Data Structure and Algorithms

Session-22

Dr. Subhra Rani Patra
SCOPE, VIT Chennai

Counting sort

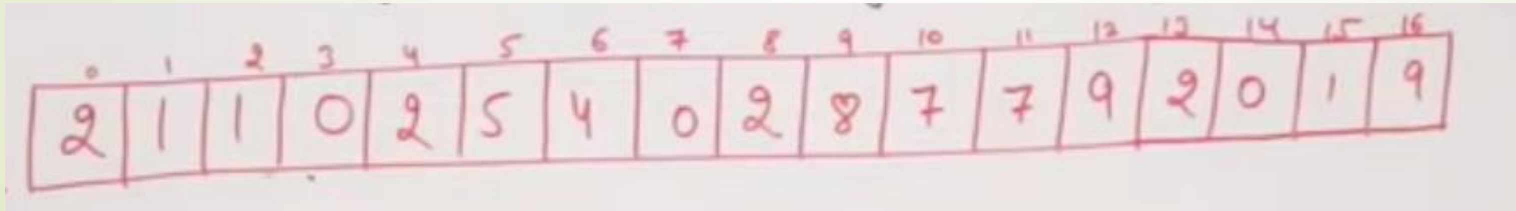
- Counting sort assumes that each of the n input elements is an integer in the range 0 to k. that is n is the number of elements and k is the highest value element.
- Consider the input set : 4, 1, 3, 4, 3. Then n=5 and k=4

- The algorithm uses three array:

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

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

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



Counting Sort

```
Counting-Sort(A, B, k)
Let C[0.....k] be a new array
for i=0 to k
      C[i]= 0;
for j=1 to A.length  or n
      C[ A[j] ] = C[ A[j] ]  + 1;
for i=1 to k
      C[i] = C[i] + C[i-1];
for j=n or A.length down to 1
      B[  C[ A[j] ]  ] = A[j];
       C[ A[j] ] = C[ A[j] ]  - 1;
```

Counting Sort

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

Counting-sort example

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

	0	1	2	3	4	5
C:						

	1	2	3	4	5	6	7	8
B:								

Executing Loop 1

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

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

	0	1	2	3	4	5
C:	0	0	0	0	0	0

	1	2	3	4	5	6	7	8
B:								

Executing Loop 2

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

A:

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

C:

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

B:

1	2	3	4	5	6	7	8

Executing Loop 2

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

A:

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

C:

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

B:

1	2	3	4	5	6	7	8

Executing Loop 2

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

A:

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

C:

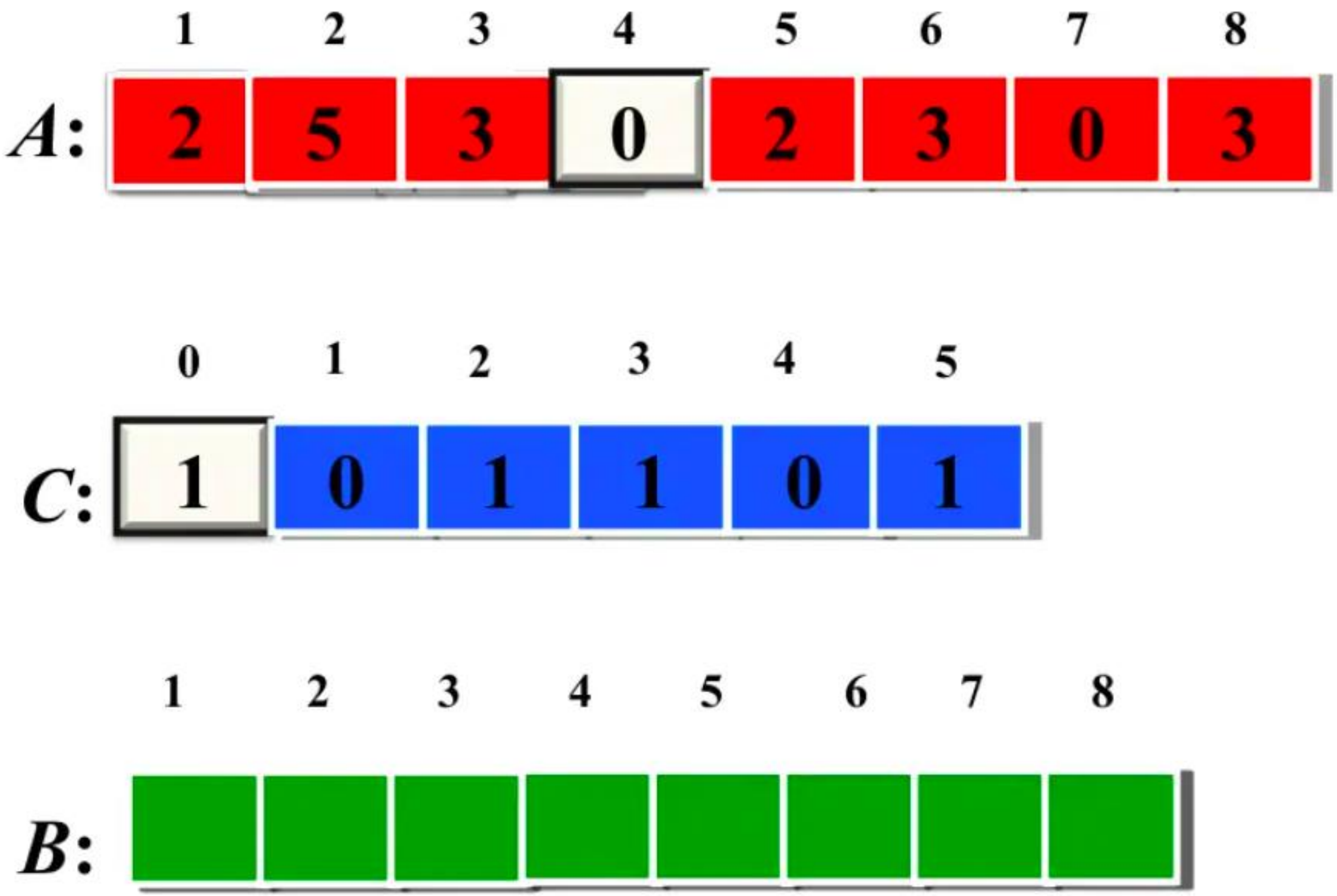
0	1	2	3	4	5
0	0	1	1	0	1

B:

1	2	3	4	5	6	7	8

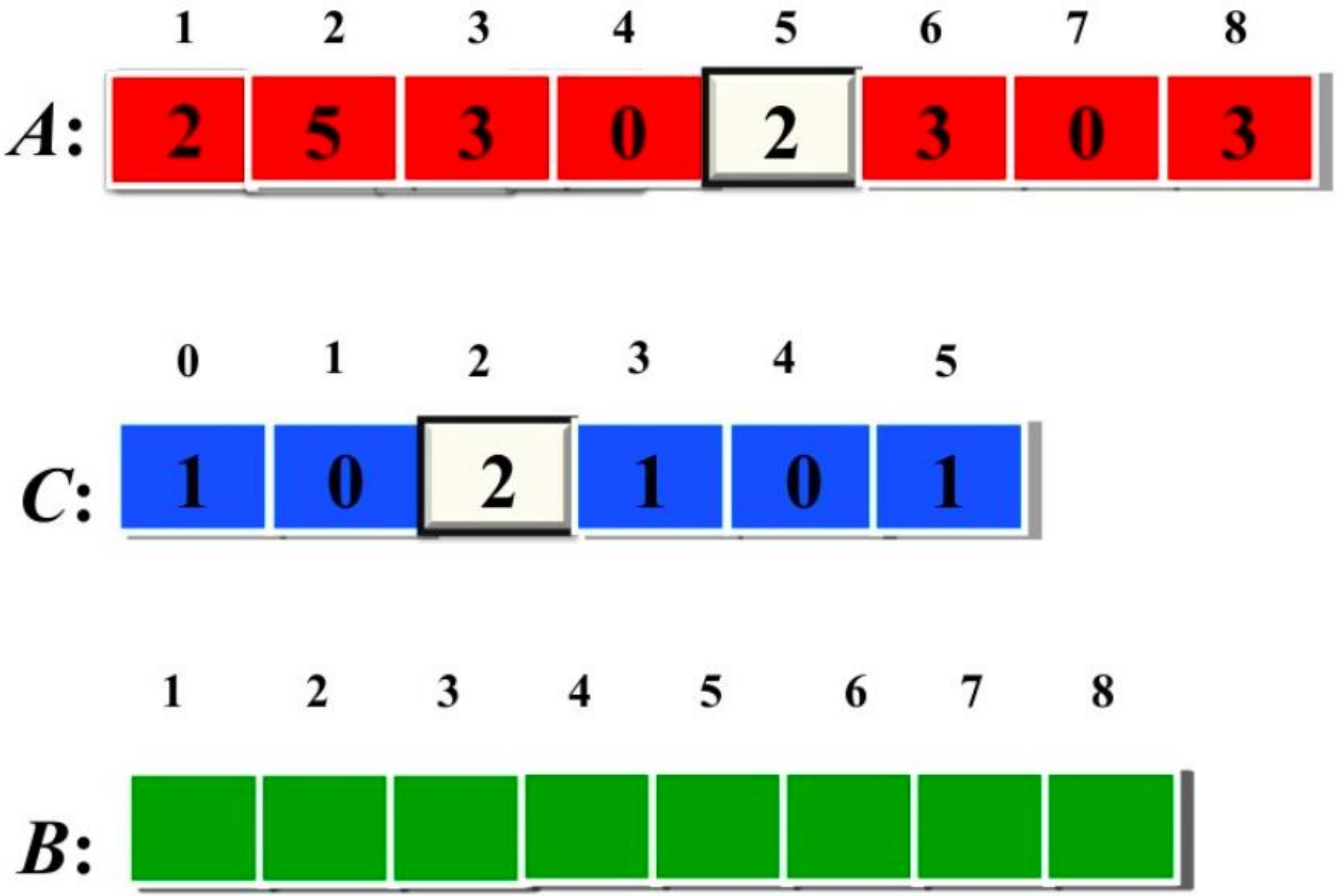
Executing Loop 2

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



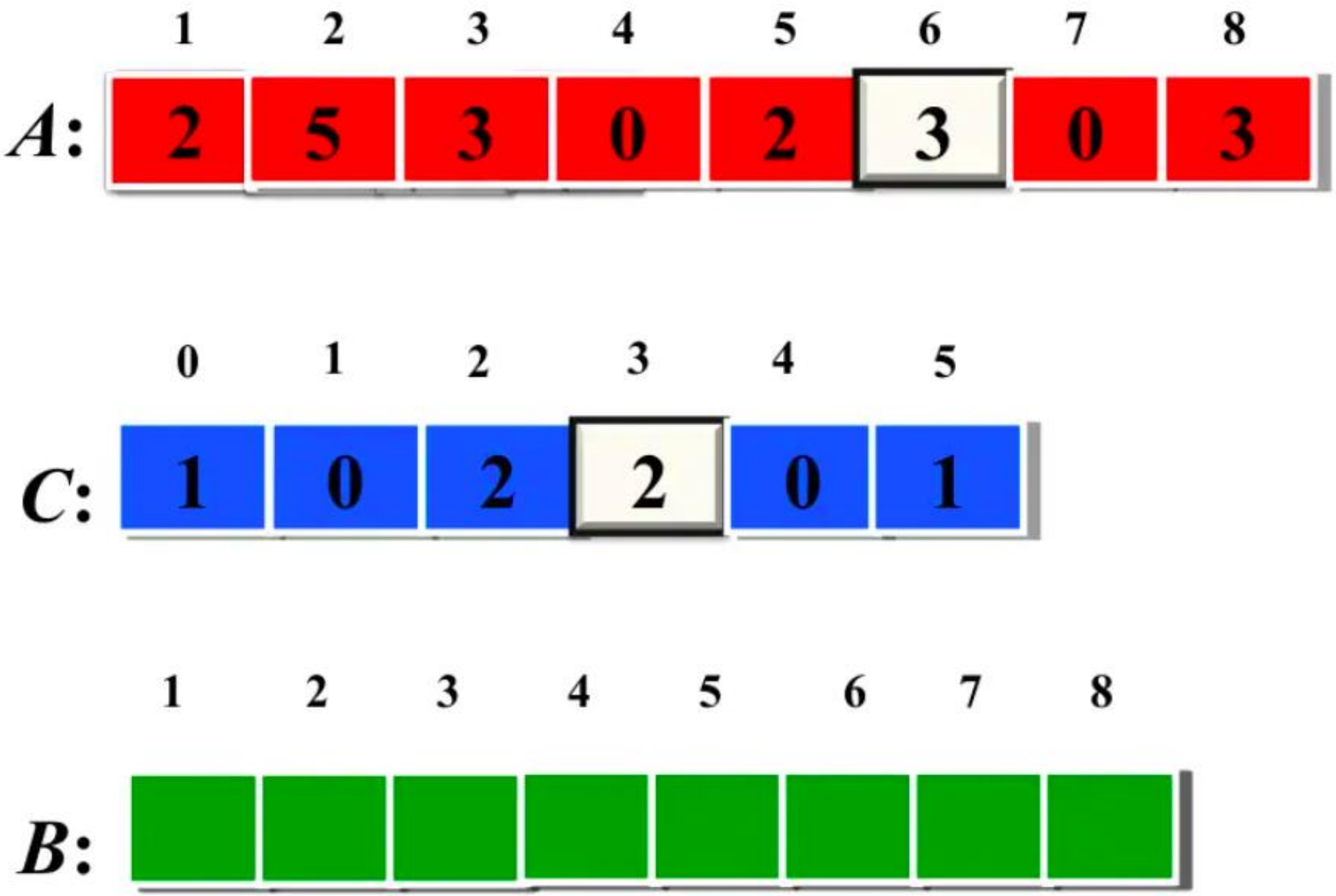
Executing Loop 2

```
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;
```

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

	0	1	2	3	4	5
C:	2	0	2	2	0	1

	1	2	3	4	5	6	7	8
B:								

Executing Loop 2

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

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

	0	1	2	3	4	5
C:	2	0	2	3	0	1

	1	2	3	4	5	6	7	8
B:								

End of Loop 2

A:

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

C:

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

B:

1	2	3	4	5	6	7	8

Executing Loop 3

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

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

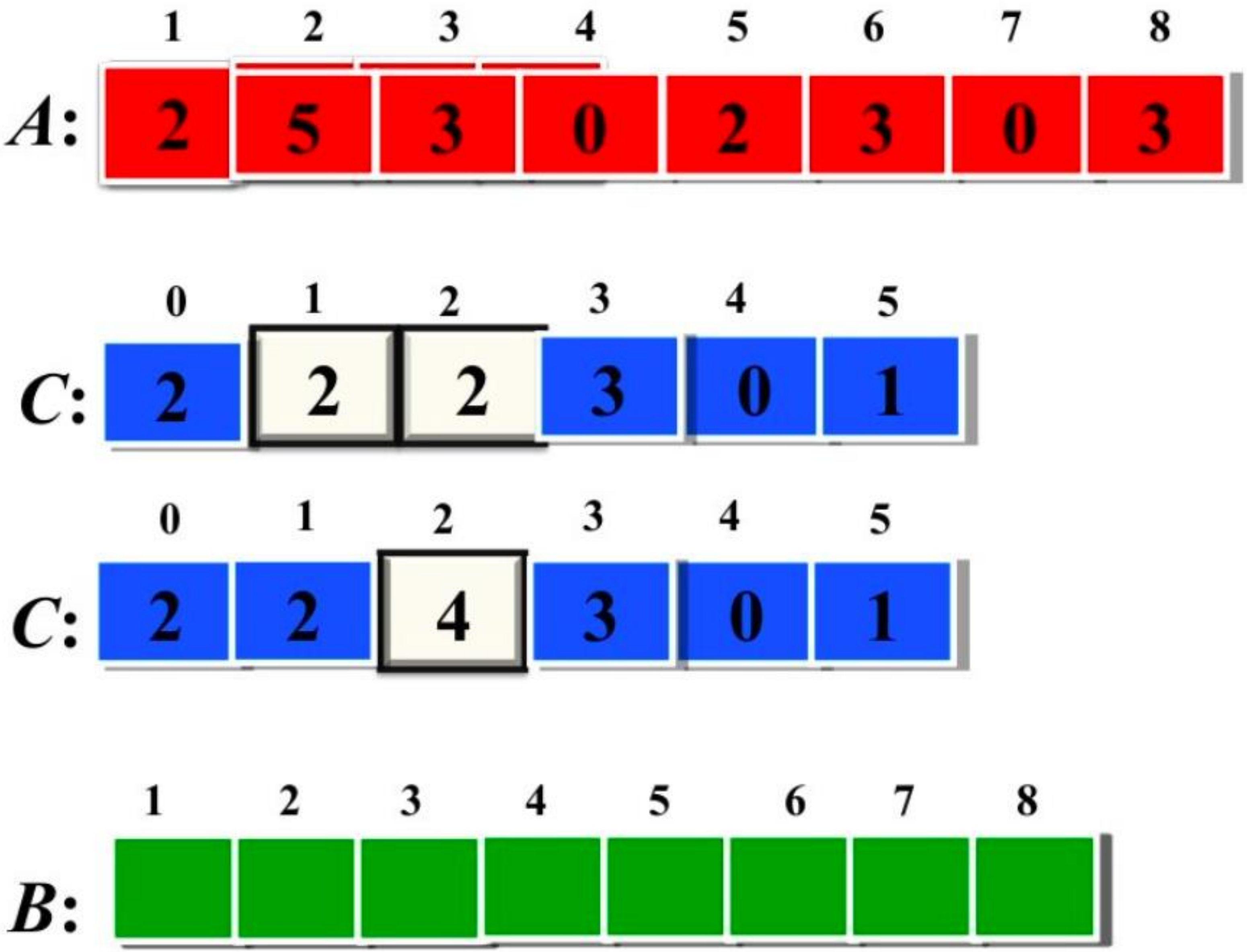
	0	1	2	3	4	5
C:	2	0	2	3	0	1

	0	1	2	3	4	5
C:	2	2	2	3	0	1

	1	2	3	4	5	6	7	8
B:								

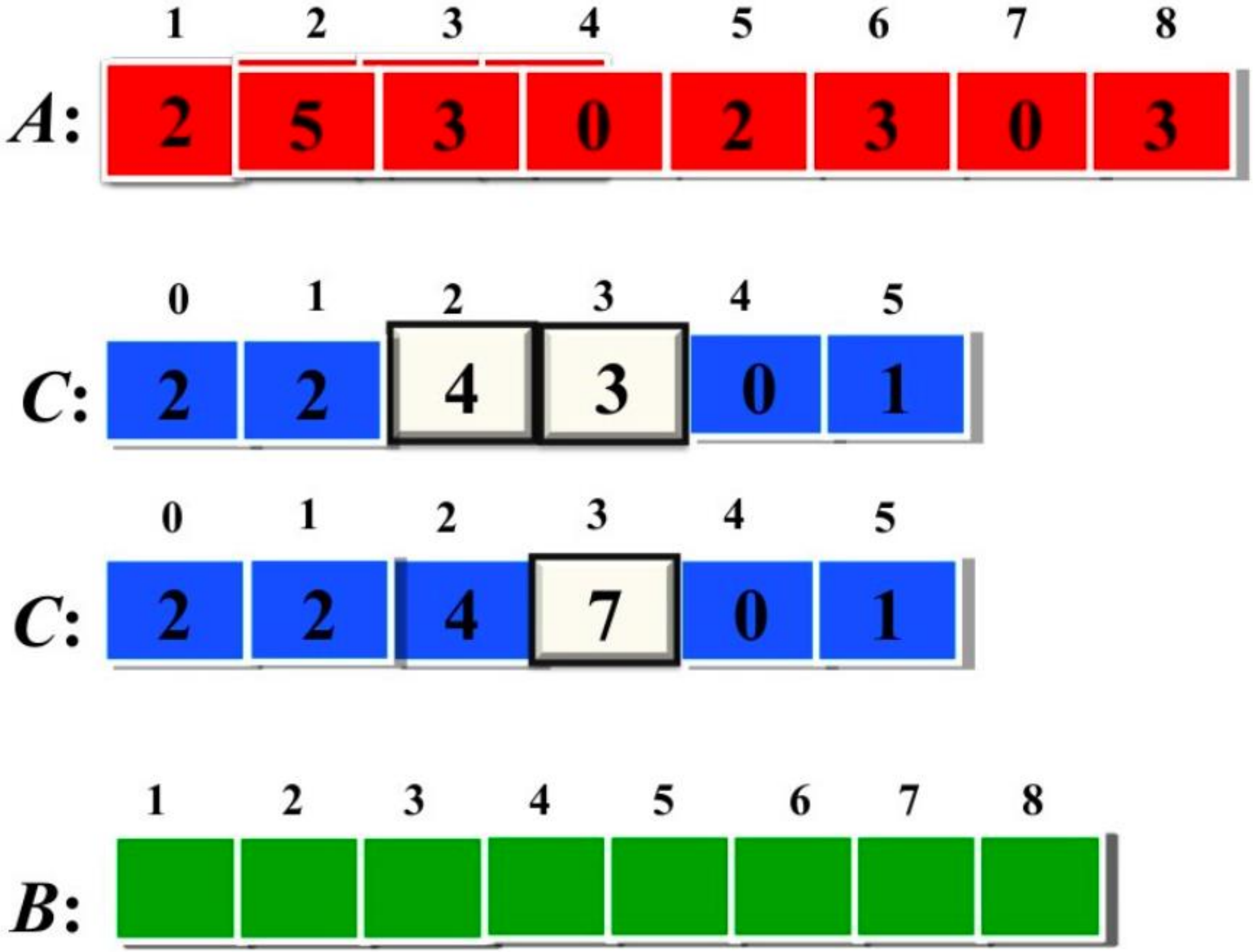
Executing Loop 3

```
for i=1 to k
        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];
```

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

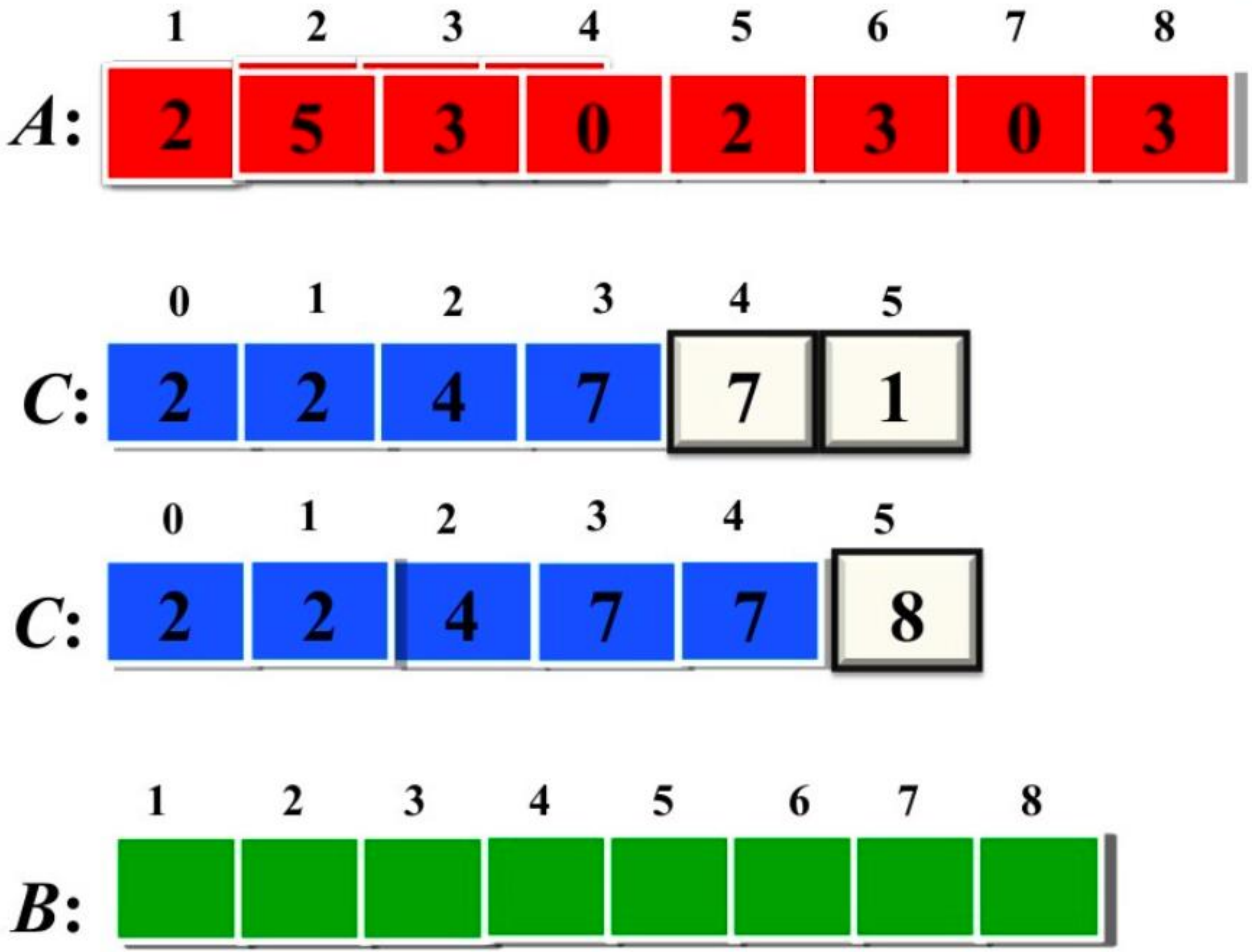
	0	1	2	3	4	5
C:	2	2	4	7	0	1

	0	1	2	3	4	5
C:	2	2	4	7	7	1

	1	2	3	4	5	6	7	8
B:								

Executing Loop 3

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



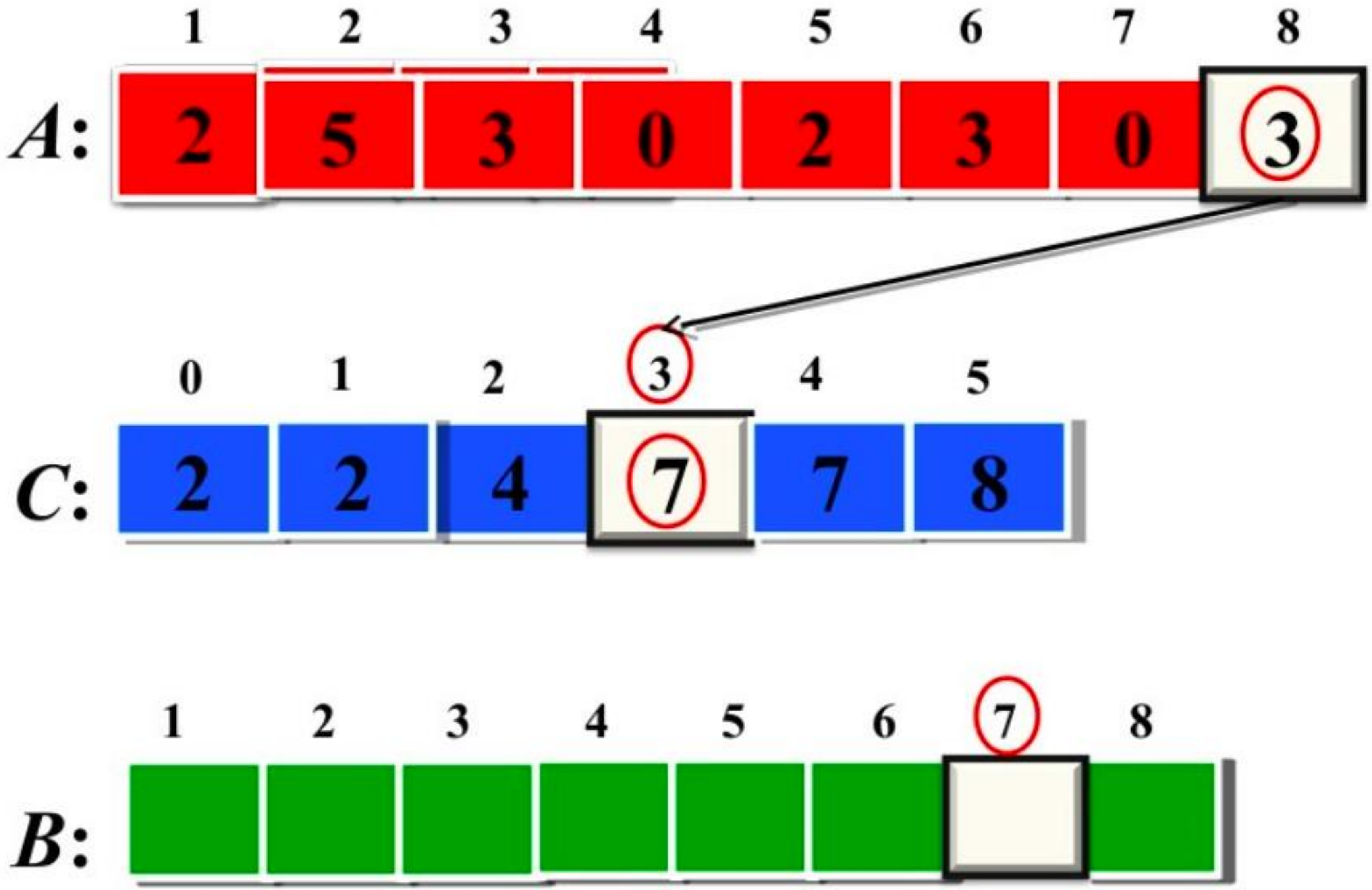
End of Loop 3

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

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

	1	2	3	4	5	6	7	8
B:								

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;
```

A:

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

C:

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

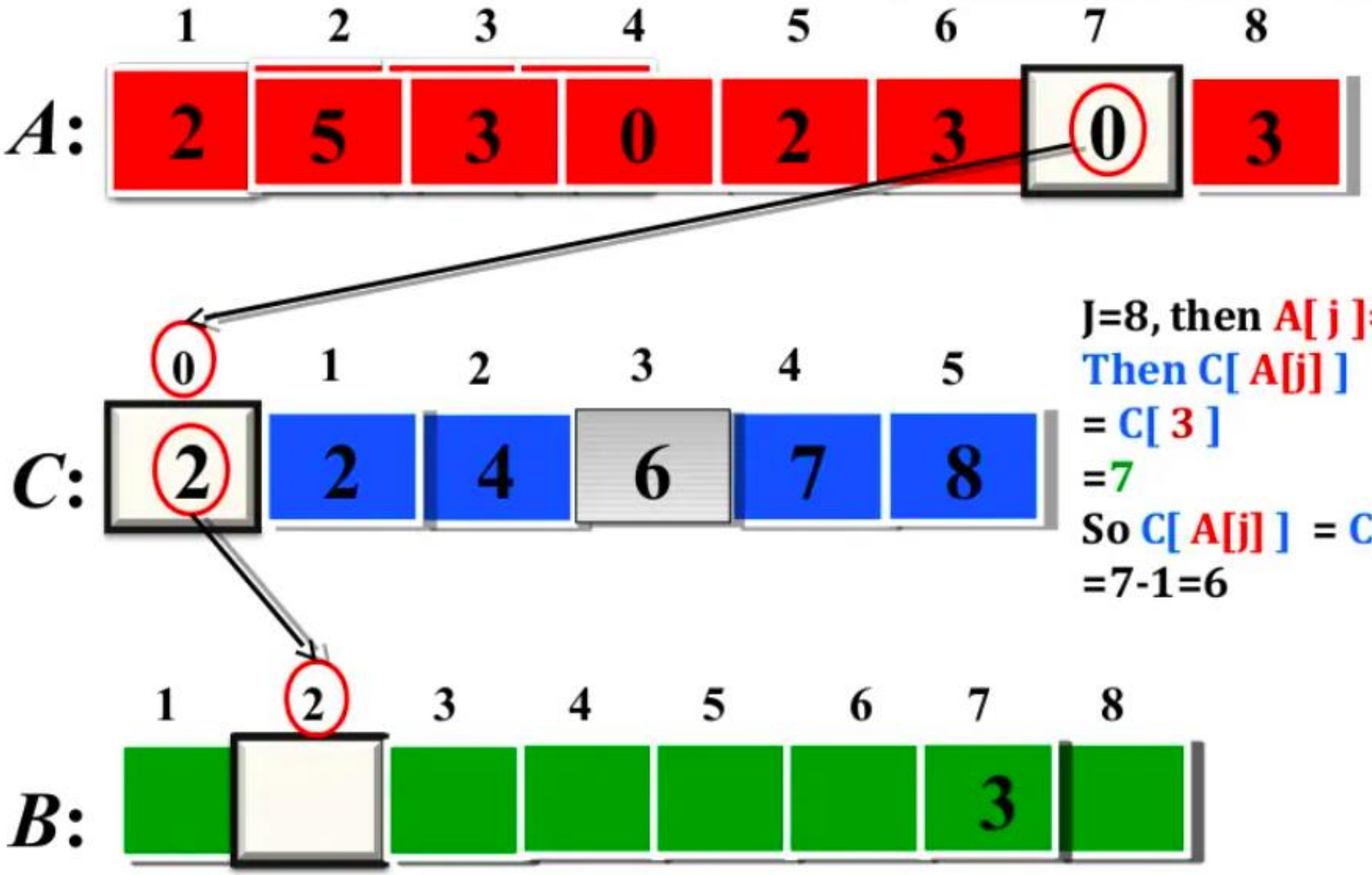
B:

1	2	3	4	5	6	7	8

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

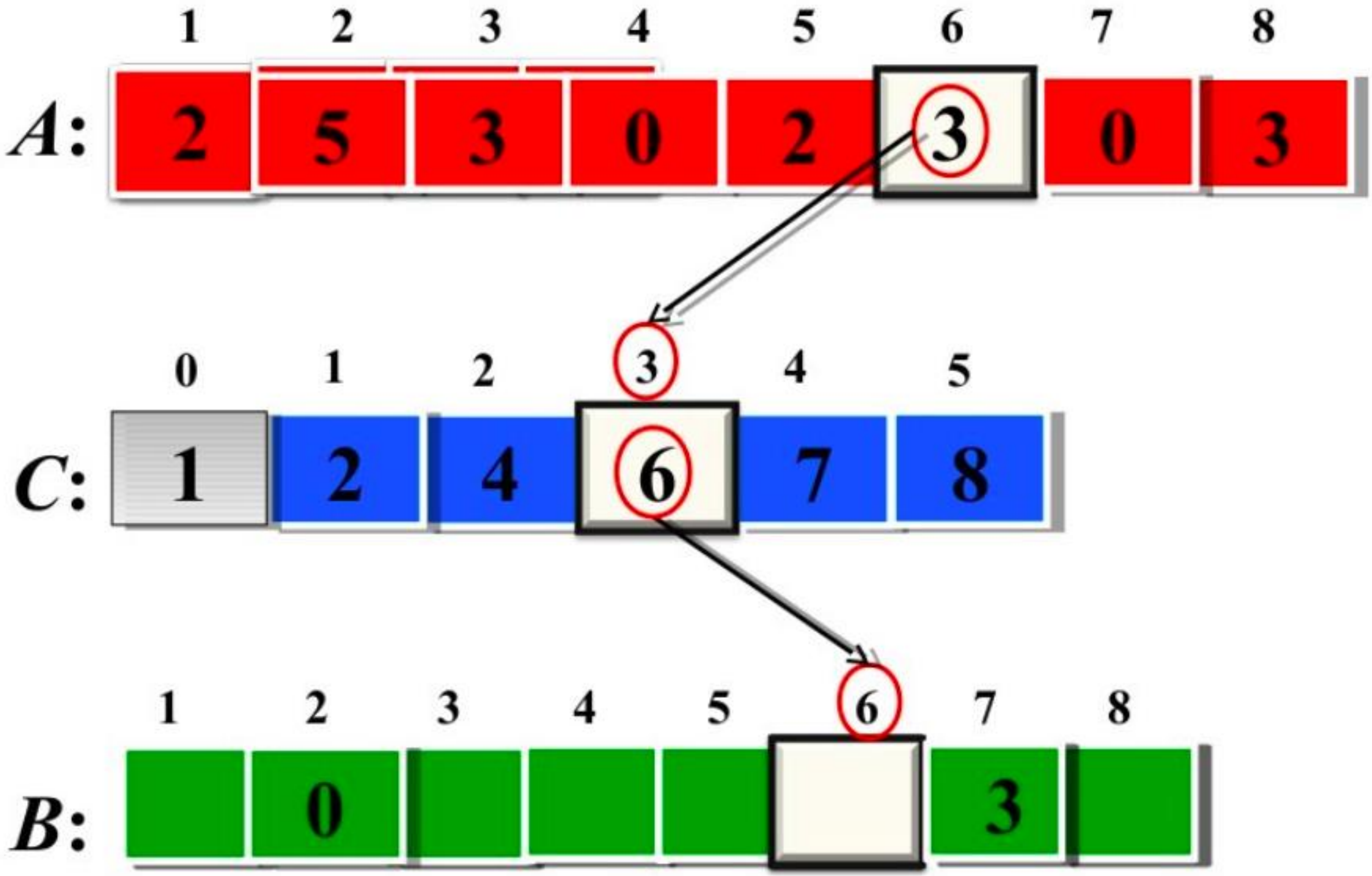
Executing Loop 4

```
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;
```

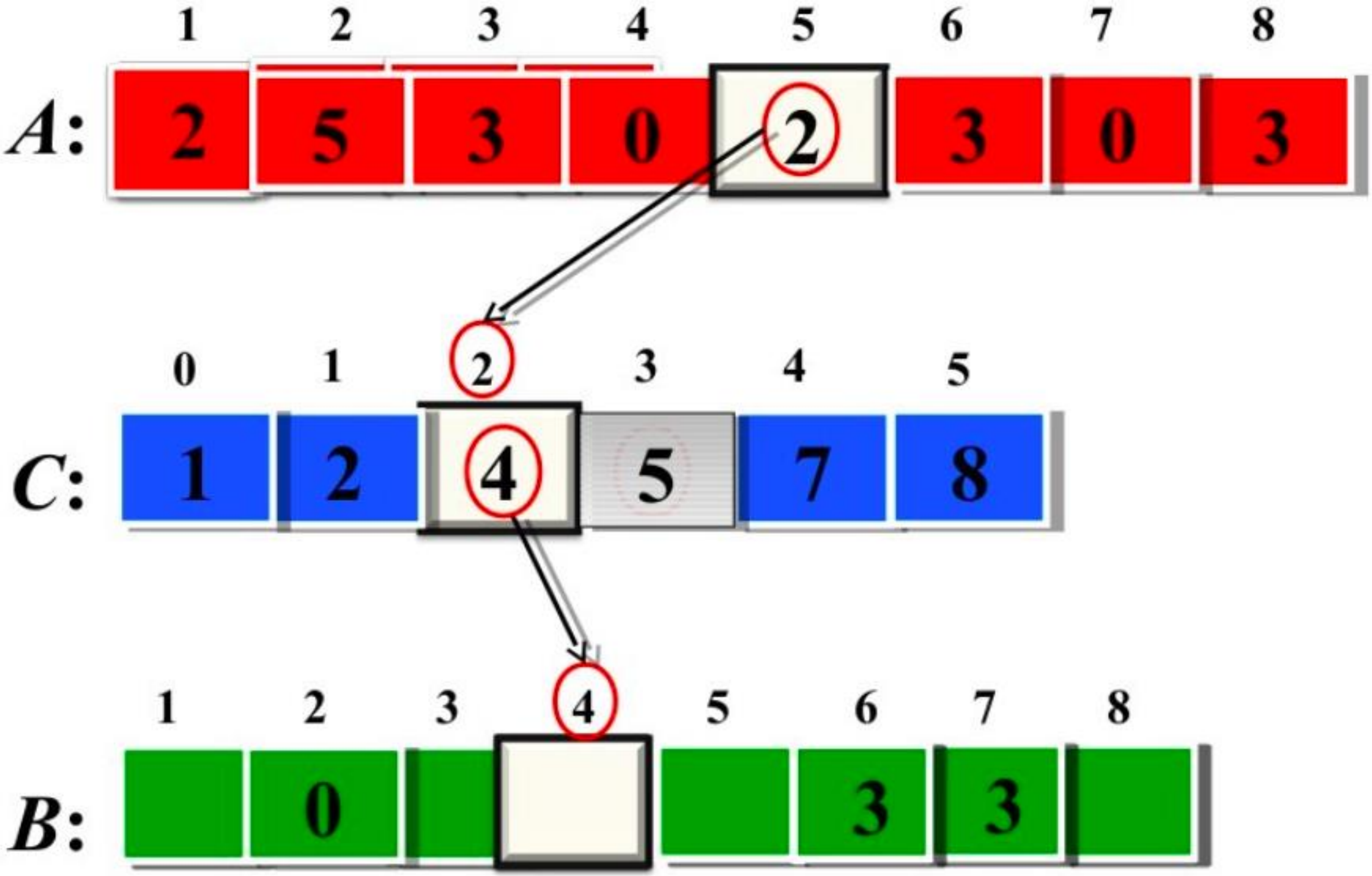


J=8, then A[j]=A[8]=3
Then C[A[j]]
= C[3]
=7
So C[A[j]] = C[A[j]] -1
=7-1=6

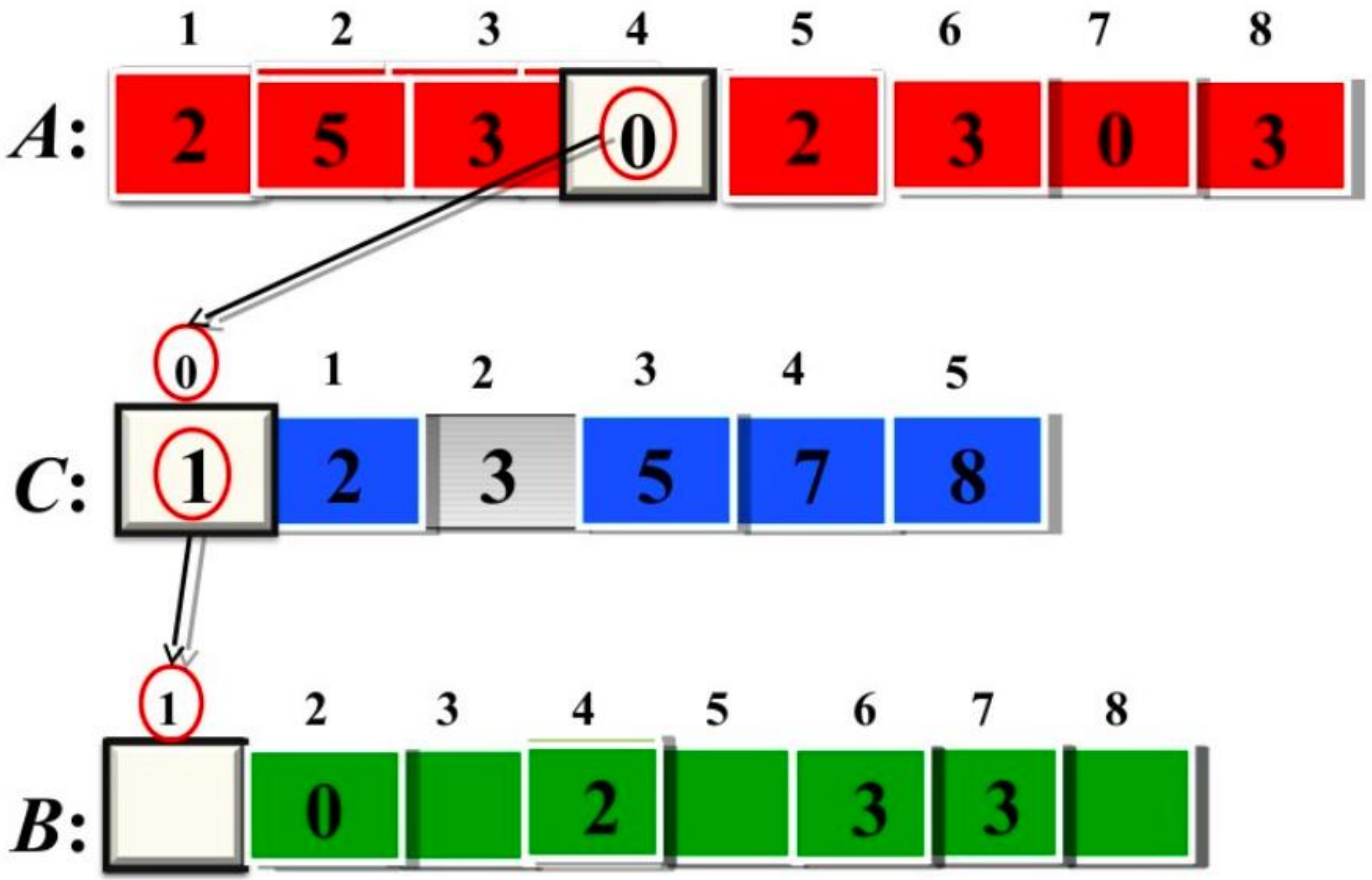
Executing Loop 4



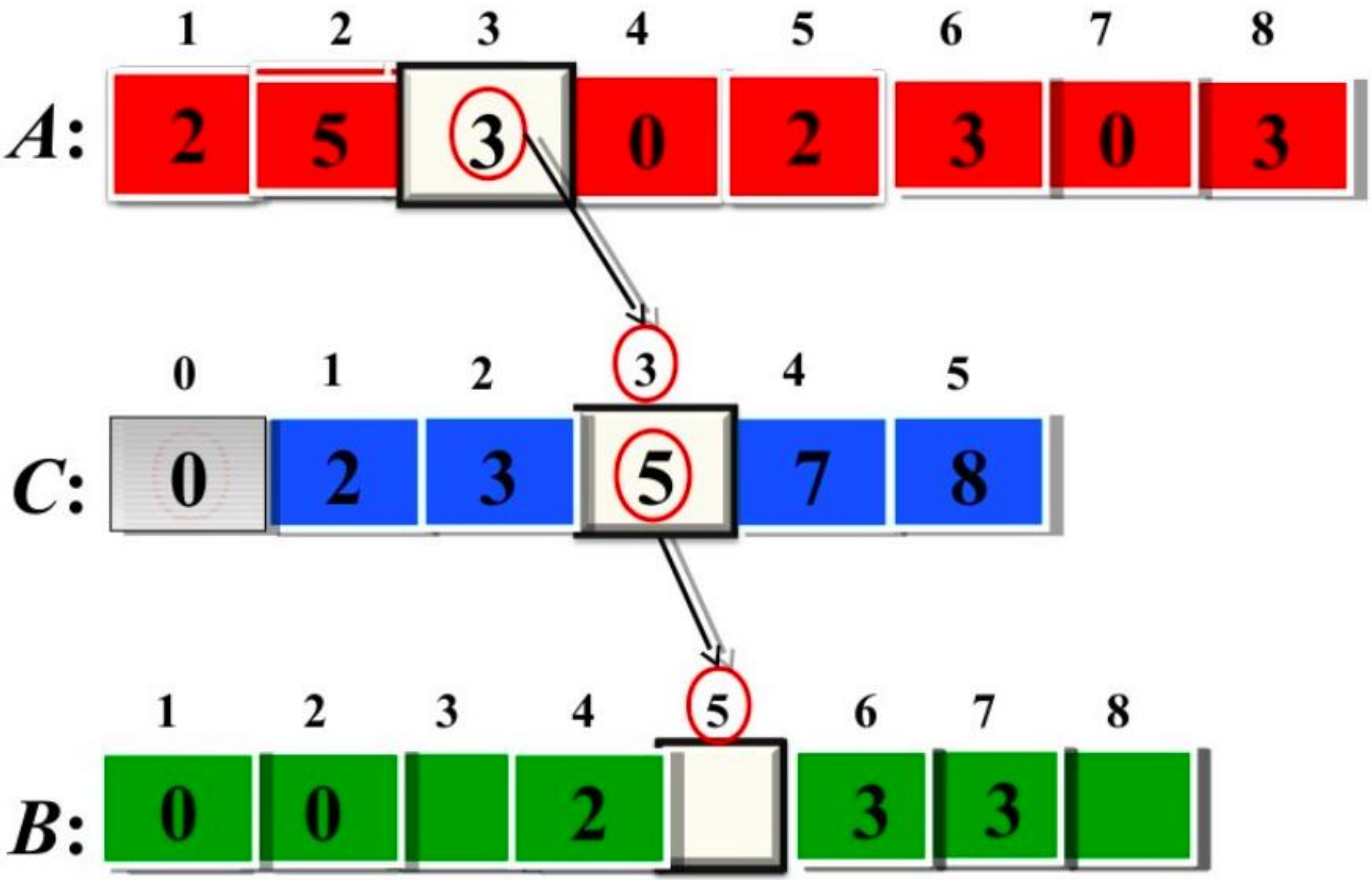
Executing Loop 4



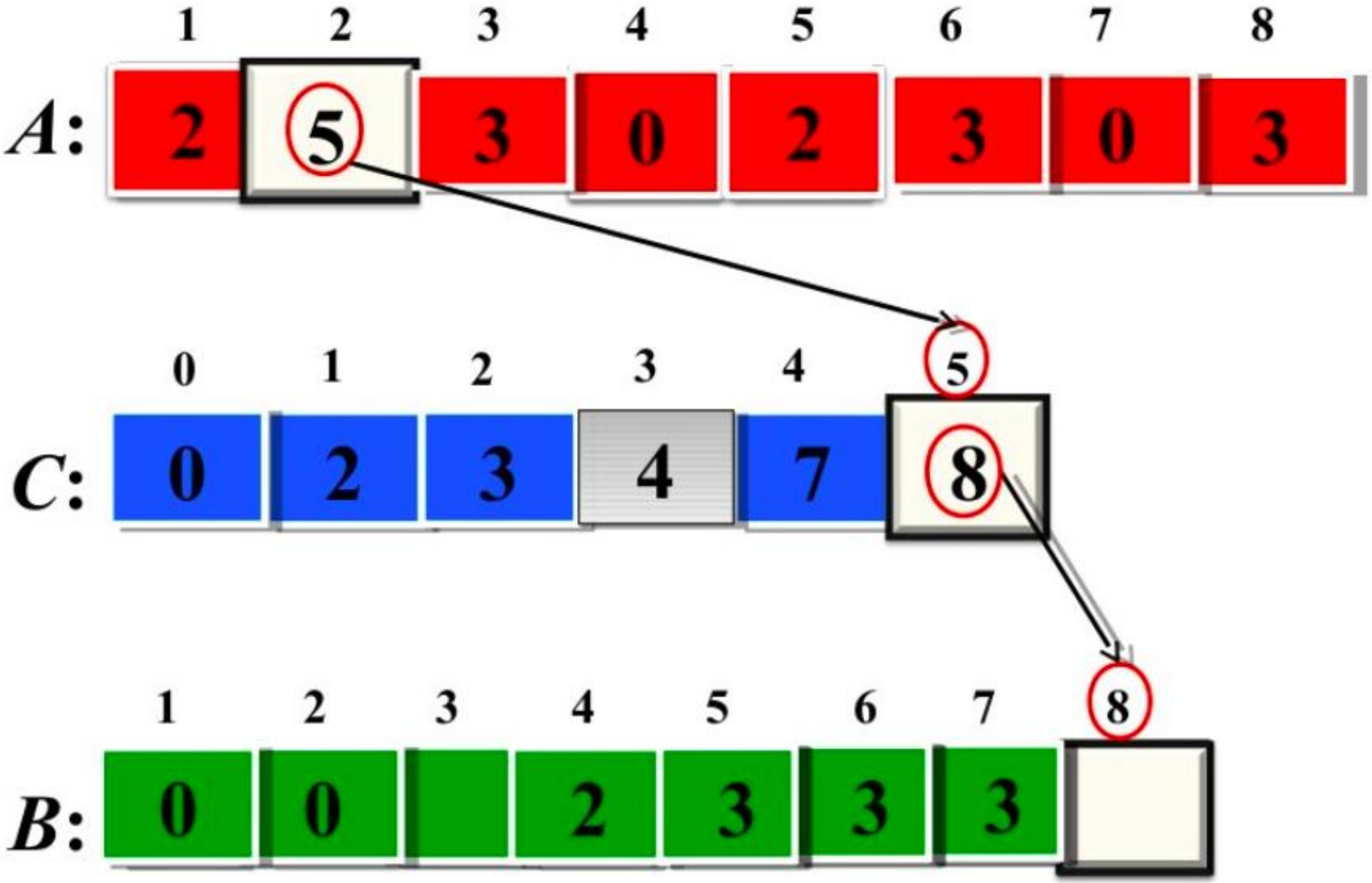
Executing Loop 4



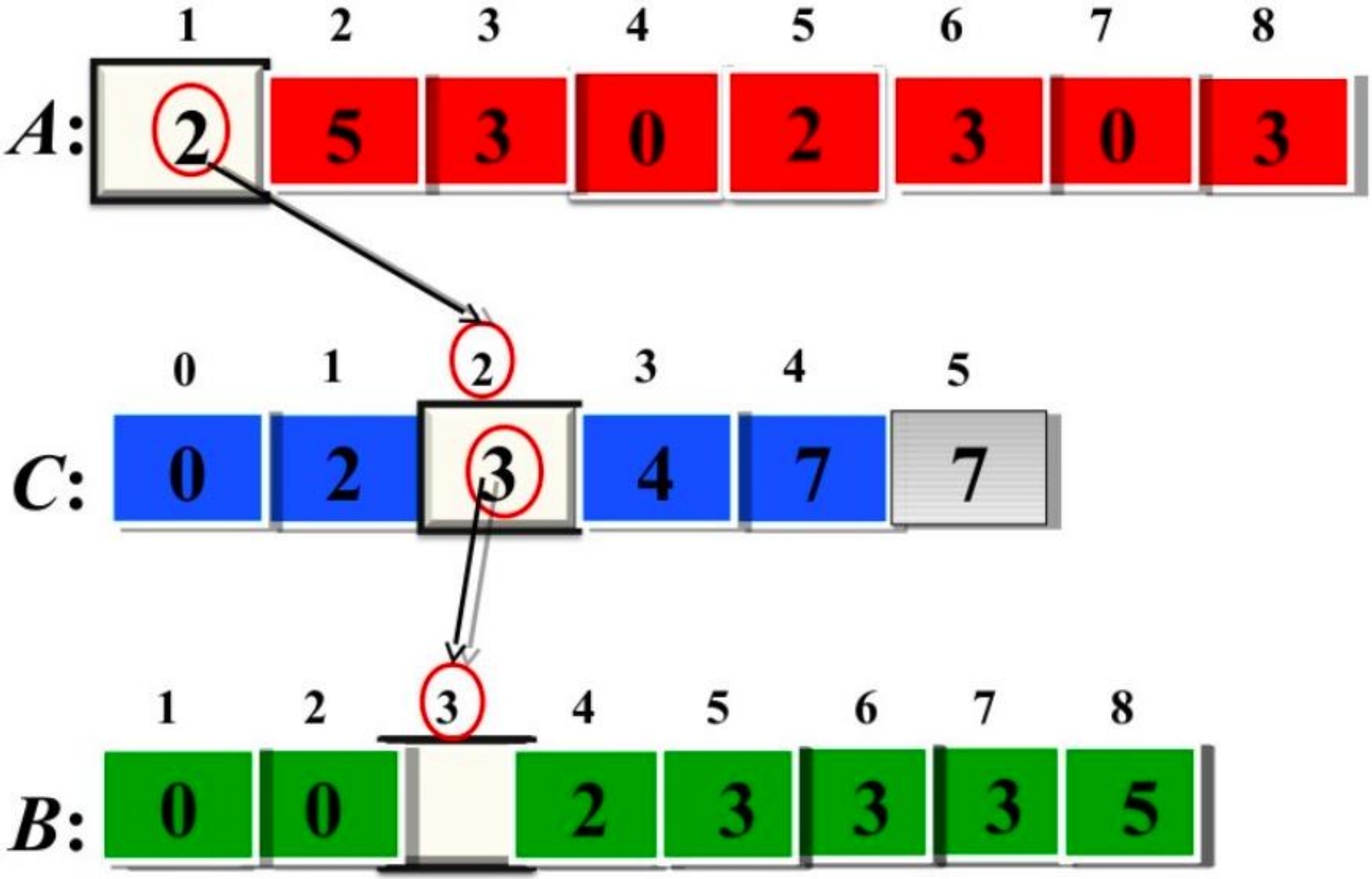
Executing Loop 4



Executing Loop 4



Executing Loop 4



End of Loop 4

A:

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

C:

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

B:

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

Sorted data in Array B

Time Complexity Analysis

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

Loop 1 and 3 takes O(k) time

Loop 2 and 4 takes O(n) time

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.

Longest Common Subsequence (LCS):

✓ **Problem Statement:**

- We are given two strings 's1' and 's2'.
- We need to find the length of the longest subsequence which is common in both the strings.
- subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements

✓ **Example 1:**

- s1 = "elephant"
- s2 = "eretpat"
- Output: 5
- Explanation: The longest substring is "eepat".

✓ **Example 2:**

- s1 = "houdini "
- s2 = "hdupti"
- Output: 3
- Explanation: The longest substring is "hui".

1. 1 + F(2,8 , 2,7)
2. 0 + F(3,8 , 2,7)
   0 + F(2,8 , 3,7)

Longest Common Subsequence (Divide & Conquer):

```
int findLCSLengthAux(String s1, String s2, int i1, int i2) {
    if (i1 == s1.length() || i2 == s2.length())//Base Case
        return 0;
    int c3 = 0;

    if (s1.charAt(i1) == s2.charAt(i2)) {//If current character in both the string matches, then increase the index by 1 in both the strings.
        c3 = 1 + findLCSLengthAux(s1, s2, i1 + 1, i2 + 1);
    int c2 = findLCSLengthAux(s1, s2, i1 + 1, i2);//Increase index of 1st String
    int c1 = findLCSLengthAux(s1, s2, i1, i2 + 1);//Increase index of 2nd String
    return Max(c1, c2, c3)
```

Closest-Pair Problem: Divide and Conquer

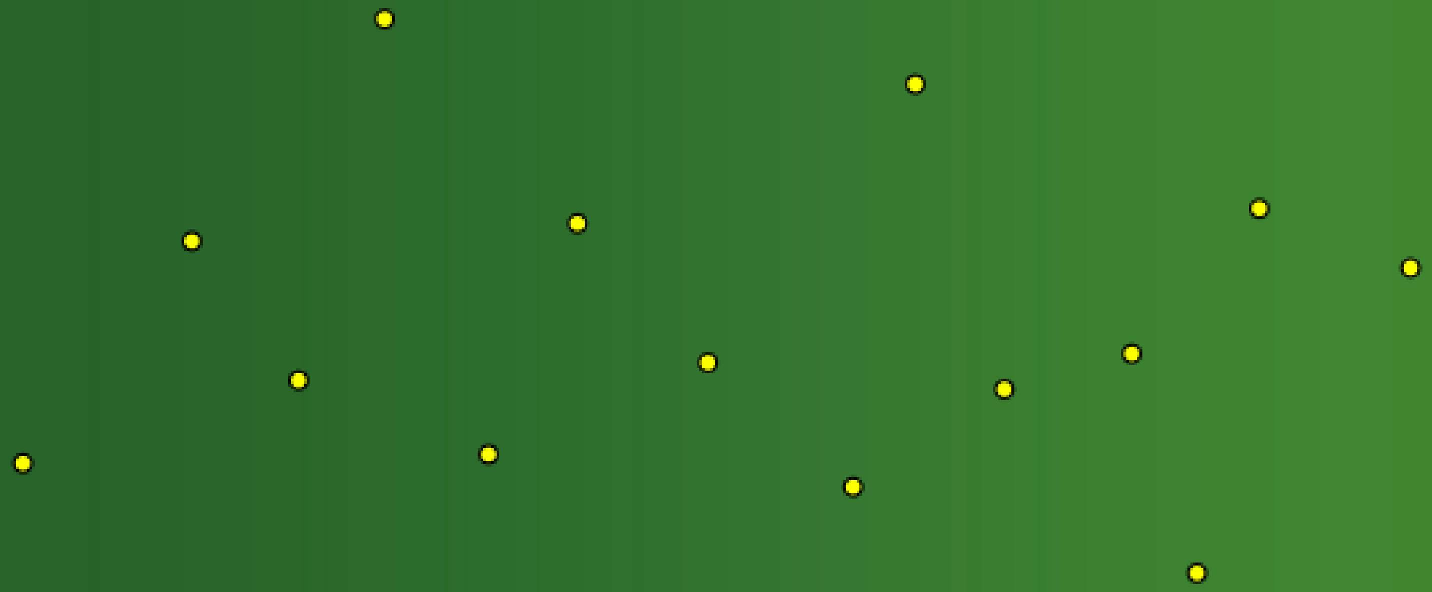
- Brute force approach requires comparing every point with every other point
- Given n points, we must perform 1 + 2 + 3 + … + n-2 + n-1 comparisons.

$$\sum_{k=1}^{n-1} k = \frac{(n-1)\cdot n}{2}$$

- Brute force → $O(n^2)$
- The Divide and Conquer algorithm yields → $O(n \log n)$
- Reminder: if n = 1,000,000 then
  - n^2 = 1,000,000,000,000 whereas
  - n log n = 20,000,000

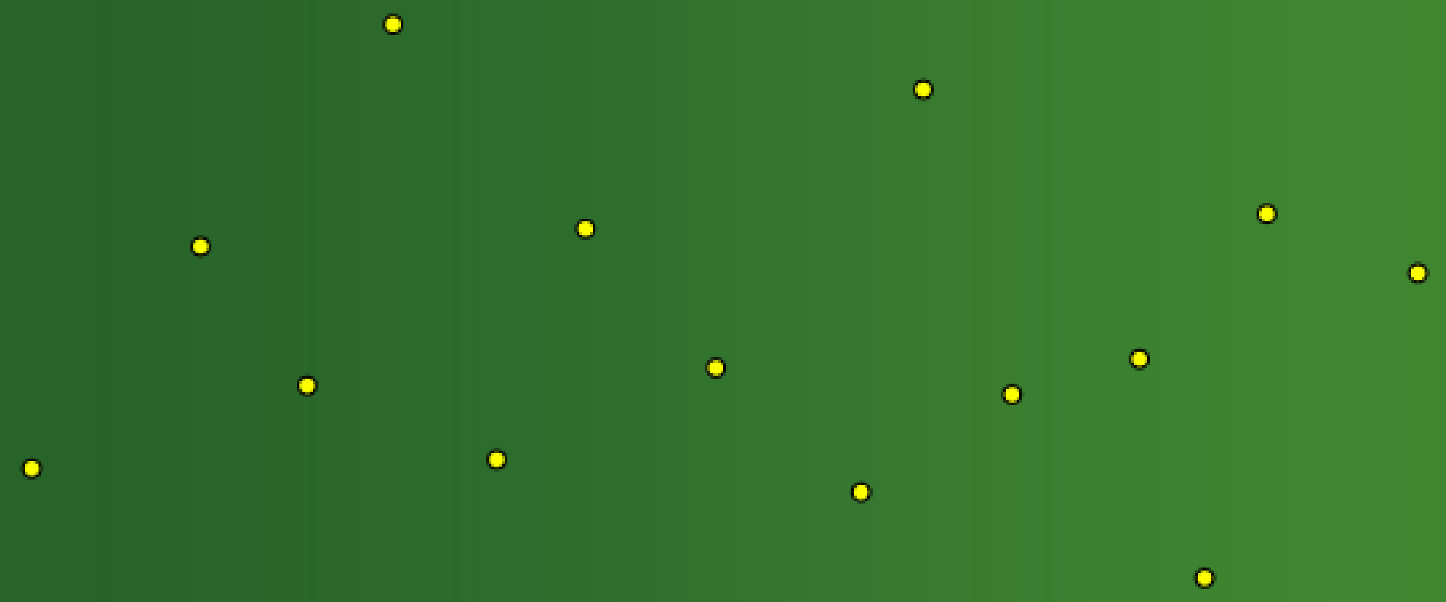
Closest-Pair Algorithm

Given: A set of points in 2-D



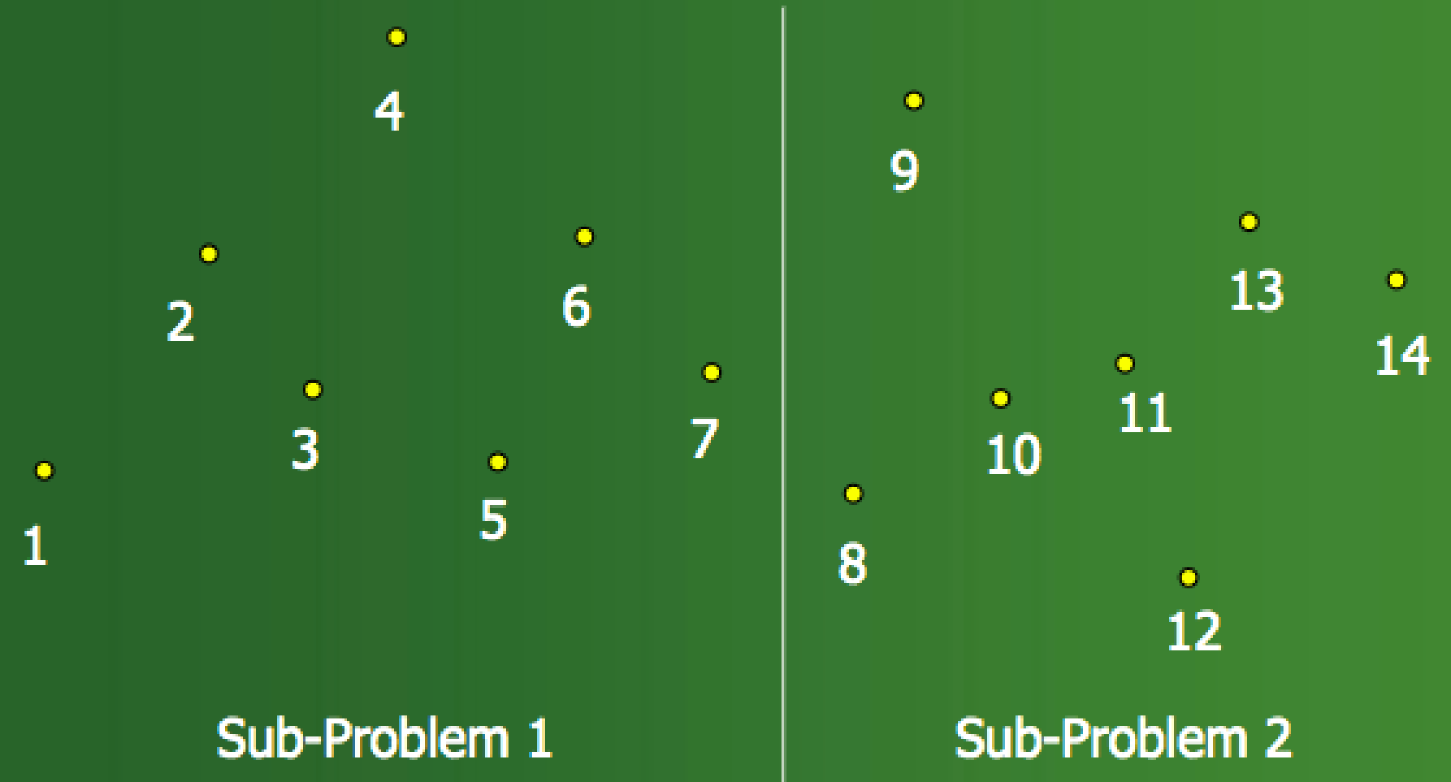
Closest-Pair Algorithm

Step 1: Sort the points in one D



Closest-Pair Algorithm

Step 2: Split the points, i.e.,
Draw a line at the mid-point between 7 and 8



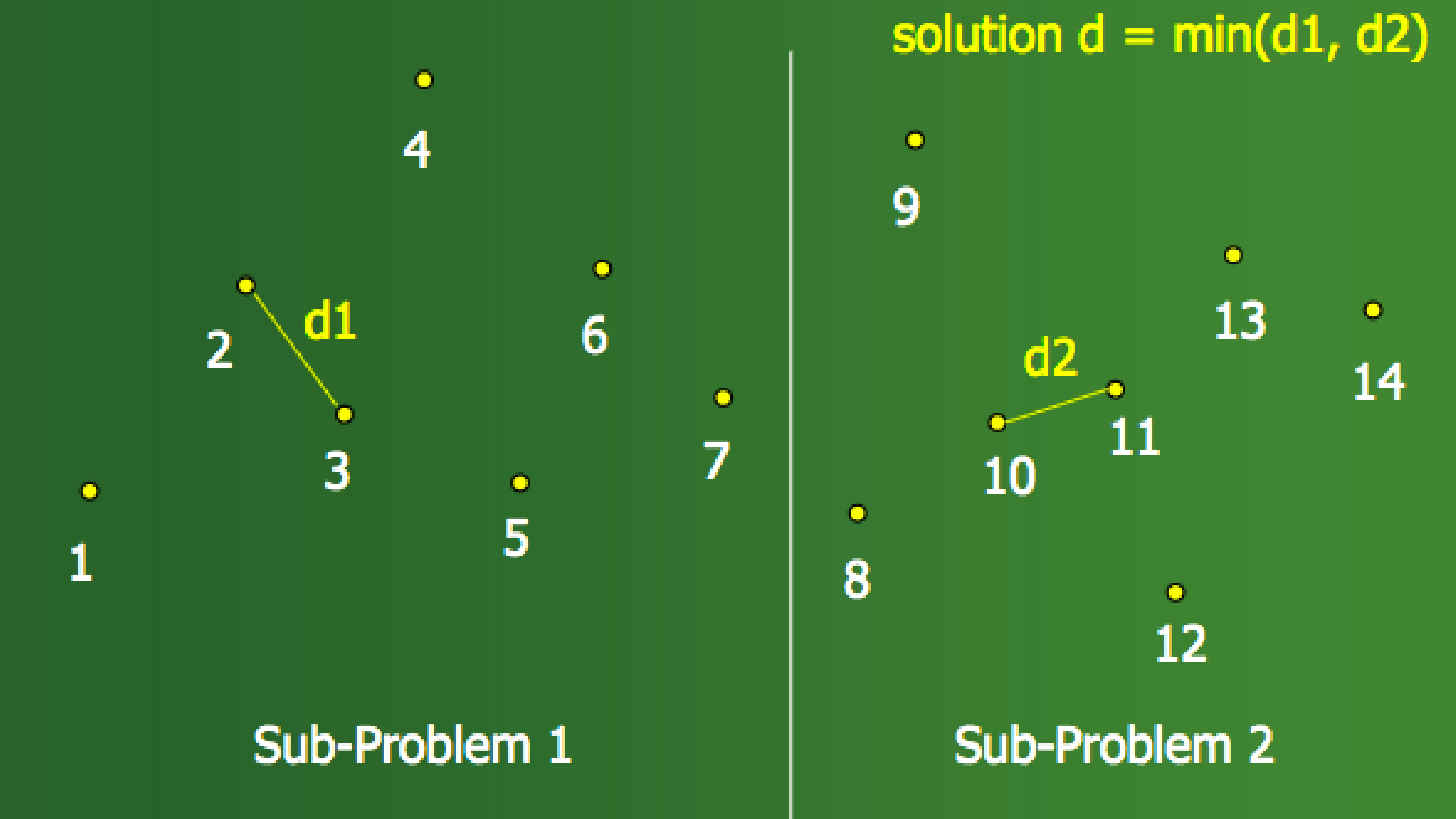
Closest-Pair Algorithm

Advantage: Normally, we'd have to compare each of the 14 points with every other point.

(n-1)n/2 = 13*14/2 = **91 comparisons**

Closest-Pair Algorithm

Advantage: Now, we have two sub-problems of half the size. Thus, we have to do 6*7/2 comparisons twice, which is 42 comparisons



Closest-Pair Algorithm

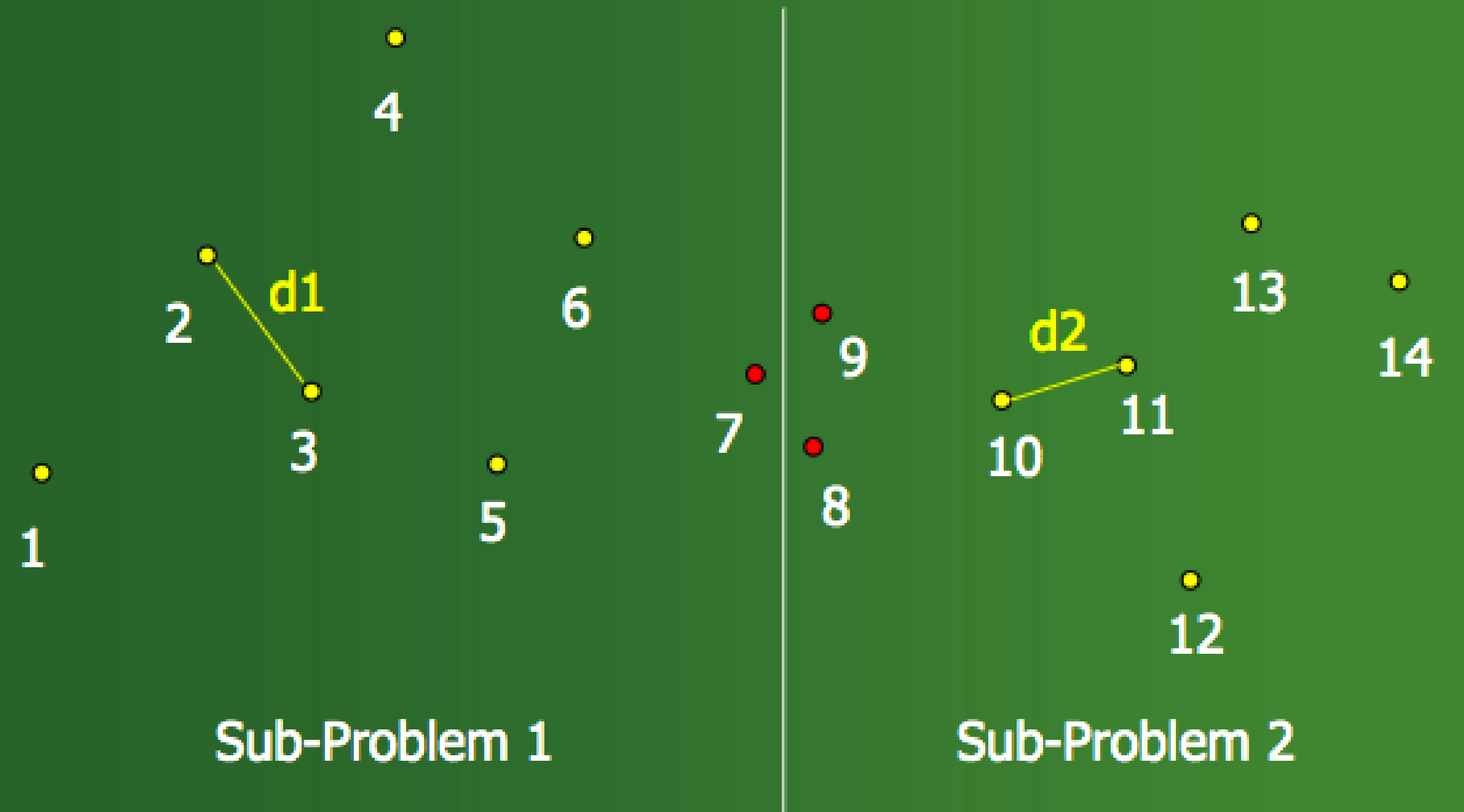
Advantage: With just one split we cut the number of comparisons in half. Obviously, we gain an even greater advantage if we split the sub-problems.

Closest-Pair Algorithm

Problem: However, what if the closest two points are each from different sub-problems?

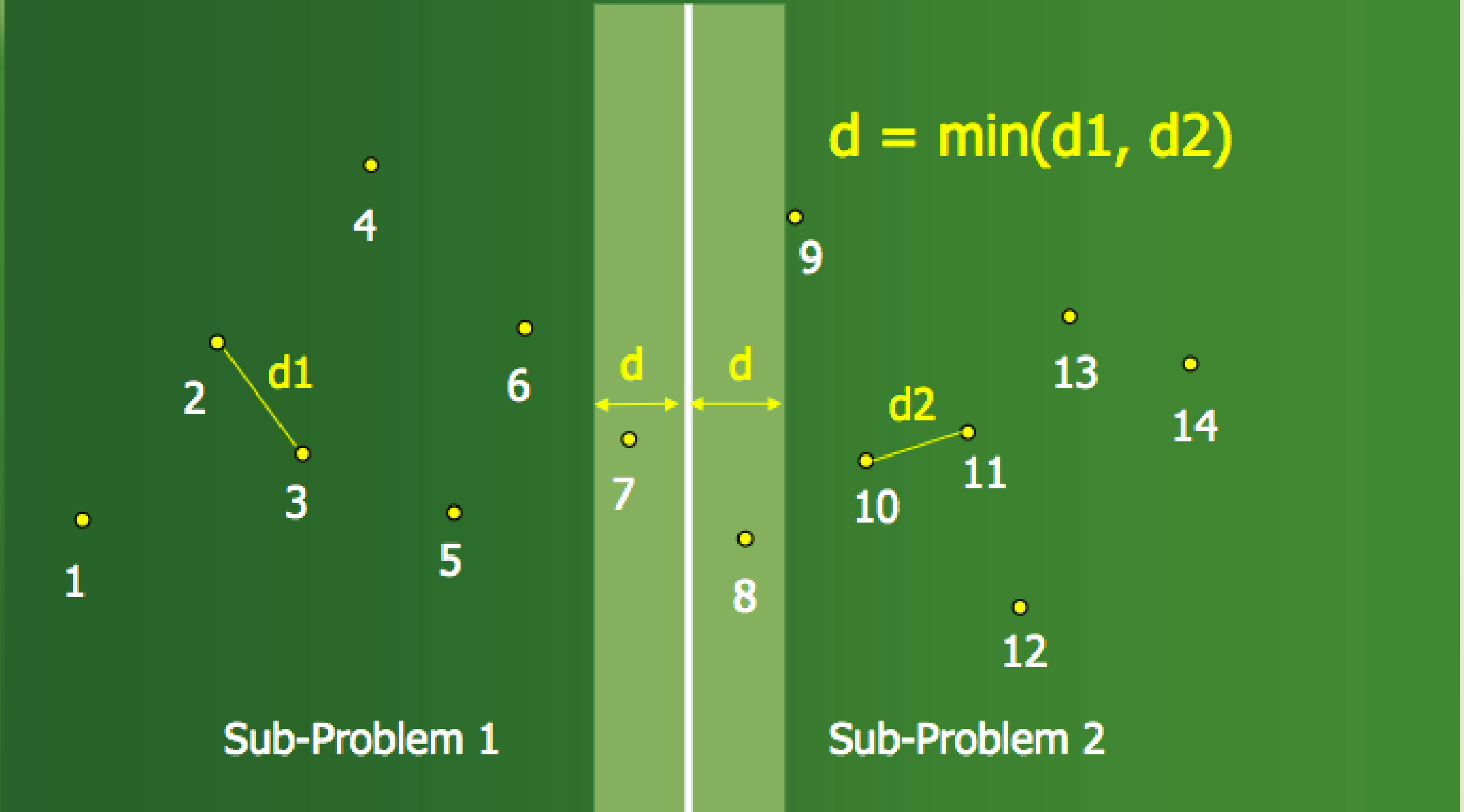
Closest-Pair Algorithm

Here is an example where we have to compare points from sub-problem 1 to the points in sub-problem 2.



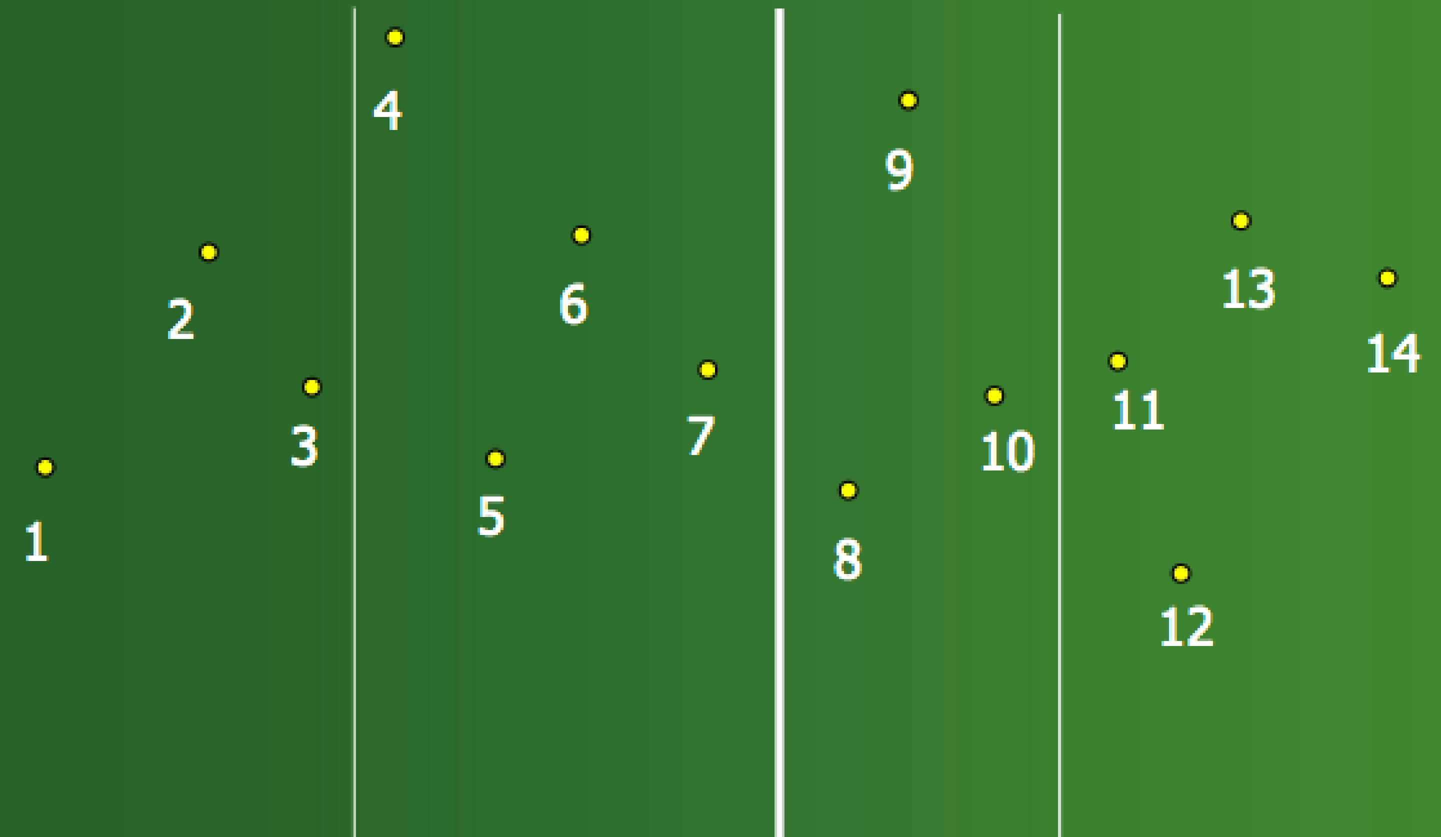
Closest-Pair Algorithm

However, we only have to compare points inside the following "strip."



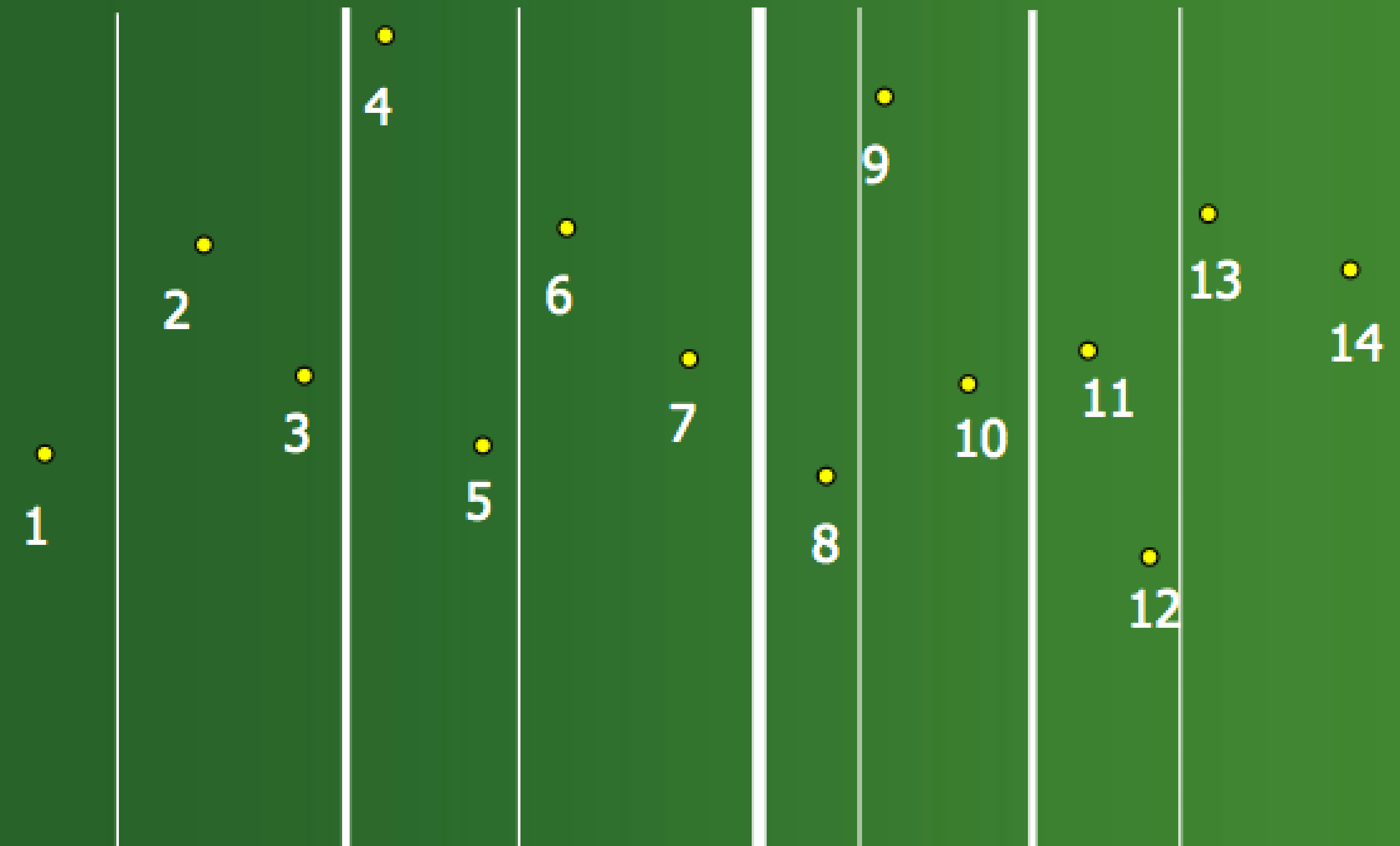
Closest-Pair Algorithm

Step 3: But, we can continue the advantage by splitting the sub-problems.



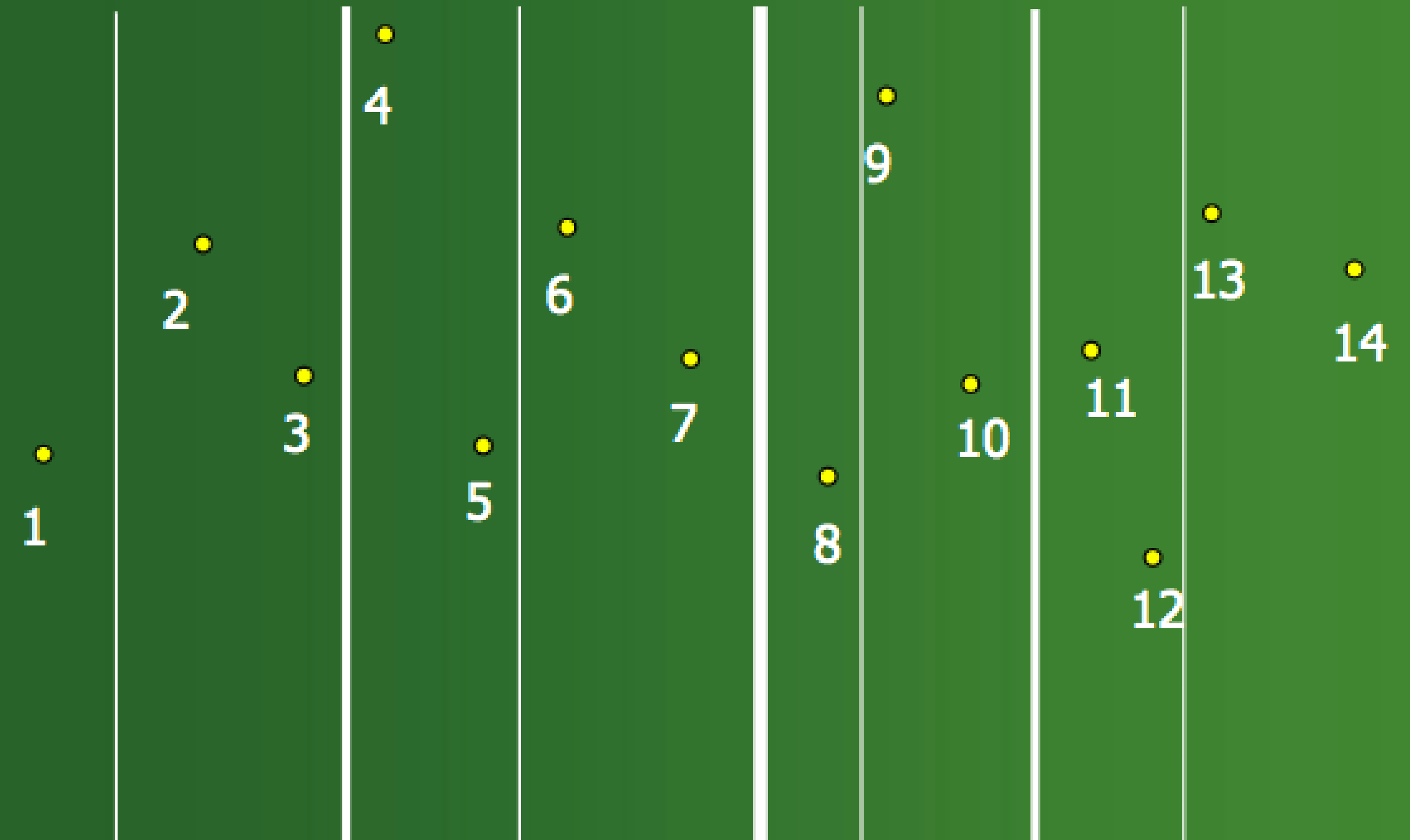
Closest-Pair Algorithm

Step 3: In fact we can continue to split until each sub-problem is trivial, i.e., takes one comparison.



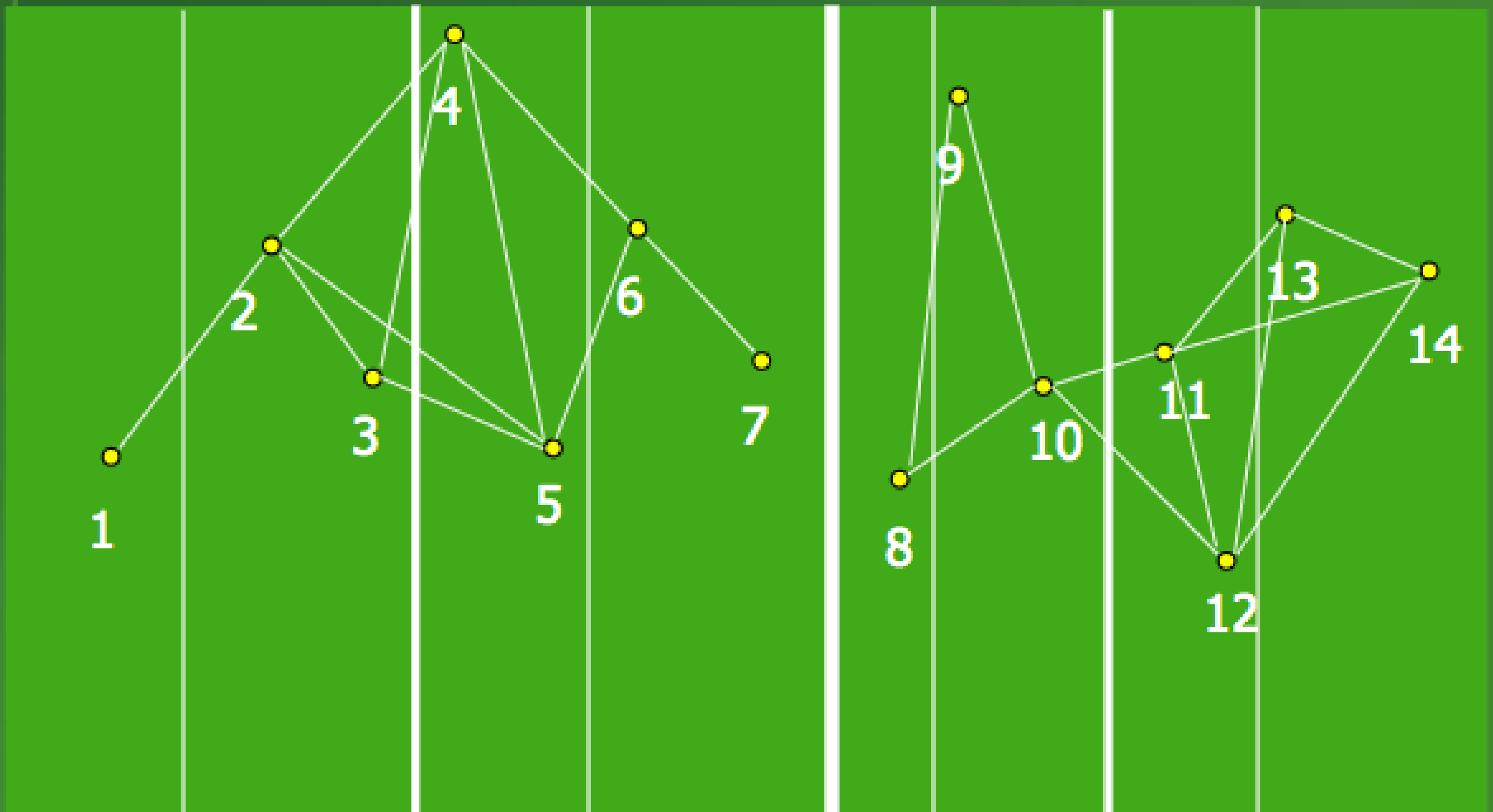
Closest-Pair Algorithm

Finally: The solution to each sub-problem is combined until the final solution is obtained



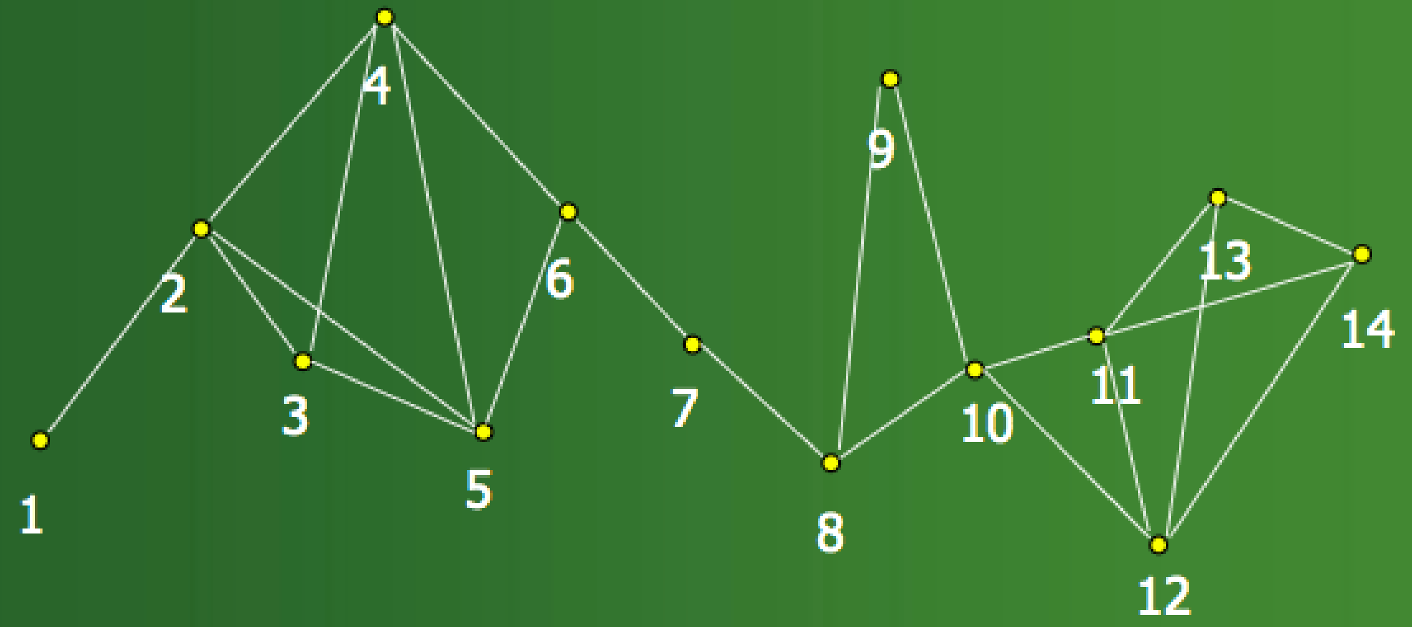
Closest-Pair Algorithm

Finally: On the last step the 'strip' will likely be very small. Thus, combining the two largest sub-problems won't require much work.



Closest-Pair Algorithm

- In this example, it takes 22 comparisons to find the closets-pair.
- The brute force algorithm would have taken 91 comparisons.
- But, the real advantage occurs when there are millions of points.




Thank you