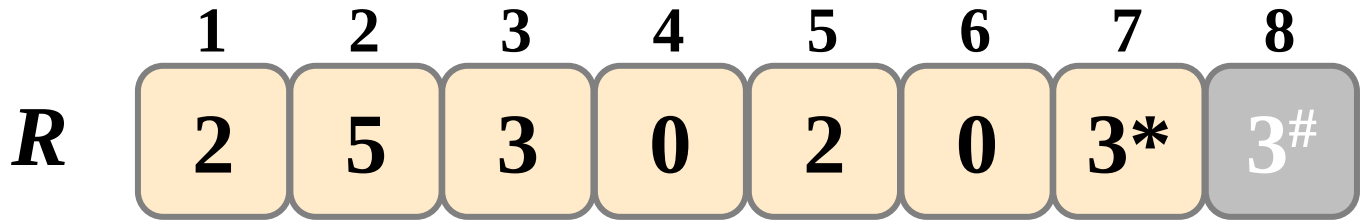


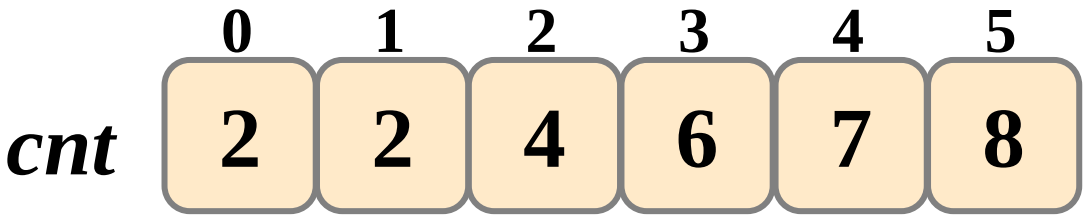

```
for(int i=n; i>=1; i--){
    int K = R[i];
    B[cnt[K]] = K;
    cnt[K]--;
}
```

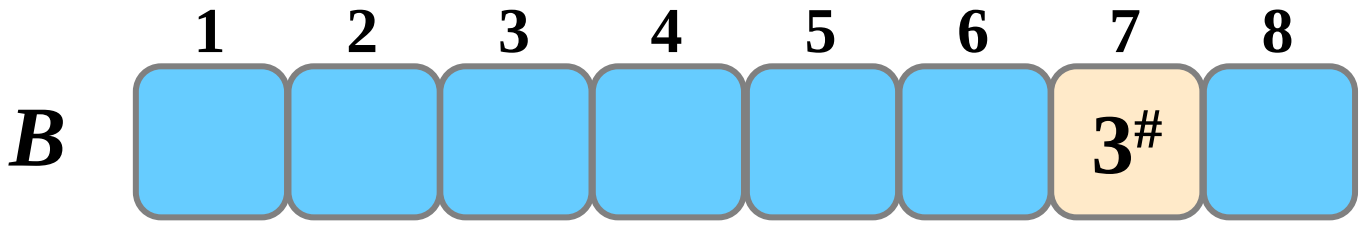
计数排序（Counting Sort）

B

③从右往左扫描R的每个元素K, 基于cnt[K]将K放入排序后的数组





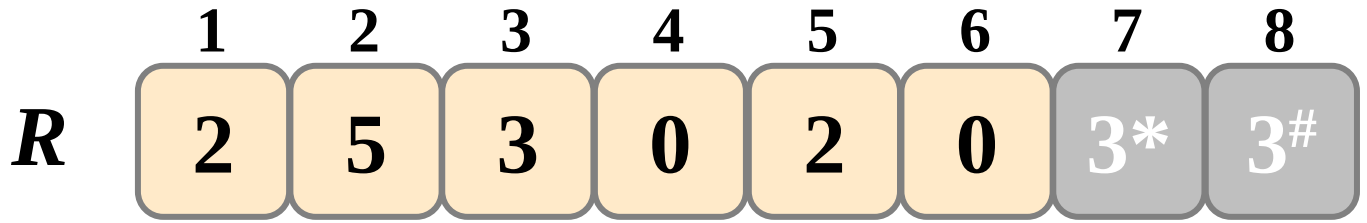


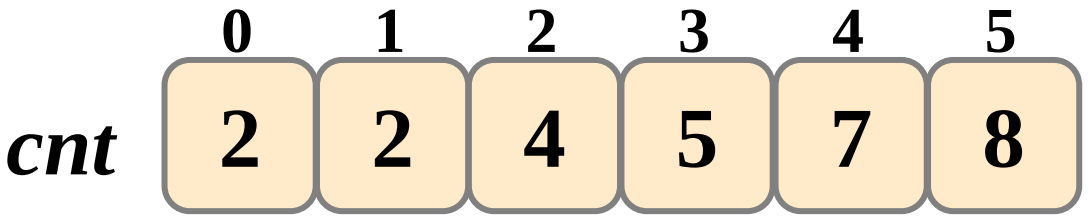

```
for(int i=n; i>=1; i--){
    int K = R[i];
    B[cnt[K]] = K;
    cnt[K]--;
}
```

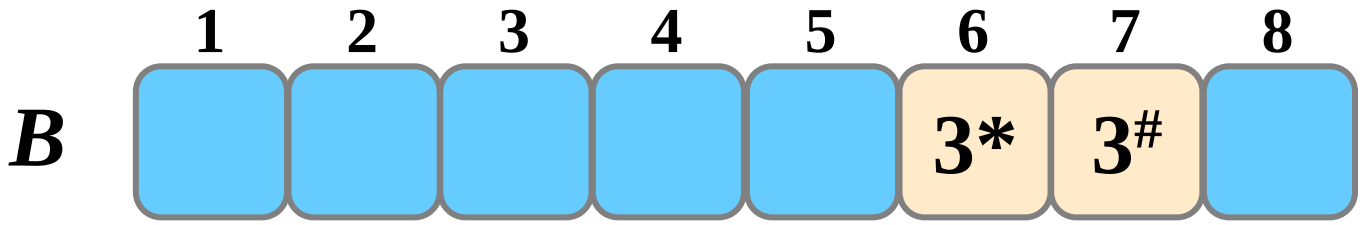
计数排序（Counting Sort）

B

③从右往左扫描R的每个元素K, 基于cnt[K]将K放入排序后的数组







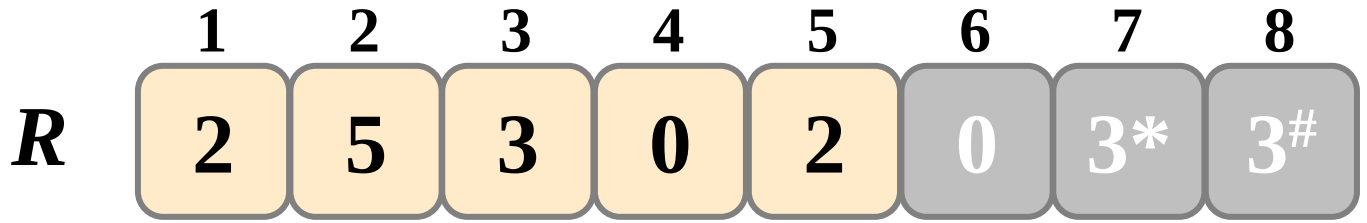

```
for(int i=n; i>=1; i--){
    int K = R[i];
    B[cnt[K]] = K;
    cnt[K]--;
}
```

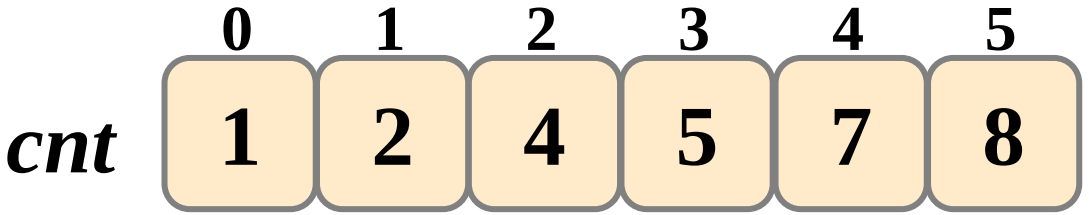
稳定

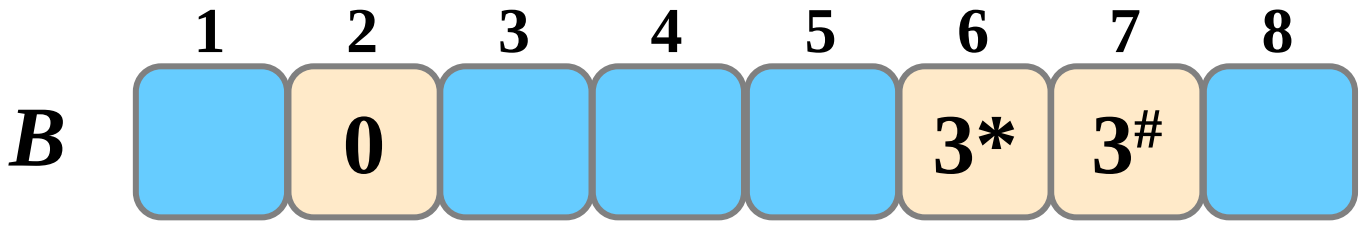
计数排序（Counting Sort）

B

③从右往左扫描R的每个元素K, 基于cnt[K]将K放入排序后的数组





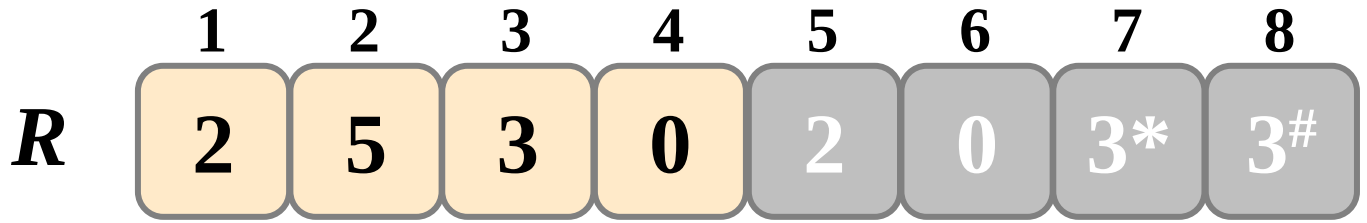


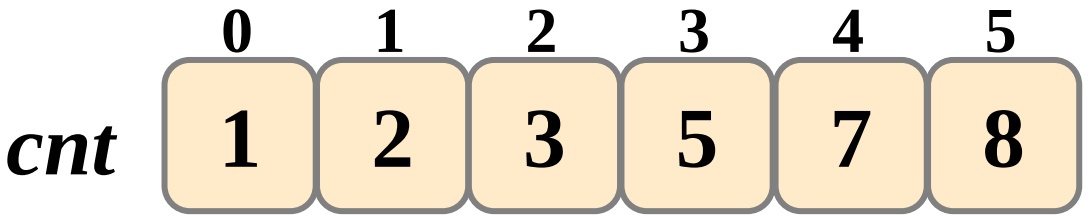

```
for(int i=n; i>=1; i--){
    int K = R[i];
    B[cnt[K]] = K;
    cnt[K]--;
}
```

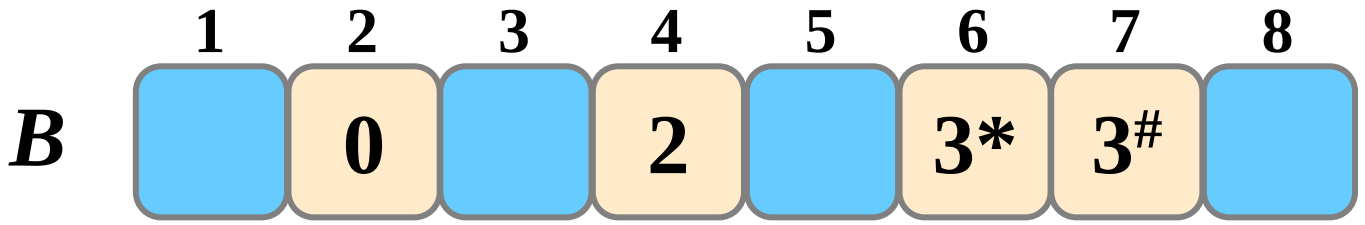
计数排序（Counting Sort）

B

③从右往左扫描R的每个元素K, 基于cnt[K]将K放入排序后的数组





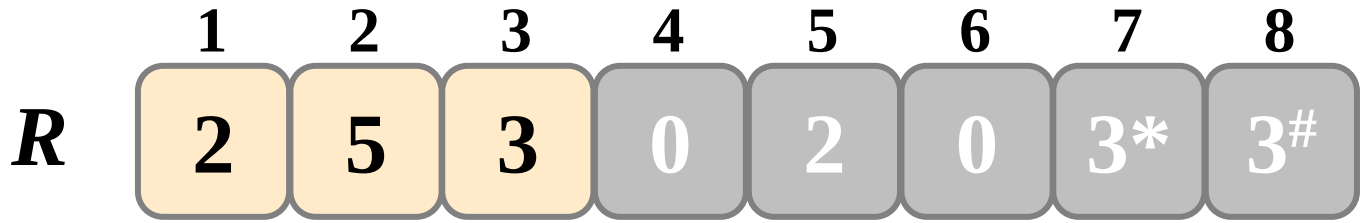


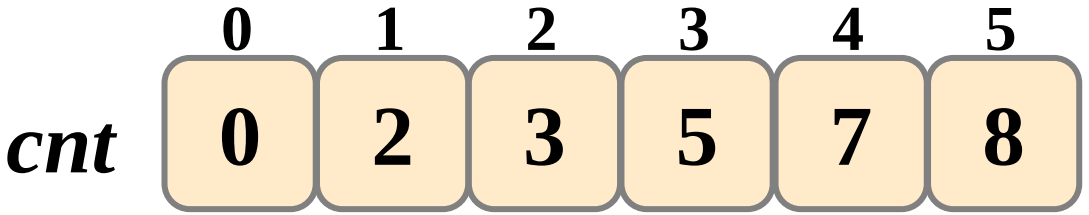

```
for(int i=n; i>=1; i--){
    int K = R[i];
    B[cnt[K]] = K;
    cnt[K]--;
}
```

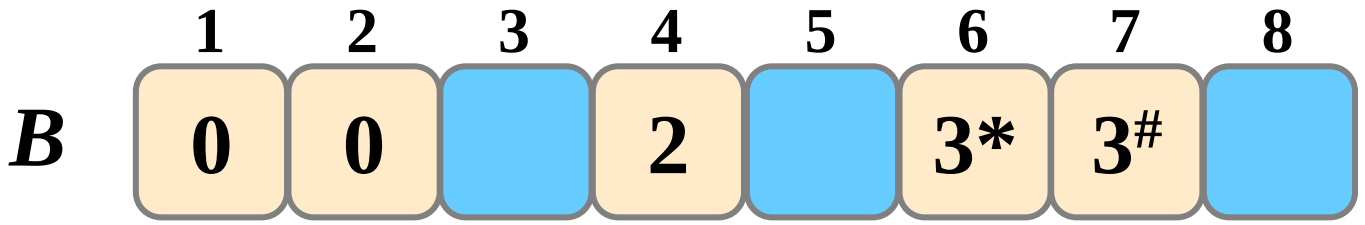
计数排序（Counting Sort）

B

③从右往左扫描R的每个元素K, 基于cnt[K]将K放入排序后的数组





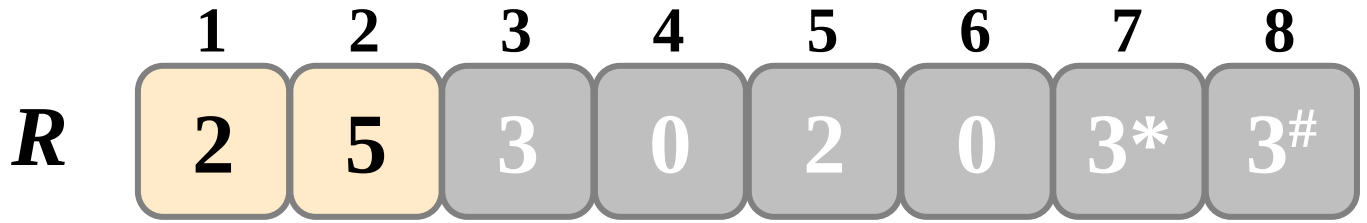


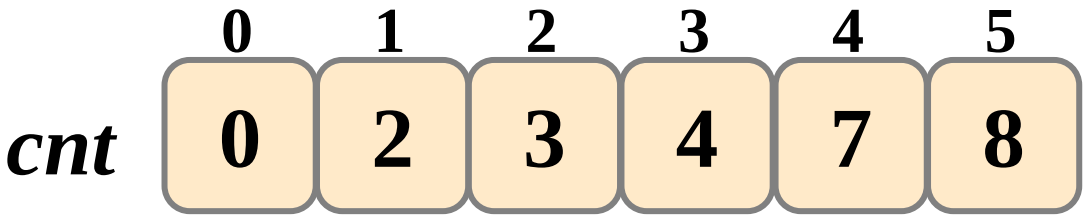

```
for(int i=n; i>=1; i--){
    int K = R[i];
    B[cnt[K]] = K;
    cnt[K]--;
}
```

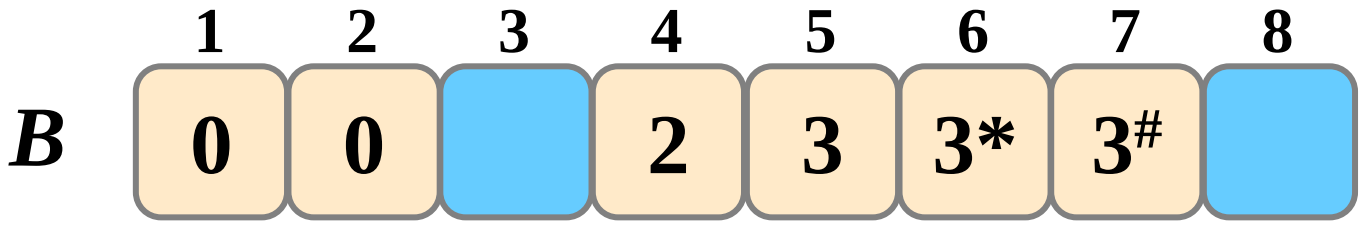
计数排序（Counting Sort）

B

③从右往左扫描R的每个元素K, 基于cnt[K]将K放入排序后的数组





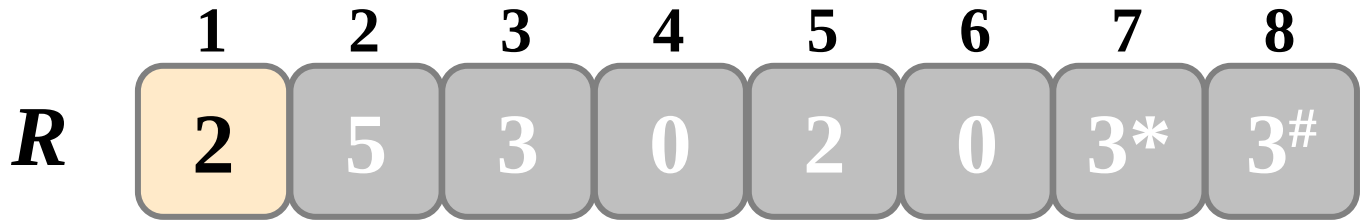


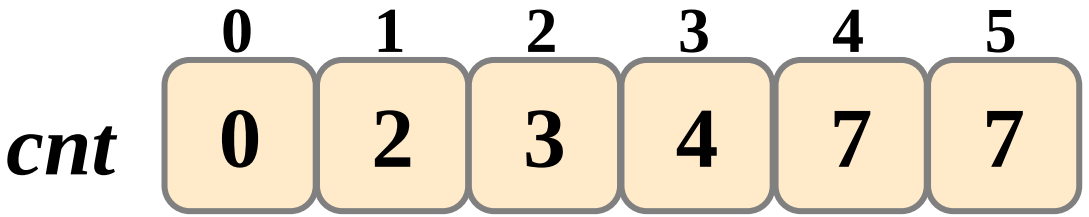

```
for(int i=n; i>=1; i--){
    int K = R[i];
    B[cnt[K]] = K;
    cnt[K]--;
}
```

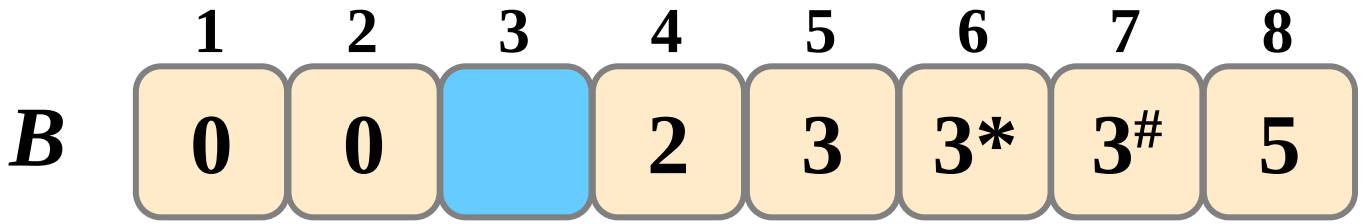
计数排序（Counting Sort）

B

③从右往左扫描R的每个元素K, 基于cnt[K]将K放入排序后的数组





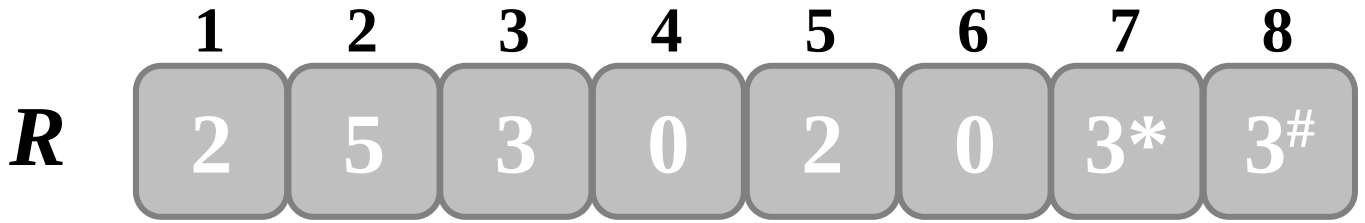


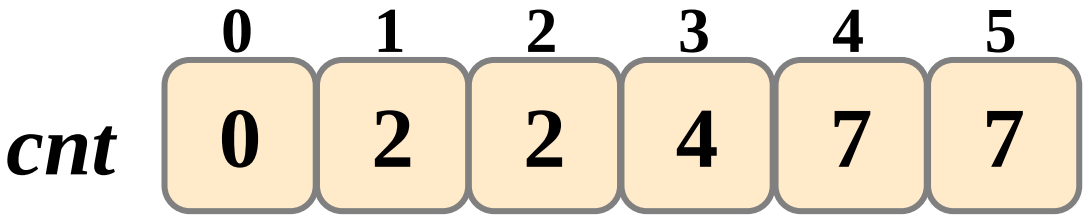

```
for(int i=n; i>=1; i--){
    int K = R[i];
    B[cnt[K]] = K;
    cnt[K]--;
}
```

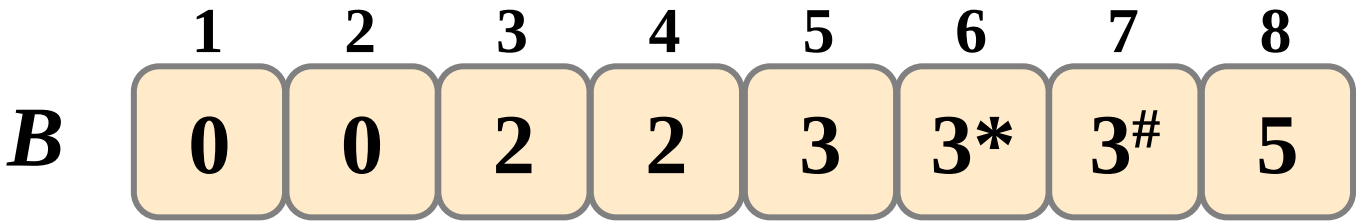
计数排序（Counting Sort）

B

③从右往左扫描R的每个元素K, 基于cnt[K]将K放入排序后的数组








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

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

计数排序（Counting Sort）

B

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$\leq i$的元素有多少个；

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。

```
const int maxm=1e5+10;
void CountingSort(int R[],int n, int m, int B[]) {
    int cnt[maxm]={0};
    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

B

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

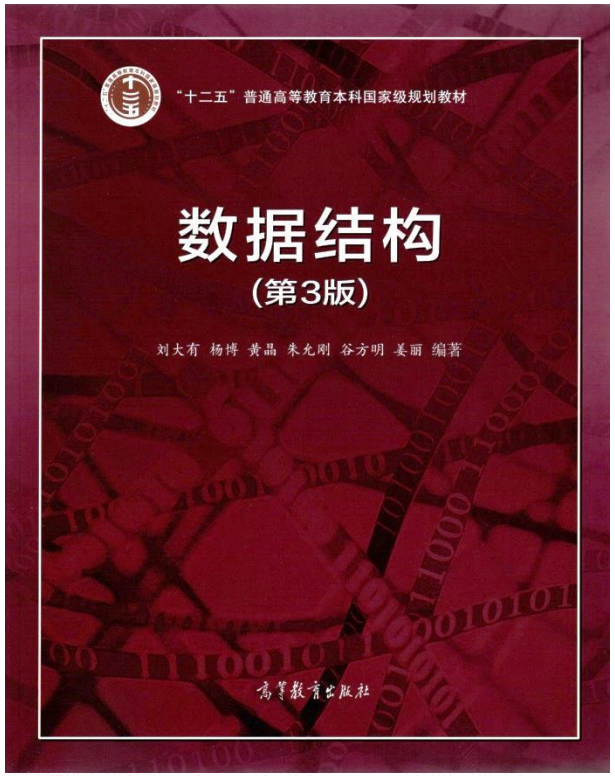
思考

A

- 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

zhuyungang@jlu.edu.cn

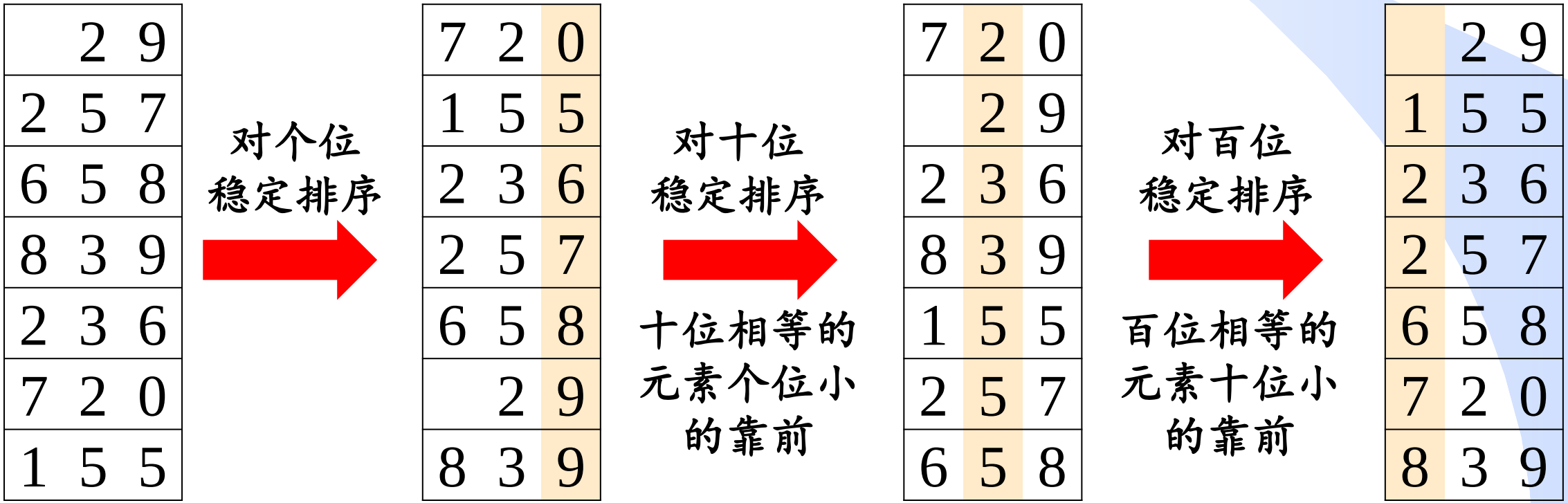
Last updated on 2023.12

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$K = K_d, K_{d-1}, \ldots, K_2, K_1$
  - ✓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



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

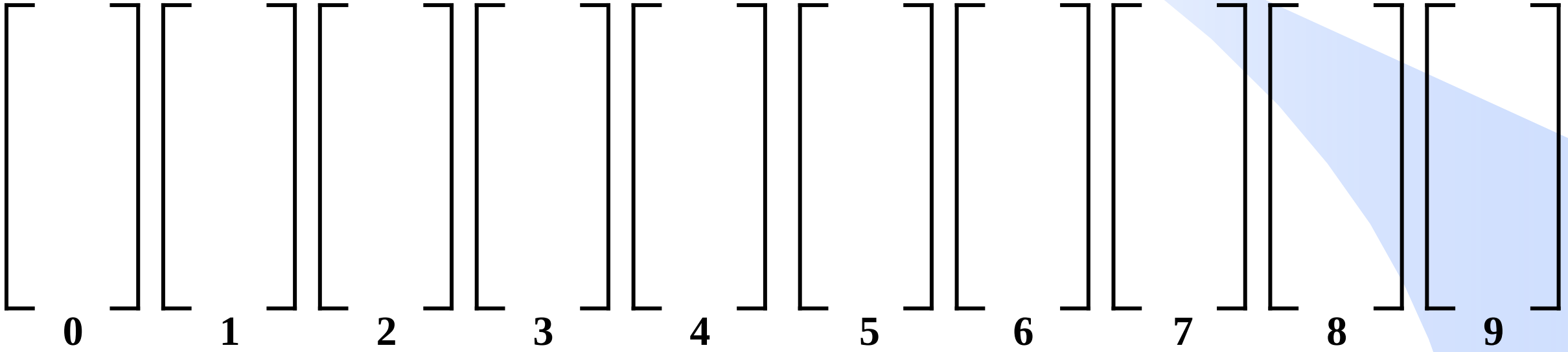
	2	9
2	5	7
6	5	8
8	3	9
2	3	6
7	2	0
1	5	5

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

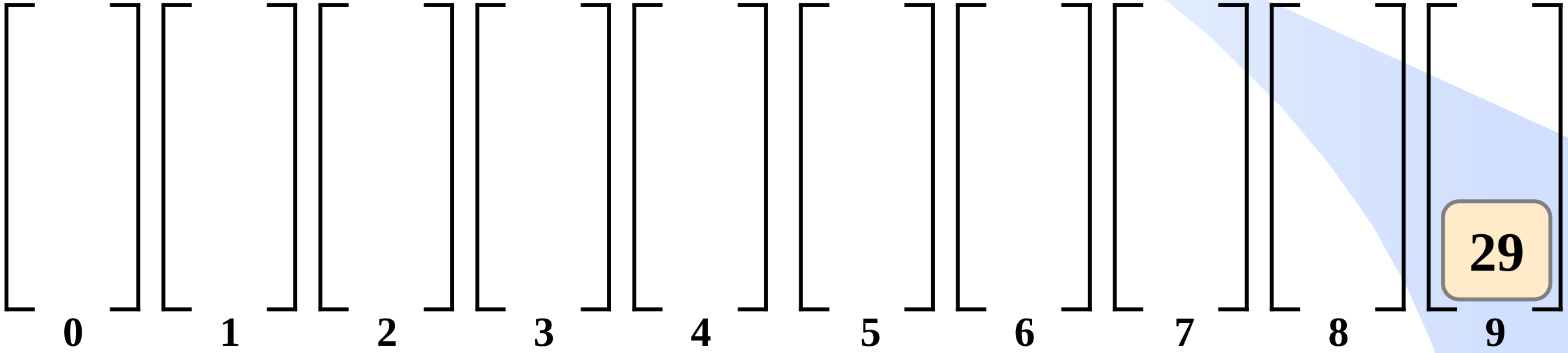


基于桶排序的基数排序示例：① 按个位排序

A

257 658 839 236 720 155

按个位分配

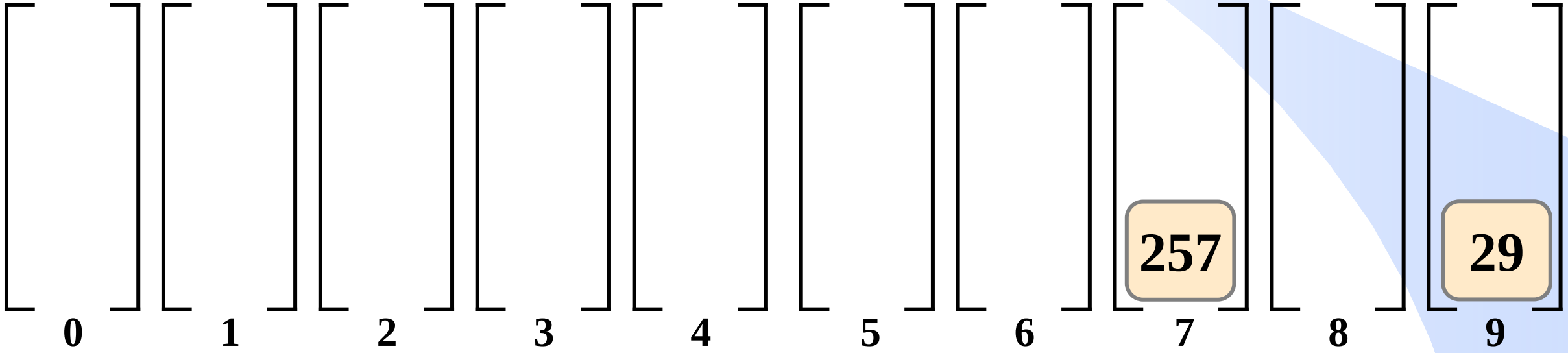


基于桶排序的基数排序示例：① 按个位排序

A

658 839 236 720 155

按个位分配

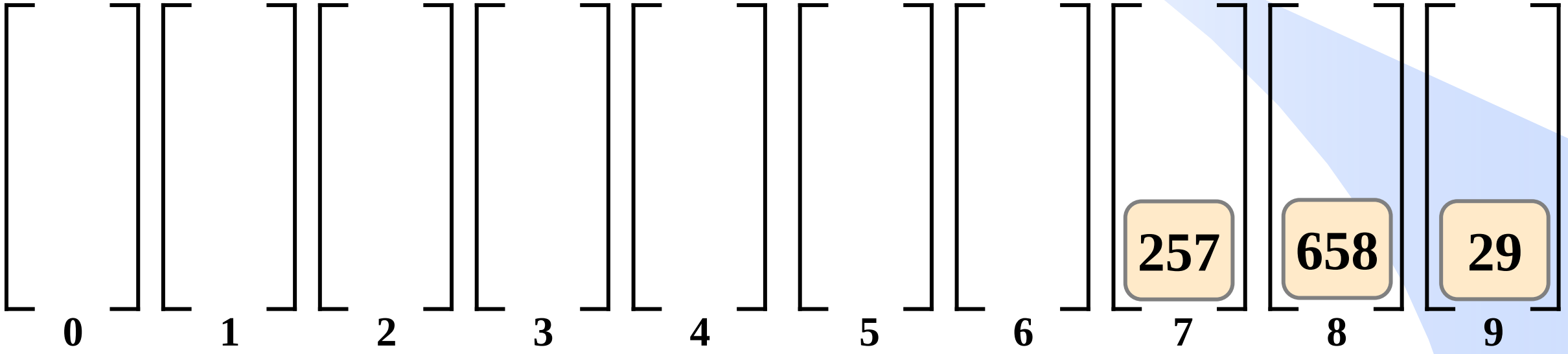


基于桶排序的基数排序示例：① 按个位排序

A

839 236 720 155

按个位分配

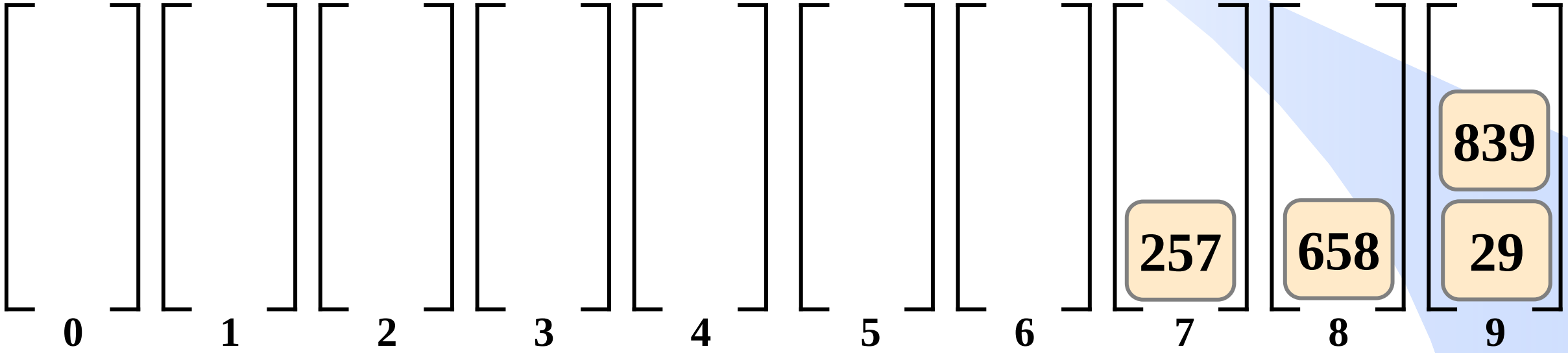


基于桶排序的基数排序示例：① 按个位排序

A

236 720 155

按个位分配

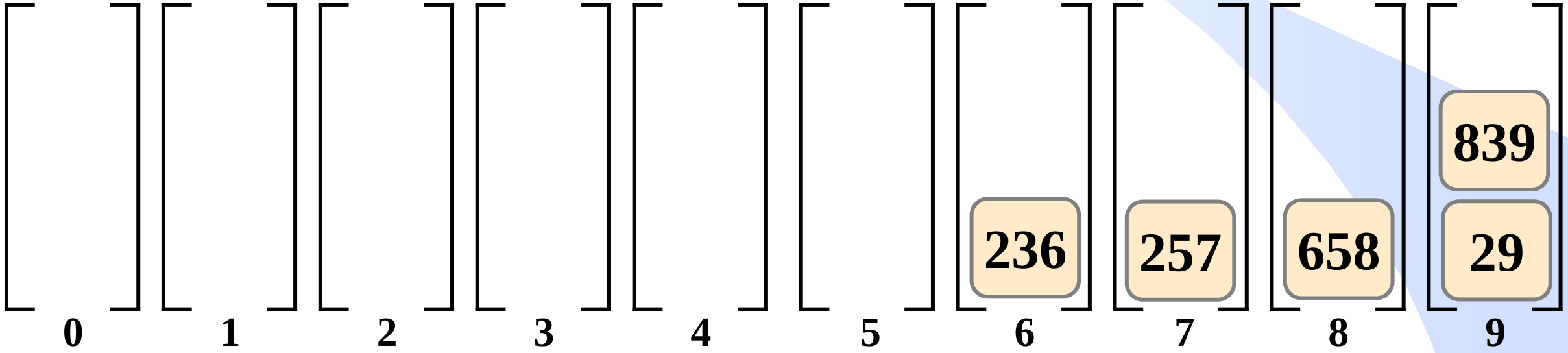


基于桶排序的基数排序示例：① 按个位排序

A

720 155

按个位分配

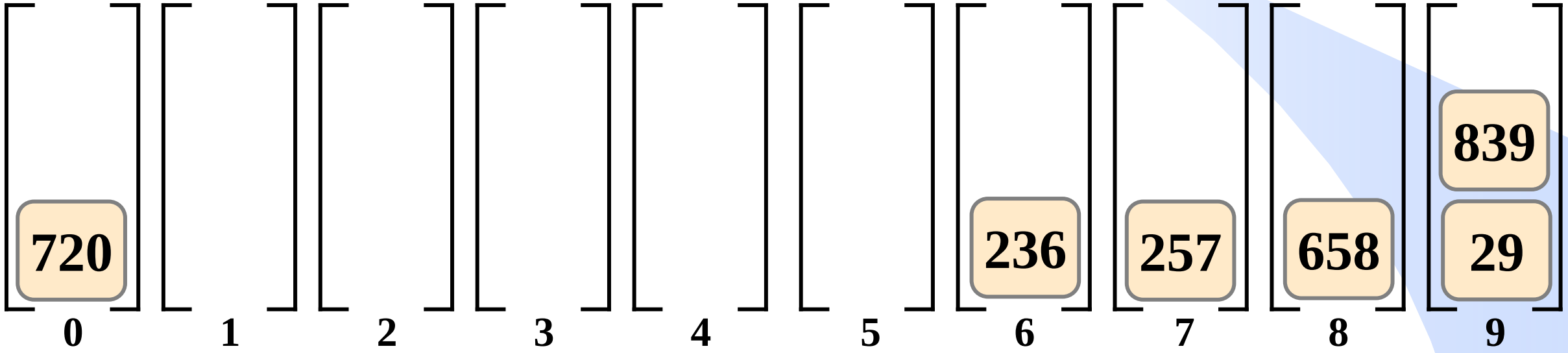


基于桶排序的基数排序示例：① 按个位排序

A

155

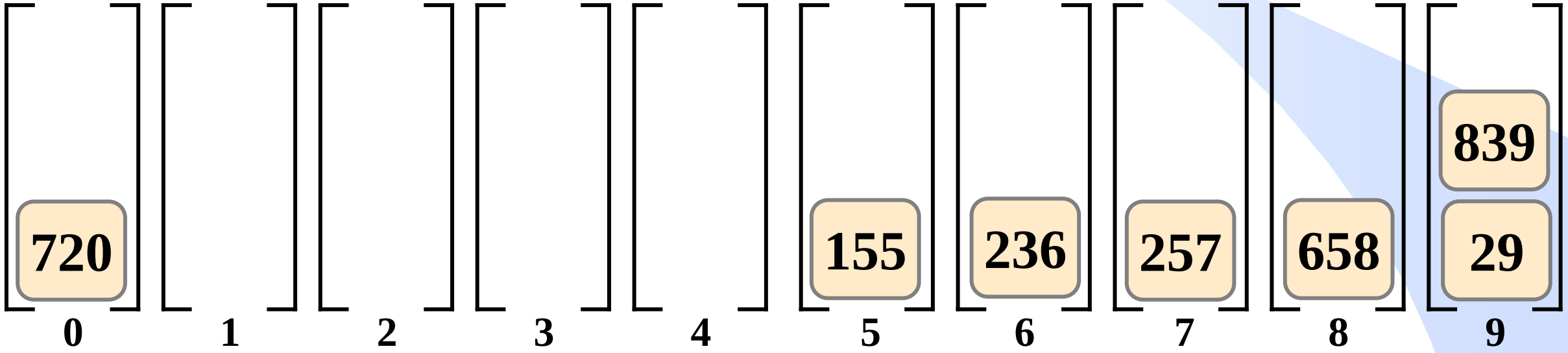
按个位分配



基于桶排序的基数排序示例：① 按个位排序

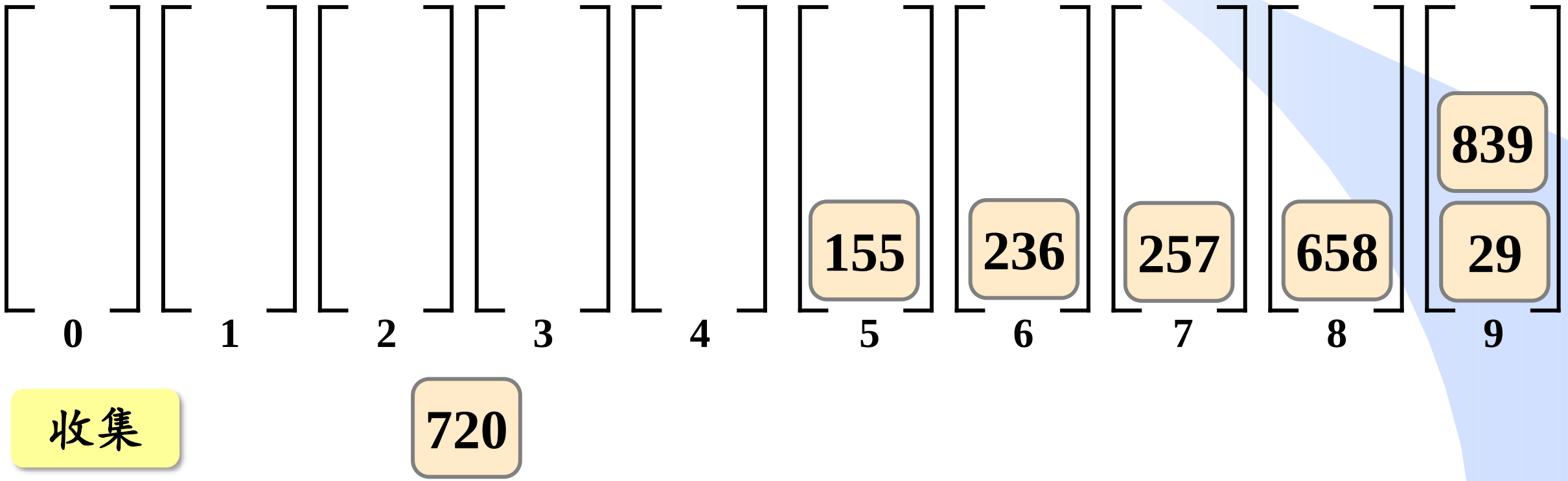
A

按个位分配



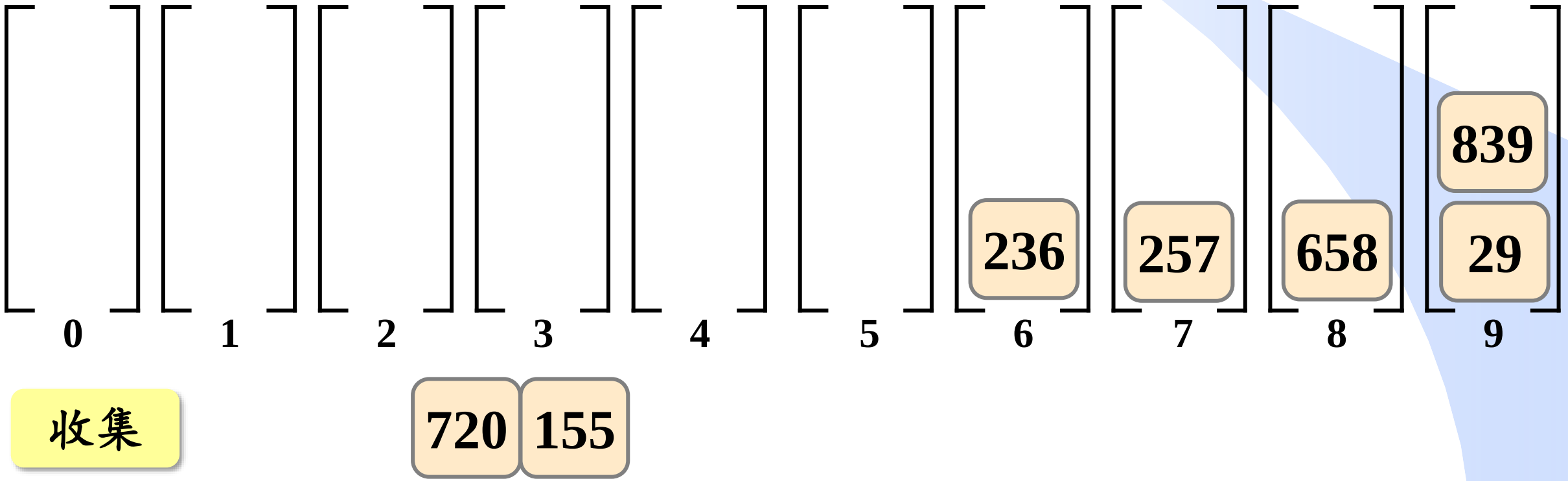
基于桶排序的基数排序示例：① 按个位排序

A



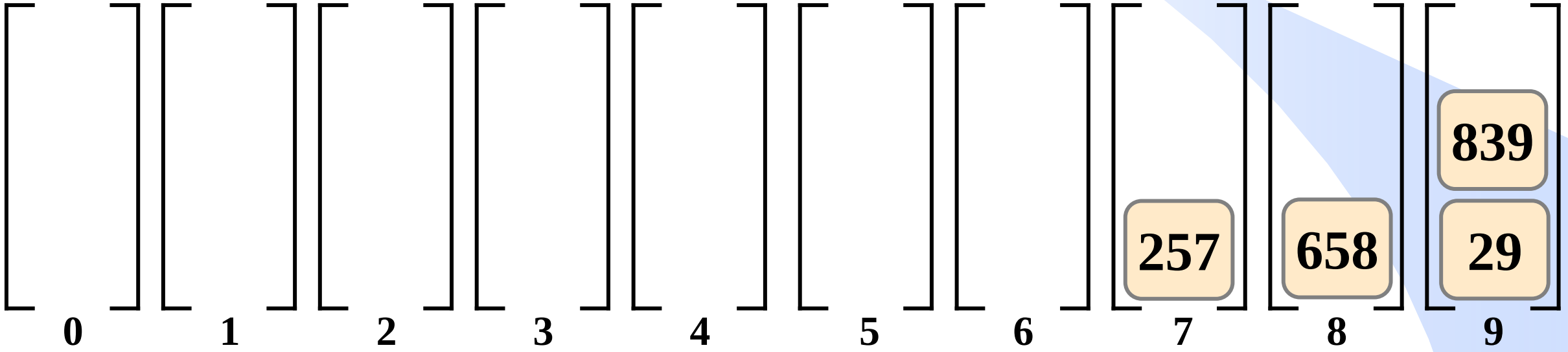
基于桶排序的基数排序示例：① 按个位排序

A



基于桶排序的基数排序示例：① 按个位排序

A

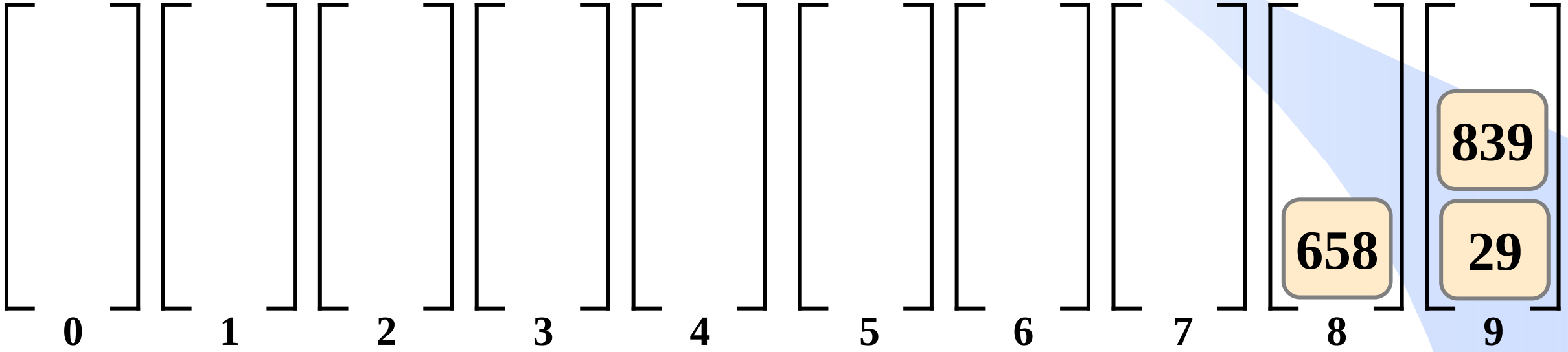


收集

720 155 236

基于桶排序的基数排序示例：① 按个位排序

A

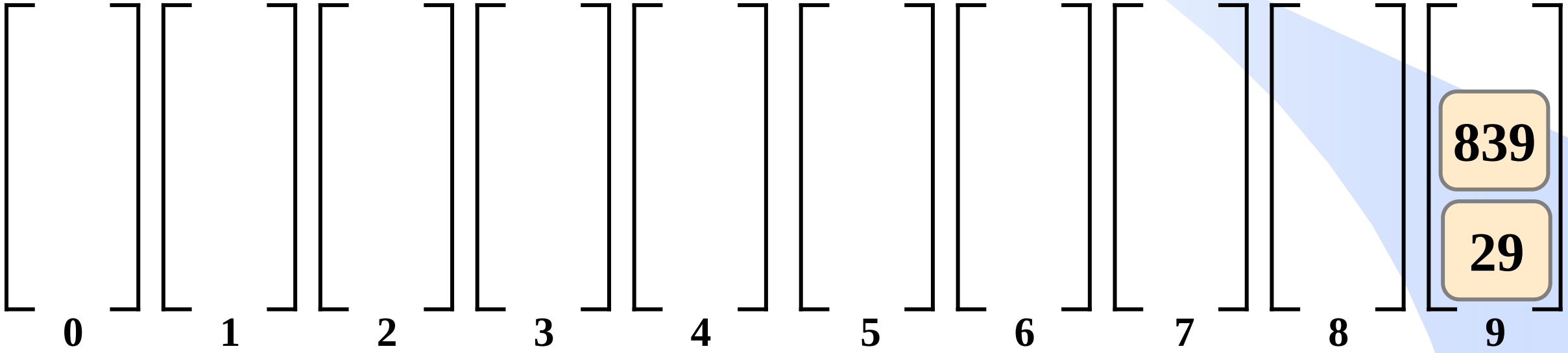


收集

720 155 236 257

基于桶排序的基数排序示例：① 按个位排序

A

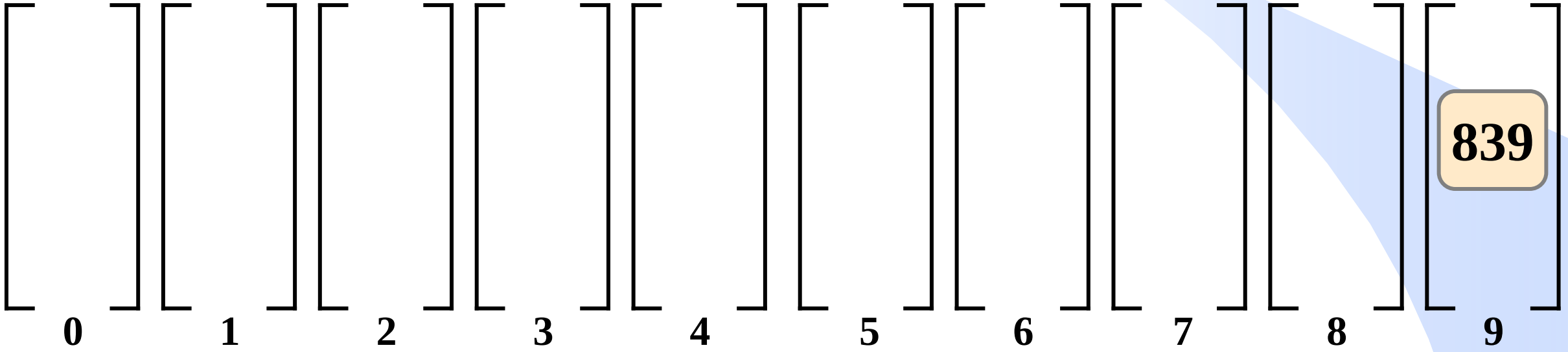


收集

720 155 236 257 658

基于桶排序的基数排序示例：① 按个位排序

A

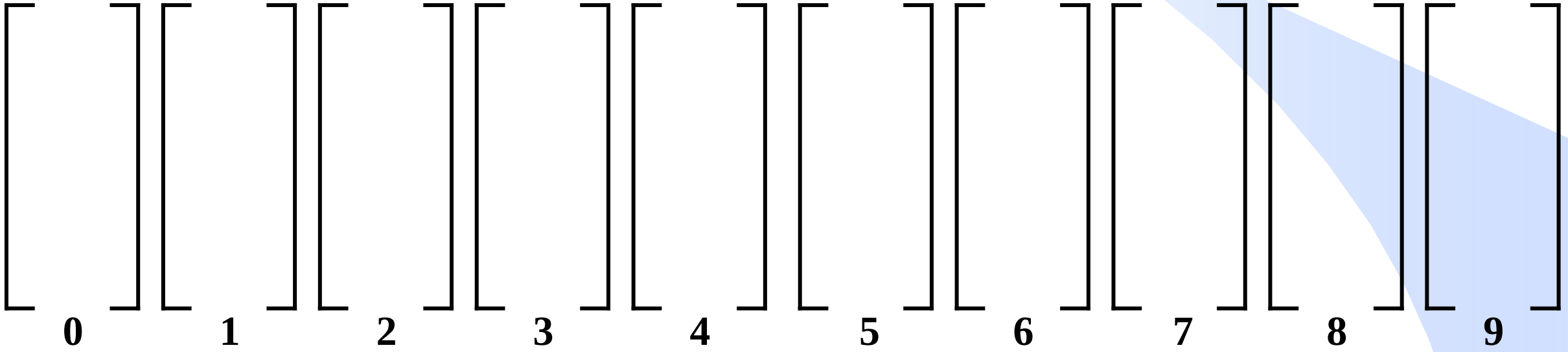


收集

720 155 236 257 658 29

基于桶排序的基数排序示例：① 按个位排序

A



收集

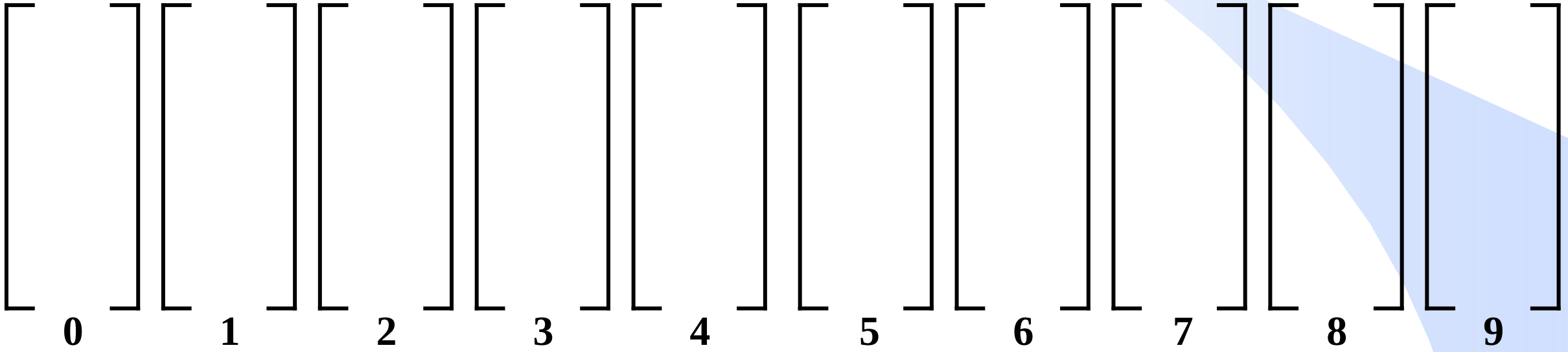
720 155 236 257 658 29 839

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

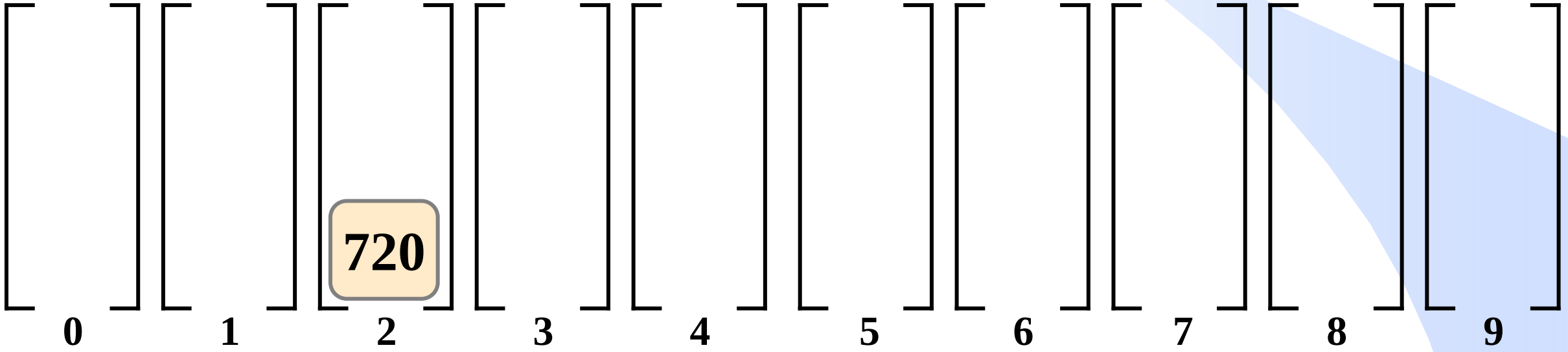


基于桶排序的基数排序示例：② 按十位排序

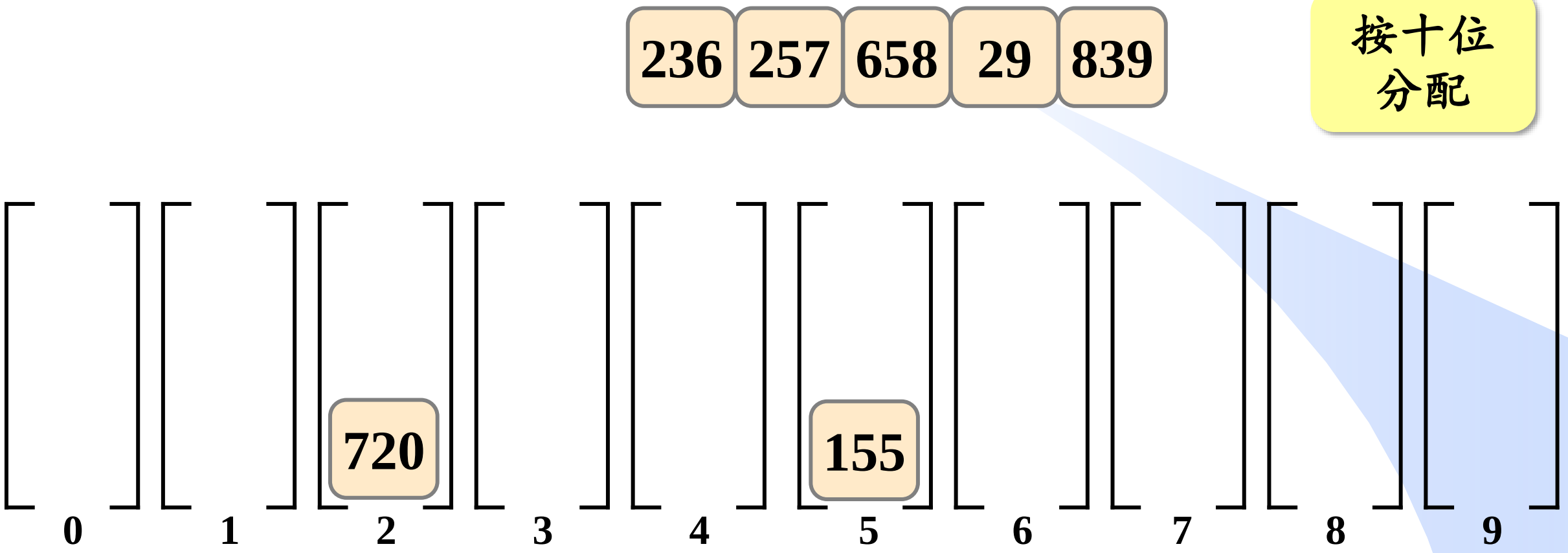
A

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

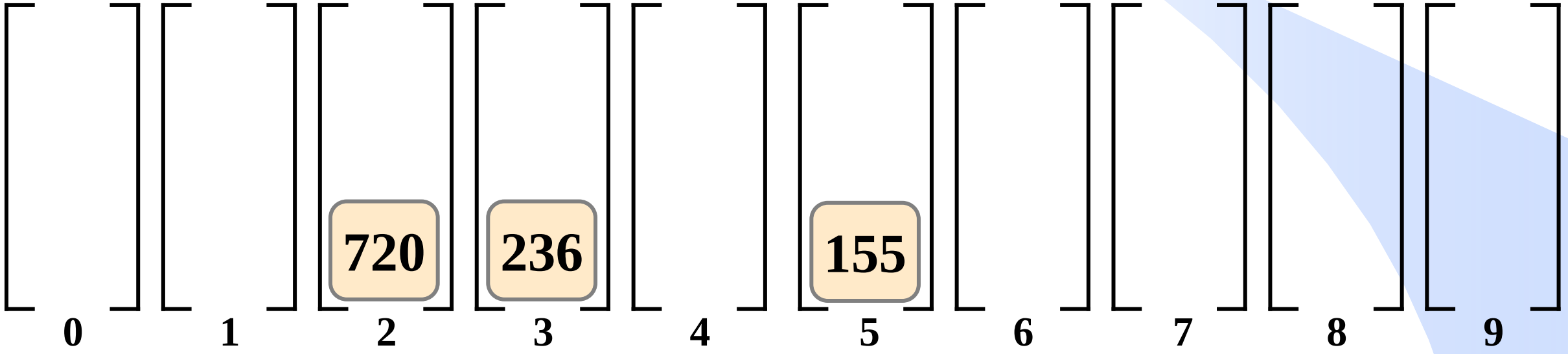


基于桶排序的基数排序示例：② 按十位排序

A

257 658 29 839

按十位分配

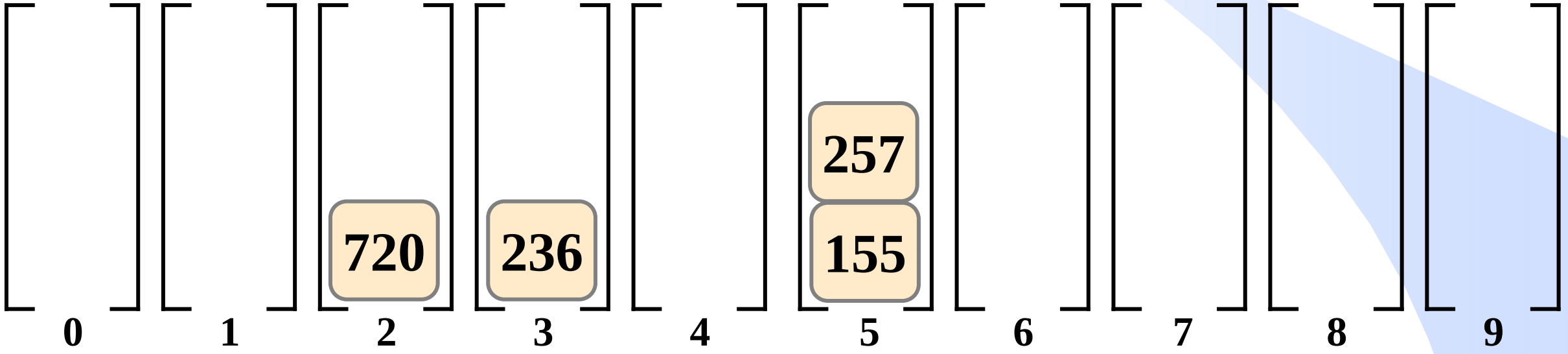


基于桶排序的基数排序示例：② 按十位排序

A

658 29 839

按十位分配

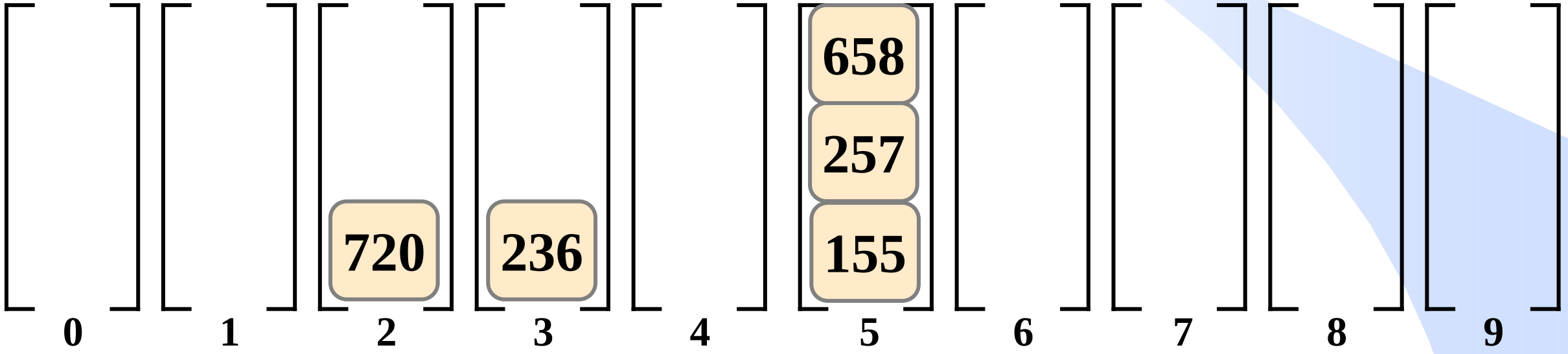


基于桶排序的基数排序示例：② 按十位排序

A

29 839

按十位分配

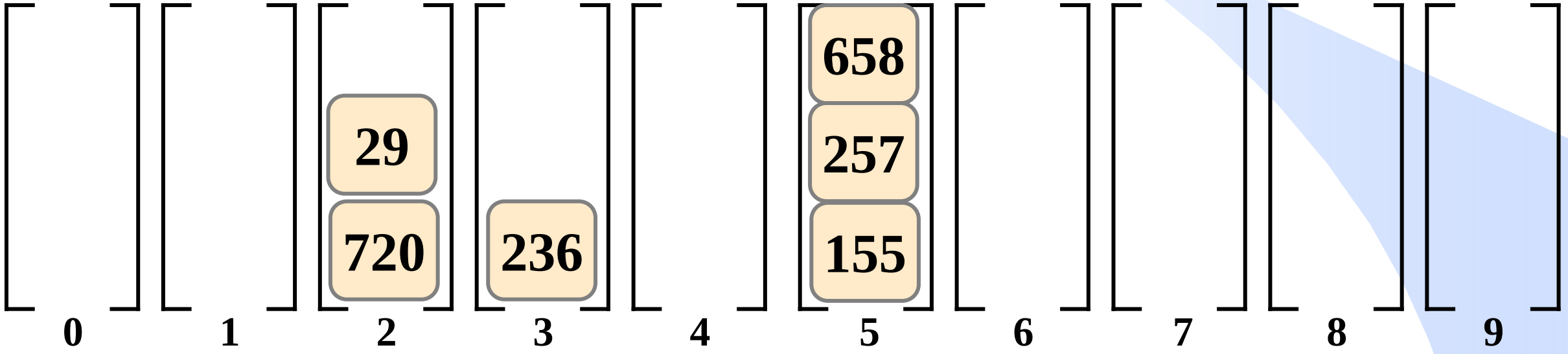


基于桶排序的基数排序示例：② 按十位排序

A

839

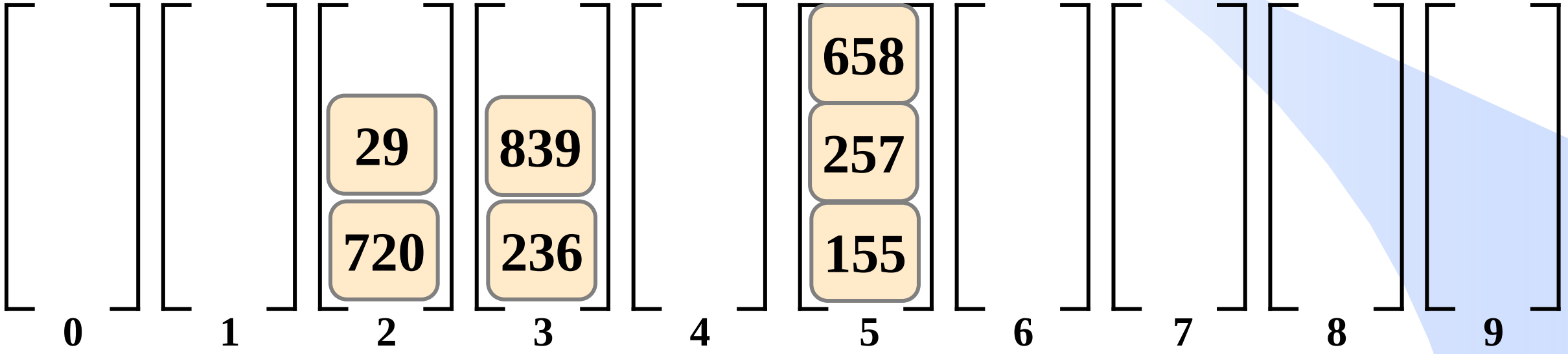
按十位分配



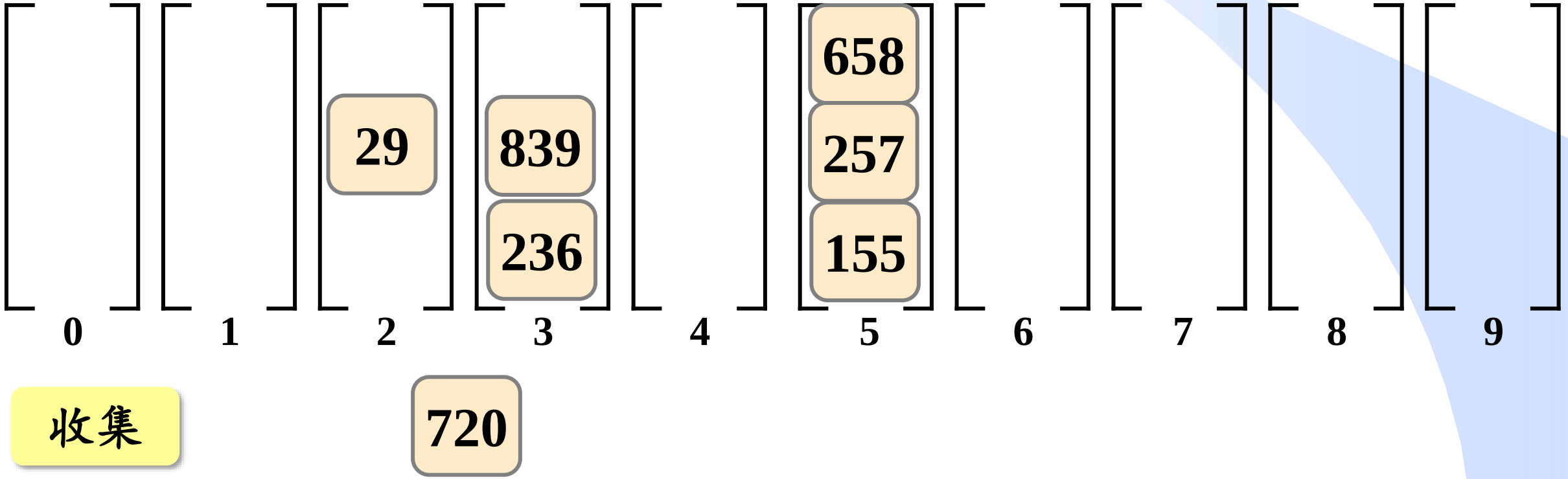
基于桶排序的基数排序示例：② 按十位排序

A

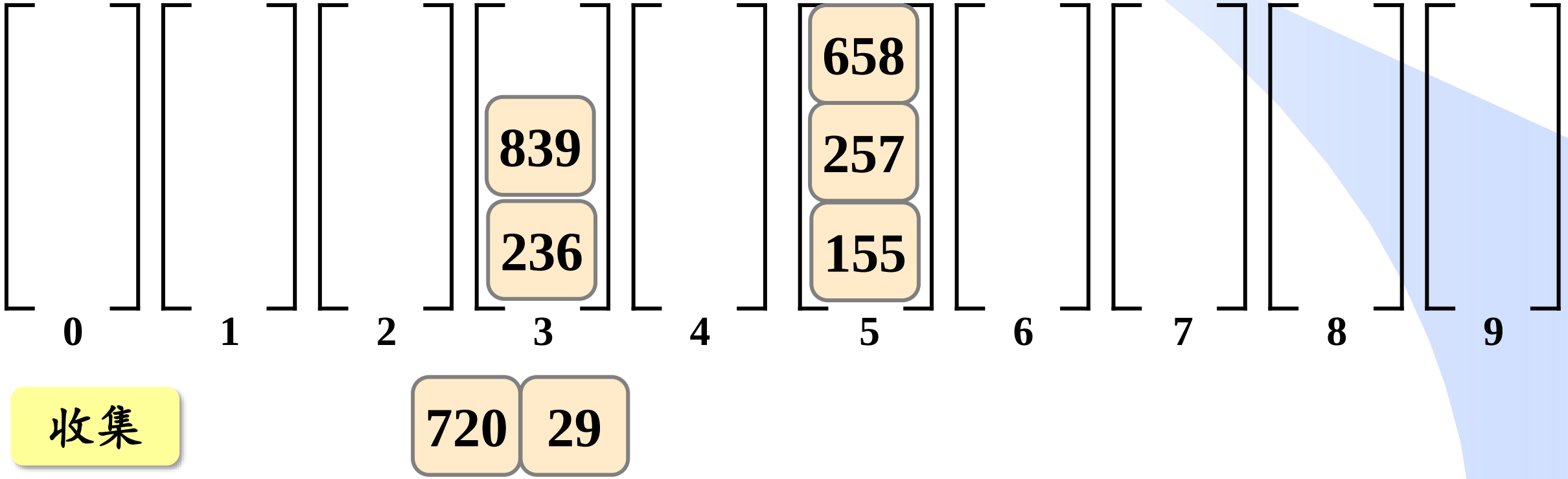
按十位
分配



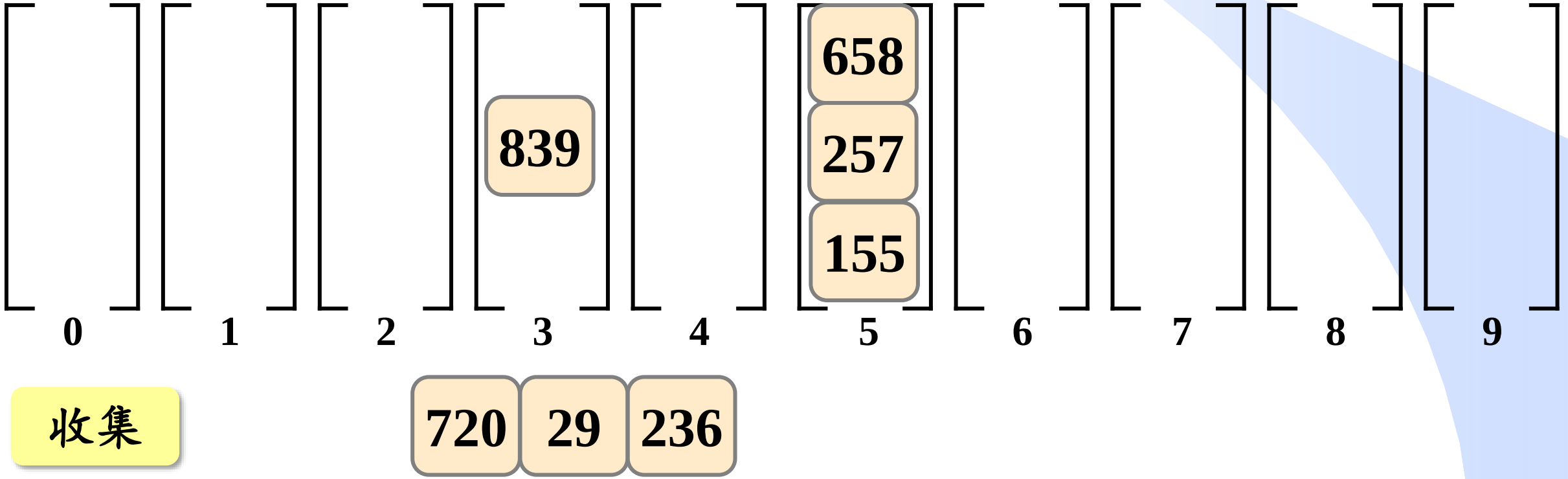
基于桶排序的基数排序示例：② 按十位排序



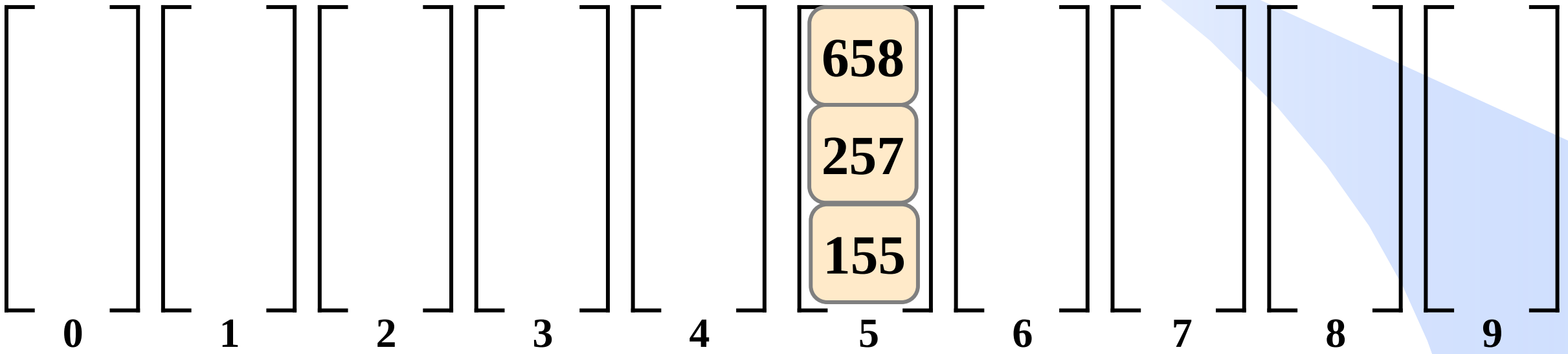
基于桶排序的基数排序示例：② 按十位排序



基于桶排序的基数排序示例：② 按十位排序



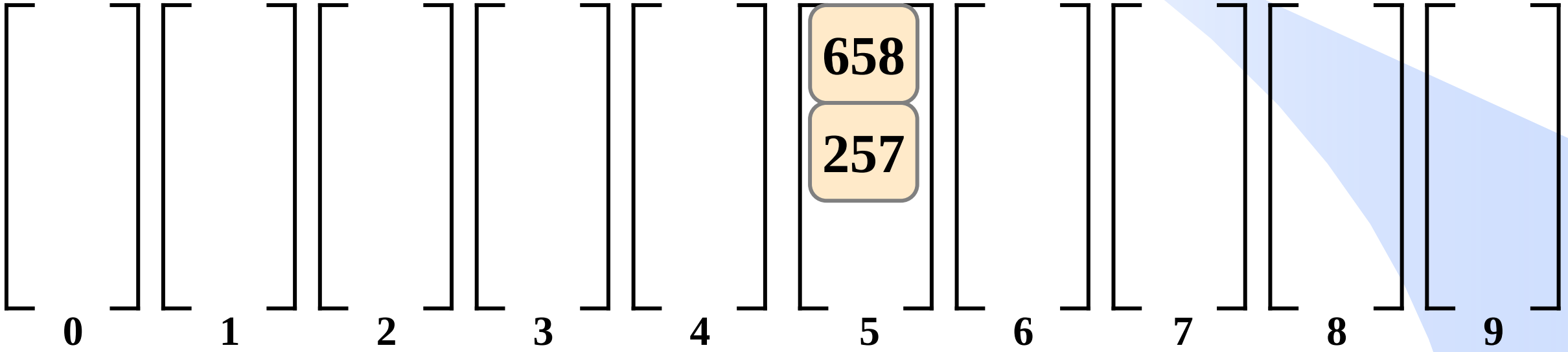
基于桶排序的基数排序示例：② 按十位排序



收集

720 29 236 839

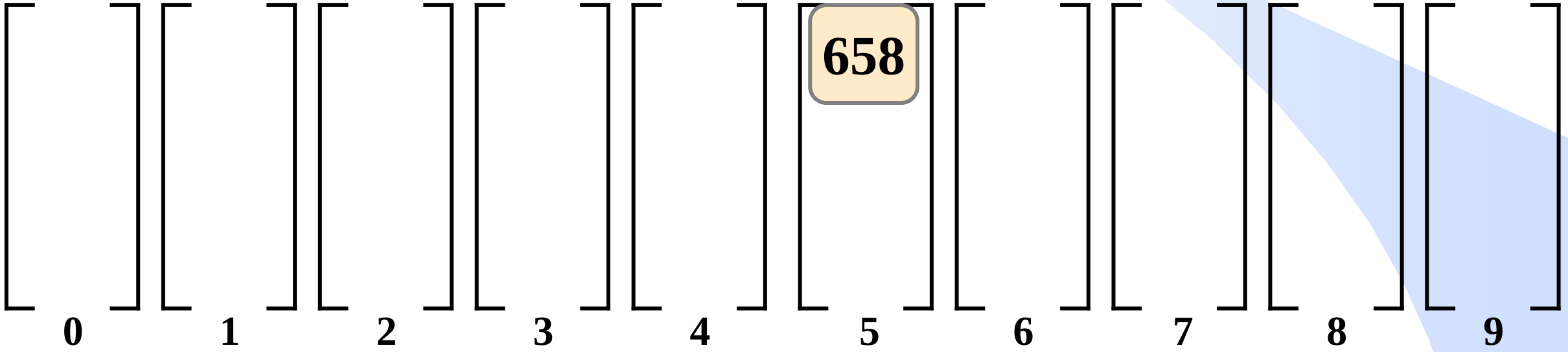
基于桶排序的基数排序示例：② 按十位排序



收集

720 29 236 839 155

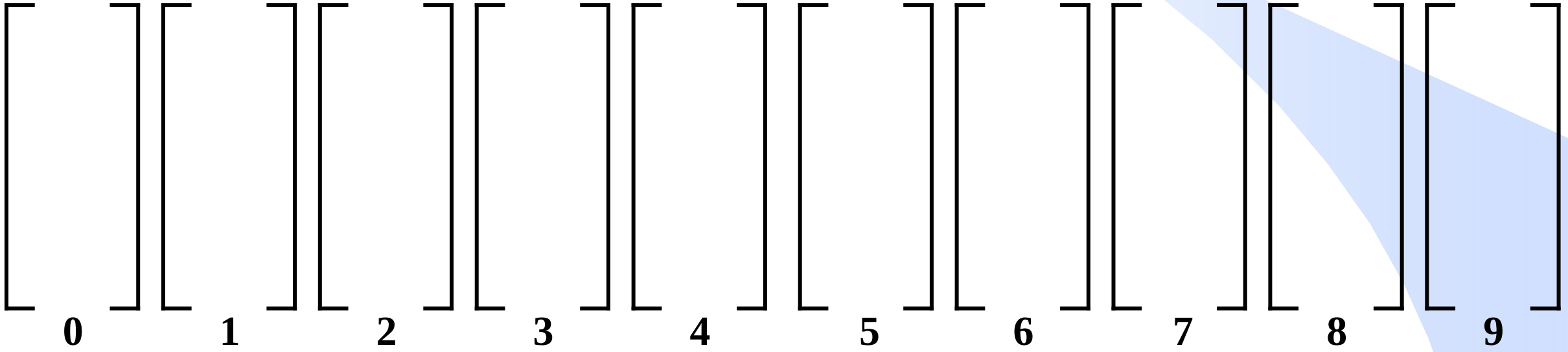
基于桶排序的基数排序示例：② 按十位排序



收集

720 29 236 839 155 257

基于桶排序的基数排序示例：② 按十位排序



收集

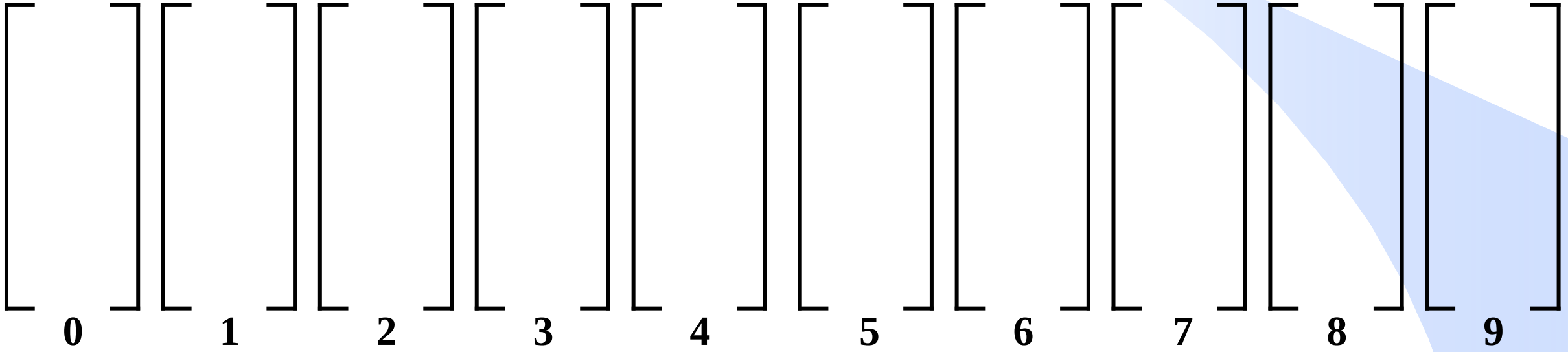
720 29 236 839 155 257 658

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

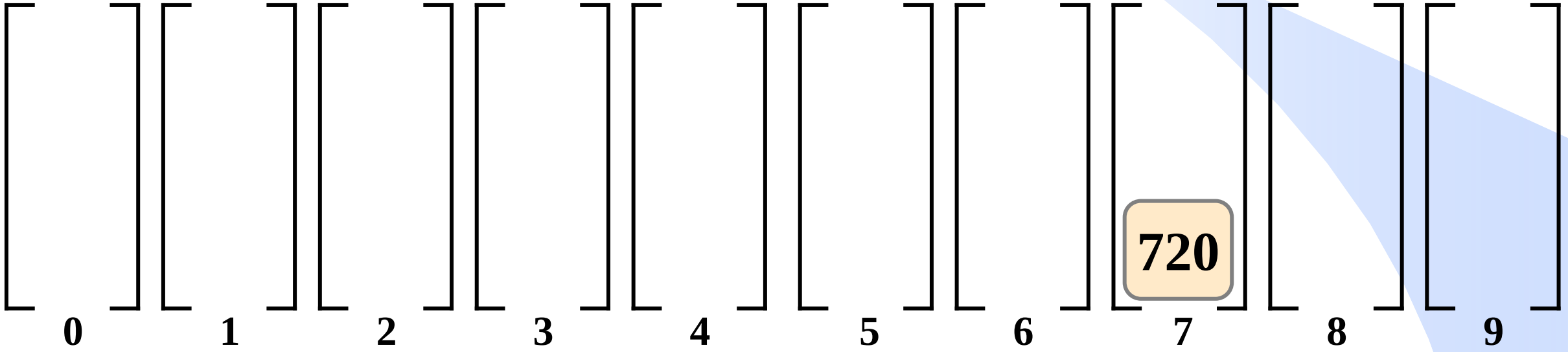


基于桶排序的基数排序示例：③ 按百位排序

A

29 236 839 155 257 658

按百位分配

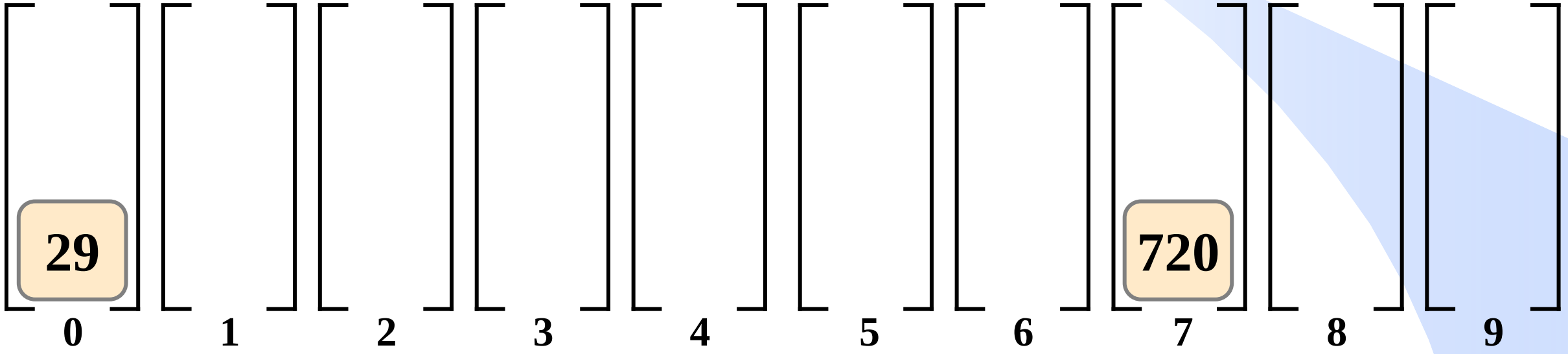


基于桶排序的基数排序示例：③ 按百位排序

A

236 839 155 257 658

按百位分配

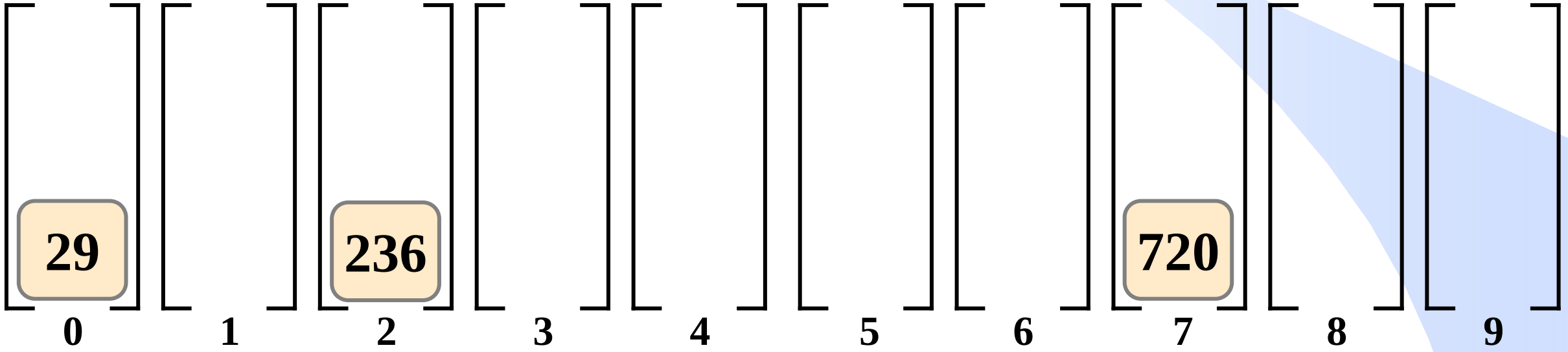


基于桶排序的基数排序示例：③ 按百位排序

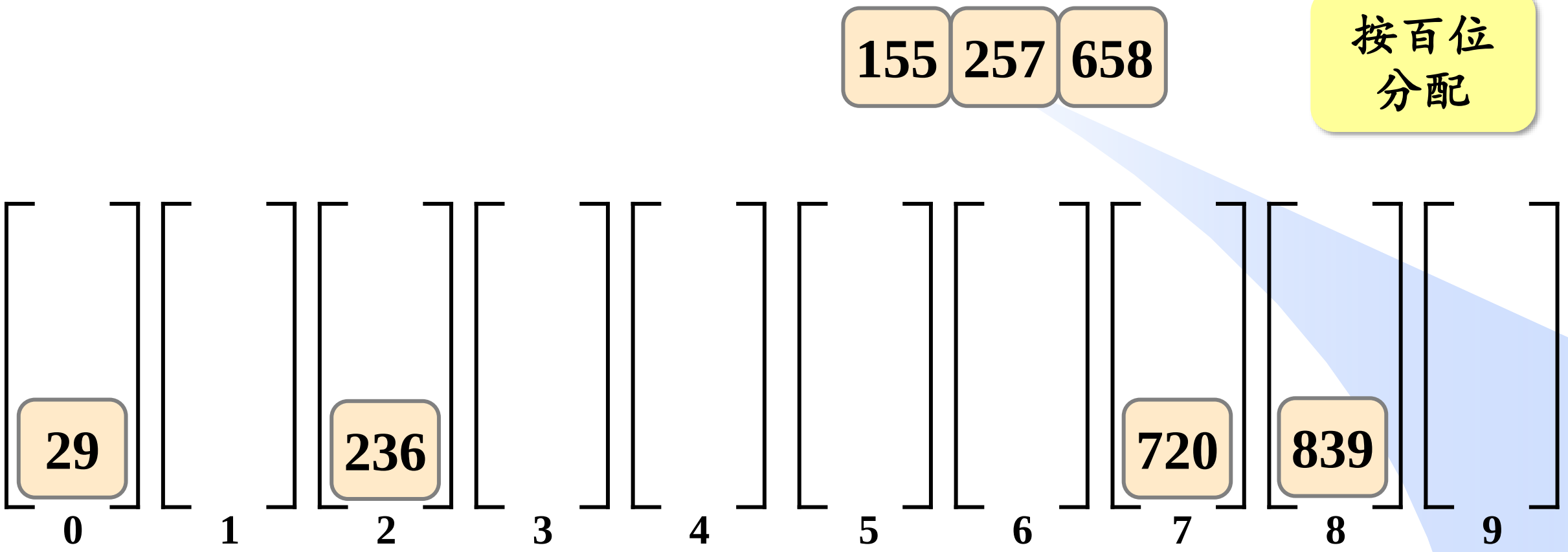
A

839 155 257 658

按百位分配



基于桶排序的基数排序示例：③ 按百位排序

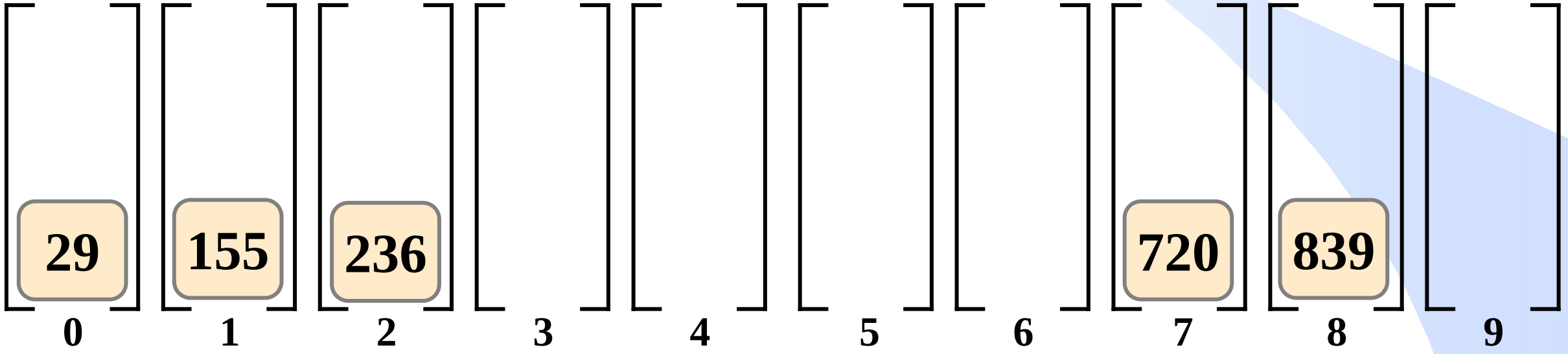


基于桶排序的基数排序示例：③ 按百位排序

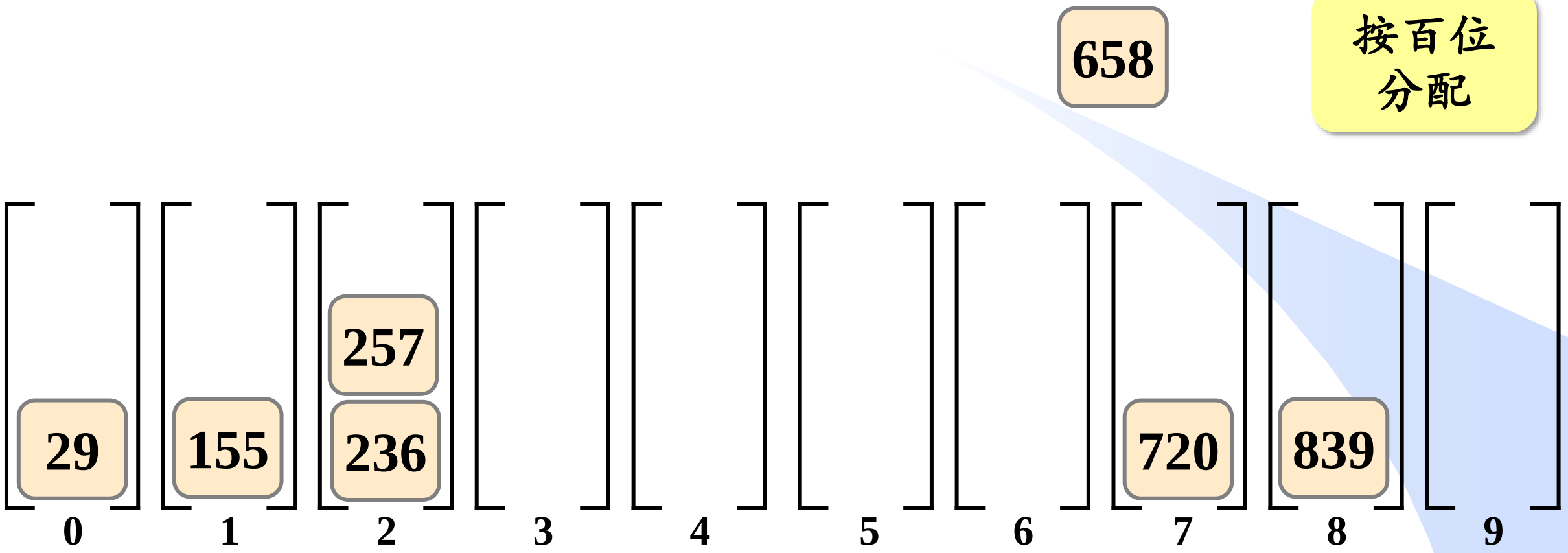
A

257 658

按百位分配



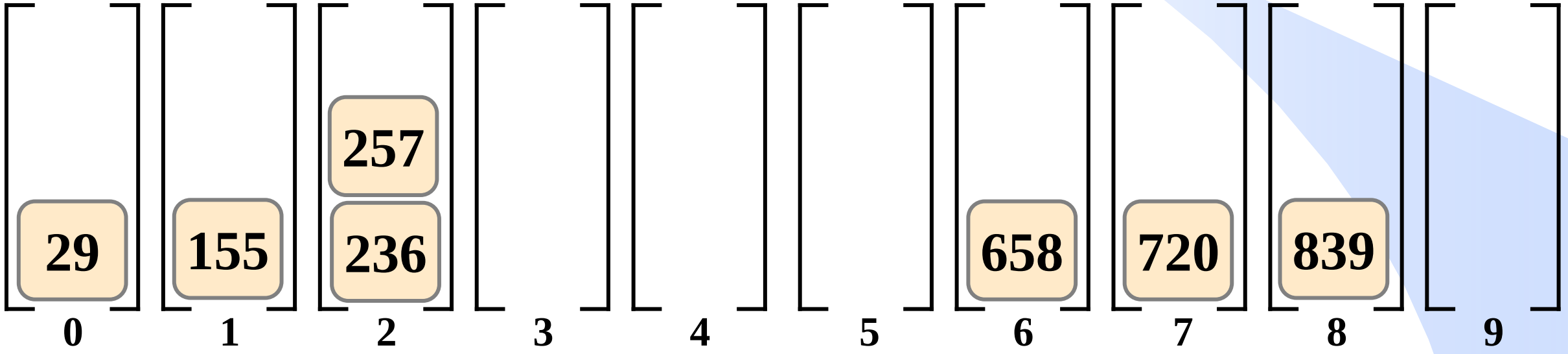
基于桶排序的基数排序示例：③ 按百位排序



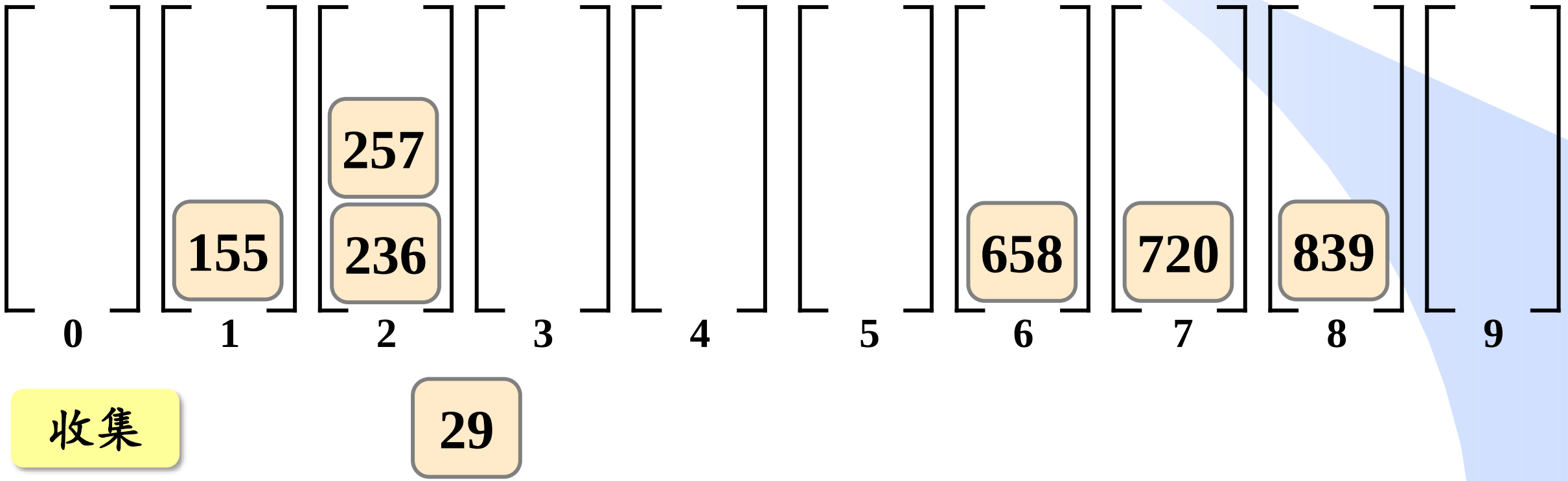
基于桶排序的基数排序示例：③ 按百位排序

A

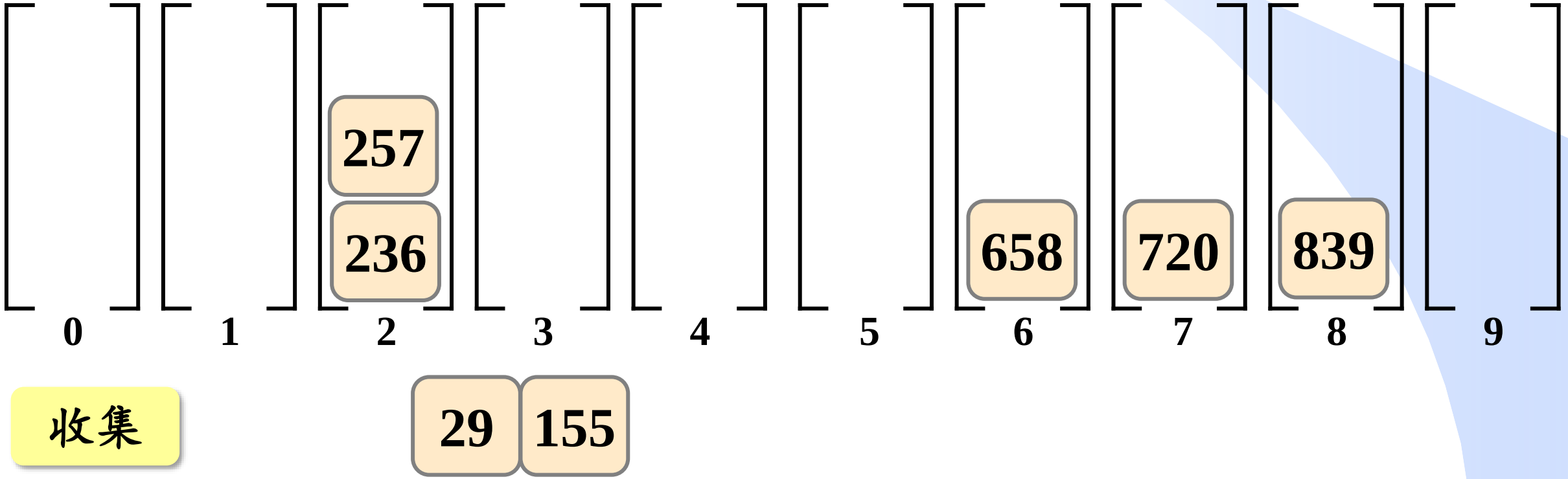
按百位
分配



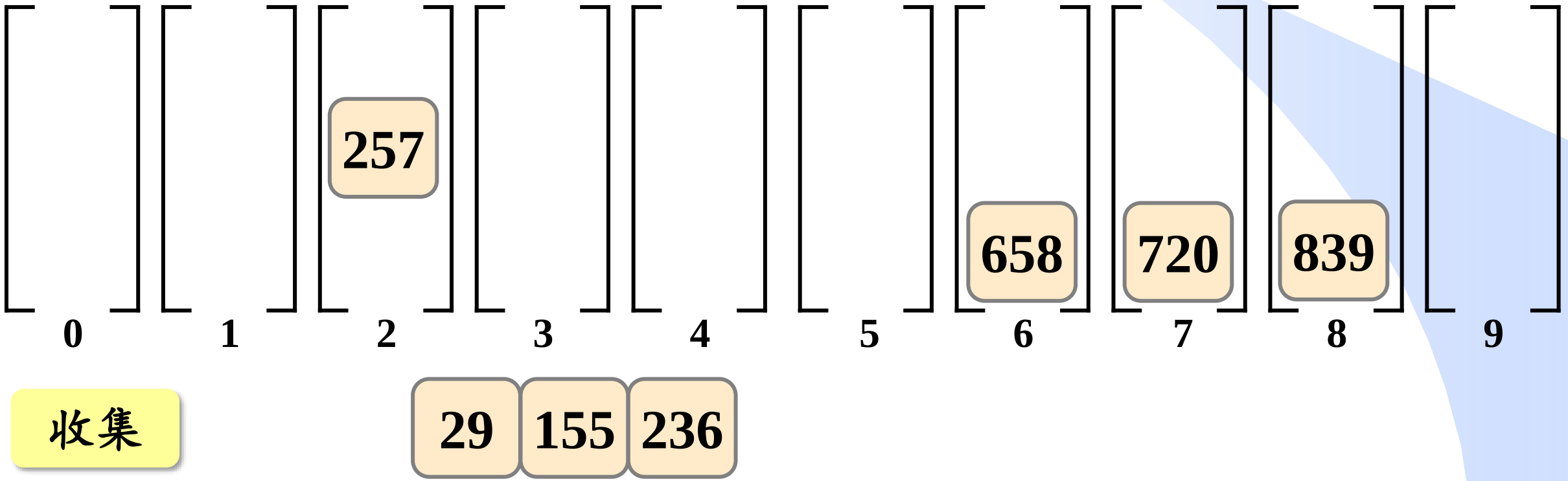
基于桶排序的基数排序示例：③ 按百位排序



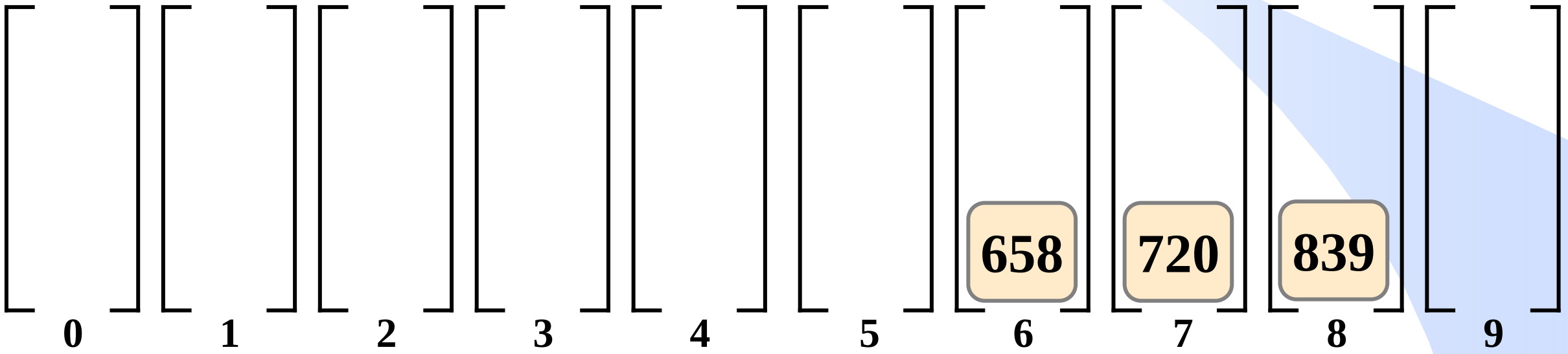
基于桶排序的基数排序示例：③ 按百位排序



基于桶排序的基数排序示例：③ 按百位排序



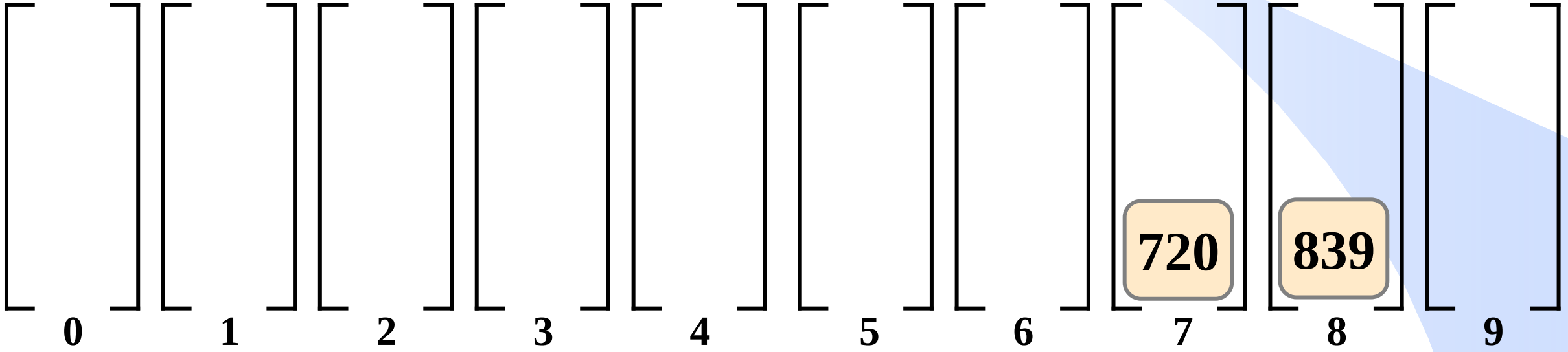
基于桶排序的基数排序示例：③ 按百位排序



收集

29 155 236 257

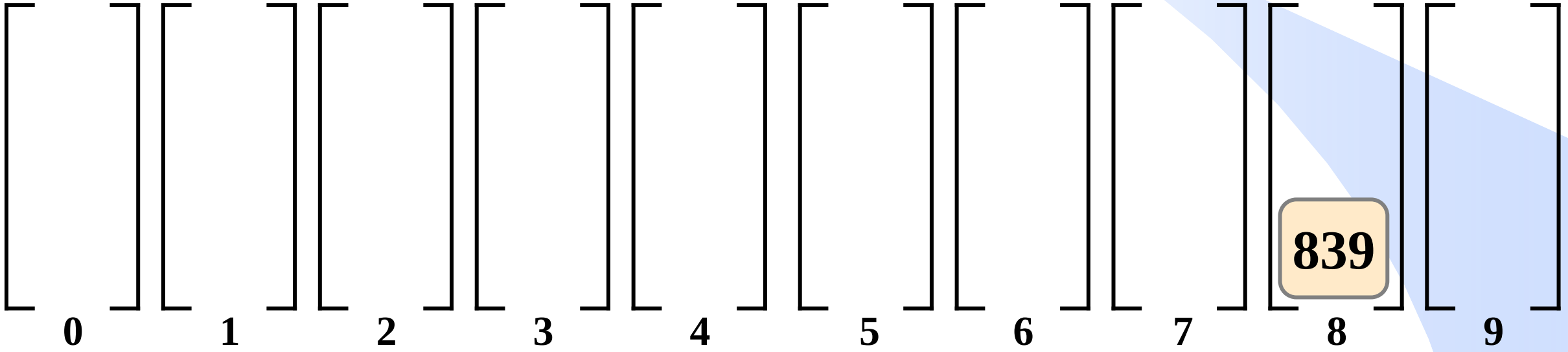
基于桶排序的基数排序示例：③ 按百位排序



收集

29 155 236 257 658

基于桶排序的基数排序示例：③ 按百位排序



收集

29 155 236 257 658 720

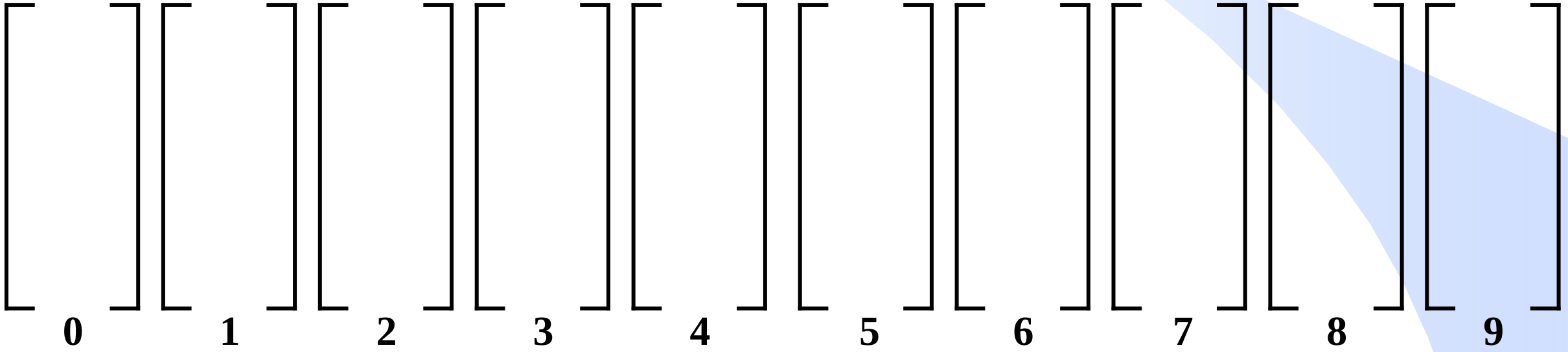
基于桶排序的基数排序示例：③ 按百位排序

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

B

```
const int r=10;
void RadixSort(int R[], int n, int d){//每个元素含d位
    int B[N], base=1;
    for(int k=1; k<=d; k++) { //对第k位排序
        int cnt[r]={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];
        base *= r;
    }
}
```

取$R[i]$的第k位

对第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

A

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

A

- 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

A

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