Lecture # 9

Counting Sort

Theorem 8.1

- *Any comparison-based sorting algorithm has worst-case running time* Ω(n log n)
  - Proof: (Book Page No 167, 2nd Edition)

- The lower bound implies that if we hope to sort numbers faster than O(n log n), we cannot do it by making comparisons alone.

- Is it possible to sort without making comparisons?
- The answer is yes, but only under very restrictive circumstances.

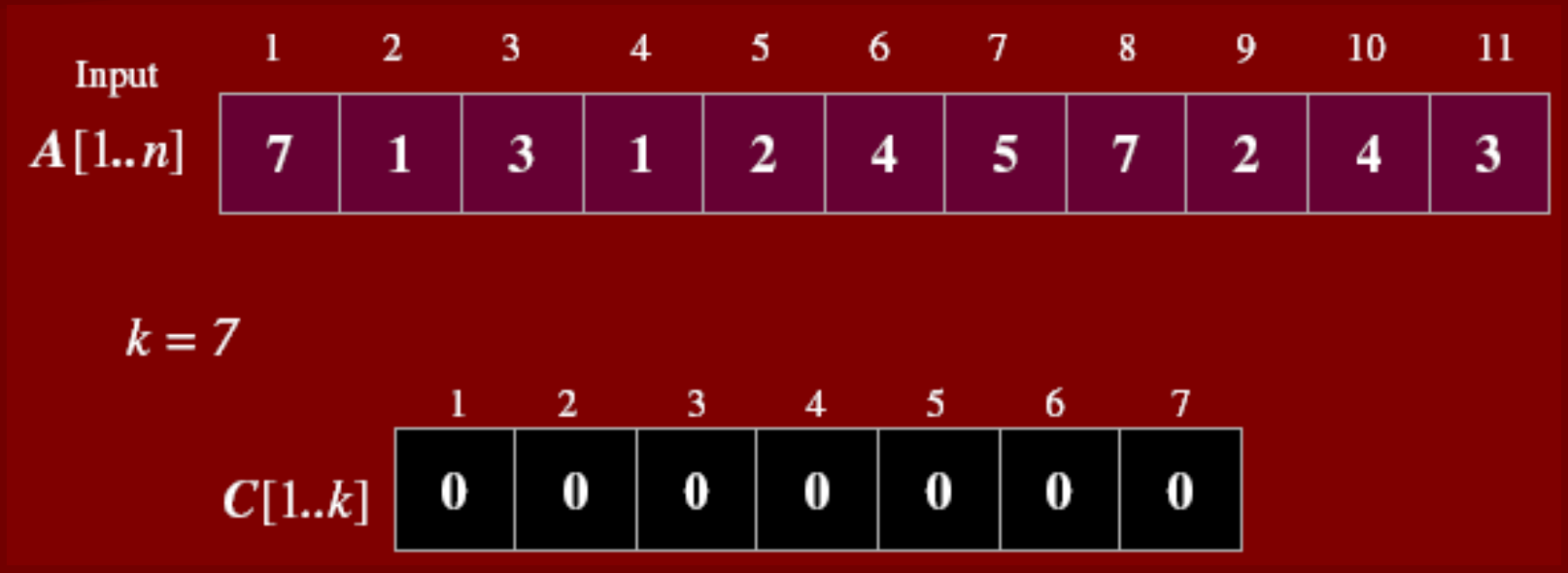
Counting Sort

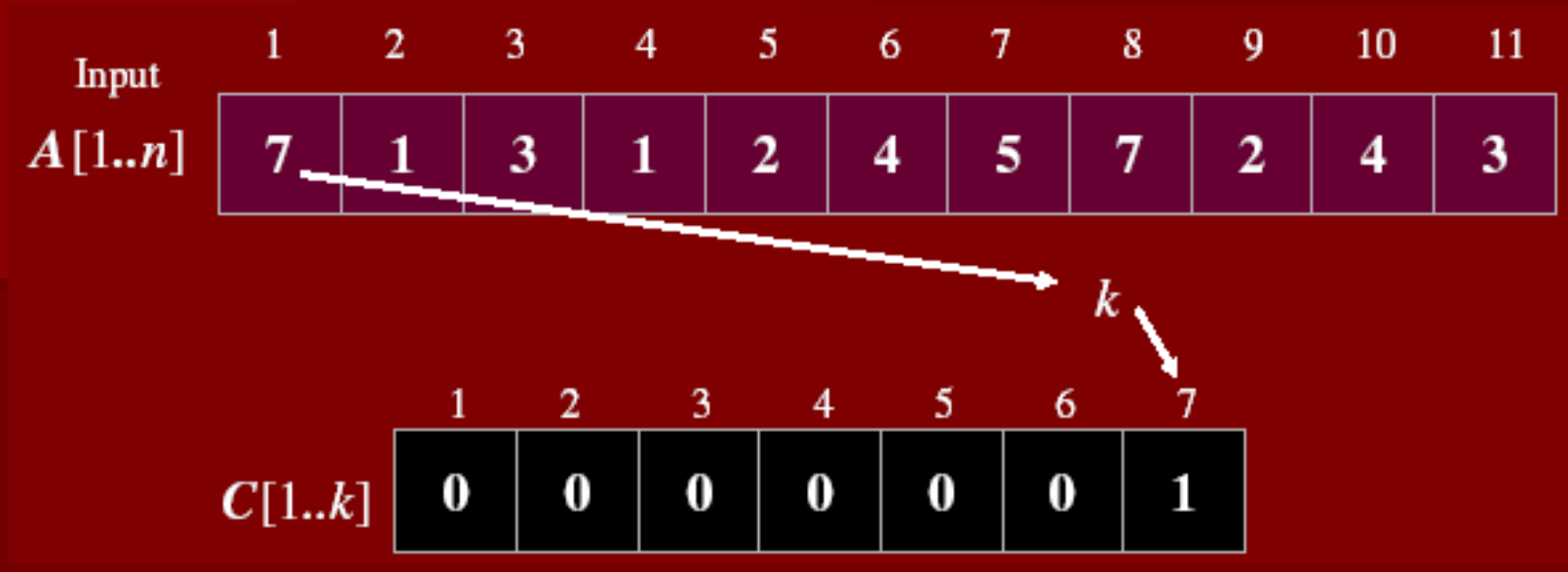
- We will consider Counting Sort algorithm that is faster and work by not making comparisons.
- Counting sort assumes that the numbers to be sorted are in the range 1 to k where k is small. The basic idea is to determine the rank of each number in final sorted array.
- The rank of an item is the *number of elements that are less than or equal to it*.
- Once we know the ranks, we simply copy numbers to their final position in an output array.

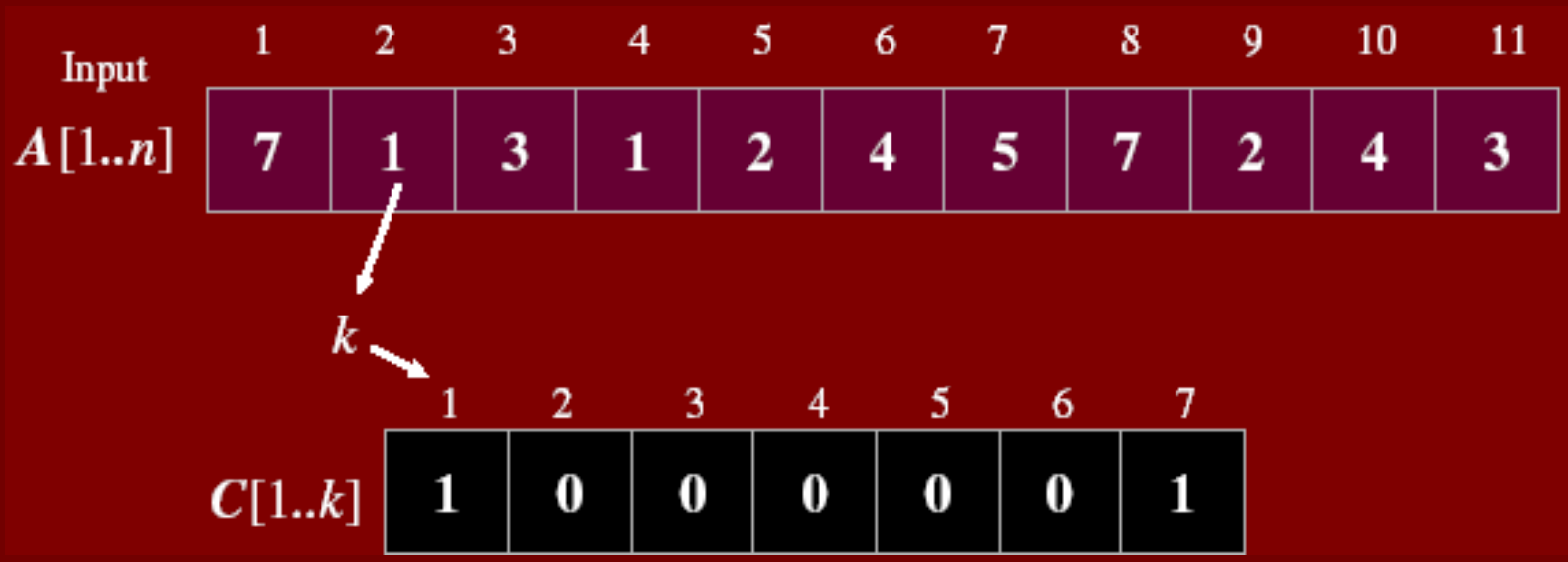
Counting Sort

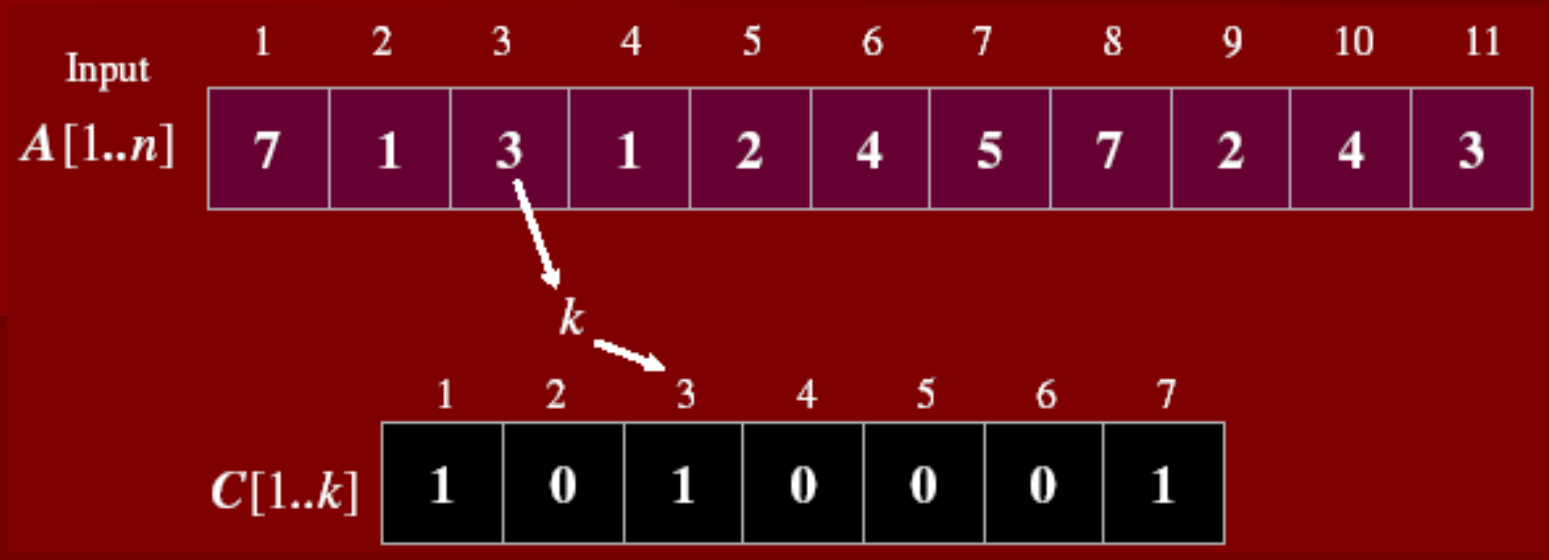
- The algorithm uses three arrays. As usual,
- A[1..n] holds the initial input,
- B[1..n] holds the sorted output and
- C[1..k] is an array of integers. C[x] is the rank of x in A, where x $\in$ [1..k].

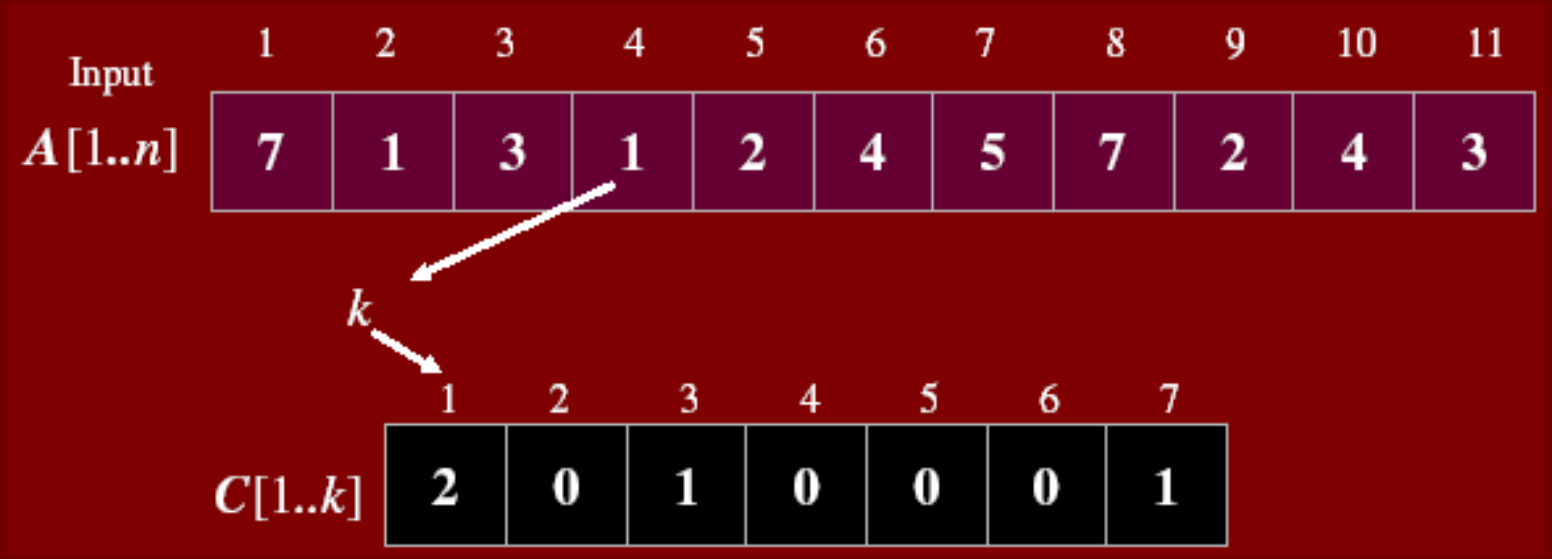
Example

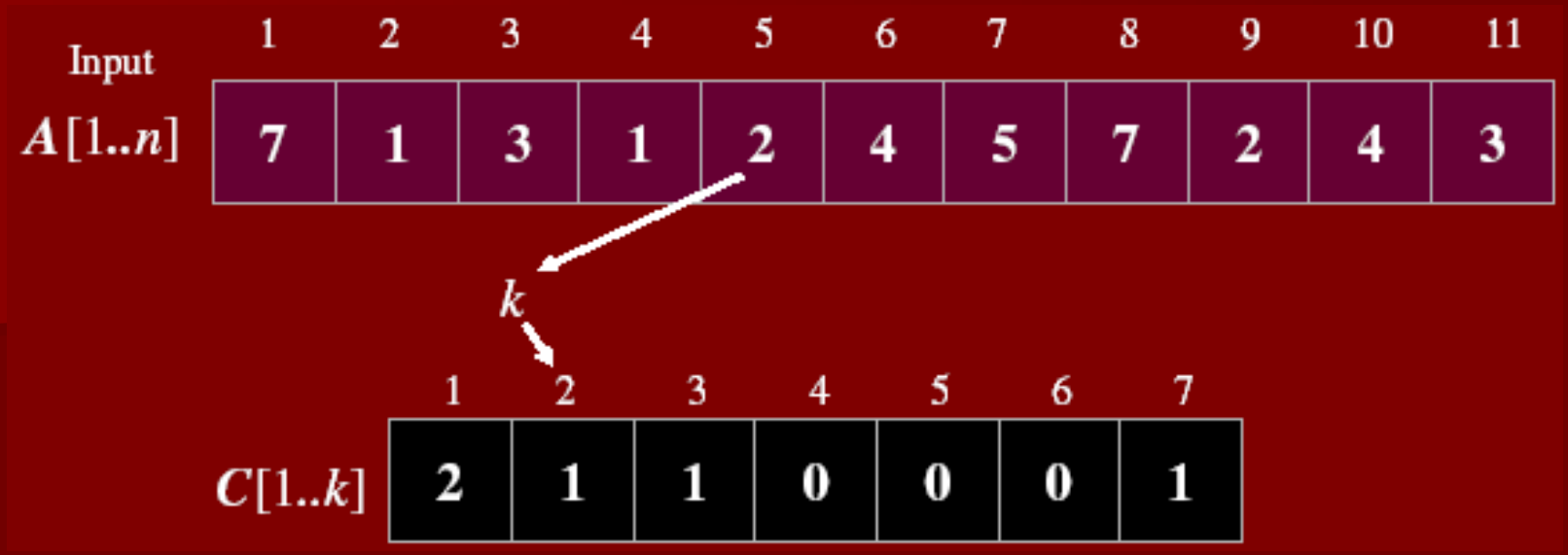

Input
1 2 3 4 5 6 7 8 9 10 11
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
k = 7
1 2 3 4 5 6 7
C[1..k]
0 0 0 0 0 0 0

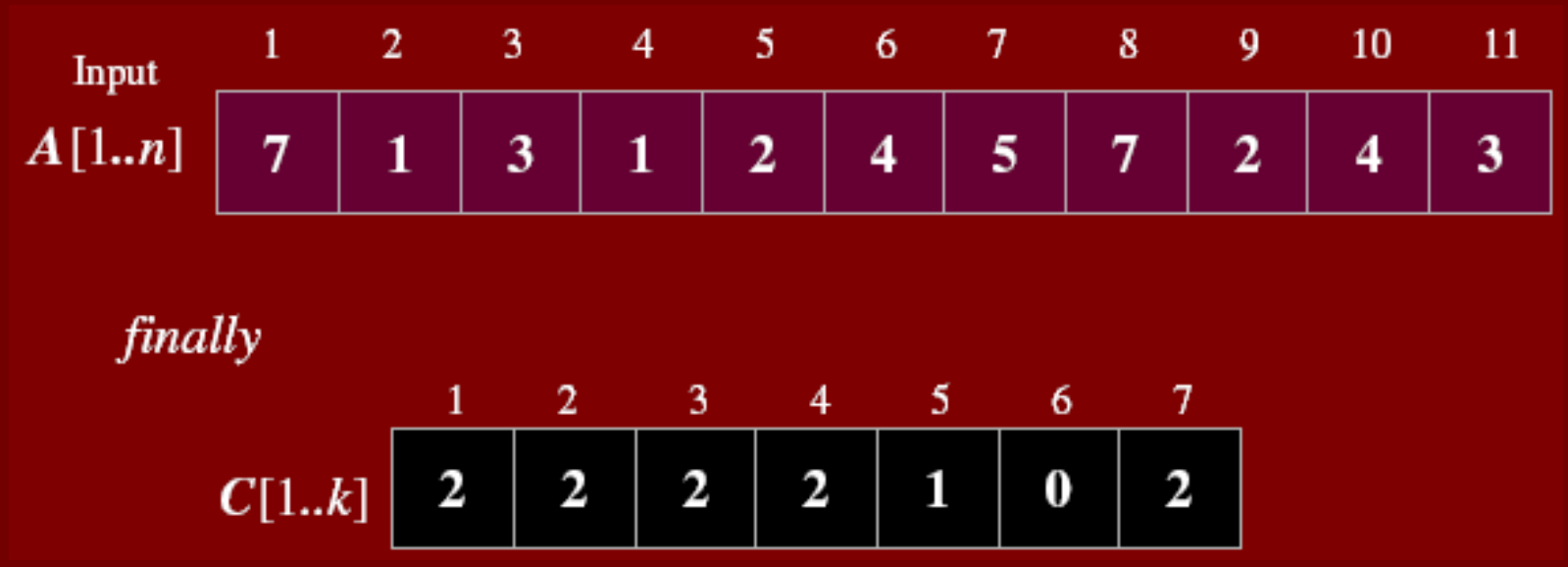

Input
A[1..n]
1 2 3 4 5 6 7 8 9 10 11
7 1 3 1 2 4 5 7 2 4 3
k
1 2 3 4 5 6 7
C[1..k]
0 0 0 0 0 0 1


Input
A[1..n]
1 2 3 4 5 6 7 8 9 10 11
7 1 3 1 2 4 5 7 2 4 3
k
1 2 3 4 5 6 7
C[1..k]
1 0 0 0 0 0 1


Input
A[1..n]
1 2 3 4 5 6 7 8 9 10 11
7 1 3 1 2 4 5 7 2 4 3
k
1 2 3 4 5 6 7
C[1..k]
1 0 1 0 0 0 1

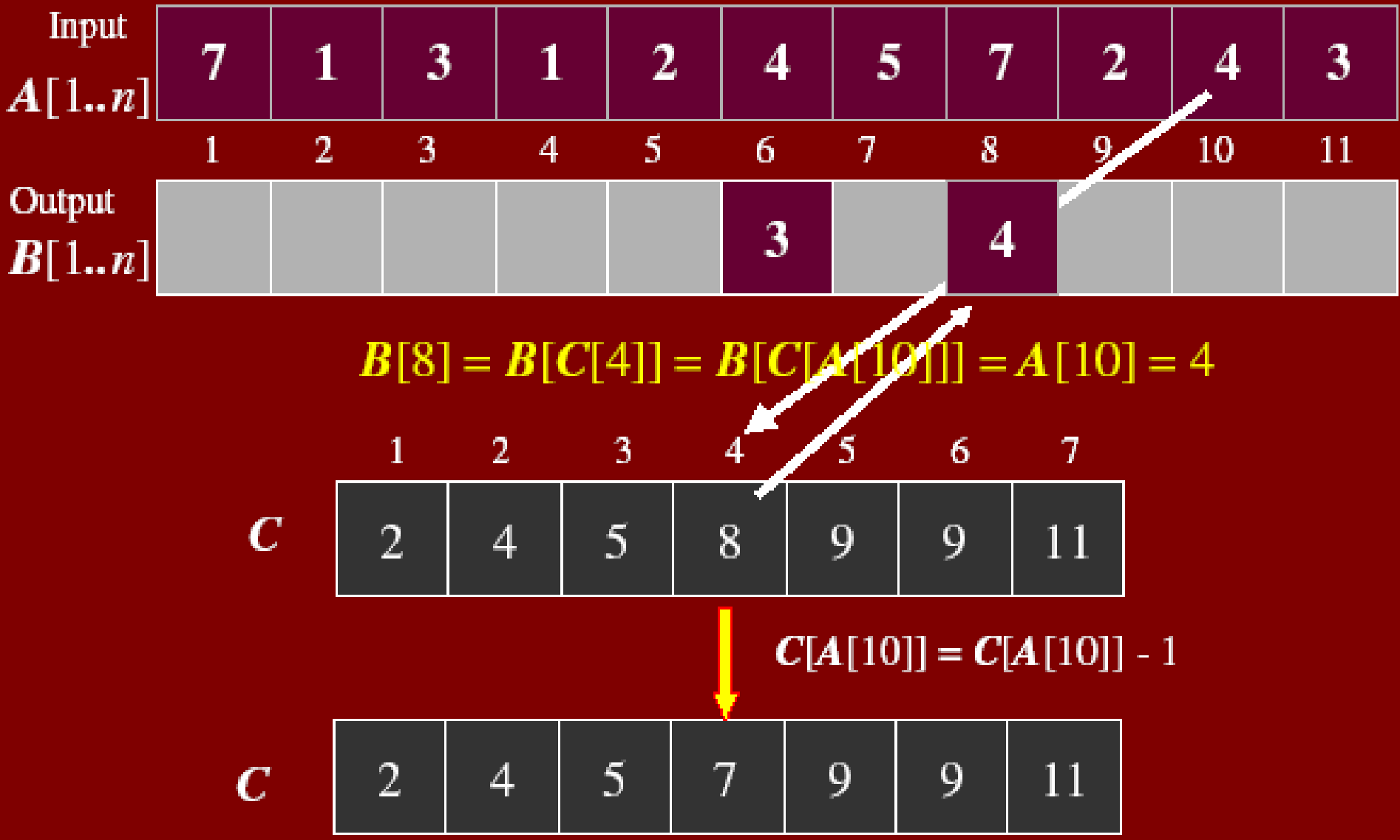

Input
A[1..n]
1 2 3 4 5 6 7 8 9 10 11
7 1 3 1 2 4 5 7 2 4 3
k
1 2 3 4 5 6 7
C[1..k]
2 0 1 0 0 0 1


Input
A[1..n]
1 2 3 4 5 6 7 8 9 10 11
7 1 3 1 2 4 5 7 2 4 3
k
1 2 3 4 5 6 7
C[1..k]
2 1 1 0 0 0 1


Input
A[1..n]
1 2 3 4 5 6 7 8 9 10 11
7 1 3 1 2 4 5 7 2 4 3
finally
1 2 3 4 5 6 7
C[1..k]
2 2 2 2 1 0 2

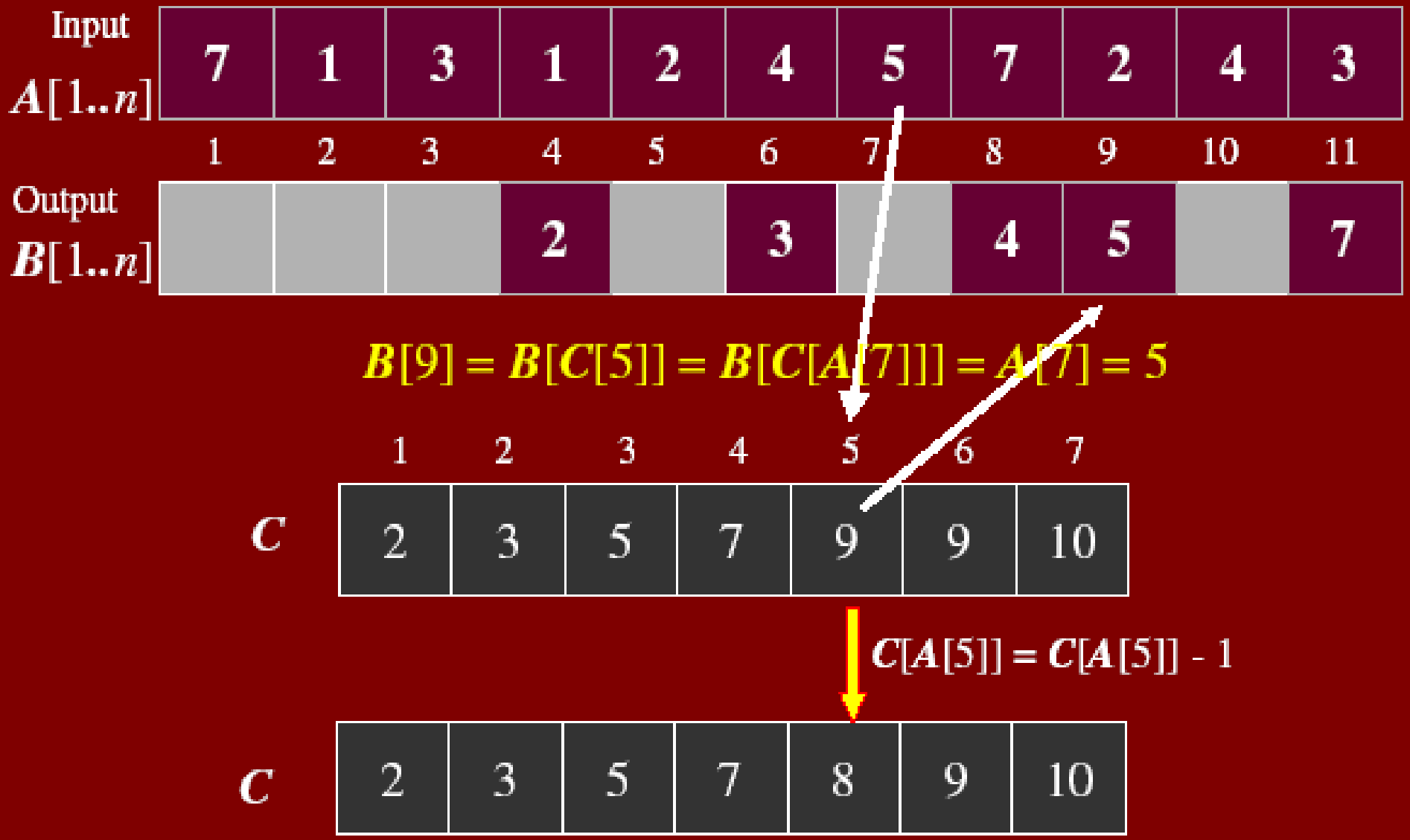

Input
A[1..n]
1 2 3 4 5 6 7 8 9 10 11
7 1 3 1 2 4 5 7 2 4 3
C[1..k]
1 2 3 4 5 6 7
2 2 2 2 1 0 2
for i = 2 to 7
do C[i] = C[i] + C[i-1]
C
1 2 3 4 5 6 7
2 4 6 8 9 9 11
6 elements <= 3


Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
3
B[6] = B[C[3]] = B[C[A[11]]] = A[11] = 3
1 2 3 4 5 6 7
C
2 4 6 8 9 9 11
C[A[11]] = C[A[11]] - 1
C
2 4 5 8 9 9 11

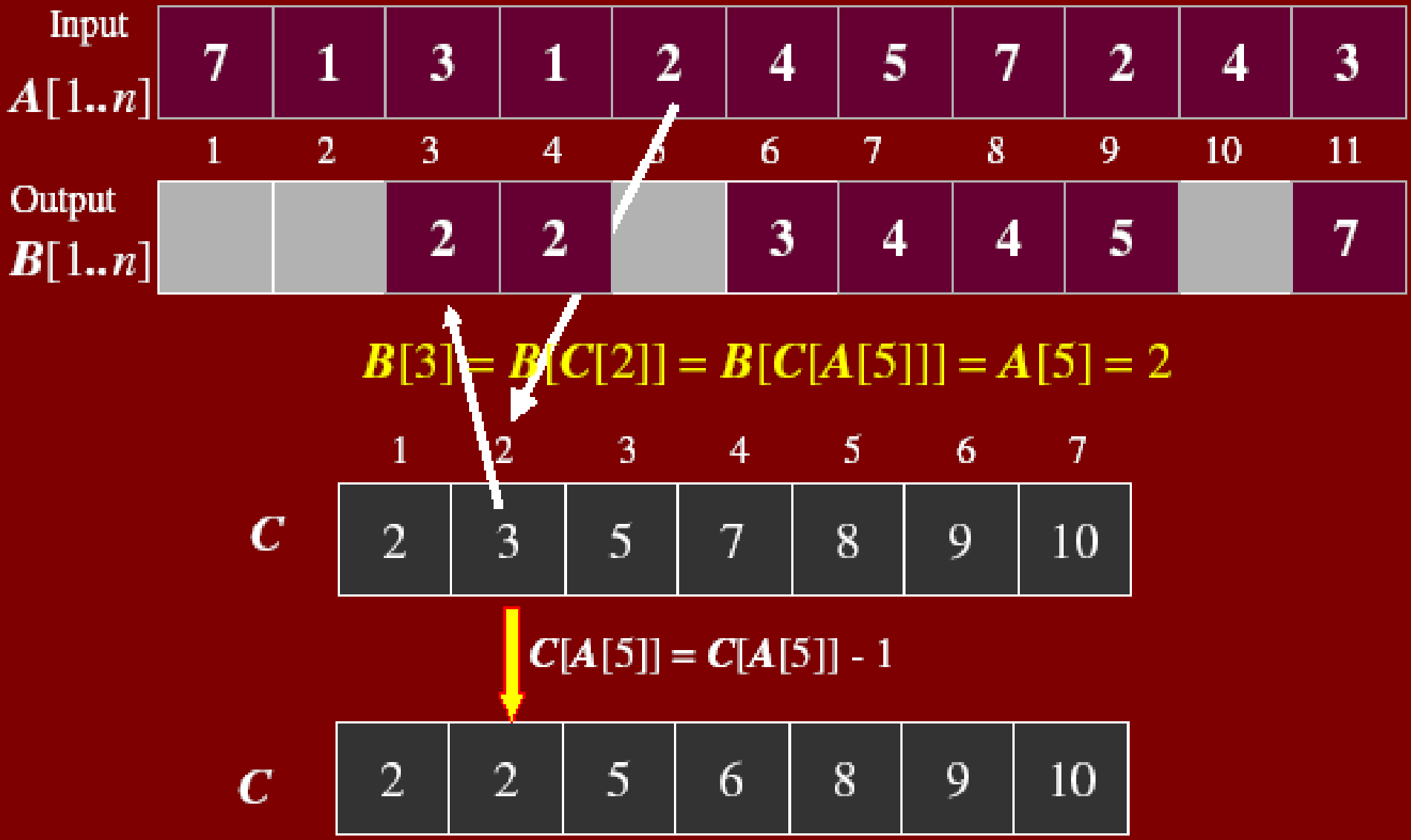

Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
3 4
B[8] = B[C[4]] = B[C[A[10]]] = A[10] = 4
1 2 3 4 5 6 7
C
2 4 5 8 9 9 11
C[A[10]] = C[A[10]] - 1
C
2 4 5 7 9 9 11


Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
2 3 4
B[4] = B[C[2]] = B[C[A[9]]] = A[9] = 2
1 2 3 4 5 6 7
C
2 4 5 7 9 9 11
C[A[9]] = C[A[9]] - 1
C
2 3 5 7 9 9 11


Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
2 3 4 7
B[11] = B[C[7]] = B[C[A[8]]] = A[8] = 7
1 2 3 4 5 6 7
C
2 3 5 7 9 9 11
C[A[8]] = C[A[8]] - 1
C
2 3 5 7 9 9 10

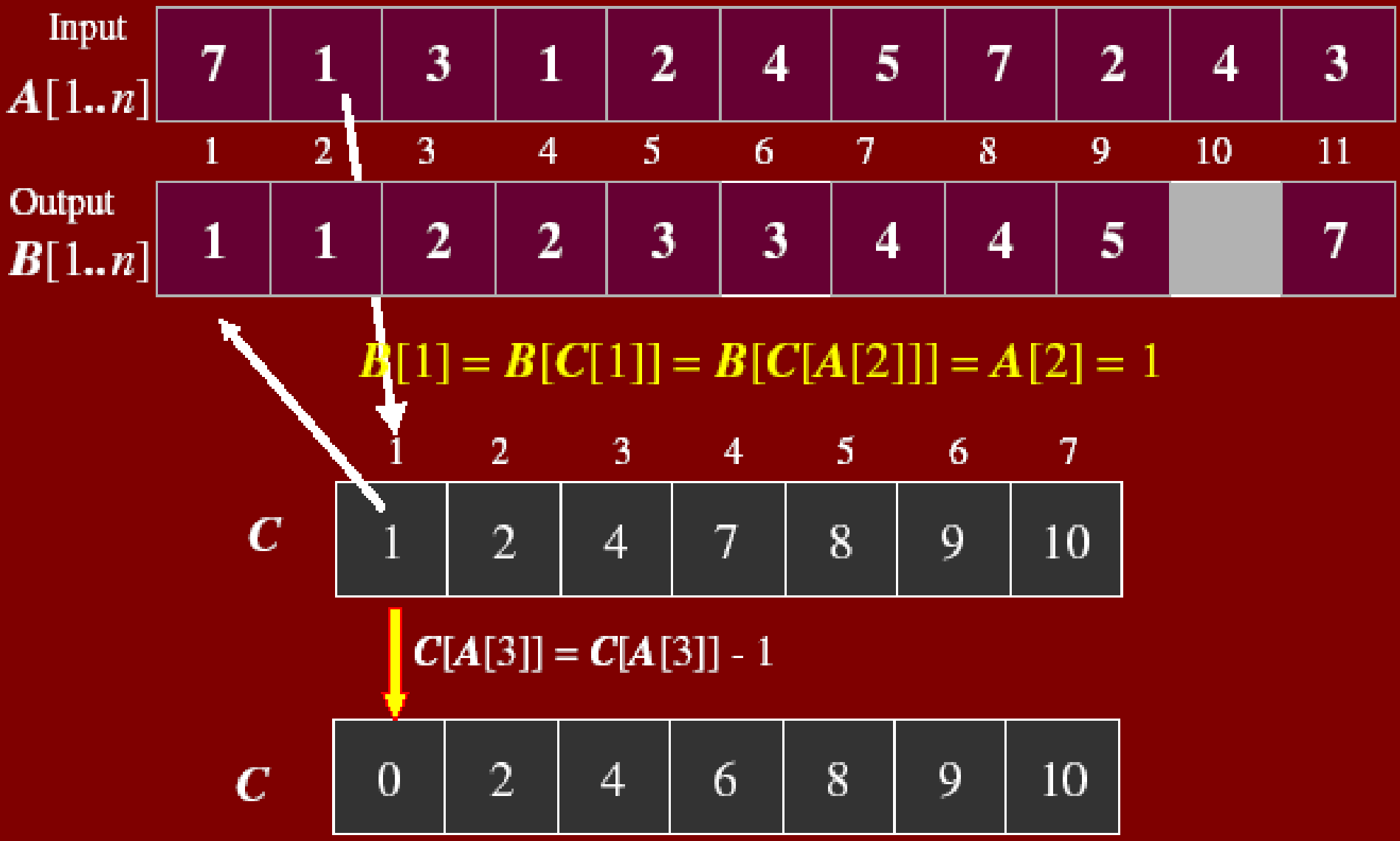

Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
2 3 4 5 7
B[9] = B[C[5]] = B[C[A[7]]] = A[7] = 5
1 2 3 4 5 6 7
C
2 3 5 7 9 9 10
C[A[5]] = C[A[5]] - 1
C
2 3 5 7 8 9 10

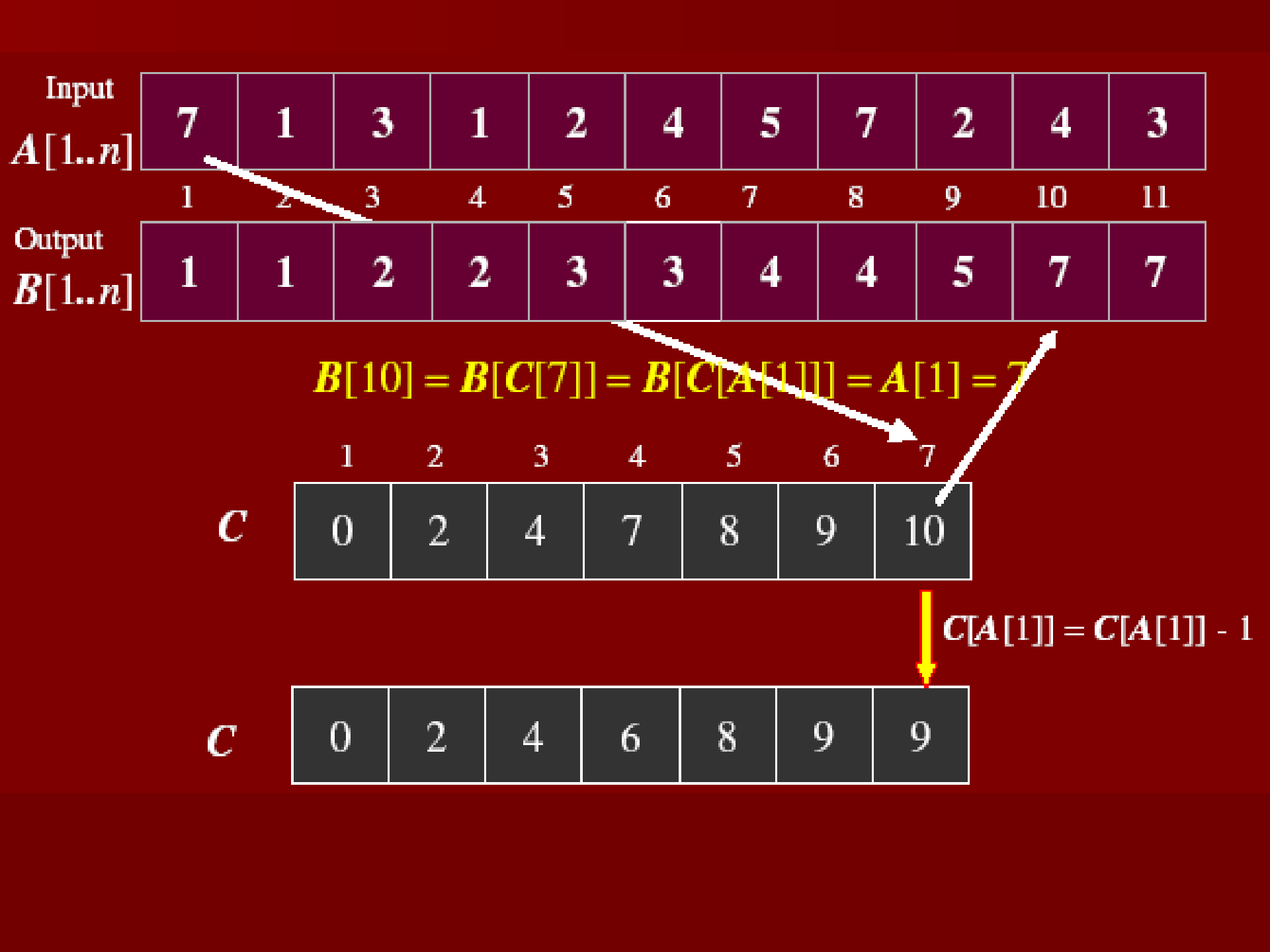

Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
2 3 4 4 5 7
B[7] = B[C[4]] = B[C[A[6]]] = A[6] = 4
1 2 3 4 5 6 7
C
2 3 5 7 8 9 10
C[A[6]] = C[A[6]] - 1
C
2 3 5 6 8 9 10


Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
2 2 3 4 4 5 7
B[3] = B[C[2]] = B[C[A[5]]] = A[5] = 2
1 2 3 4 5 6 7
C
2 3 5 7 8 9 10
C[A[5]] = C[A[5]] - 1
C
2 2 5 6 8 9 10


Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
1 2 2 3 4 4 5 7
B[2] = B[C[1]] = B[C[A[4]]] = A[4] = 1
1 2 3 4 5 6 7
C
2 2 5 7 8 9 10
C[A[4]] = C[A[4]] - 1
C
1 2 5 6 8 9 10


Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
1 2 2 3 3 4 4 5 7
B[5] = B[C[3]] = B[C[A[3]]] = A[3] = 3
1 2 3 4 5 6 7
C
1 2 5 7 8 9 10
C[A[3]] = C[A[3]] - 1
C
1 2 4 6 8 9 10


Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
1 1 2 2 3 3 4 4 5 7
B[1] = B[C[1]] = B[C[A[2]]] = A[2] = 1
1 2 3 4 5 6 7
C
1 2 4 7 8 9 10
C[A[3]] = C[A[3]] - 1
C
0 2 4 6 8 9 10


Input
A[1..n]
7 1 3 1 2 4 5 7 2 4 3
1 2 3 4 5 6 7 8 9 10 11
Output
B[1..n]
1 1 2 2 3 3 4 4 5 7 7
B[10] = B[C[7]] = B[C[A[1]]] = A[1] = 7
1 2 3 4 5 6 7
C
0 2 4 7 8 9 10
C[A[1]] = C[A[1]] - 1
C
0 2 4 6 8 9 9

Counting Sort Algorithm

```
COUNTING-SORT( array A, array B, int k)
 1   for i ← 1 to k
 2   do C[i] ← 0                                  k times
 3   for j ← 1 to length[A]
 4   do C[A[j]] ← C[A[j]] + 1                     n times
 5   // C[i] now contains the number of elements = i
 6   for i ← 2 to k
 7   do C[i] ← C[i] + C[i − 1]                    k times
 8   // C[i] now contains the number of elements ≤ i
 9   for j ← length[A] downto 1
10   do B[C[A[j]]] ← A[j]
11      C[A[j]] ← C[A[j]] − 1                     n times
```

- There are four (unnested) loops, executed k times, n times, k - 1 times, and n times, respectively,
- so the total running time is **Θ(n + k)** time.
- If k = Θ(n), then the total running time is **Θ(n)**.