Sorting Algorithms

Counting sort

- Counting sort assumes that each of the n input elements is an integer in the range 0 to k. that is n is the number of elements and k is the highest value element.
- Consider the input set : 4, 1, 3, 4, 3. Then n=5 and k=4
- Counting sort determines for each input element x, the number of elements less than x. And it uses this information to place element x directly into its position in the output array. For example if there exits 17 elements less that x then x is placed into the 18^{th} position into the output array.
- The algorithm uses three array:

  Input Array: **A[1..*n*]** store input data where A[j] $\in$ {1, 2, 3, …, *k*}

  Output Array: **B[1..*n*]** finally store the sorted data

  Temporary Array: **C[1..*k*]** store data temporarily

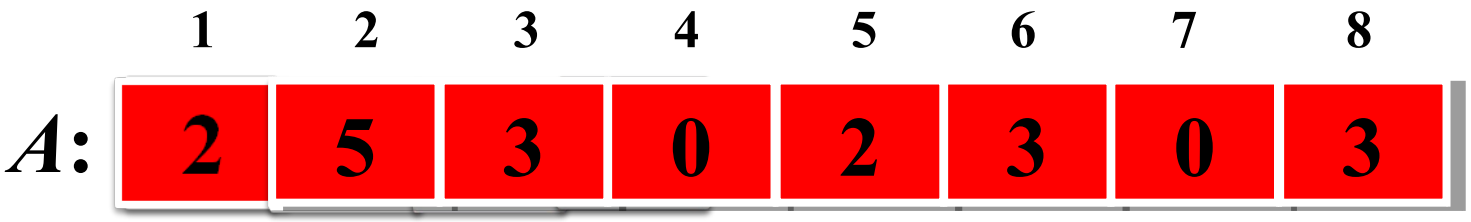
Counting Sort

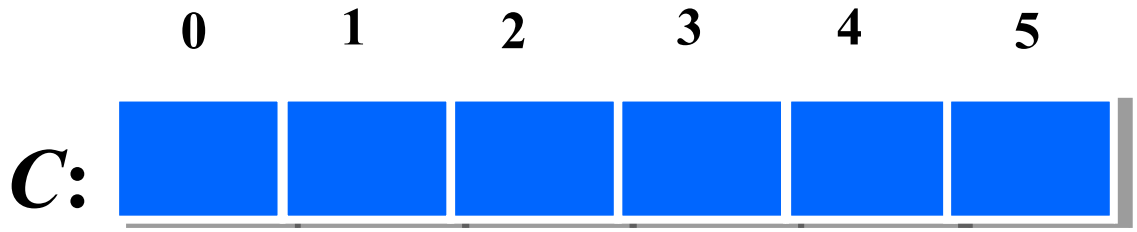
```
1.  Counting-Sort(A, B, k)
2.  Let C[0…..k] be a new array
3.  for i=0 to k
4.        C[i]= 0;
5.  for j=1 to A.length  or n
6.        C[ A[j] ] = C[ A[j] ]  + 1;
7.  for i=1 to k
8.        C[i] = C[i] + C[i-1];
9.  for j=n or A.length down to 1
10.       B[  C[ A[j] ]  ] = A[j];
11.       C[ A[j] ] = C[ A[j] ]  - 1;
```

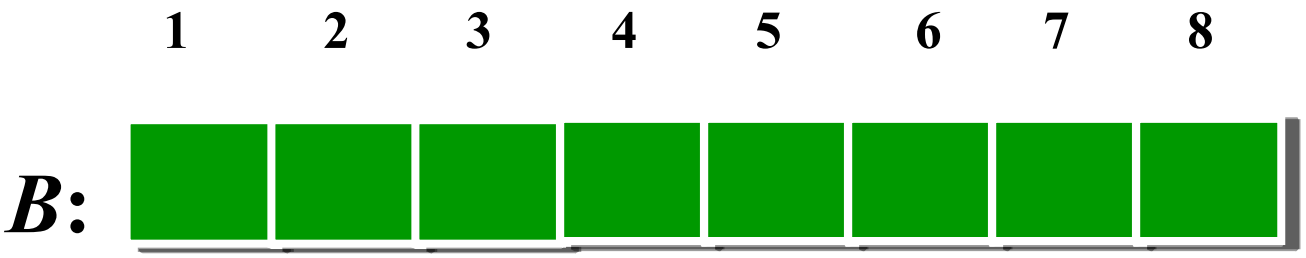
Counting Sort

```
1.  Counting-Sort(A, B, k)
2.  Let C[0…..k] be a new array
3.  for i=0 to k                                  [Loop 1]
4.        C[i]= 0;
5.  for j=1 to A.length( or n)                    [Loop 2]
6.        C[ A[j] ] = C[ A[j] ]  + 1;
7.  for i=1 to k                                  [Loop 3]
8.        C[i] = C[i] + C[i-1];
9.  for j=n or A.length down to 1                 [Loop 4]
10.       B[  C[ A[j] ]  ] = A[j];
11.       C[ A[j] ] = C[ A[j] ]  - 1;
```

Counting-sort example

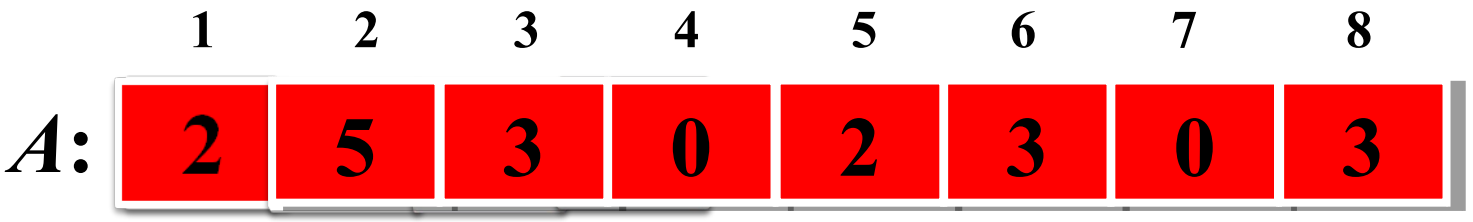


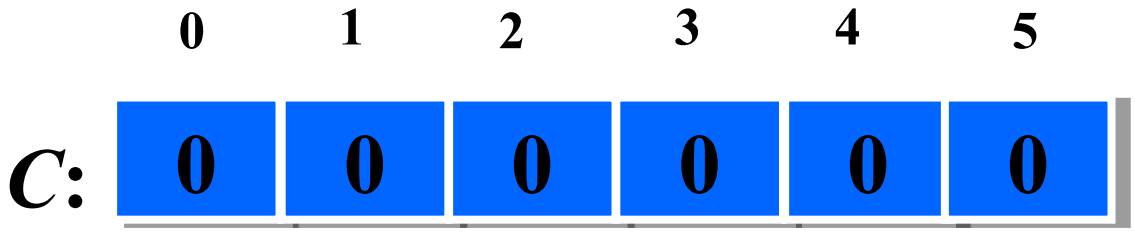


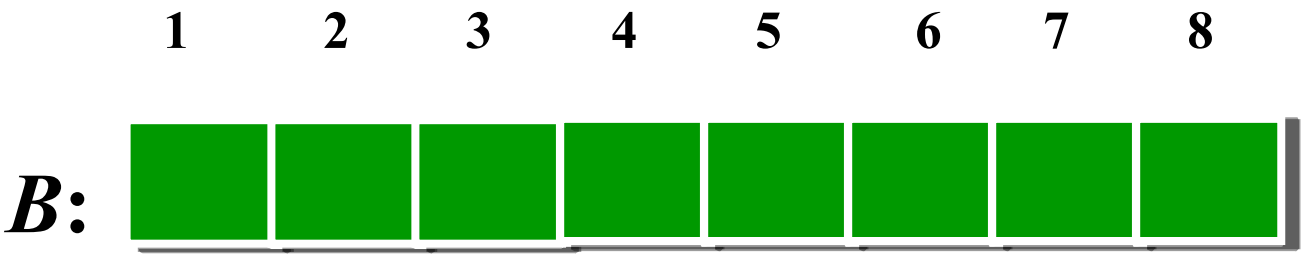


Executing Loop 1

```
3.  for i=0 to k
4.      C[i]= 0;
```

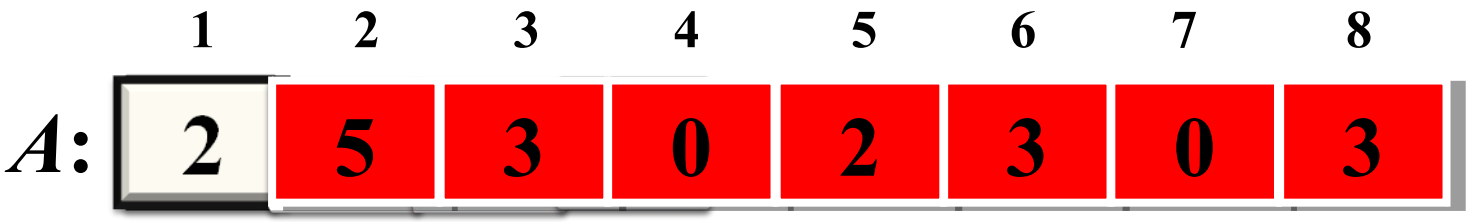


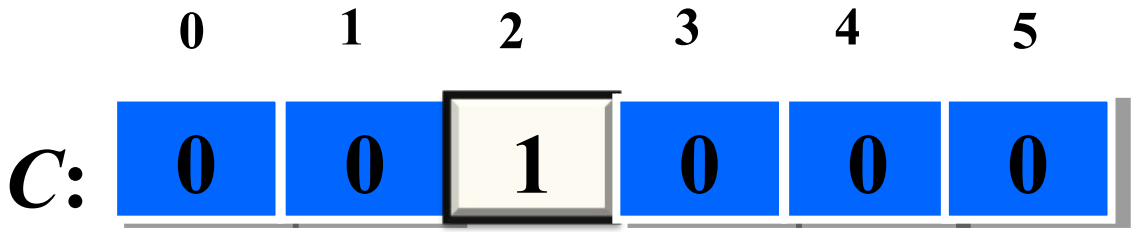


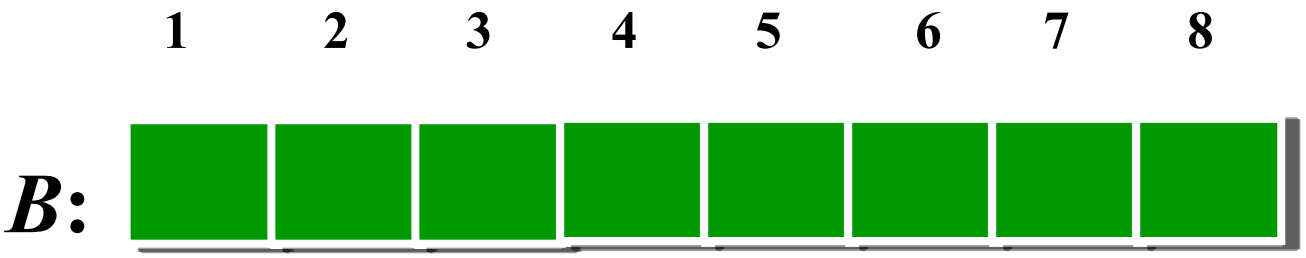


Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```

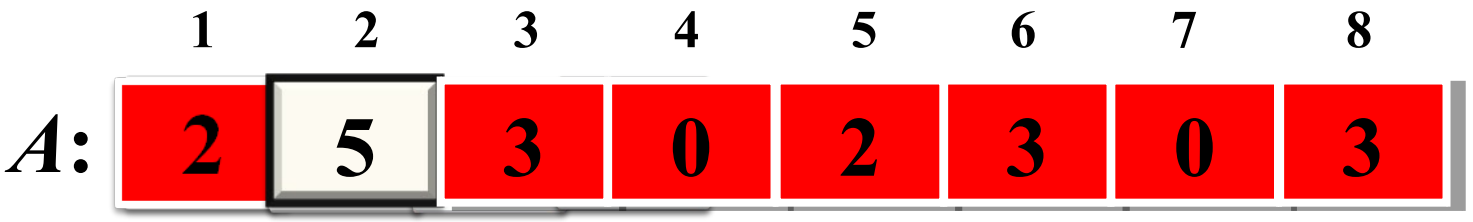


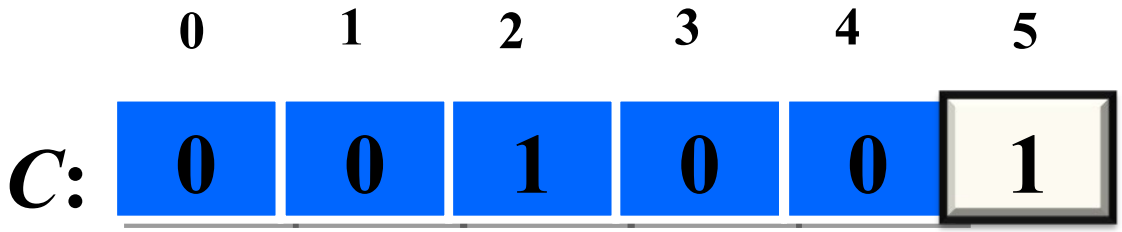


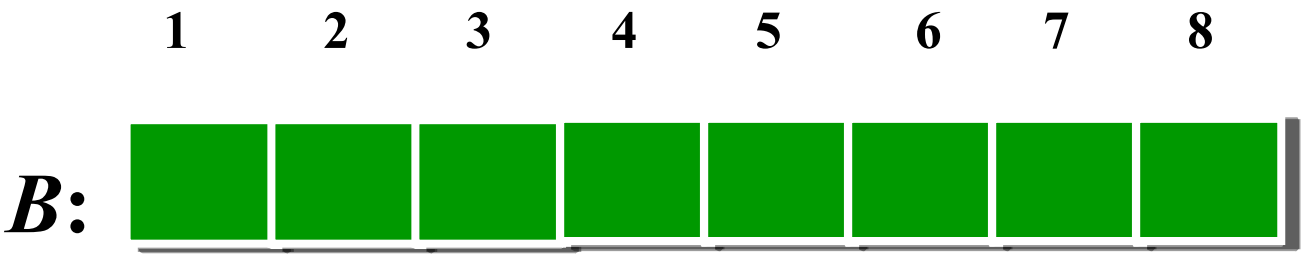


Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```

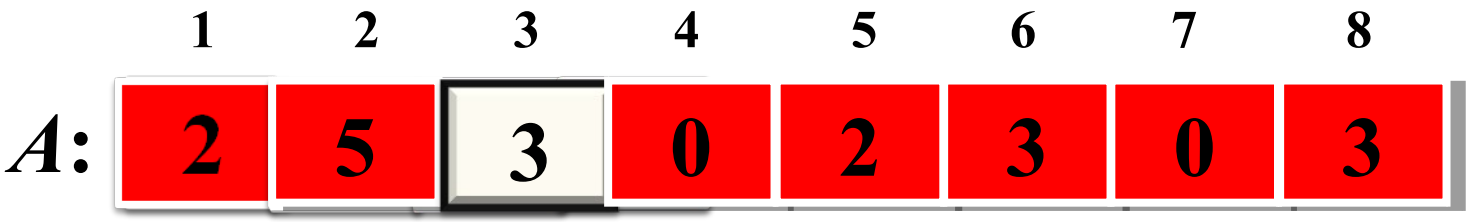


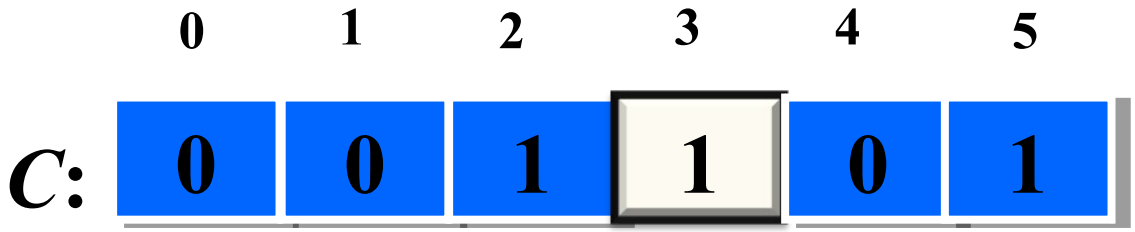


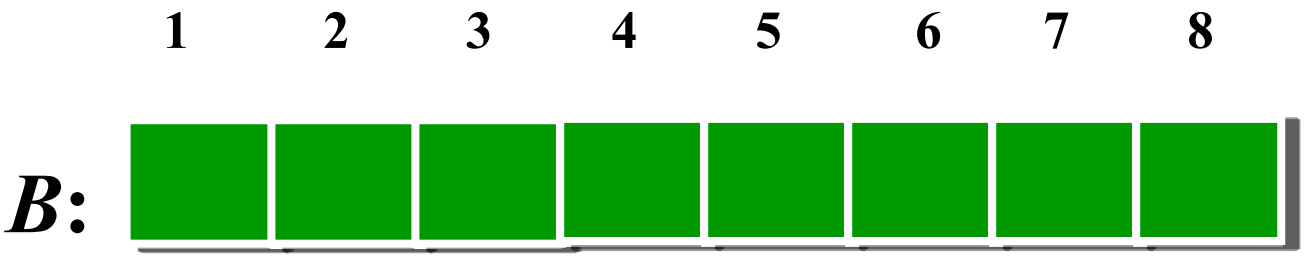


Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```

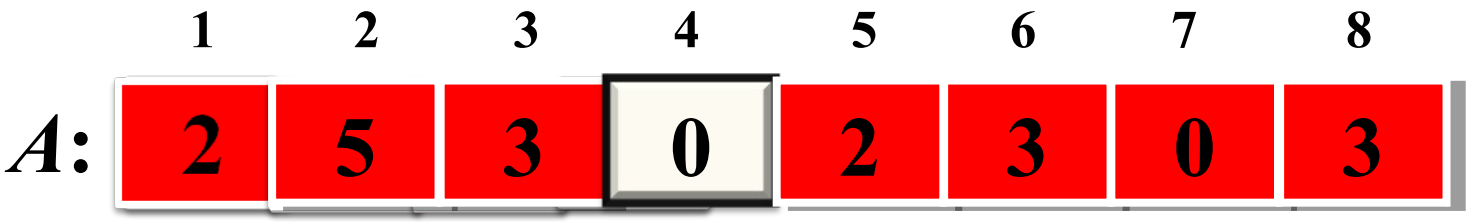


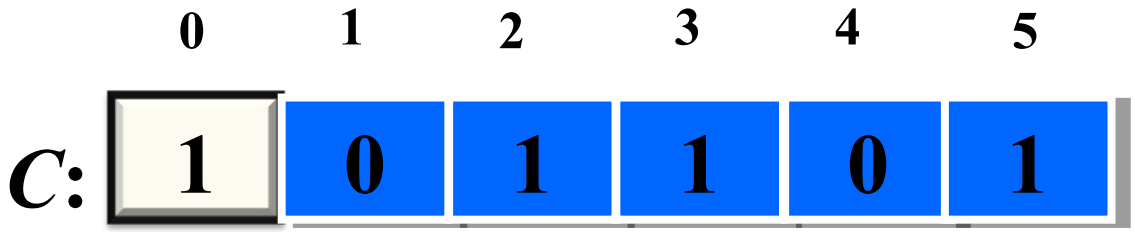


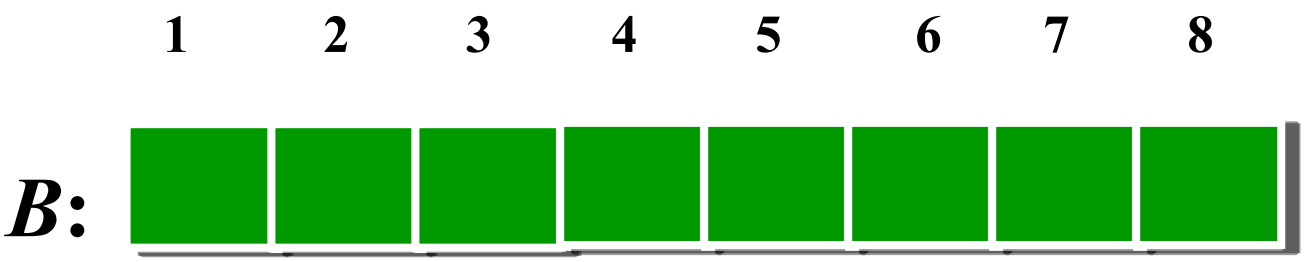


Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```

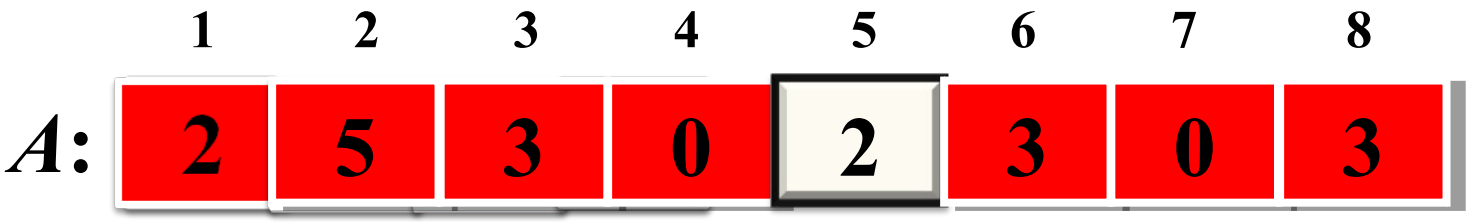


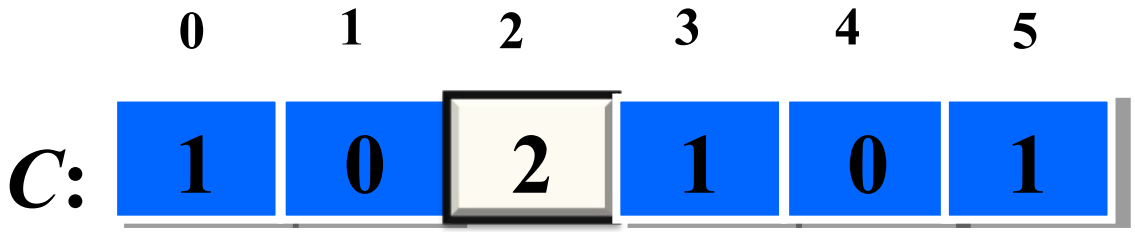


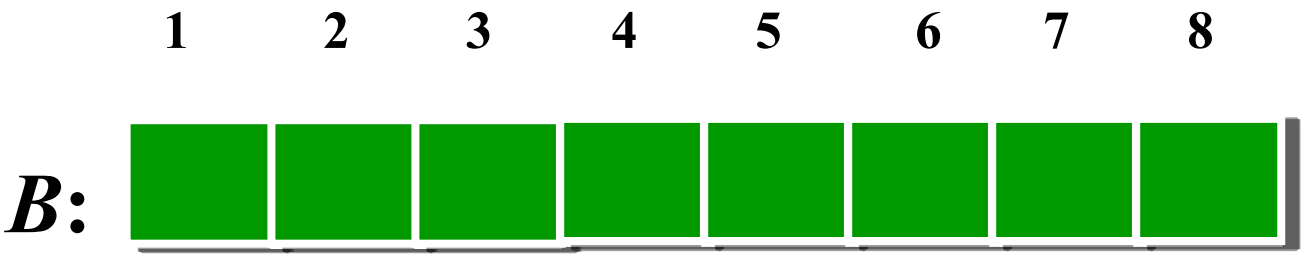


Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```

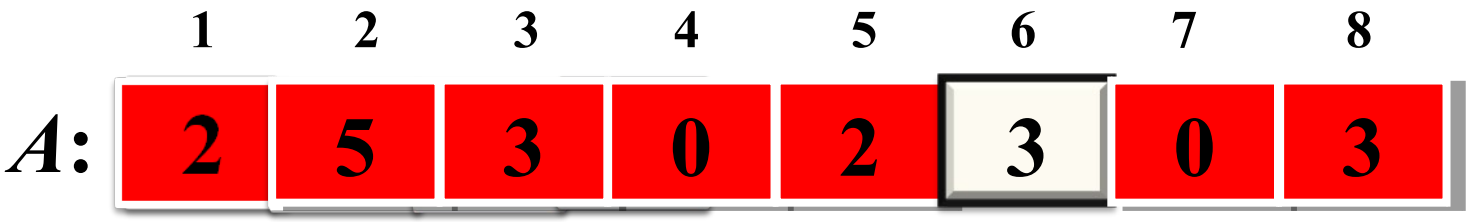


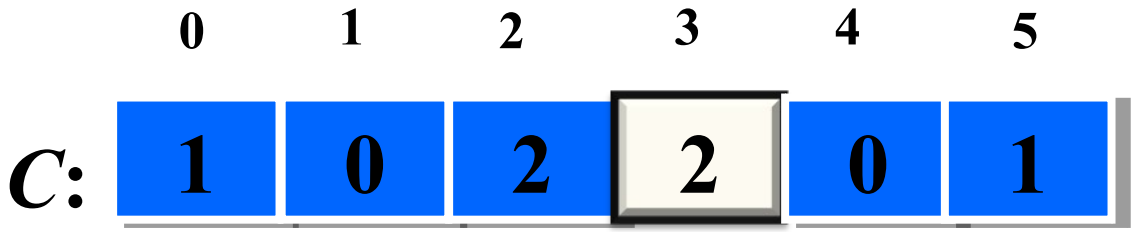


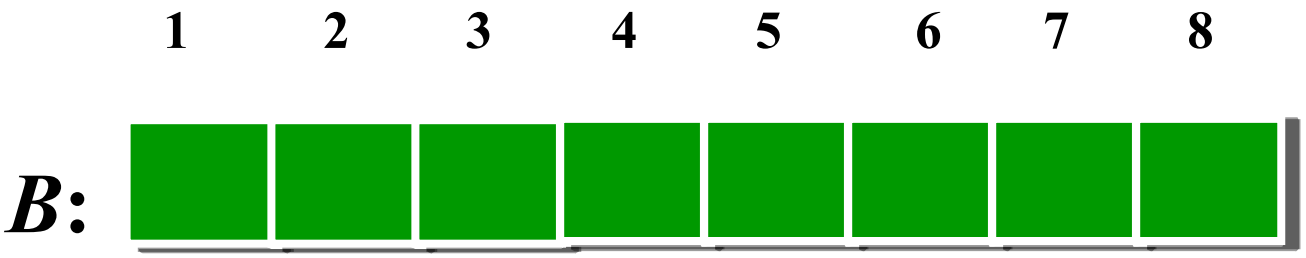


Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```

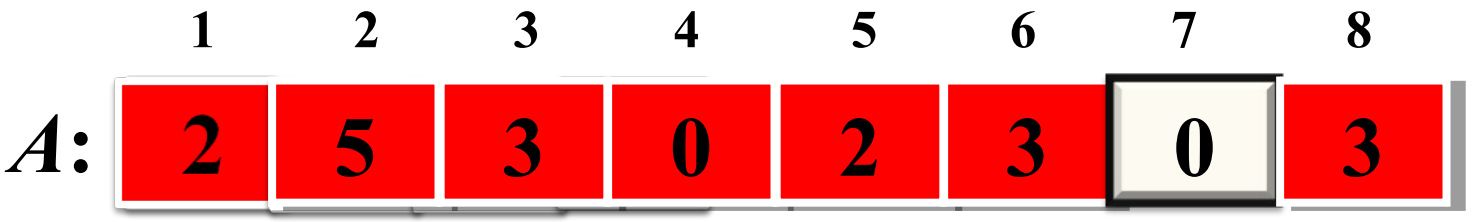


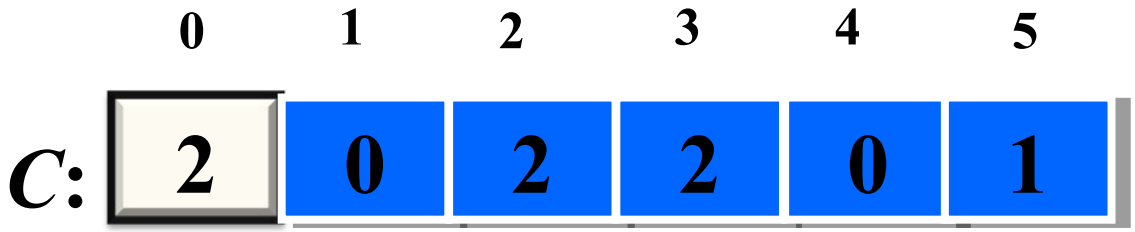


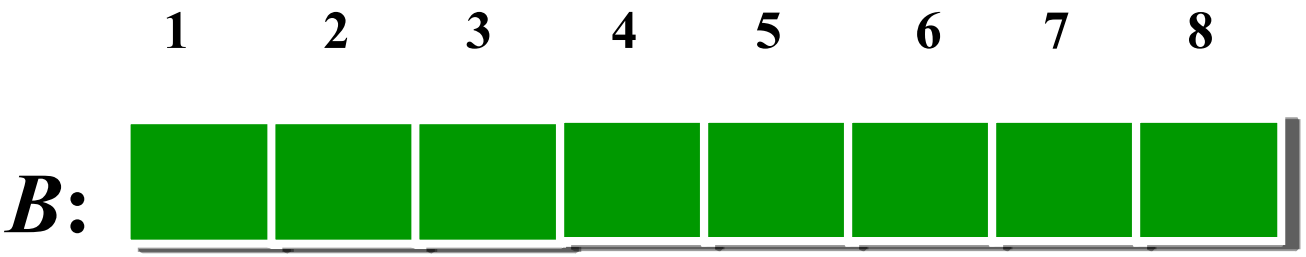


Executing Loop 2

```
5.  for j=1 to A.length  or n
6.        C[ A[j] ] = C[ A[j] ]  + 1;
```

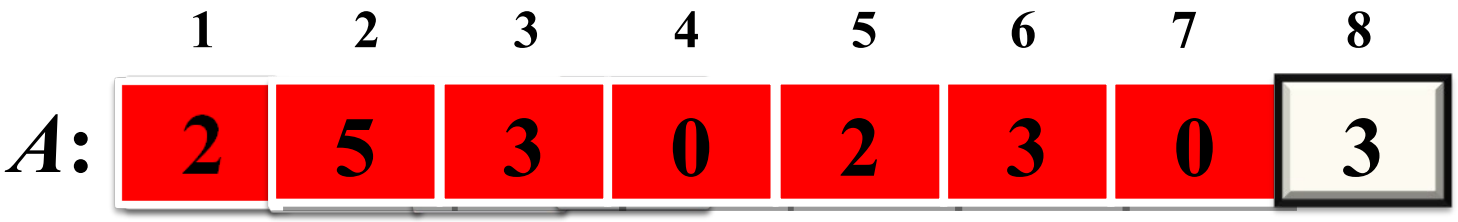


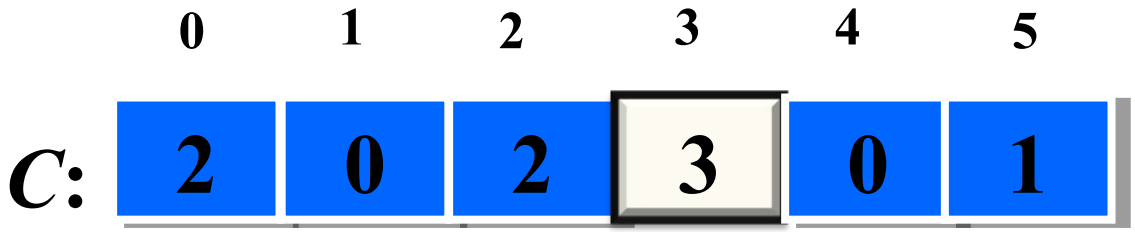


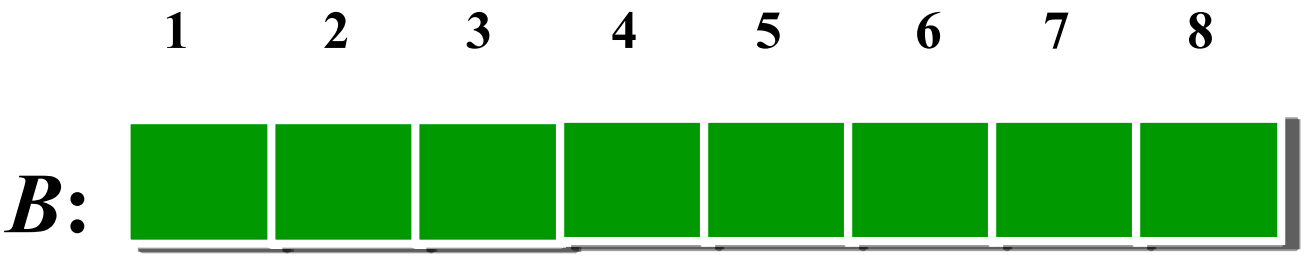


Executing Loop 2

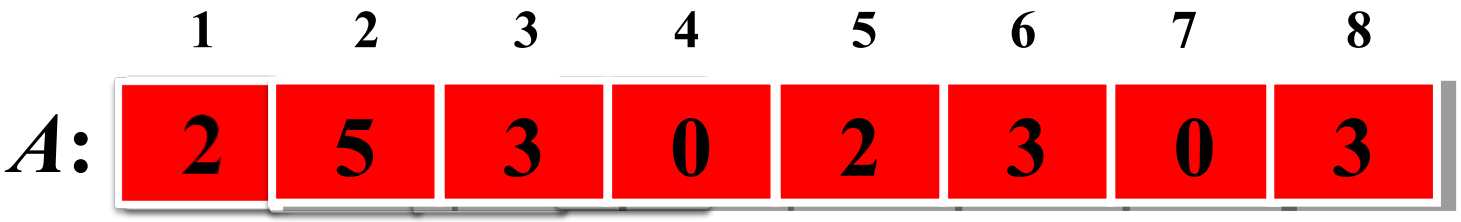
```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```

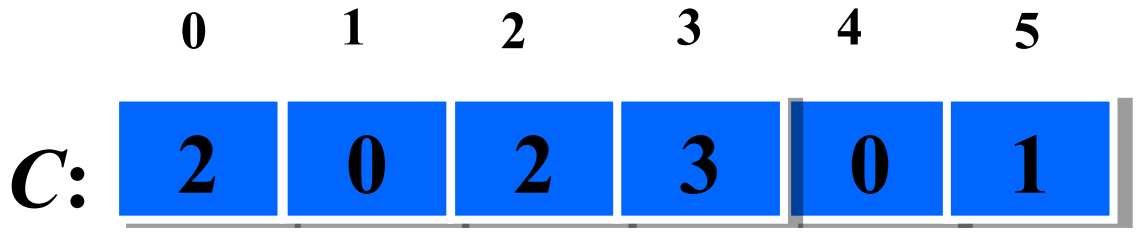


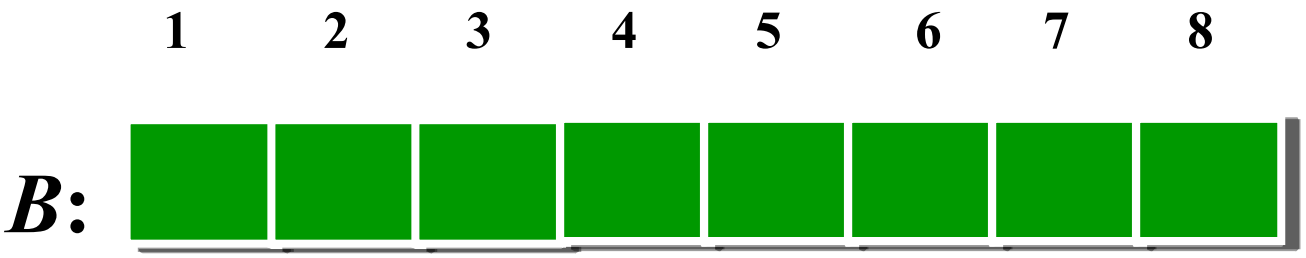




End of Loop 2

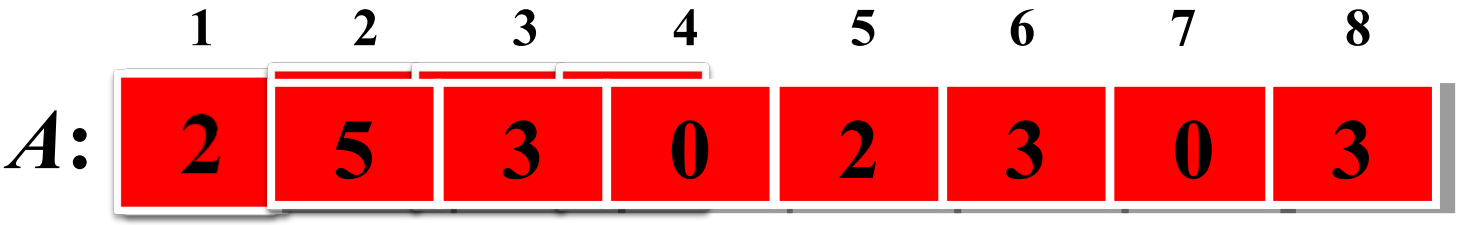


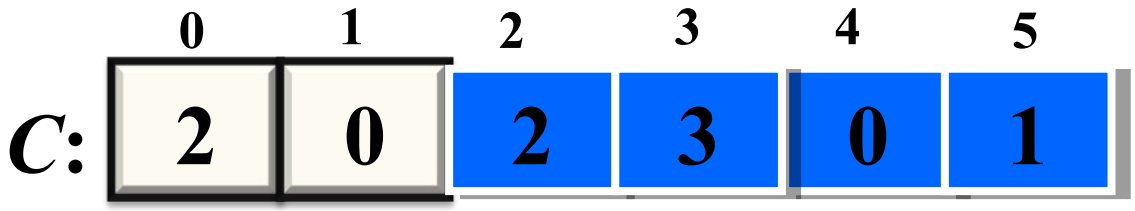


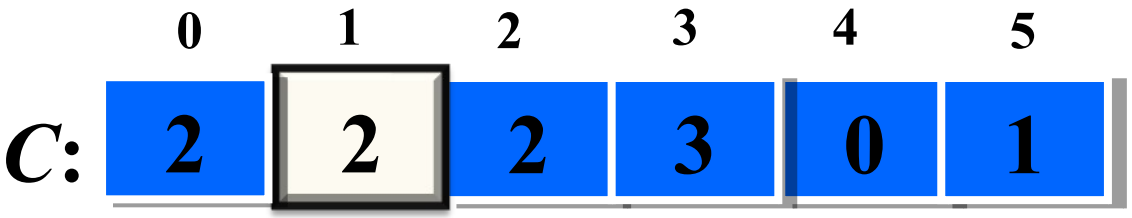


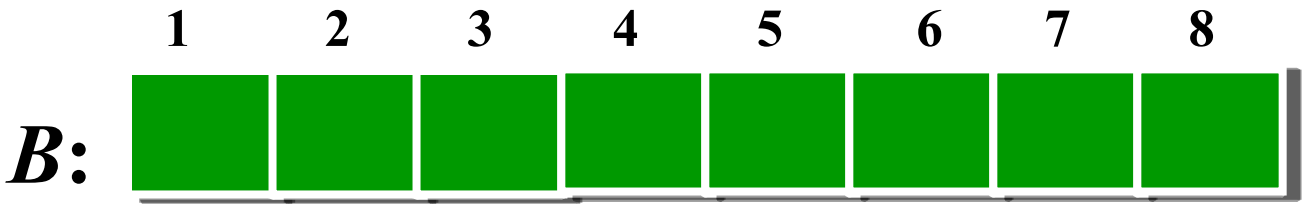
Executing Loop 3

```
7.  for i=1 to k
8.      C[i] = C[i] + C[i-1];
```



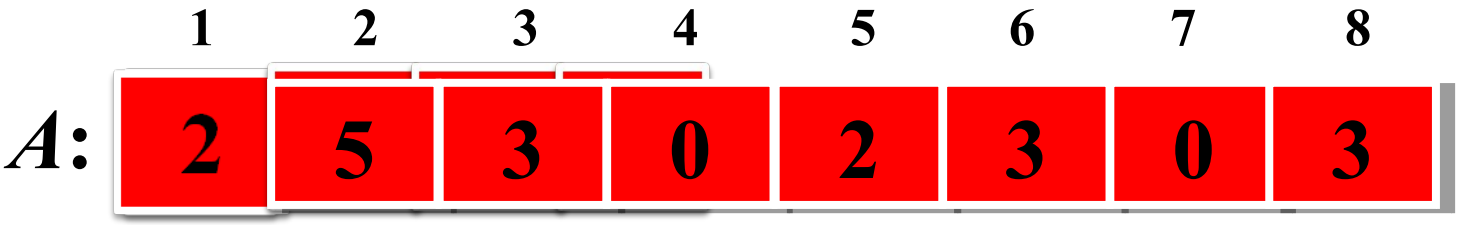


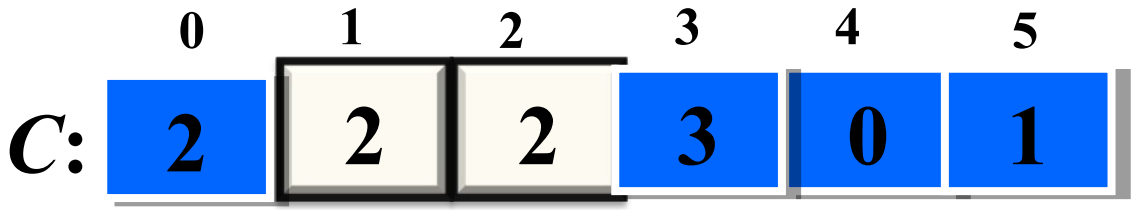


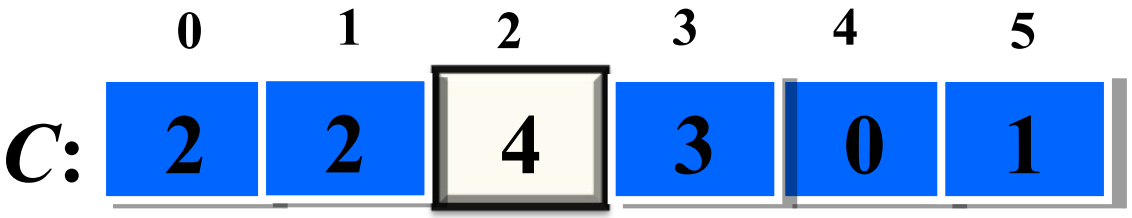


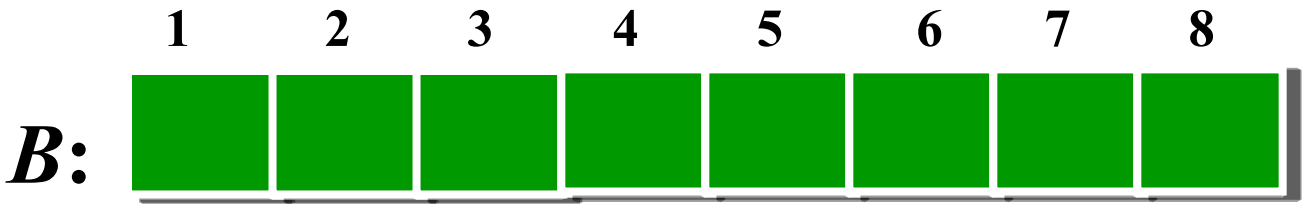
Executing Loop 3

```
7.  for i=1 to k
8.      C[i] = C[i] + C[i-1];
```



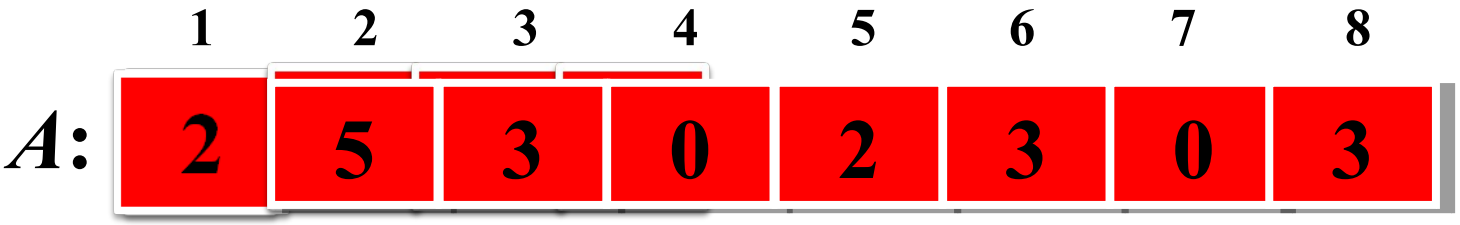


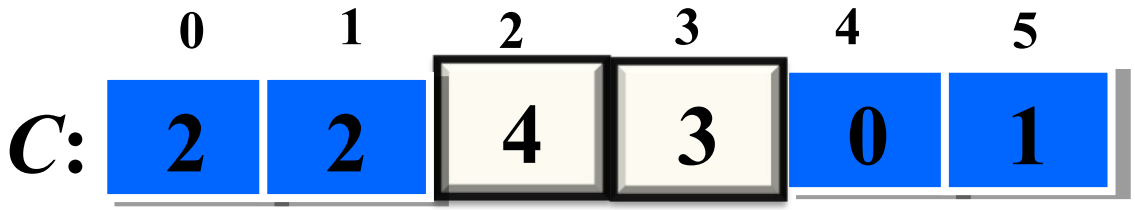


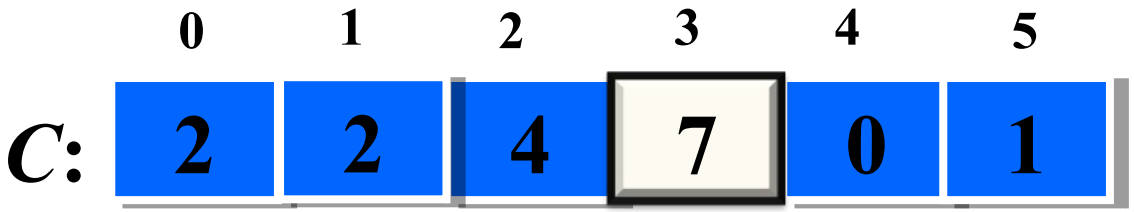


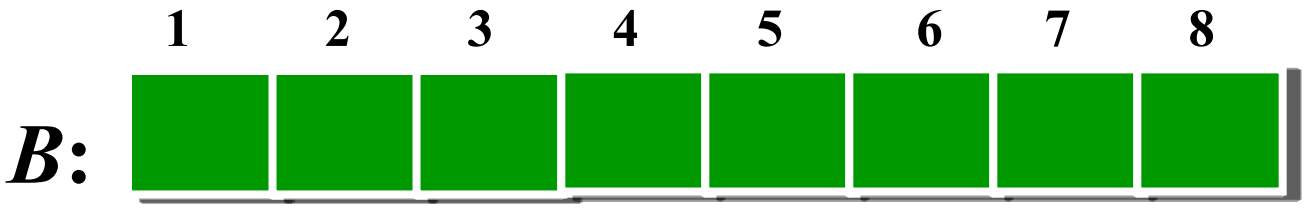
Executing Loop 3

```
7.  for i=1 to k
8.      C[i] = C[i] + C[i-1];
```



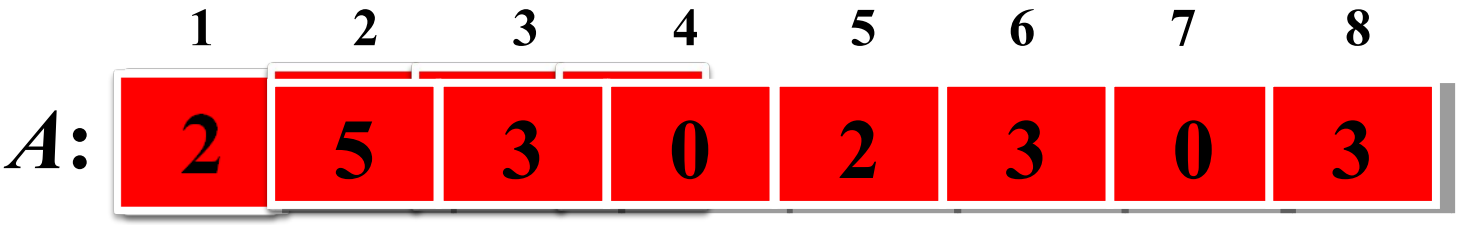


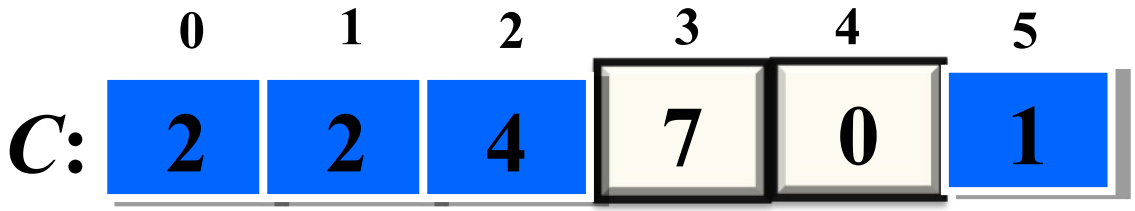


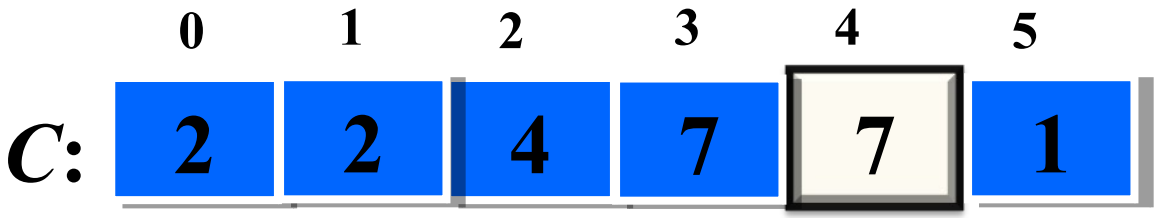


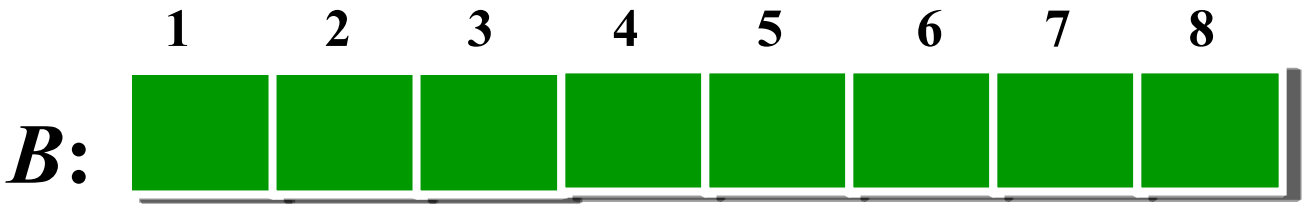
Executing Loop 3

```
7.  for i=1 to k
8.      C[i] = C[i] + C[i-1];
```



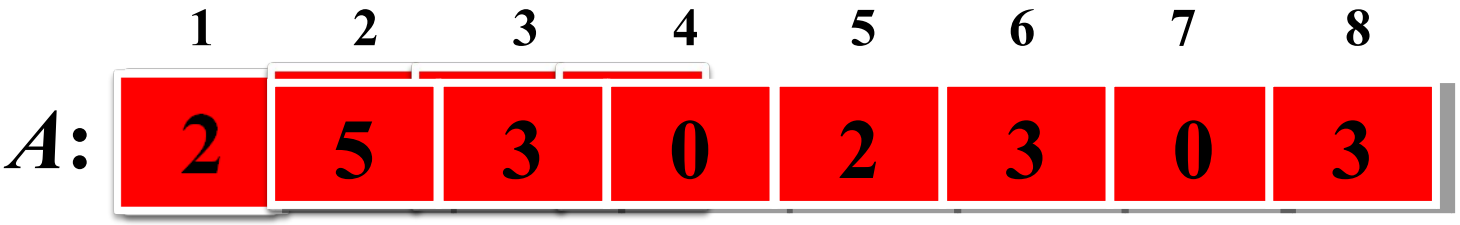


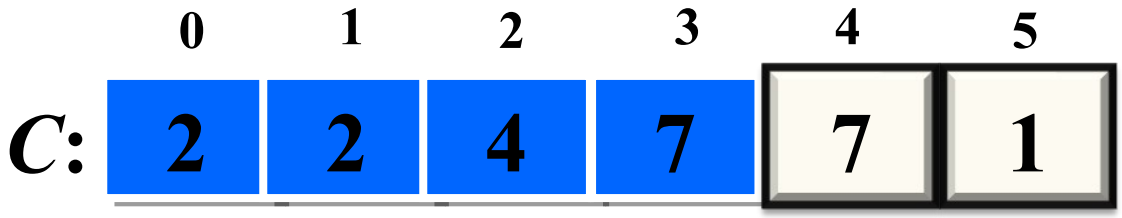


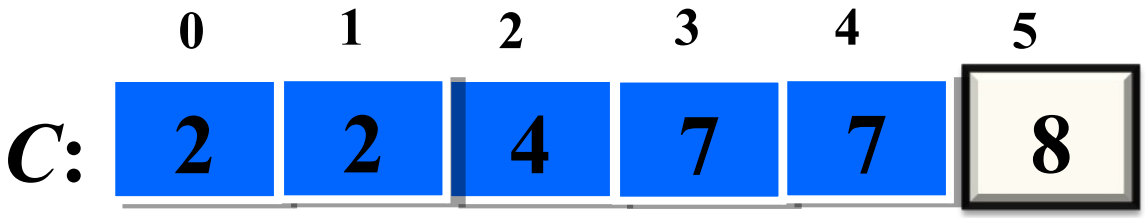


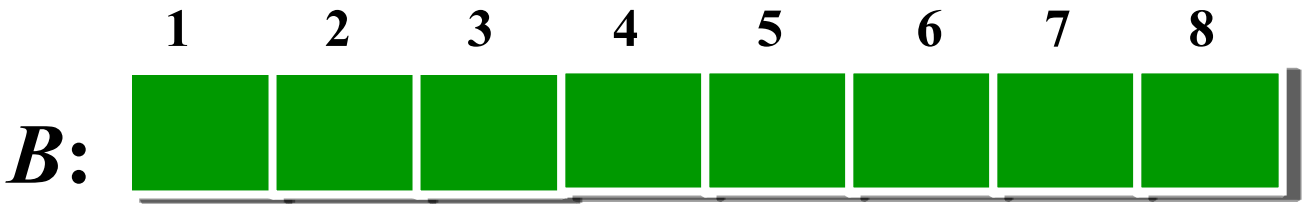
Executing Loop 3

```
7.  for i=1 to k
8.      C[i] = C[i] + C[i-1];
```

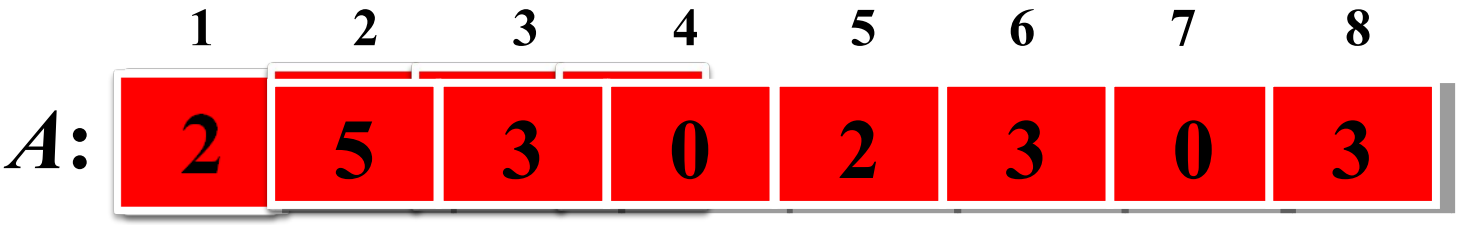


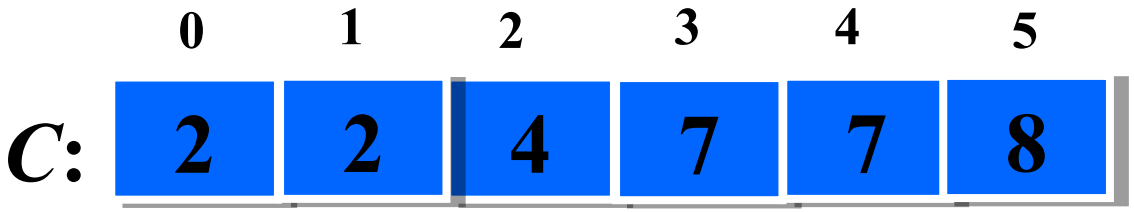


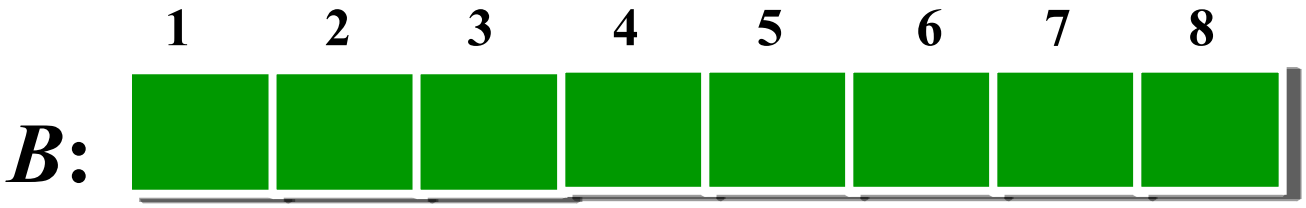




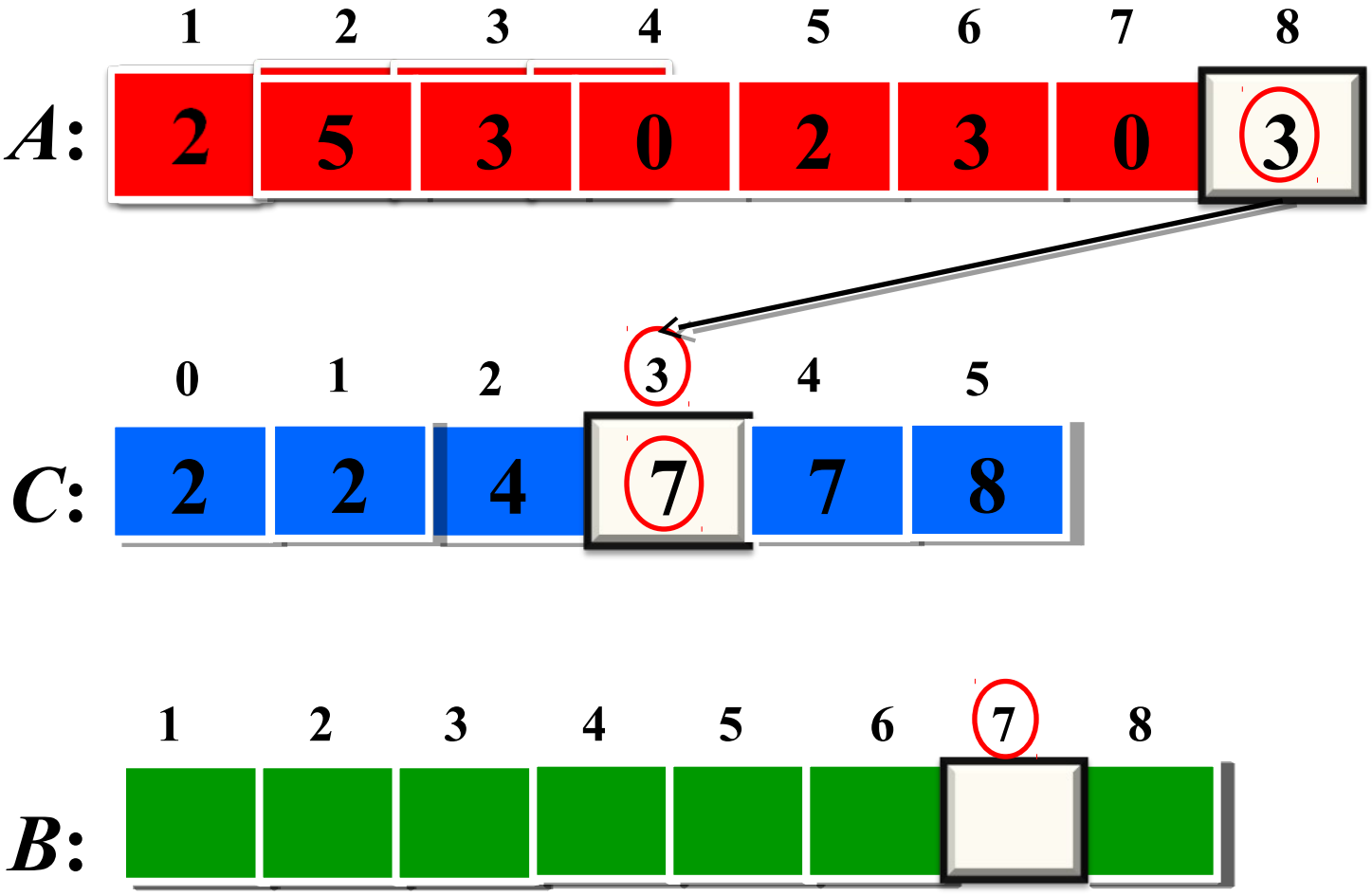
End of Loop 3





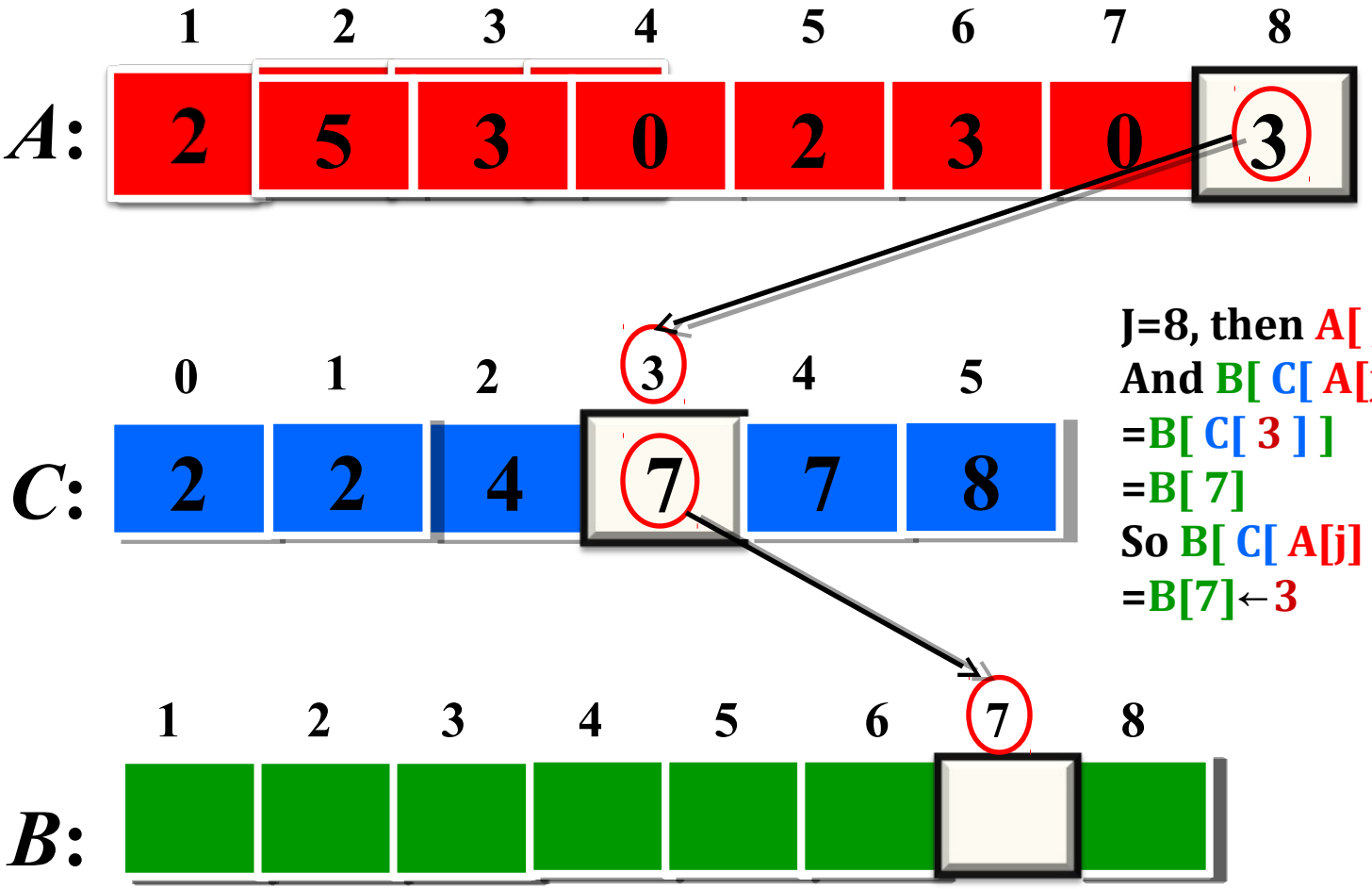


Executing Loop 4



Executing Loop 4

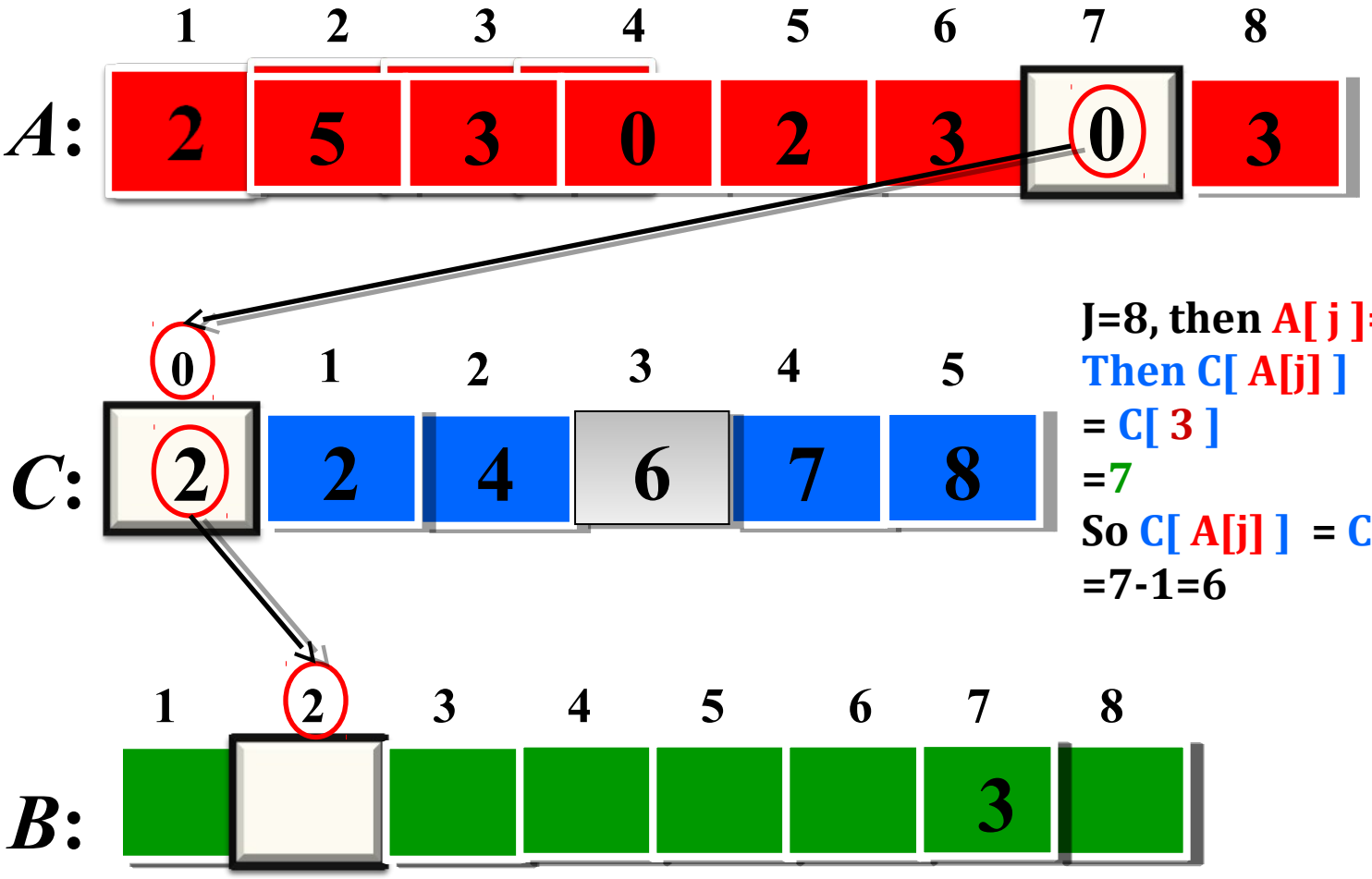
```
9.  for j=n or A.length down to 1
10.     B[ C[ A[j] ] ] = A[j];
11.     C[ A[j] ] = C[ A[j] ] - 1;
```



J=8, then A[j]=A[8]=3
And B[C[A[j]]]
=B[C[3]]
=B[7]
So B[C[A[j]]] ←A[j]
=B[7]←3

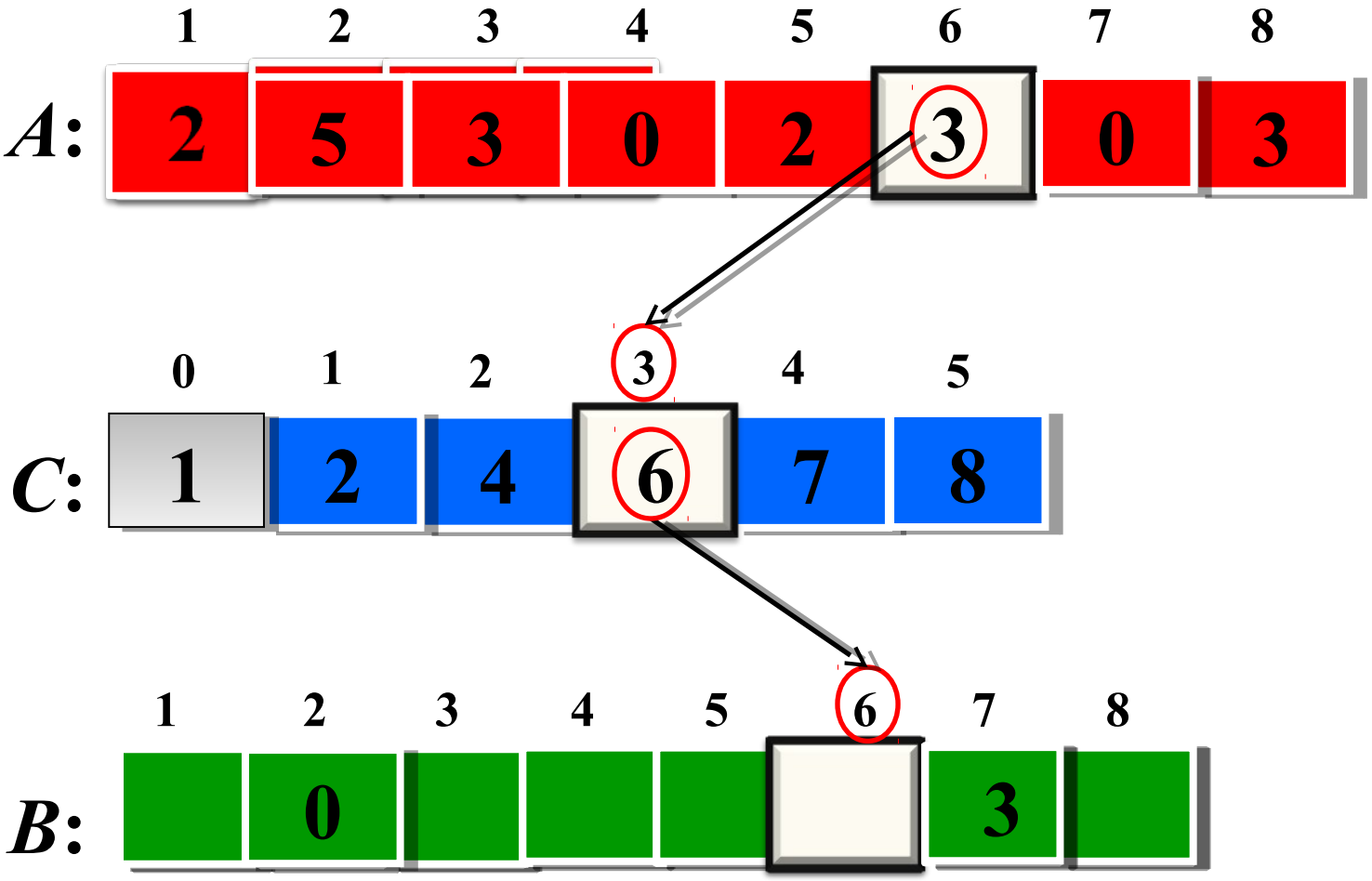
Executing Loop 4

```
9.  for j=n or A.length down to 1
10.     B[ C[ A[j] ] ] = A[j];
11.     C[ A[j] ] = C[ A[j] ] - 1;
```

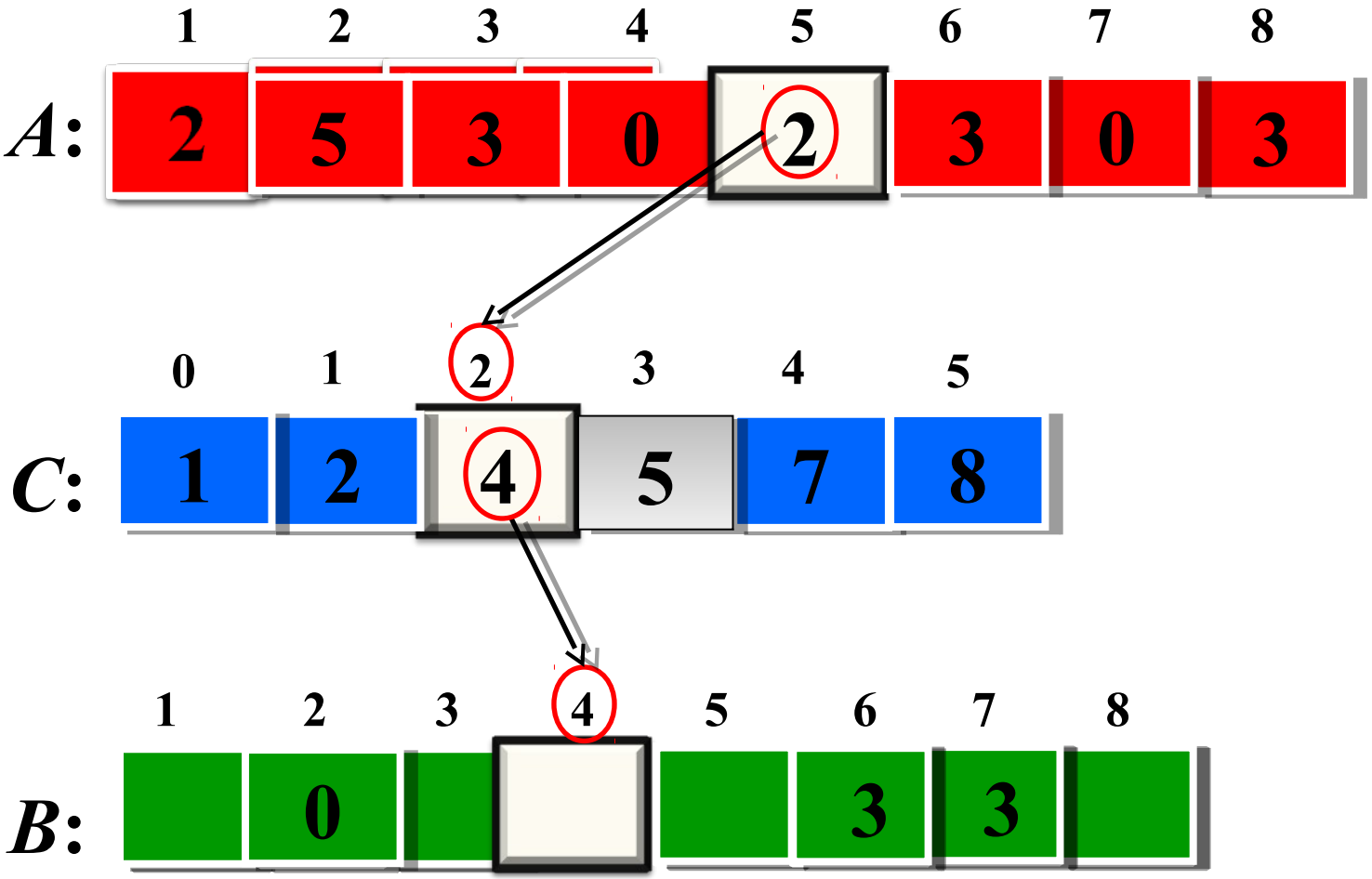


J=8, then A[j]=A[8]=3
Then C[A[j]]
= C[3]
=7
So C[A[j]] = C[A[j]] -1
=7-1=6

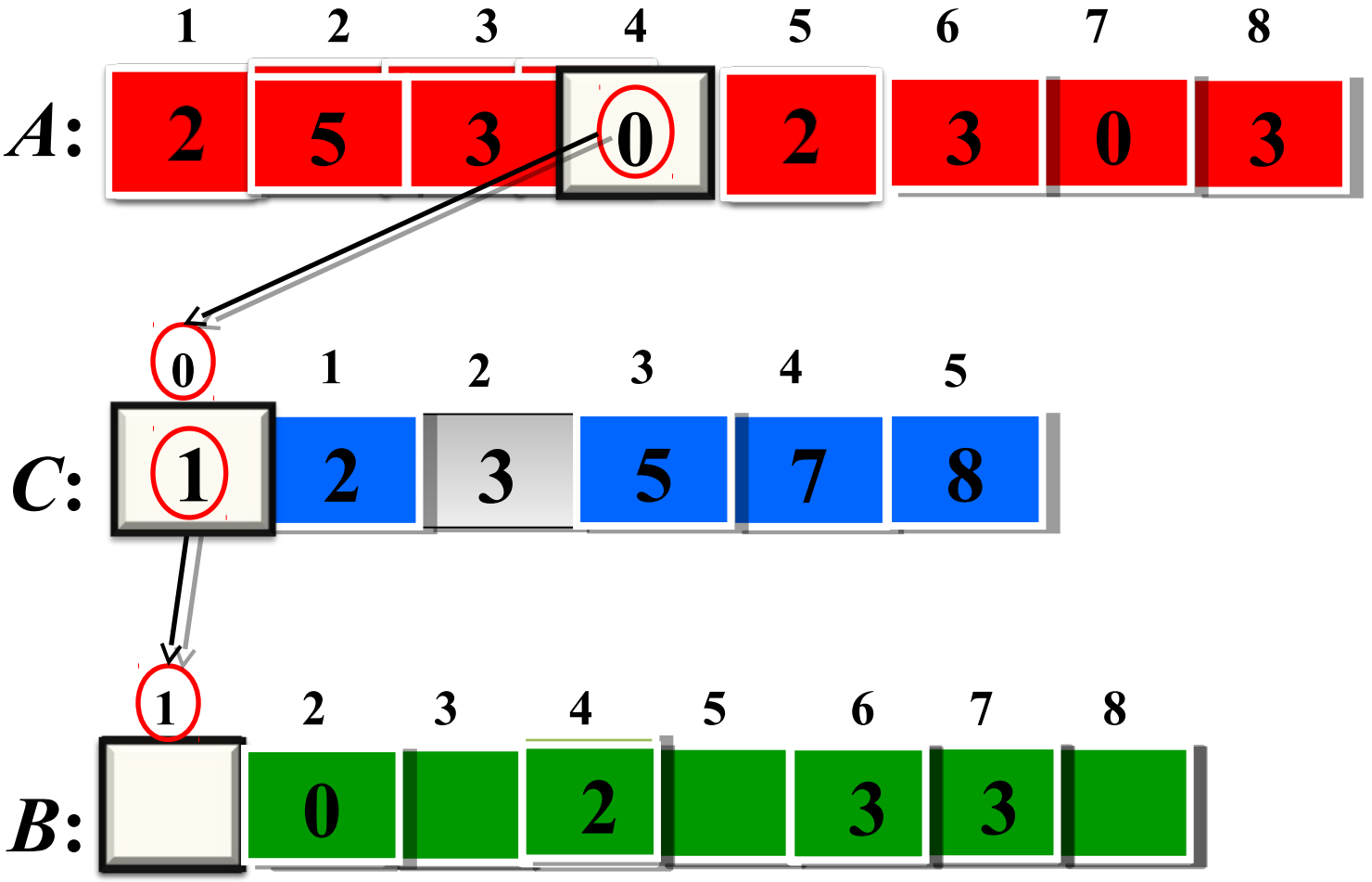
Executing Loop 4



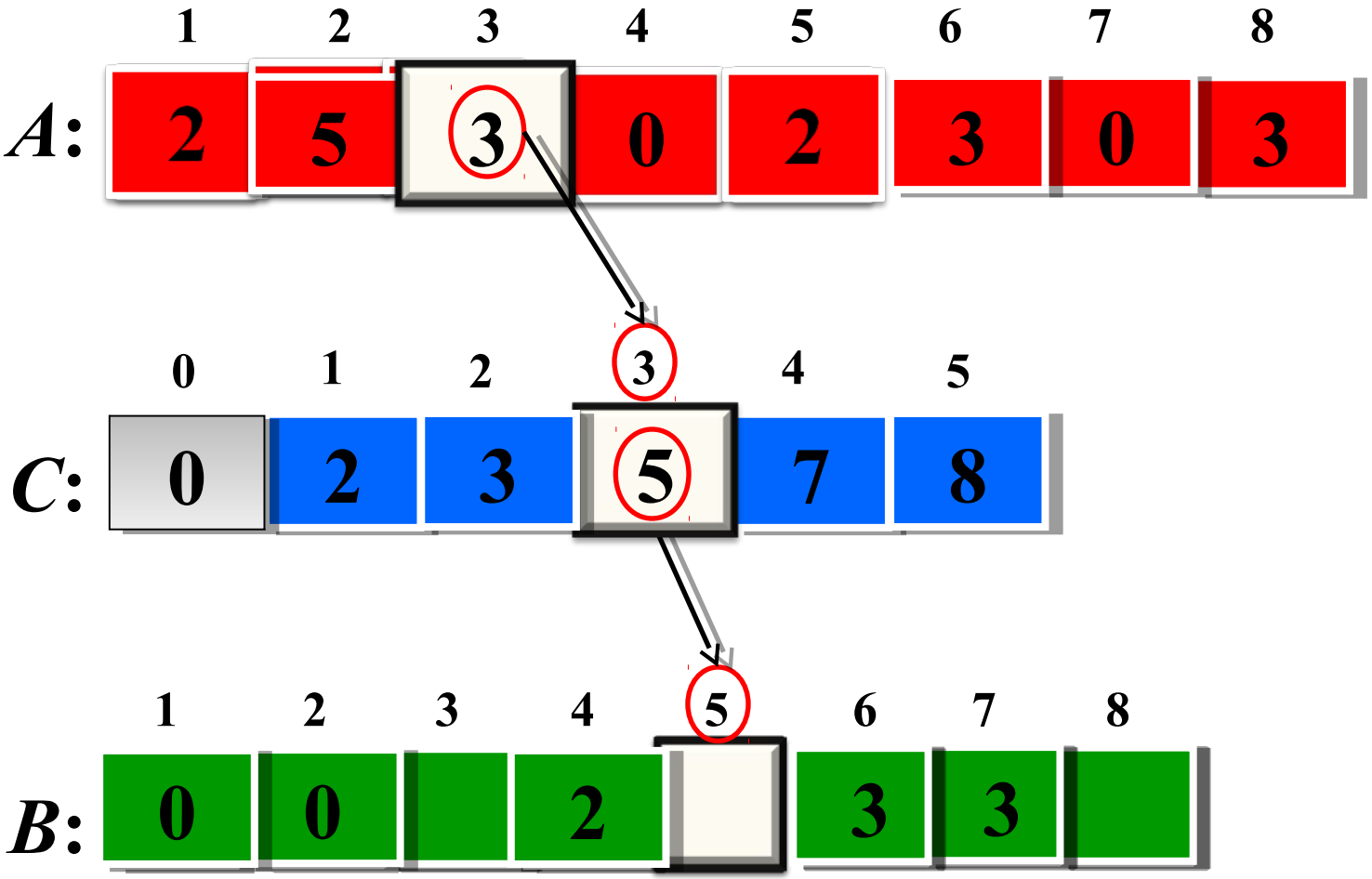
Executing Loop 4



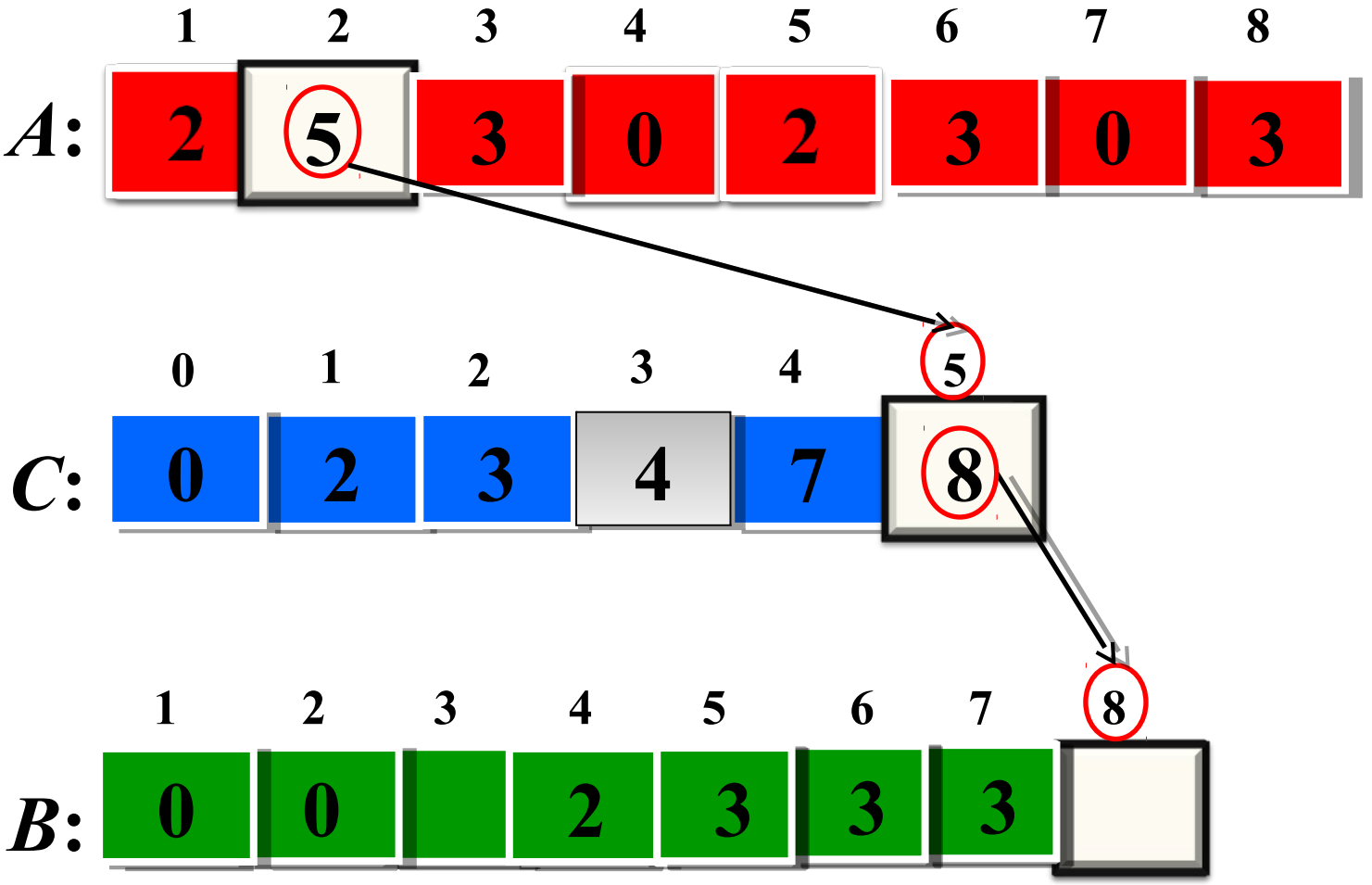
Executing Loop 4



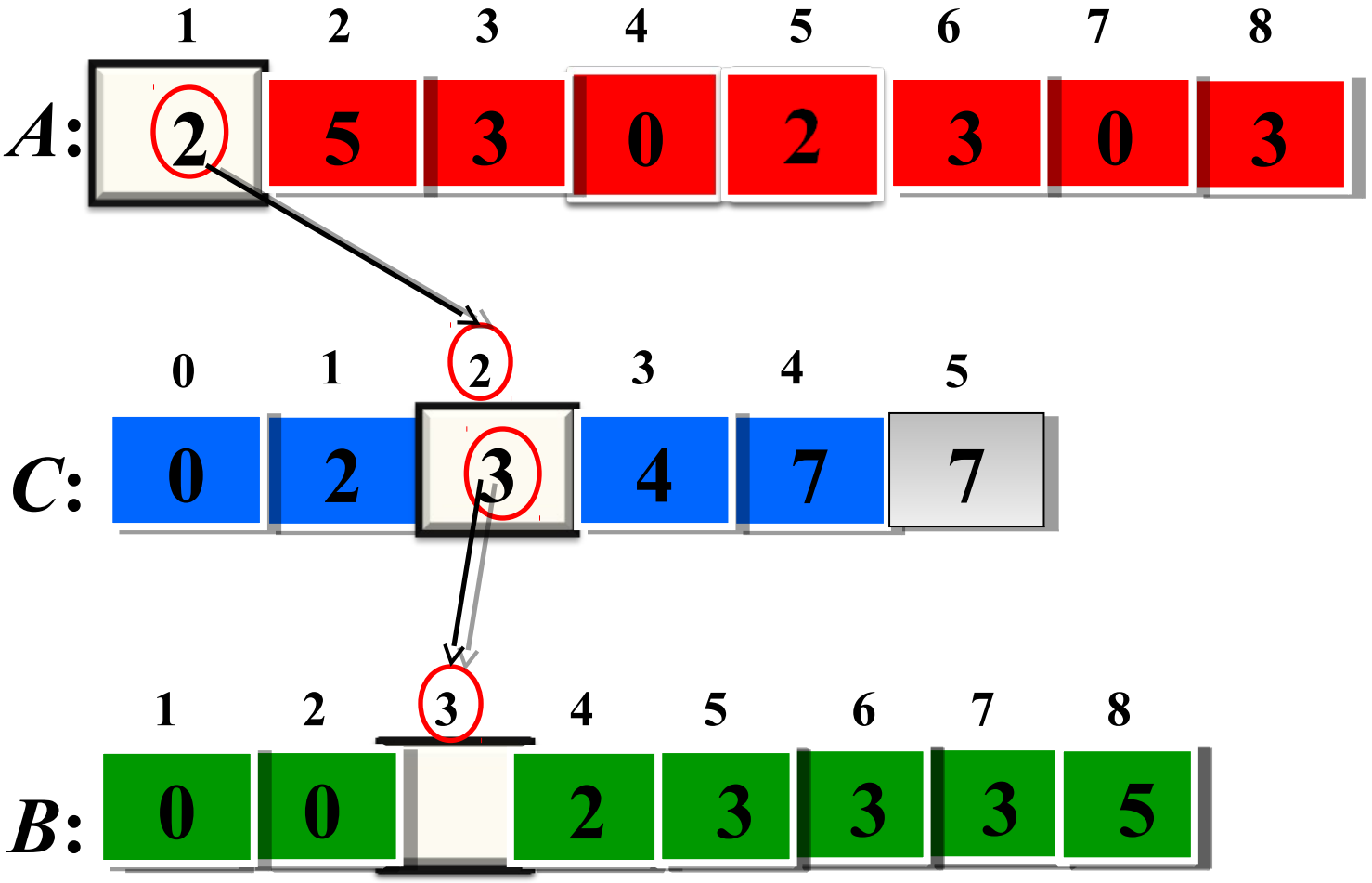
Executing Loop 4



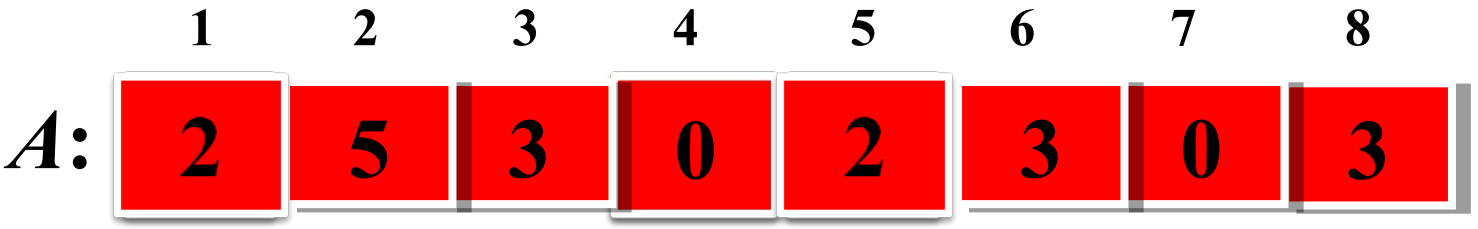
Executing Loop 4

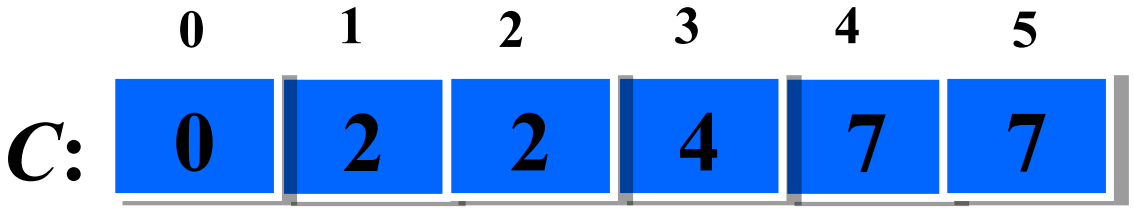


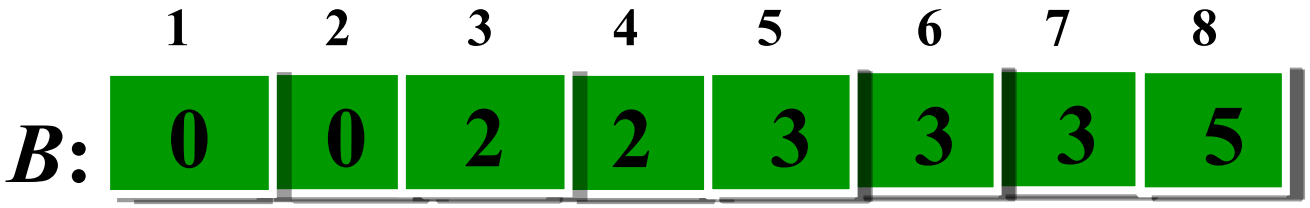
Executing Loop 4



End of Loop 4



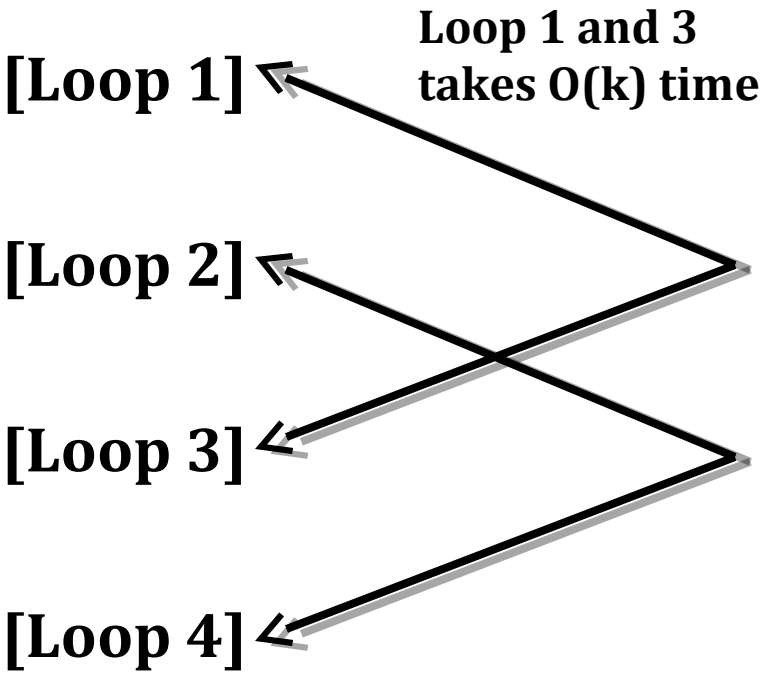




Sorted data in Array B

Time Complexity Analysis

```
1.  Counting-Sort(A, B, k)
2.  Let C[0…..k] be a new array
3.  for i=0 to k                                   [Loop 1]
4.        C[i]= 0;
5.  for j=1 to A.length  or n                      [Loop 2]
6.        C[ A[j] ] = C[ A[j] ]  + 1;
7.  for i=1 to k                                   [Loop 3]
8.        C[i] = C[i] + C[i-1];
9.  for j=n or A.length down to 1                  [Loop 4]
10.       B[  C[ A[j] ]  ] = A[j];
11.       C[ A[j] ] = C[ A[j] ]  - 1;
```



Time Complexity Analysis

- So the counting sort takes a total time of: $O(n + k)$
- Counting sort is called stable sort.
  - A sorting algorithm is ***stable*** when numbers with the same values appear in the output array in the same order as they do in the input array.

Counting Sort Review

- **Assumption:** input taken from **small** set of **numbers** of size k
- Basic idea:
  - Count number of elements less than you for each element.
  - This gives the position of that number – similar to selection sort.
- Pro's:
  - Fast
  - Asymptotically fast - $O(n+k)$
  - Simple to code
- Con's:
  - Doesn't sort in place.
  - Requires $O(n+k)$ extra storage.

Radix Sort

- Radix sort is non comparative sorting method
- Two classifications of radix sorts are least significant digit (LSD) radix sorts and most significant digit (MSD) radix sorts.
- LSD radix sorts process the integer representations starting from the least digit and move towards the most significant digit. MSD radix sorts work the other way around.

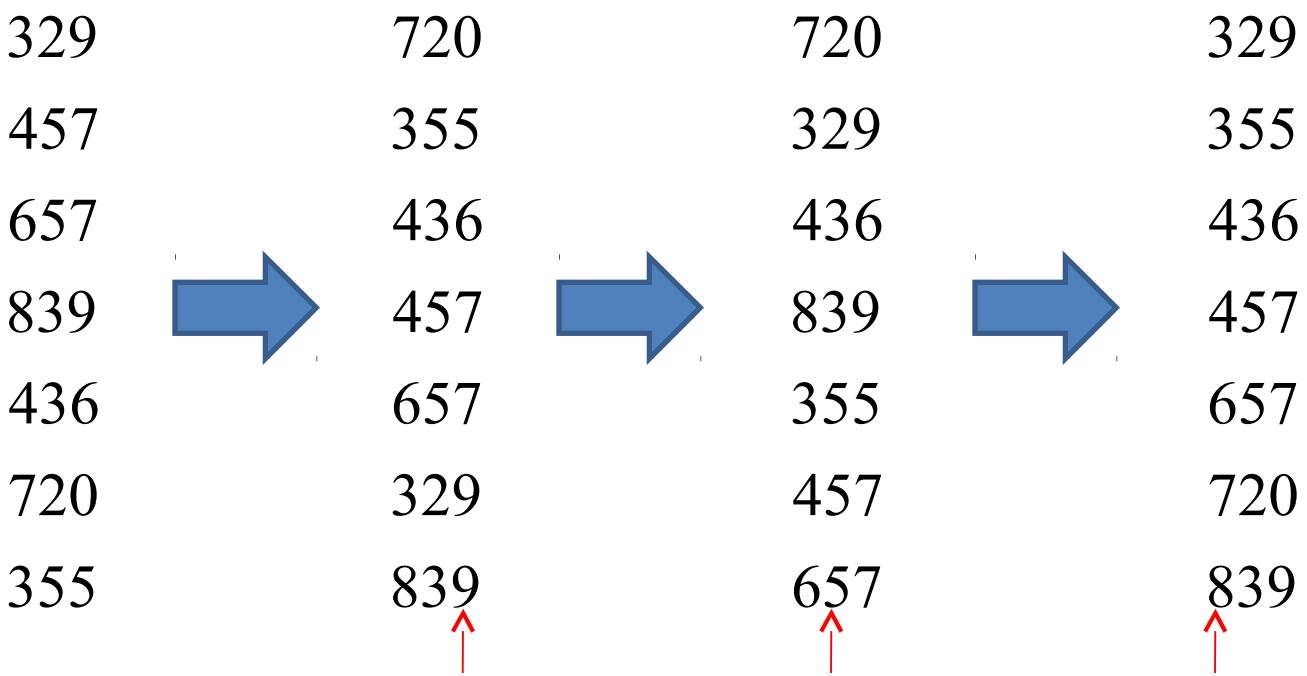
Radix Sort

In input array A, each element is a number of d digit.

Radix - Sort(A, d)

for $i \leftarrow 1$ to d

do "use a stable sort to sort array A on digit i;



The Algorithm

```
void radixsort(int a[1000],int n,int digits)
{
 for(int i =1;i<=digits;i++)
   countsort(a,n,i);
}
```

The Algorithm

```
void countsort(int a[1000],int n,int x)
{
 int d[1000],t;
 for(int s=1;s<=n;s++)        // extracting the concerned digit from
    {  t = a[s];                                the number
   t = t / (pow(10,x-1));
  d[s] = t%10;
 }
int c[10],b[1000],i,j;
for(i=0;i<=9;i++)
   c[i] = 0;
```

The Algorithm

```
for(j = 1;j<=n;++j)
   c[d[j]] = c[d[j]] + 1;    //c[i] contains no of elements
    for(i =0;i<9;i++)                    equal to i
   c[i+1] = c[i+1] + c[i];
 for(j=n;j>0;j--)
 { b[c[d[j]]] = a[j];         //shift the array's numbers
  c[d[j]] = c[d[j]] -1;
 }
 for(i=1;i<=n;i++)
  a[i] = b[i];
}
```

Time Complexity Analysis

Given n d-digit number in which each digit can take 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.

Radix Sort Review

- **Assumption:** input taken from **large** set of **numbers**
- Basic idea:
  - Sort the input on the basis of digits starting from unit's place.
  - This gives the position of that number – similar to selection sort.
- Pro's:
  - Fast
  - Asymptotically fast - $O(d(n+k))$
  - Simple to code
- Con's:
  - Doesn't sort in place.

Thank You