Welcome To My Presentation

My Presentation Topic is

Counting Sort

Presented By

Tushar Sarkar

Student ID : 18CSE35

Second Year Second Semester

Department of CSE, BSMRSTU.

Content

- Introduction
- Counting Sort Algorithm
- Working of Counting Sort
- Pseudocode
- Time Complexity
- Conclusion

Introduction

- Counting sort is a sorting algorithm that sorts the elements of an array by counting the number of occurrences of each unique element in the array.
- The count is stored in an auxiliary array and the sorting is done by mapping the count as an index of the auxiliary array.
- Counting Sort is a sorting technique based on keys between a specific range.
- It works by counting the number of objects having distinct key values (kind of hashing).
- Then doing some arithmetic to calculate the position of each object in the output sequence.

Counting Sort Algorithm

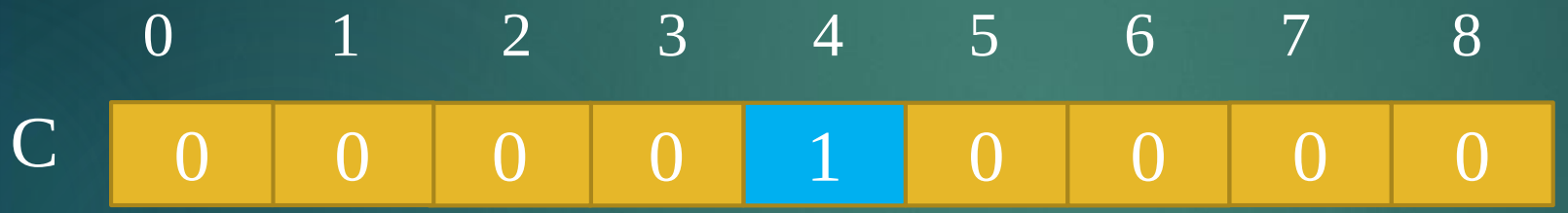
- Find out the maximum element (let it be max) from the given array.
- Initialize an array of length max+1 with all elements 0. This array is used for storing the count of the elements in the array.
- Store the count of each element at their respective index in count array.
- Store cumulative sum of the elements of the count array. It helps in placing the elements into the correct index of the sorted array.
- Find the index of each element of the original array in the count array.
- After placing each element at its correct position, decrease its count by one.

Working of Counting Sort

Here, max = 8

C for count array

Working of Counting Sort



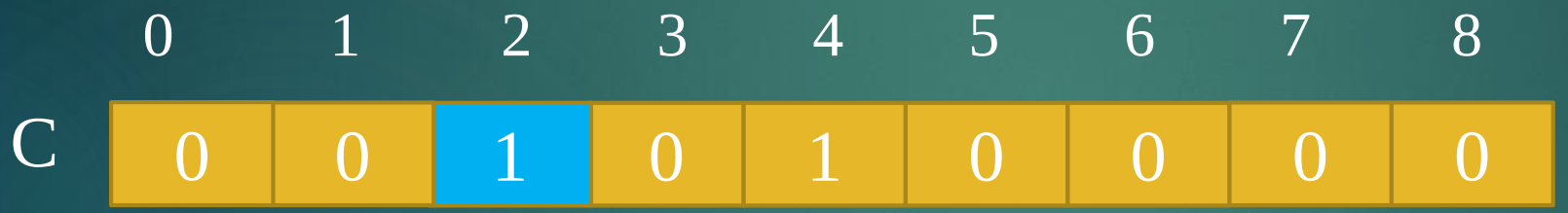
for(i = 0; i < A.length or 8; i++)
C[A[i]] = C[A[i]] + 1

C[4] = C[4] + 1

= 0 + 1

= 1

Working of Counting Sort



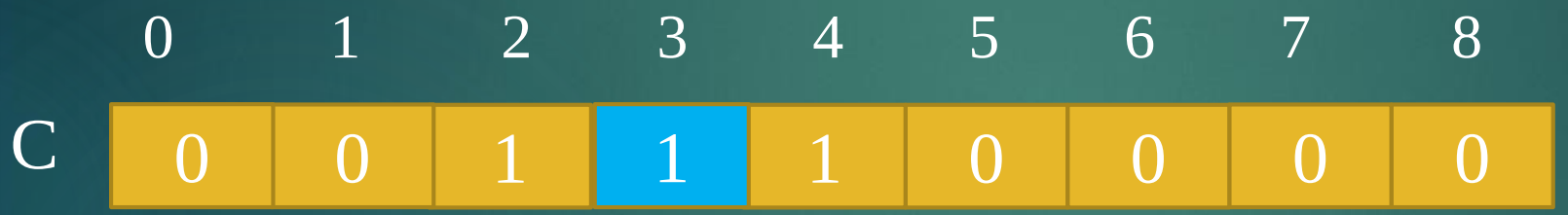
for(i = 1; i < A.length or 8; i++)
C[A[i]] = C[A[i]] + 1

C[2] = C[2] + 1

= 0 + 1

= 1

Working of Counting Sort



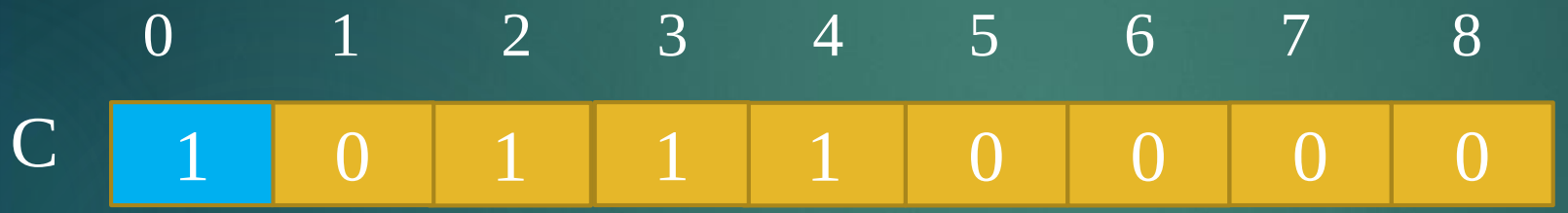
for(i = 2; i < A.length or 8; i++)
C[A[i]] = C[A[i]] + 1

C[3] = C[3] + 1

= 0 + 1

= 1

Working of Counting Sort



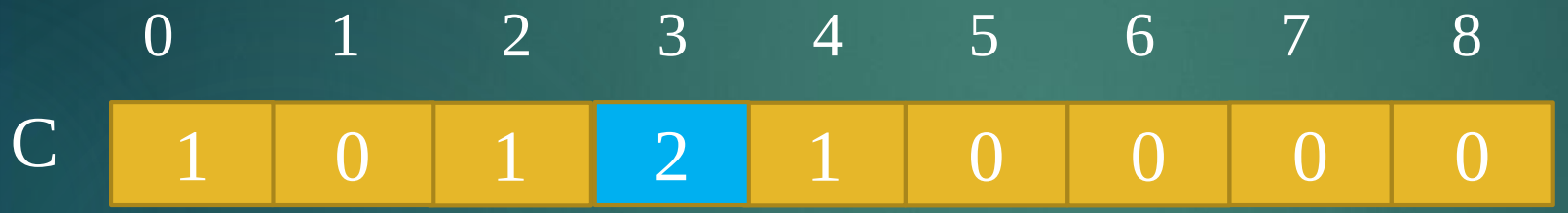
for(i = 3; i < A.length or 8; i++)
C[A[i]] = C[A[i]] + 1

C[0] = C[0] + 1

= 0 + 1

= 1

Working of Counting Sort



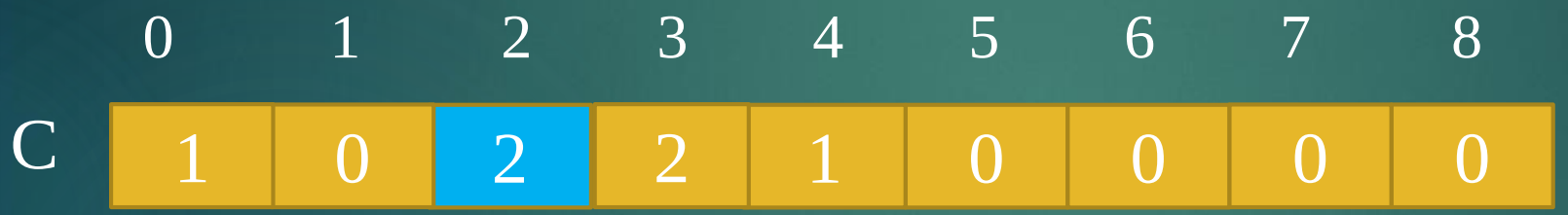
for(i = 4; i < A.length or 8; i++)
C[A[i]] = C[A[i]] + 1

C[3] = C[3] + 1

= 1 + 1

= 2

Working of Counting Sort



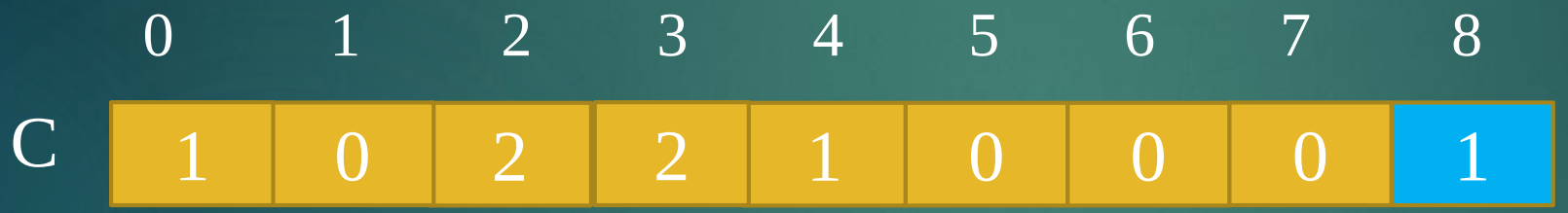
for(i = 5; i < A.length or 8; i++)
C[A[i]] = C[A[i]] + 1

C[2] = C[2] + 1

= 1 + 1

= 2

Working of Counting Sort



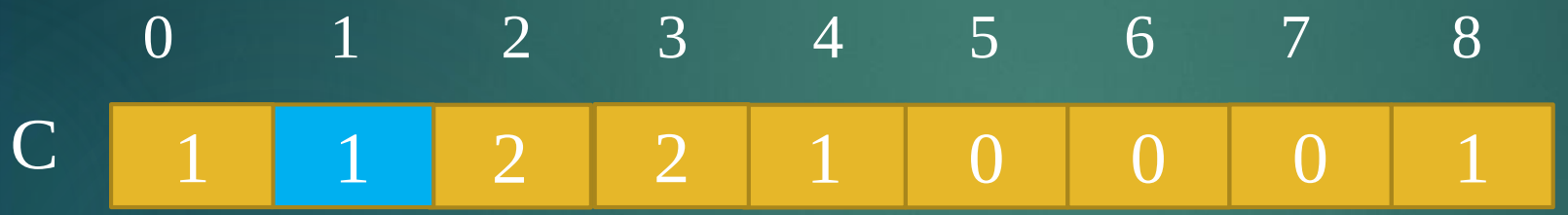
for(i = 6; i < A.length or 8; i++)
C[A[i]] = C[A[i]] + 1

C[8] = C[8] + 1

= 0 + 1

= 1

Working of Counting Sort



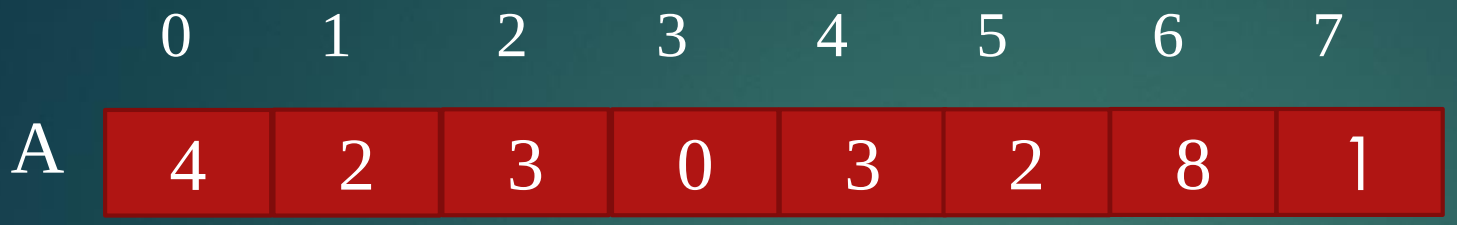
for(i = 7; i < A.length or 8; i++)
C[A[i]] = C[A[i]] + 1

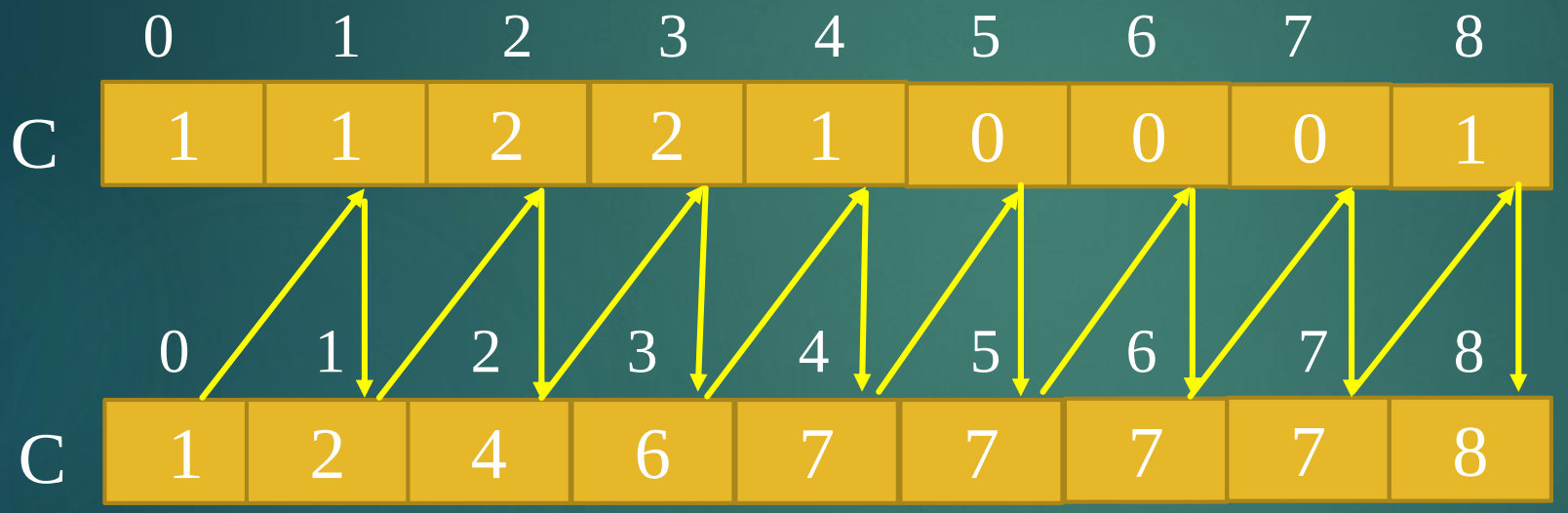
C[1] = C[1] + 1

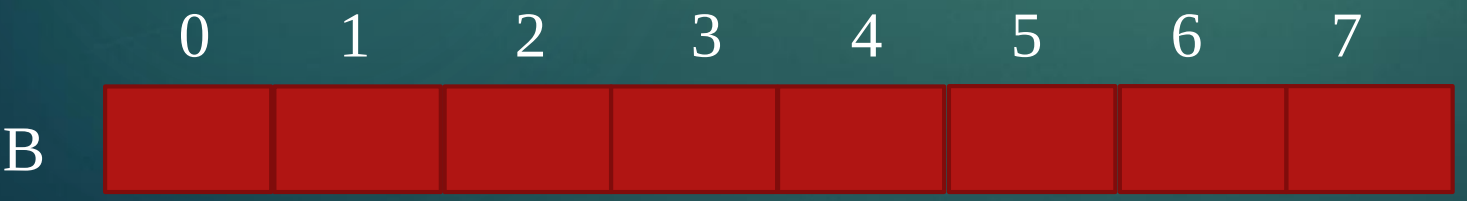
= 0 + 1

= 1

Working of Counting Sort



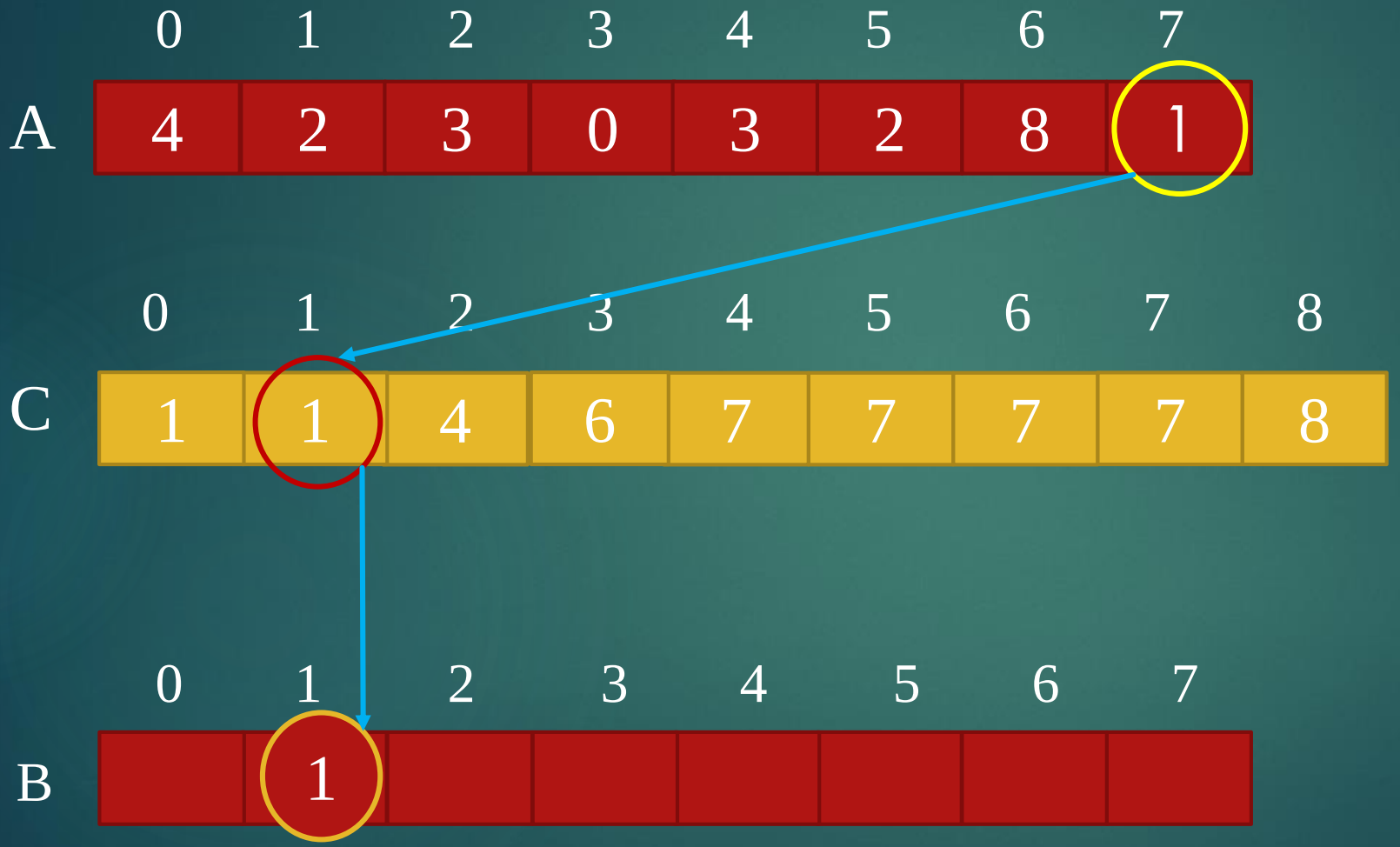




Cumulative Sum

```
for(i = 0; i <= max(A); i++)
  if(i==0)C[i] = C[i]
  else C[i] = C[i-1] + C[i]
```

Working of Counting Sort



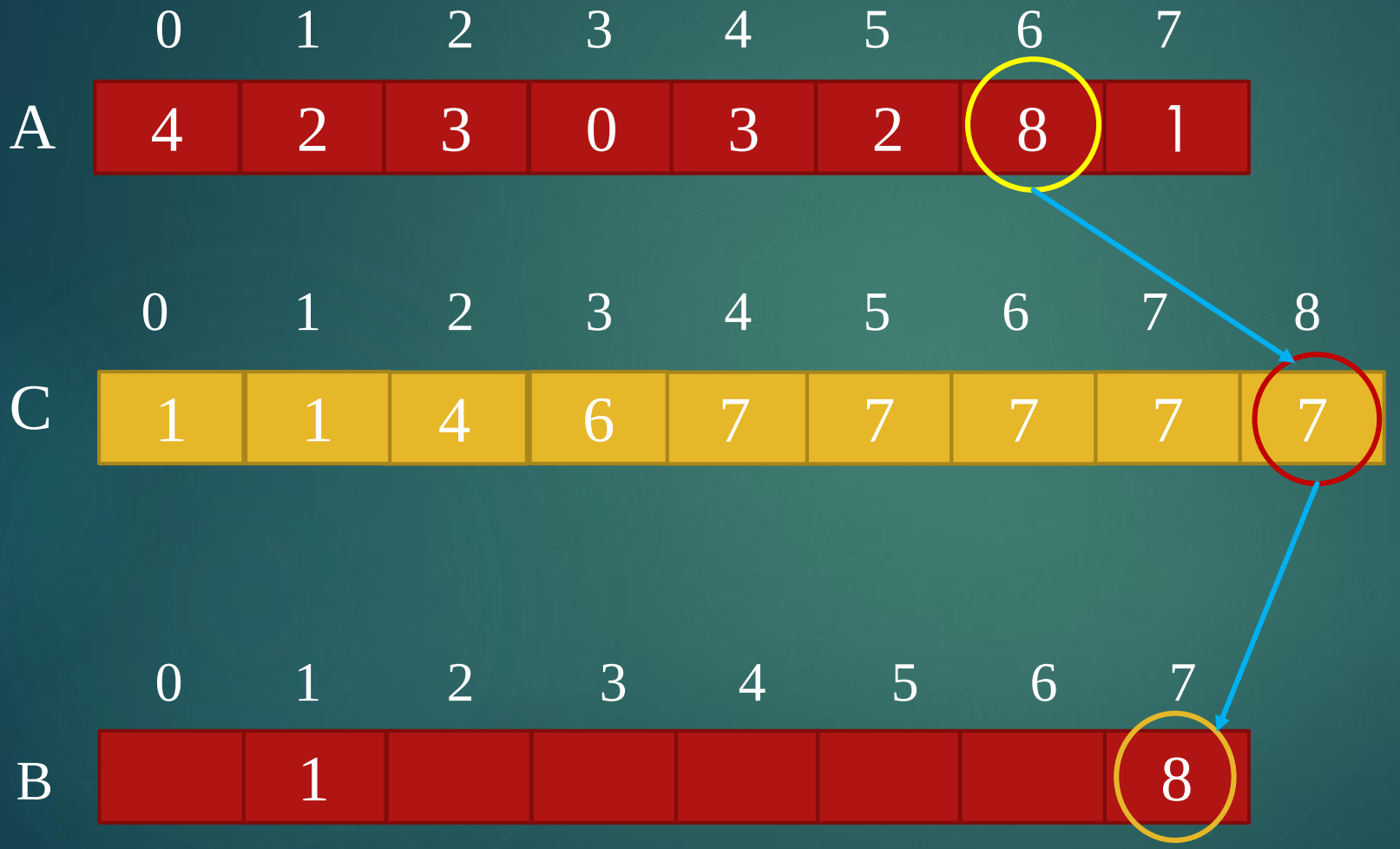

```
for(i=7; i >= 0; i--)
      C[A[i]]--;
      B[C[A[i]]] = A[i]
```

A[7] = 1;

C[1] = C[1] – 1;
= 2 – 1
= 1

B[1] = A[7] = 1

Working of Counting Sort



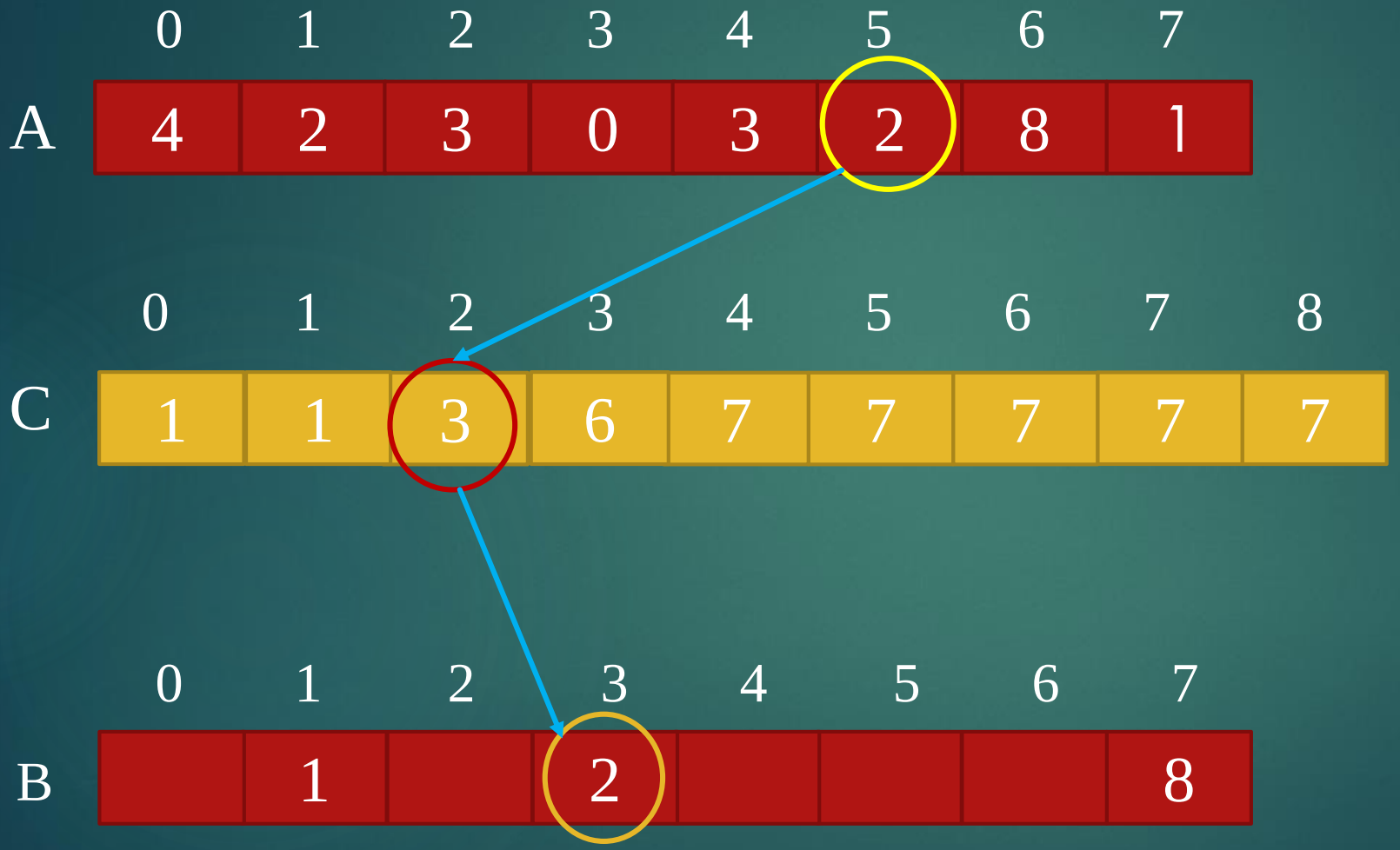

```
for(i=6; i >= 0; i--)
      C[A[i]]--;
      B[C[A[i]]] = A[i]
```

A[6] = 8;

C[8] = C[8] – 1;
= 8 – 1
= 7

B[7] = A[6] = 8

Working of Counting Sort



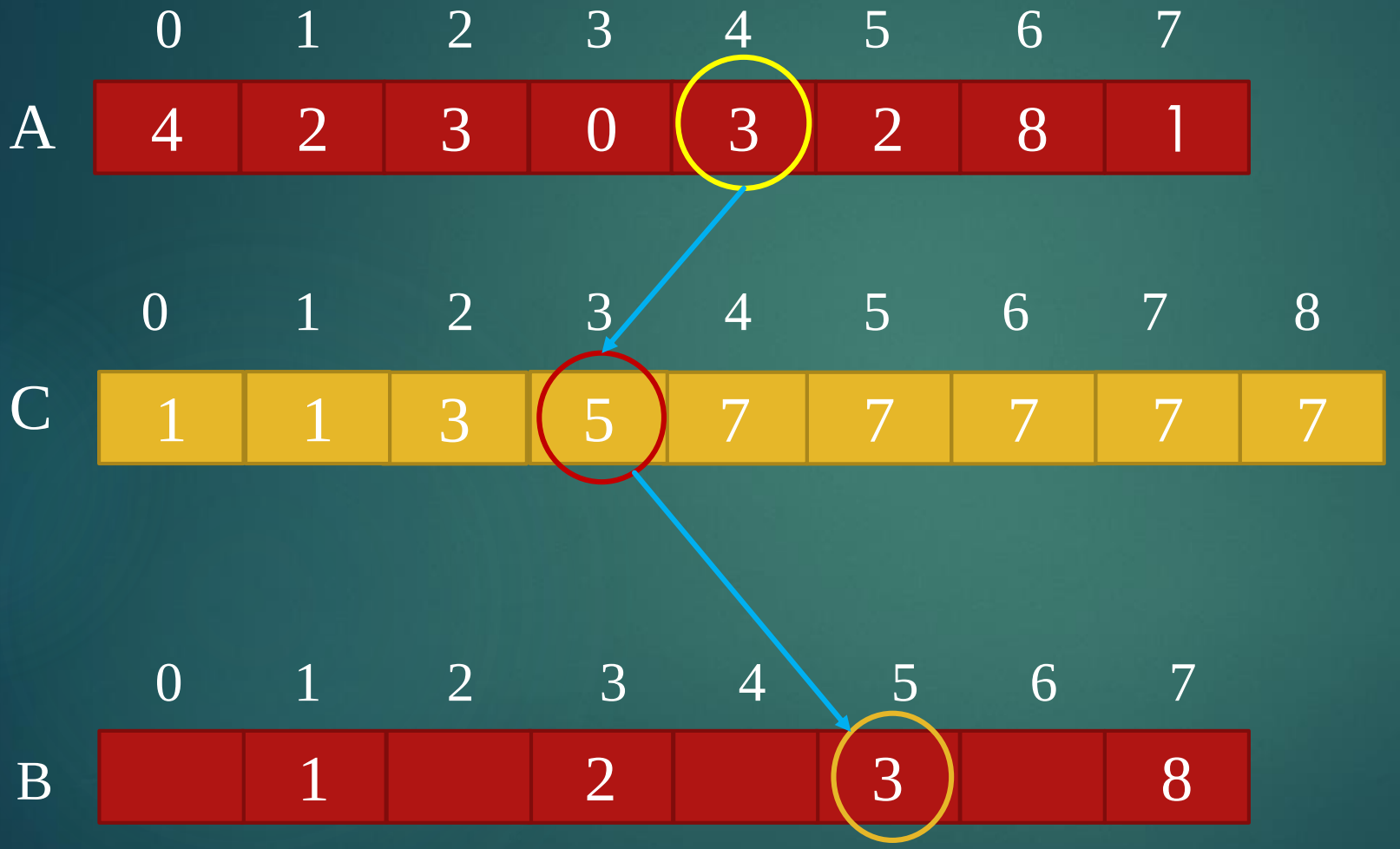

```
for(i=5; i >= 0; i--)
      C[A[i]]--;
      B[C[A[i]]] = A[i]
```

A[5] = 2;

C[2] = C[2] – 1;
= 4 – 1
= 3

B[3] = A[5] = 2

Working of Counting Sort



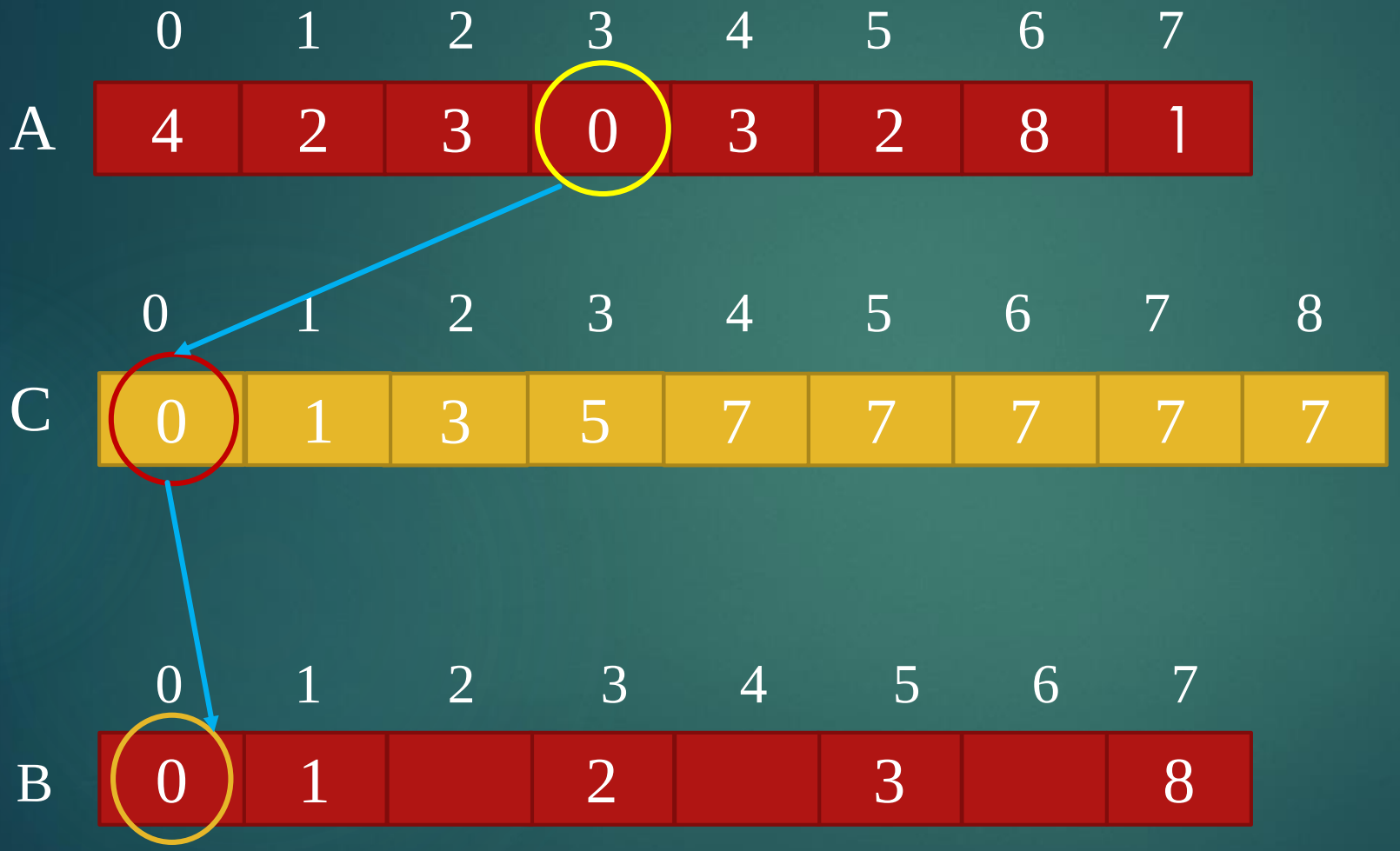

```
for(i=4; i >= 0; i--)
      C[A[i]]--;
      B[C[A[i]]] = A[i]
```

A[4] = 3;

C[3] = C[3] – 1;
= 6 – 1
= 5

B[5] = A[4] = 3

Working of Counting Sort



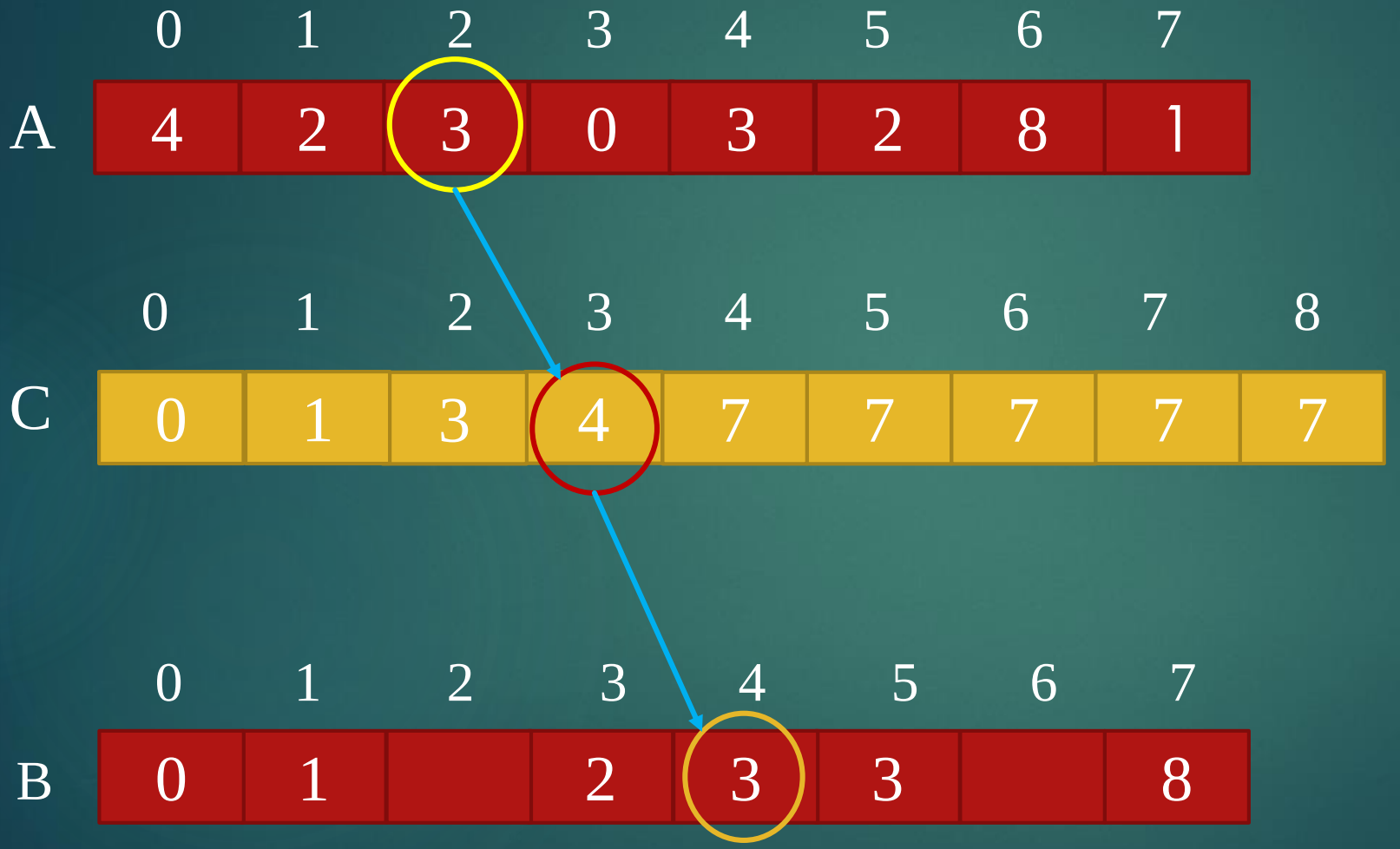

```
for(i=3; i >= 0; i--)
      C[A[i]]--;
      B[C[A[i]]] = A[i]
```

A[3] = 0;

C[0] = C[0] – 1;
= 1 – 1
= 0

B[0] = A[3] = 0

Working of Counting Sort



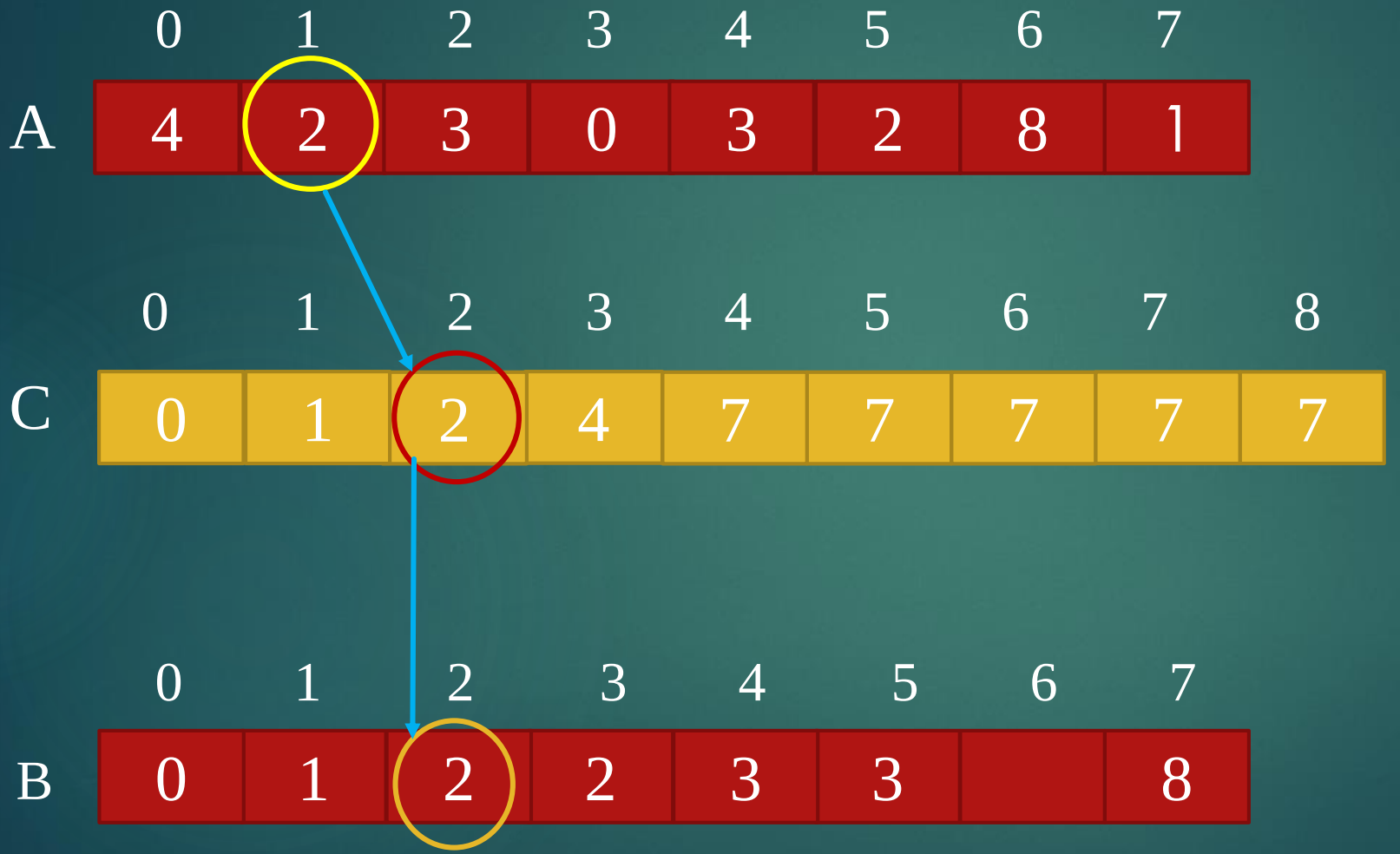

```
for(i=2; i >= 0; i--)
      C[A[i]]--;
      B[C[A[i]]] = A[i]
```

A[2] = 3;

C[3] = C[3] – 1;
= 5 – 1
= 4

B[4] = A[2] = 3

Working of Counting Sort



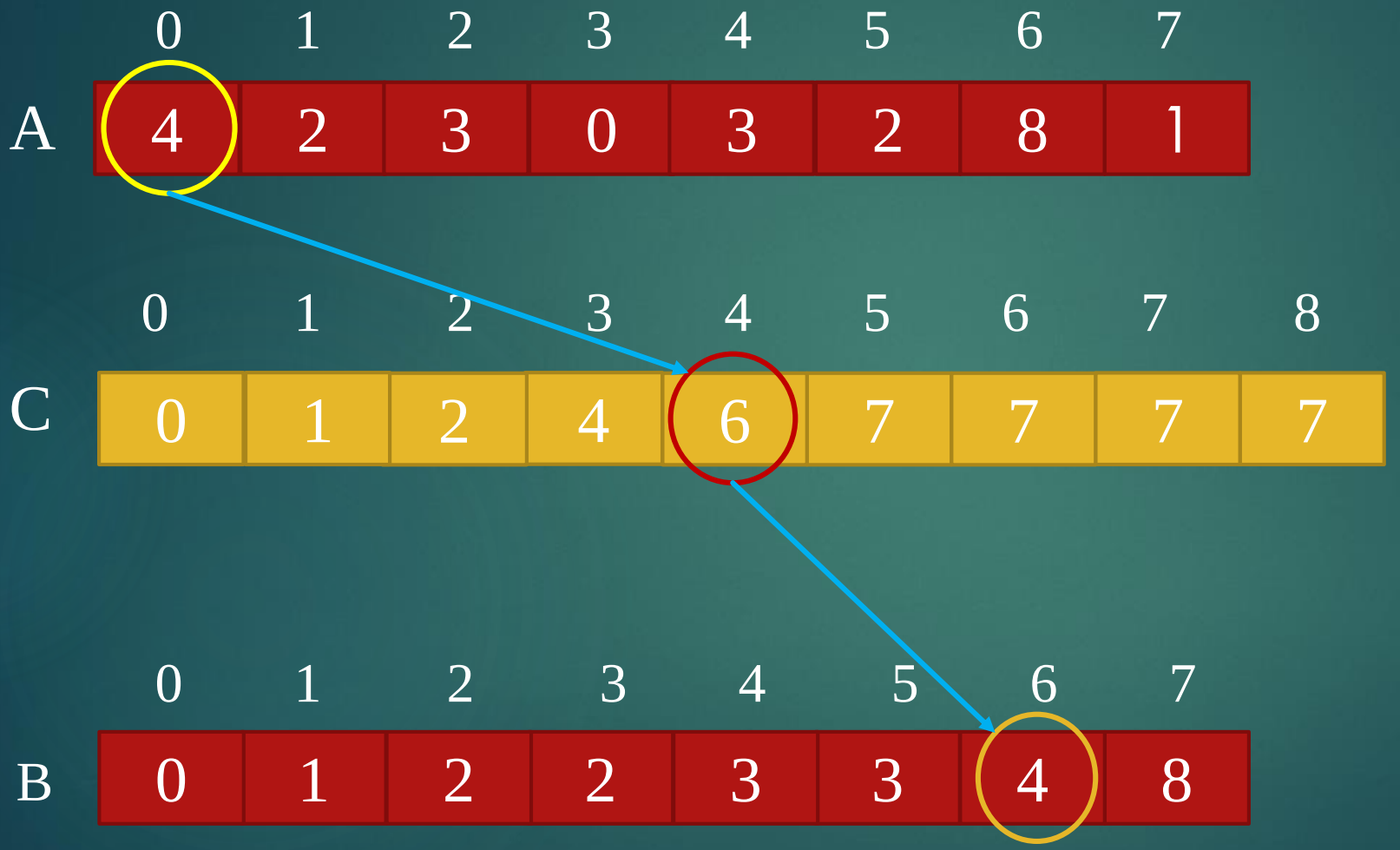

```
for(i=1; i >= 0; i--)
      C[A[i]]--;
      B[C[A[i]]] = A[i]
```

A[1] = 2;

C[2] = C[2] – 1;
= 3 – 1
= 2

B[2] = A[1] = 2

Working of Counting Sort




```
for(i=0; i >= 0; i--)
      C[A[i]]--;
      B[C[A[i]]] = A[i]
```

A[0] = 4;

C[4] = C[4] – 1;
= 7 – 1
= 6

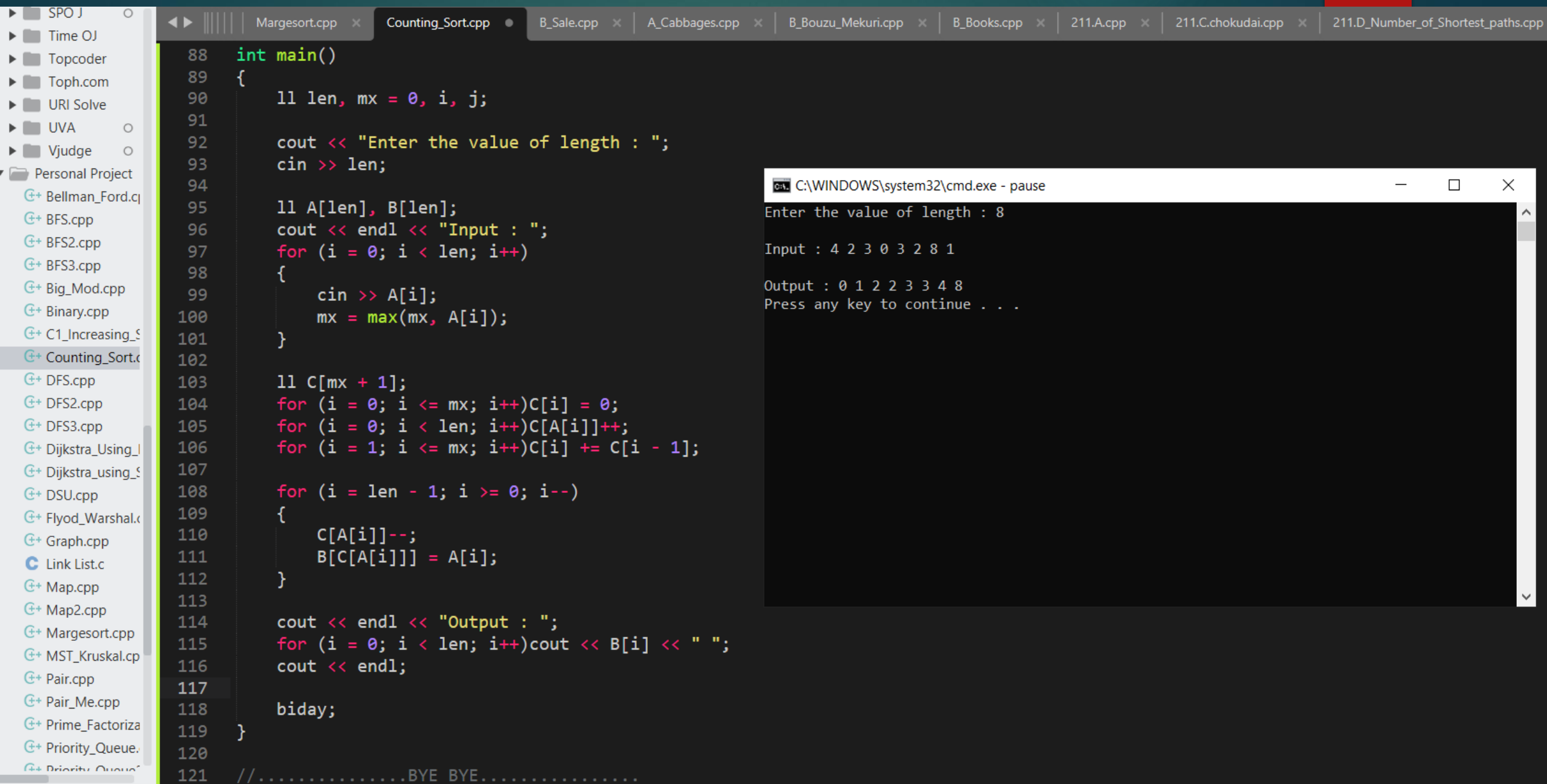
B[6] = A[0] = 4

Working of Counting Sort

This is Input & Unsorted Array

This is Output & Sorted Array

Pseudocode


SPO J
Time OJ
Topcoder
Toph.com
URI Solve
UVA
Vjudge
Personal Project
Bellman_Ford.c
BFS.cpp
BFS2.cpp
BFS3.cpp
Big_Mod.cpp
Binary.cpp
C1_Increasing_S
Counting_Sort.c
DFS.cpp
DFS2.cpp
DFS3.cpp
Dijkstra_Using_
Dijkstra_using_S
DSU.cpp
Flyod_Warshal.c
Graph.cpp
Link List.c
Map.cpp
Map2.cpp
Margesort.cpp
MST_Kruskal.cp
Pair.cpp
Pair_Me.cpp
Prime_Factoriza
Priority_Queue.
Margesort.cpp
Counting_Sort.cpp
B_Sale.cpp
A_Cabbages.cpp
B_Bouzu_Mekuri.cpp
B_Books.cpp
211.A.cpp
211.C.chokudai.cpp
211.D_Number_of_Shortest_paths.cpp
88 int main()
89 {
90 ll len, mx = 0, i, j;
91
92 cout << "Enter the value of length : ";
93 cin >> len;
94
95 ll A[len], B[len];
96 cout << endl << "Input : ";
97 for (i = 0; i < len; i++)
98 {
99 cin >> A[i];
100 mx = max(mx, A[i]);
101 }
102
103 ll C[mx + 1];
104 for (i = 0; i <= mx; i++)C[i] = 0;
105 for (i = 0; i < len; i++)C[A[i]]++;
106 for (i = 1; i <= mx; i++)C[i] += C[i - 1];
107
108 for (i = len - 1; i >= 0; i--)
109 {
110 C[A[i]]--;
111 B[C[A[i]]] = A[i];
112 }
113
114 cout << endl << "Output : ";
115 for (i = 0; i < len; i++)cout << B[i] << " ";
116 cout << endl;
117
118 biday;
119 }
120
121 //..............BYE BYE..............
C:\WINDOWS\system32\cmd.exe - pause
Enter the value of length : 8
Input : 4 2 3 0 3 2 8 1
Output : 0 1 2 2 3 3 4 8
Press any key to continue . . .

Time Complexity

- In all the above cases, the complexity is the same because no matter how the elements are placed in the array, the algorithm goes through *len+mx* times.

- Overall complexity = O(len) + O(mx) + O(len) + O(mx) + O(len) + O(len)
  = O(len + mx)

Worst Case Complexity : O(len+mx)

Best Case Complexity : O(len+mx)

Average Case Complexity: O(len+mx)

Thank You