Freie Universität Berlin

Objektorientierte Programmierung

Sortieren (Teil 3)

Imperativ!

SoSe 2020

Prof. Dr. Margarita Esponda

Sortieralgorithmen

Ohne Vergleiche

Counting-Sort	**O**(n)
Radix-Sort	**O**(n)
Bucket-Sort	**O**(n)

Counting sort

Wenn die Daten, die sortiert werden sollen, ganzzahlige Werte mit einem kleinen Wertebereich zwischen **0** und **k** sind, ist es möglich, Zahlen zu sortieren ohne diese direkt zu vergleichen.

Die Zeitkomplexität, die man dabei hat, ist linear.

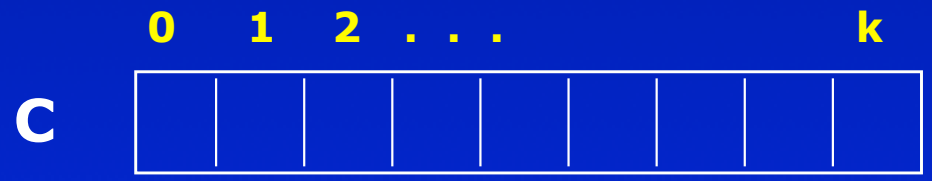
$$T(n) = O(n)$$

Anzahl der Daten, die sortiert werden sollen

Counting sort

Der "Counting sort"-Algorithmus benutzt zwei Hilfsfelder.

1. ein **C**-Feld, das so groß ist wie der Wertebereich (**k=8**) der Zahlen

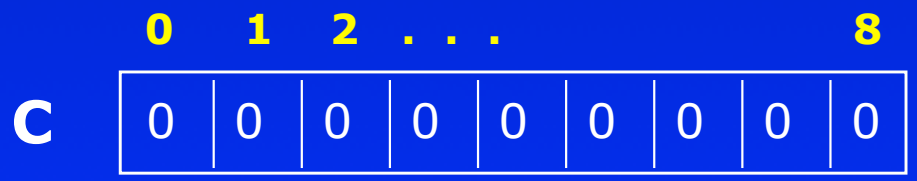


2. ein **B**-Feld, in dem die sortierten Zahlen am Ende gespeichert werden.

Counting sort

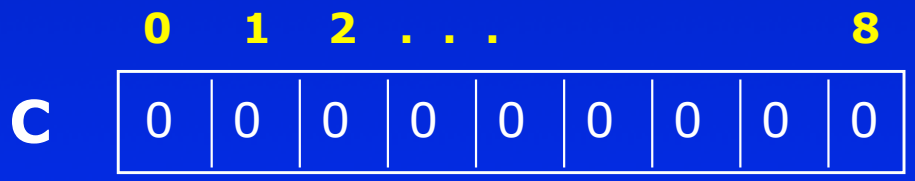
Nehmen wir an, wir wollen die Zahlen des **A**-Feldes sortieren.

Das C-Feld wird mit Nullen initialisiert.

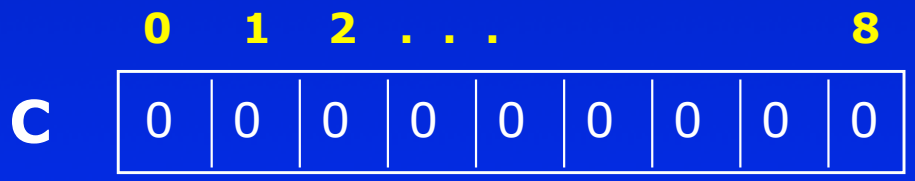


Wertebereich = (0-8)

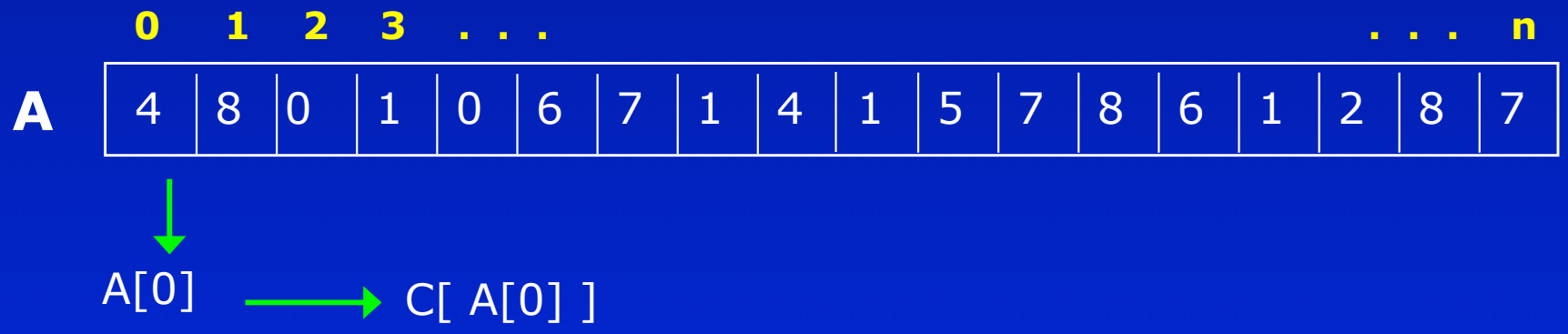
Counting sort

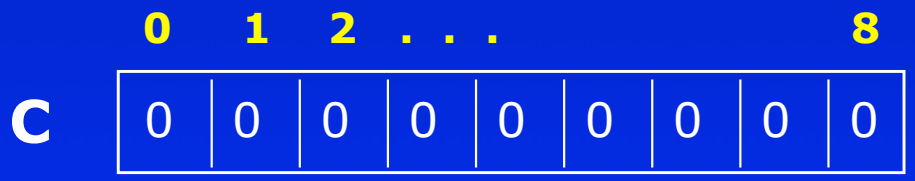


Counting sort

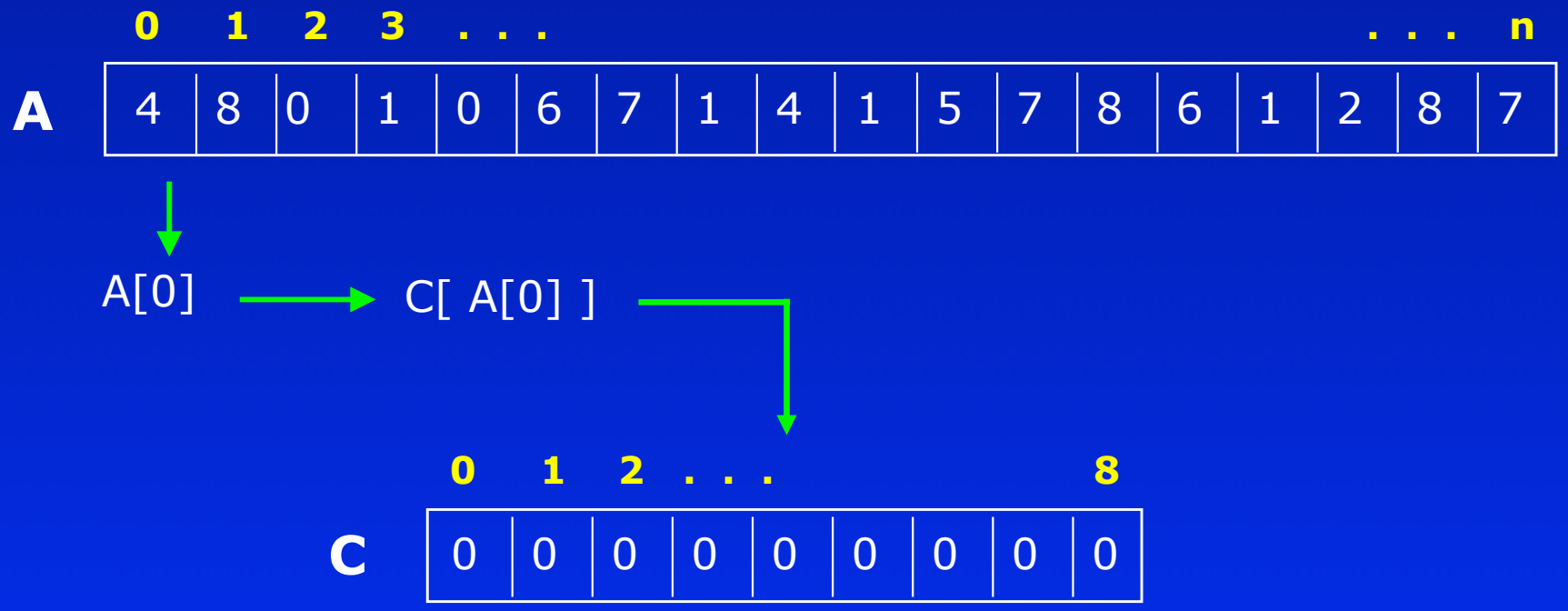


Counting sort

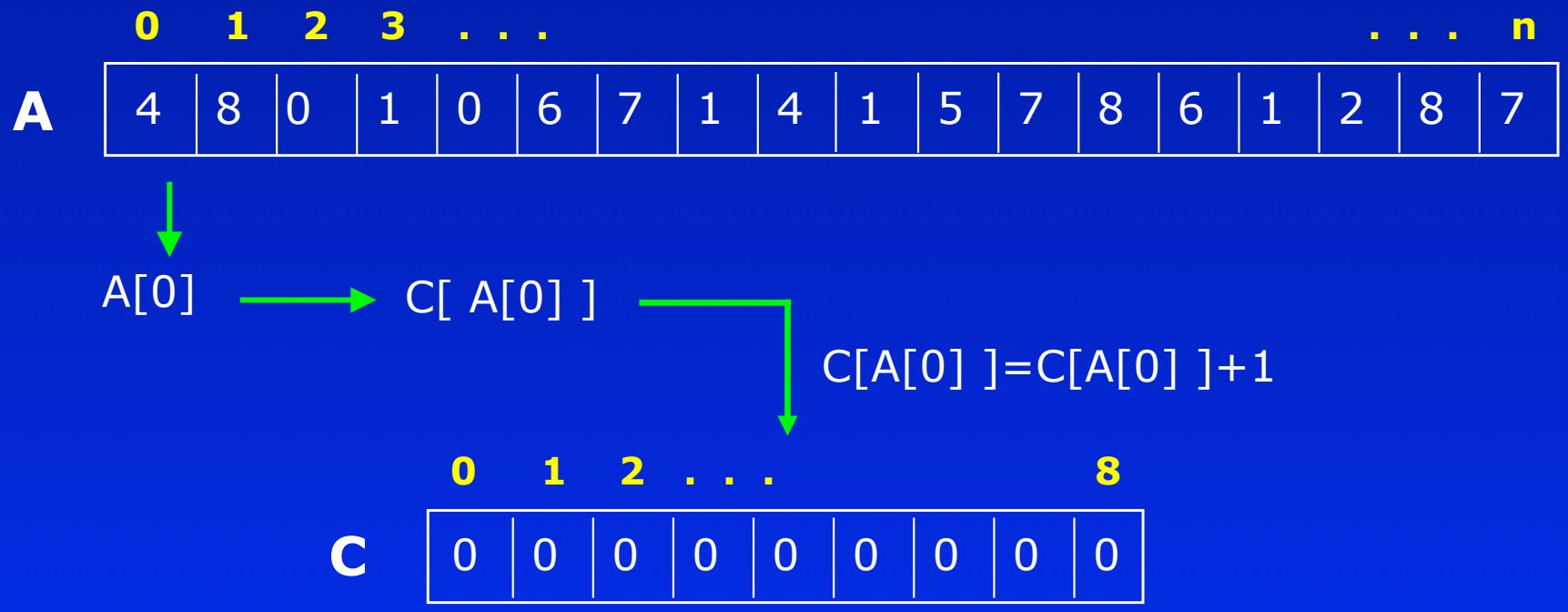




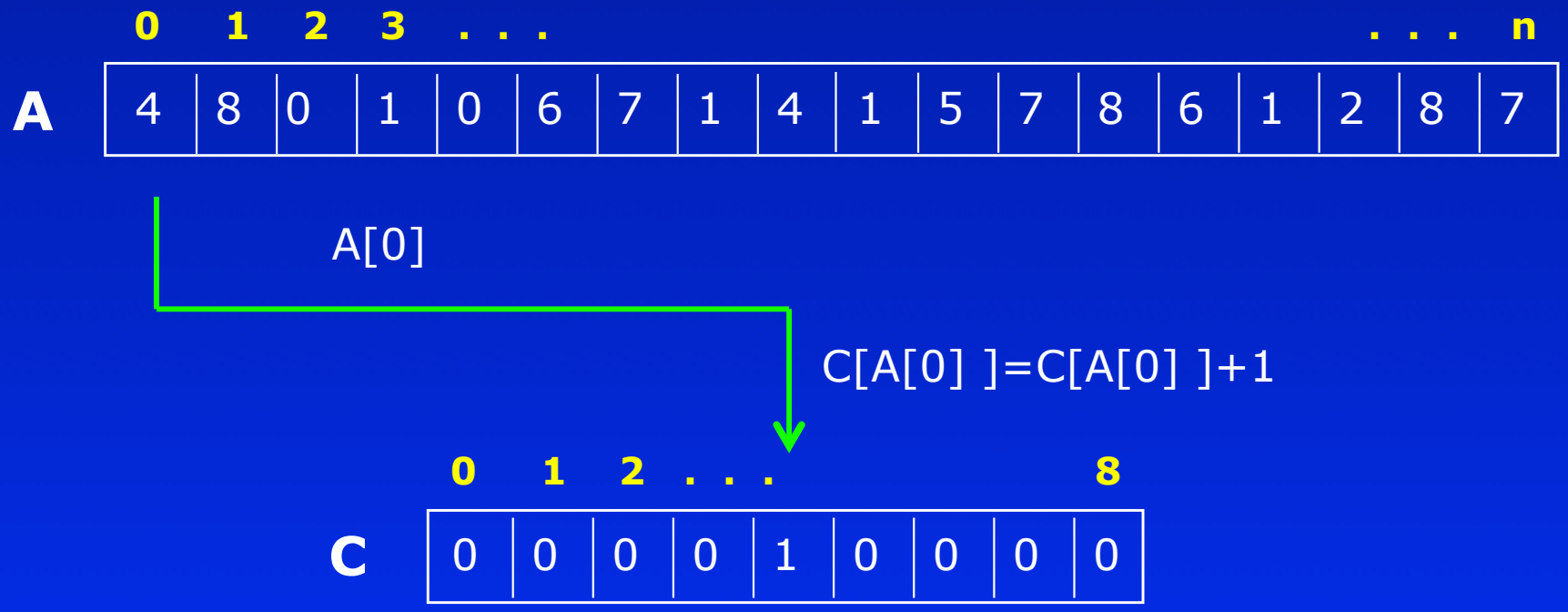
Counting sort



Counting sort

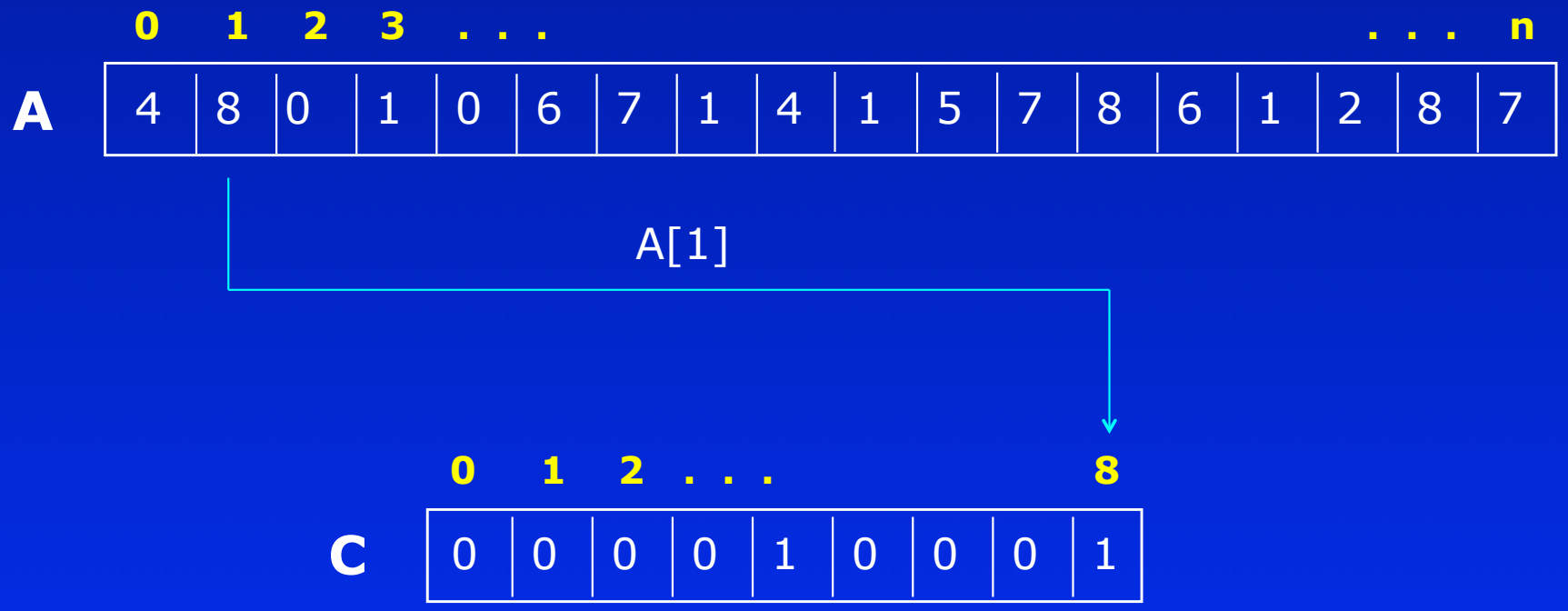


Counting sort

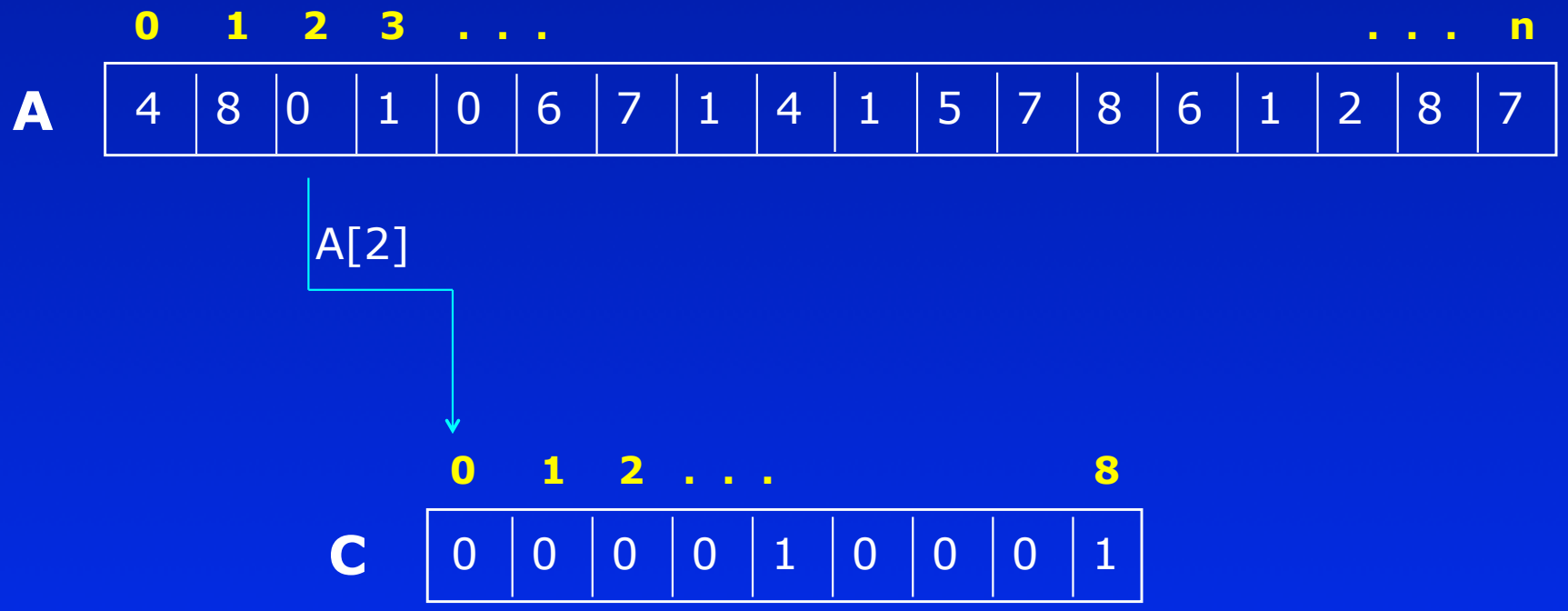


Counting sort

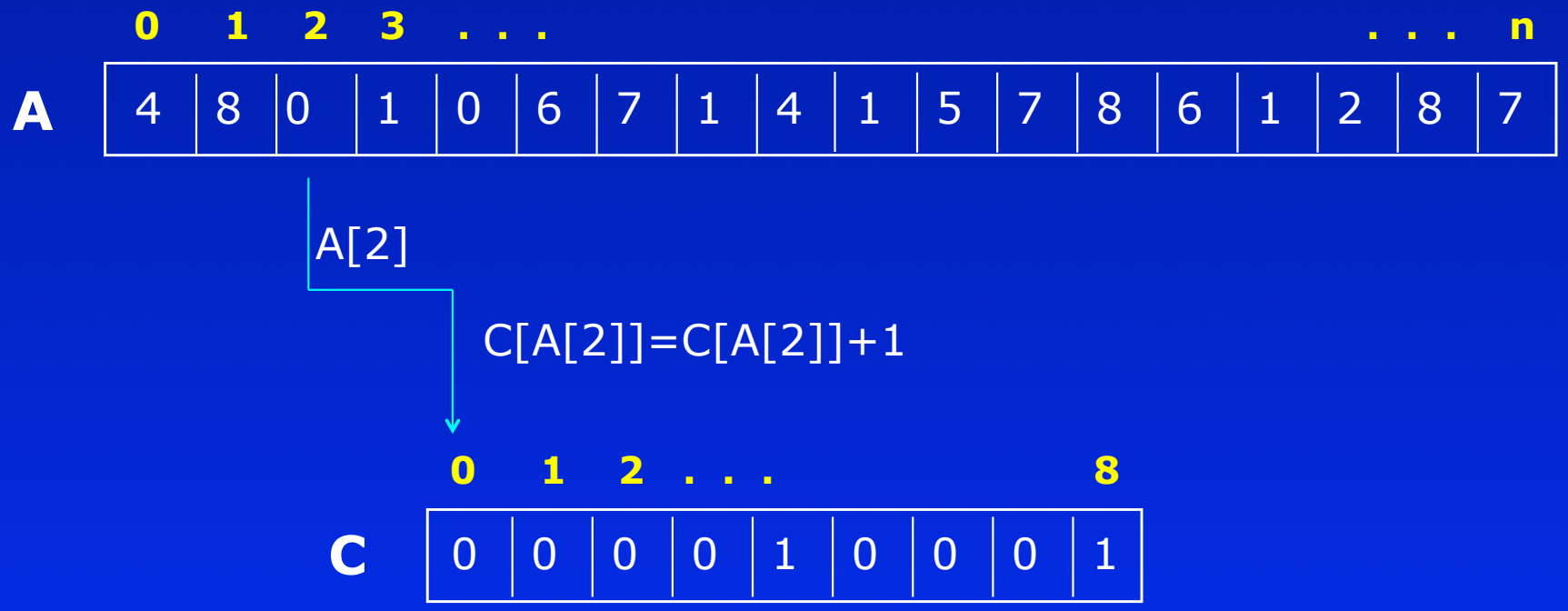
Counting sort



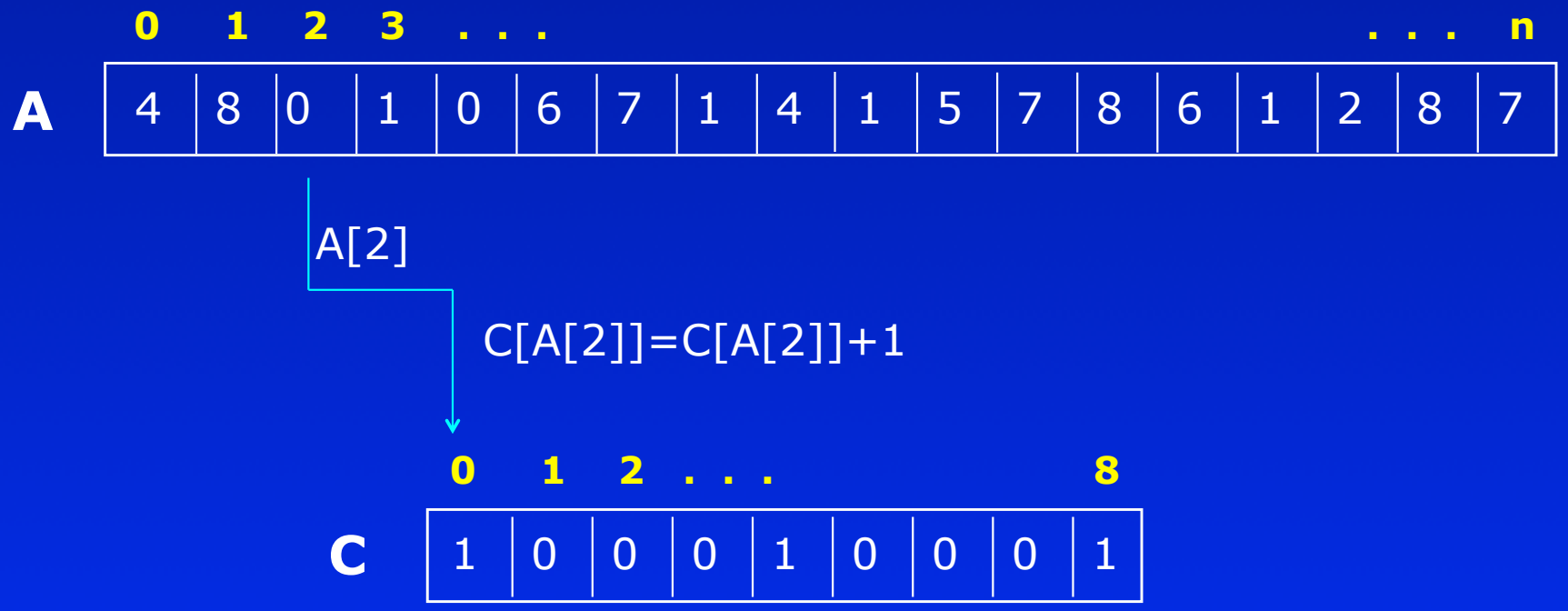
Counting sort



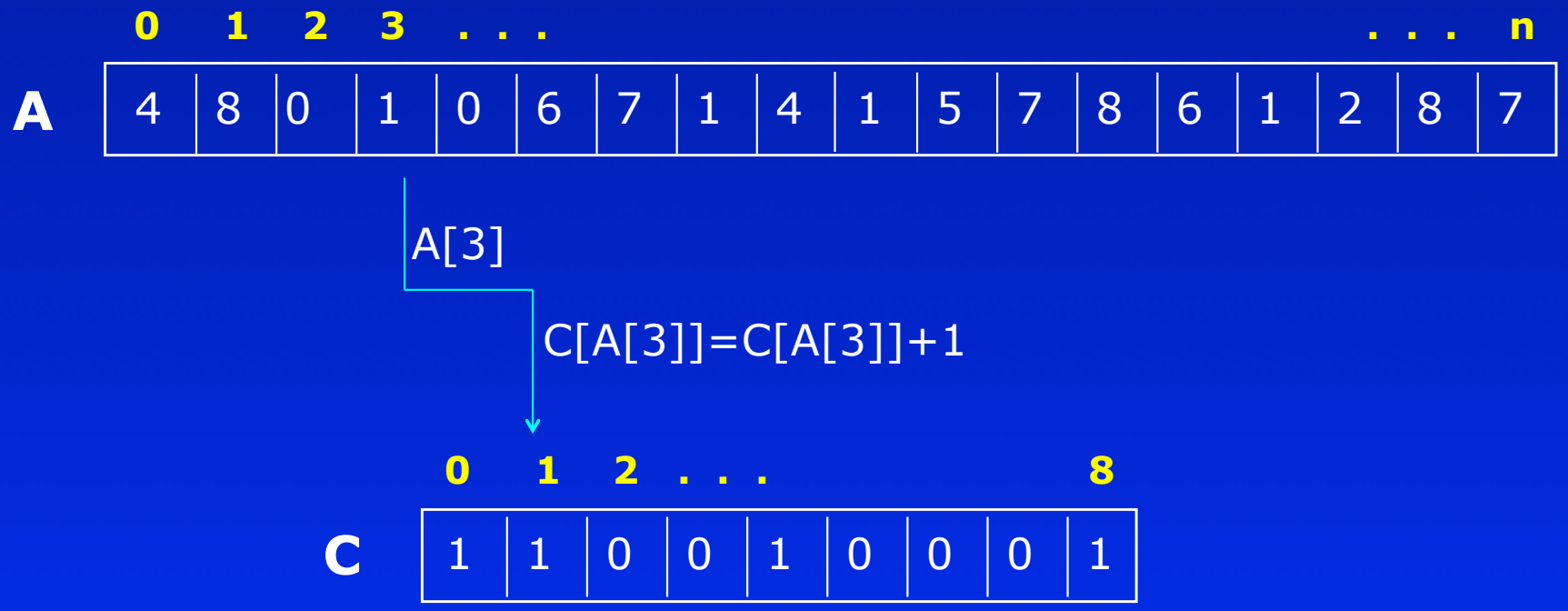
Counting sort



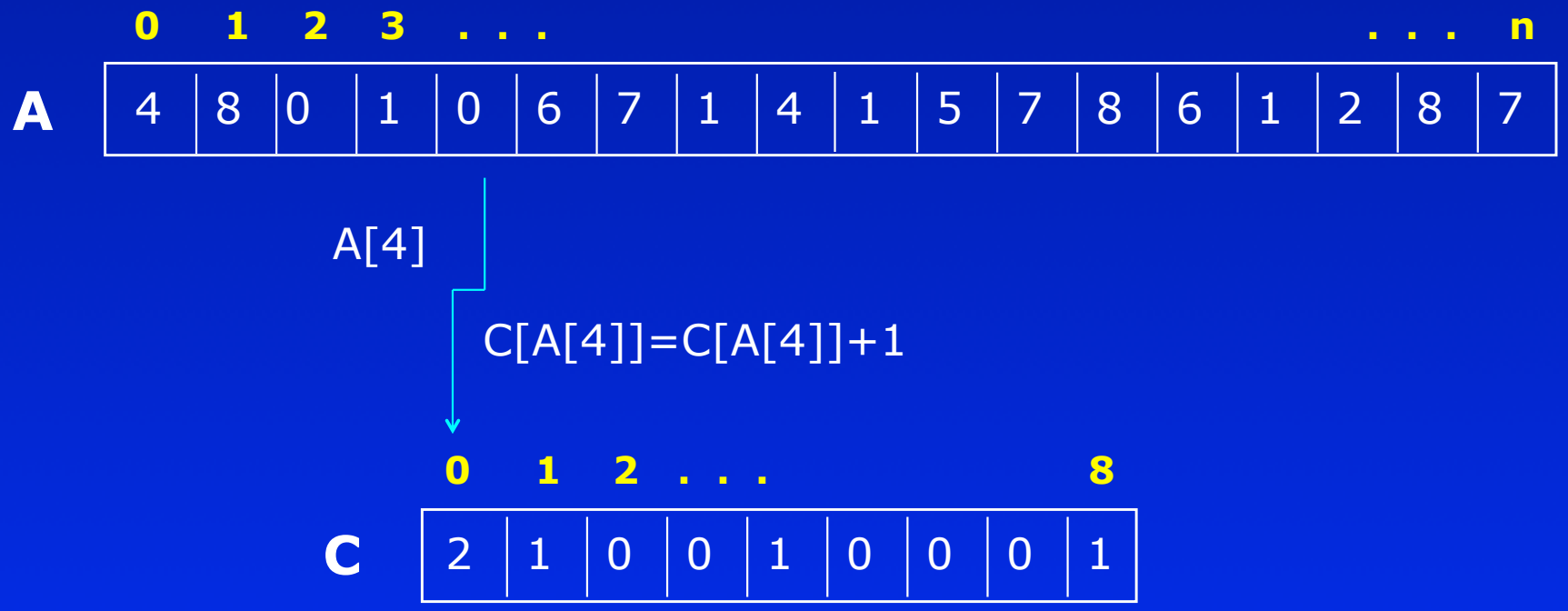
Counting sort



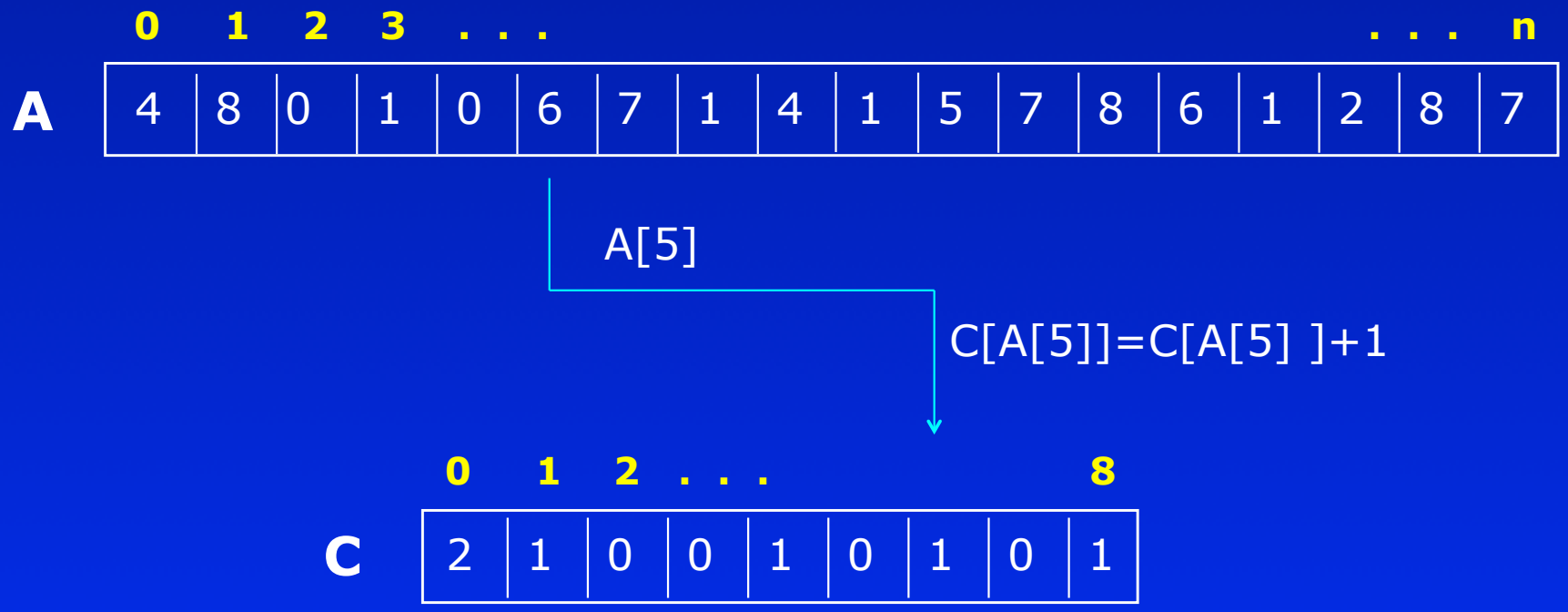
Counting sort



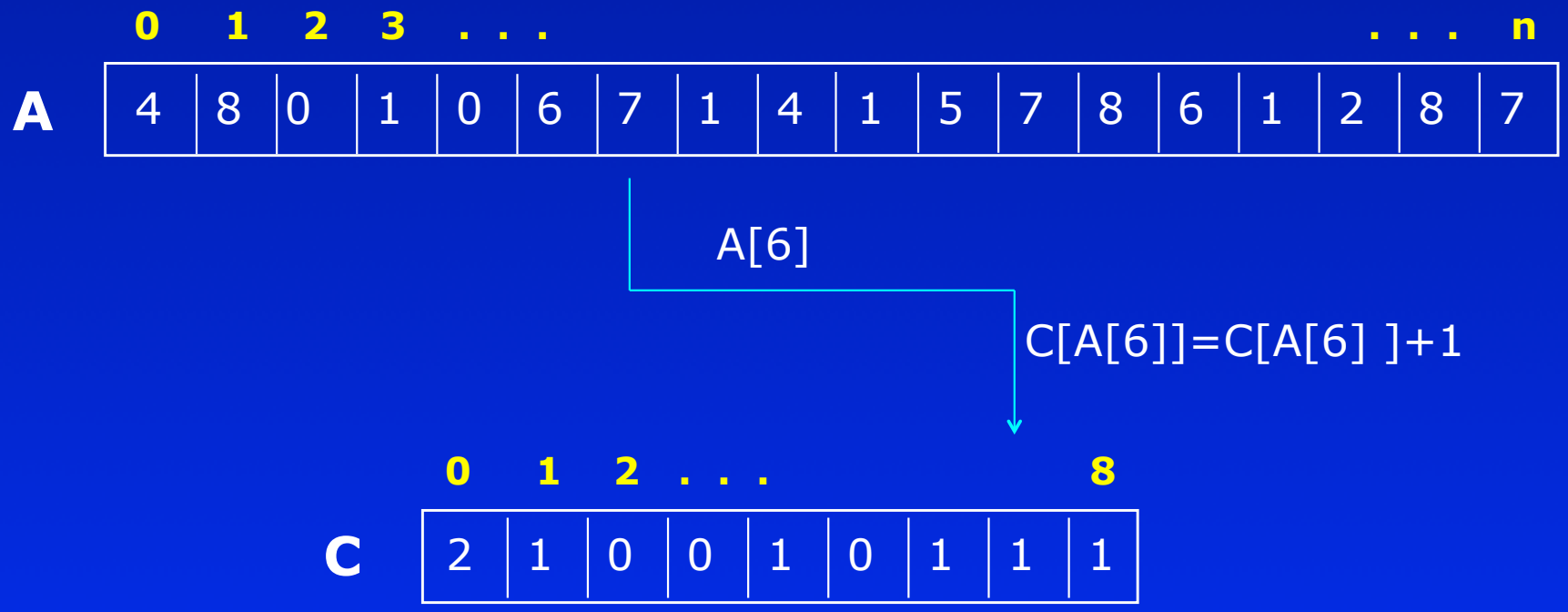
Counting sort



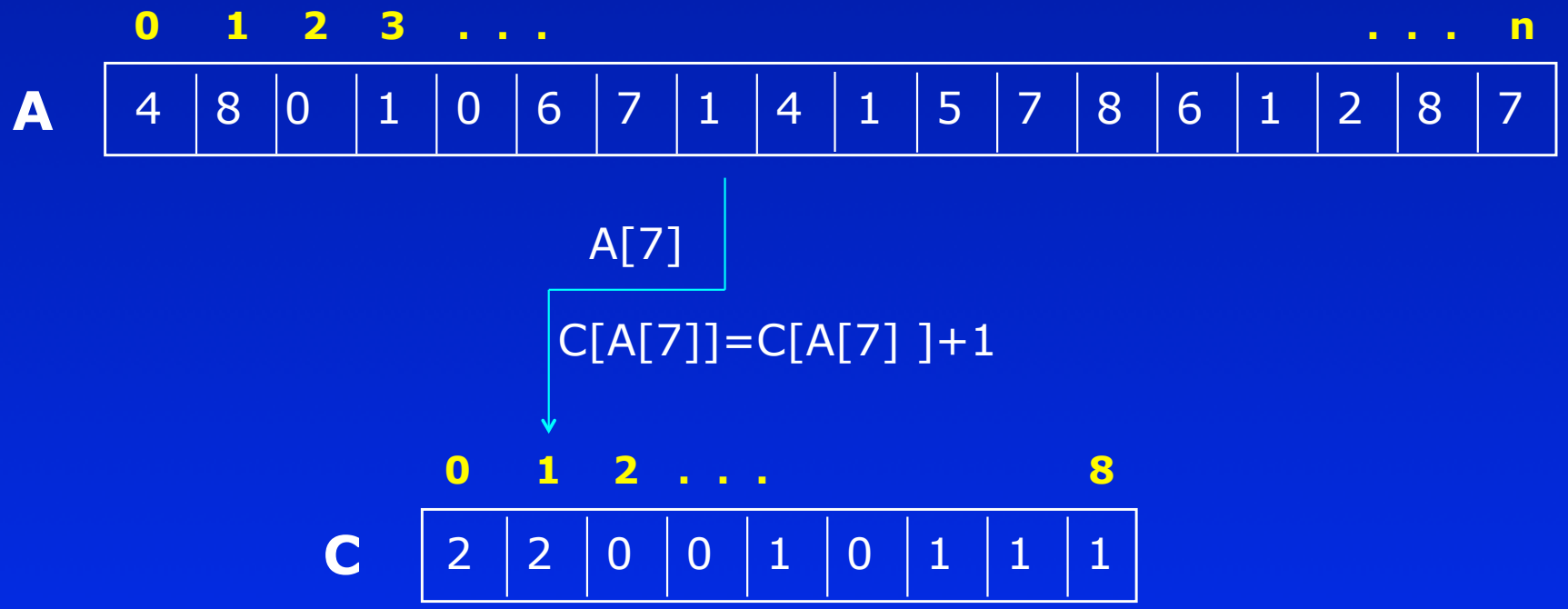
Counting sort



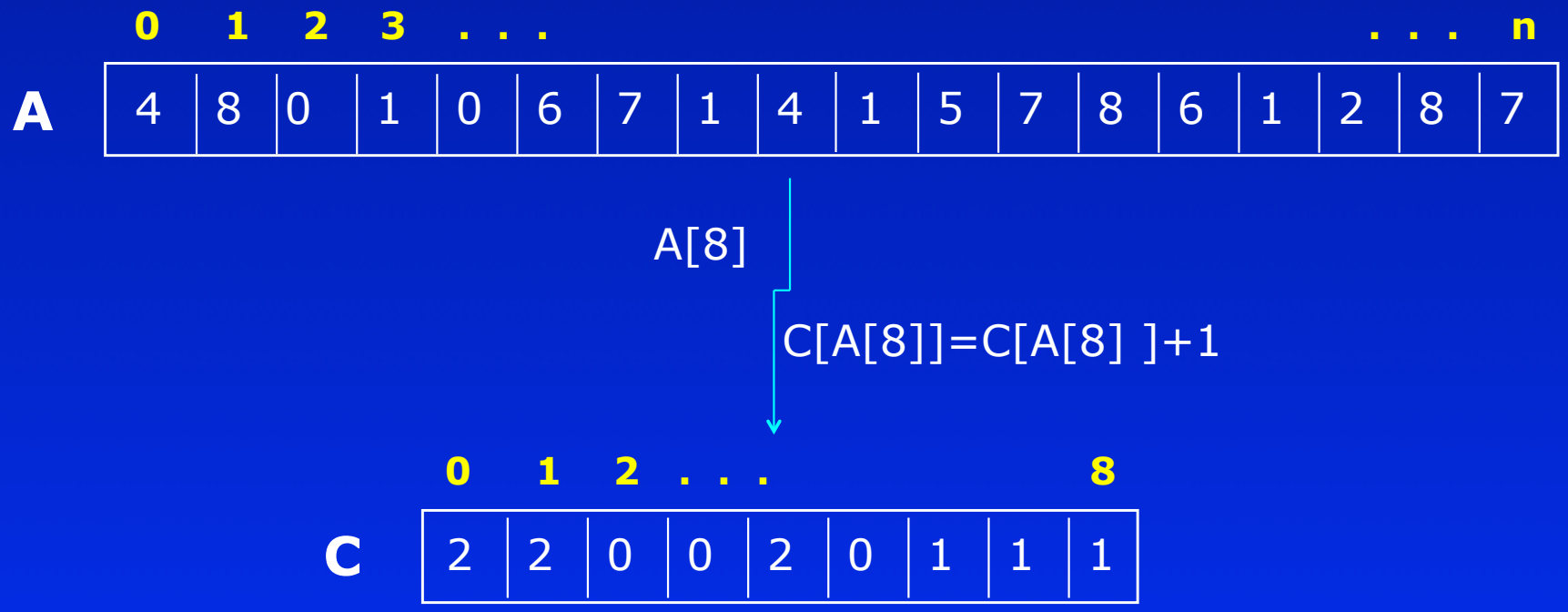
Counting sort



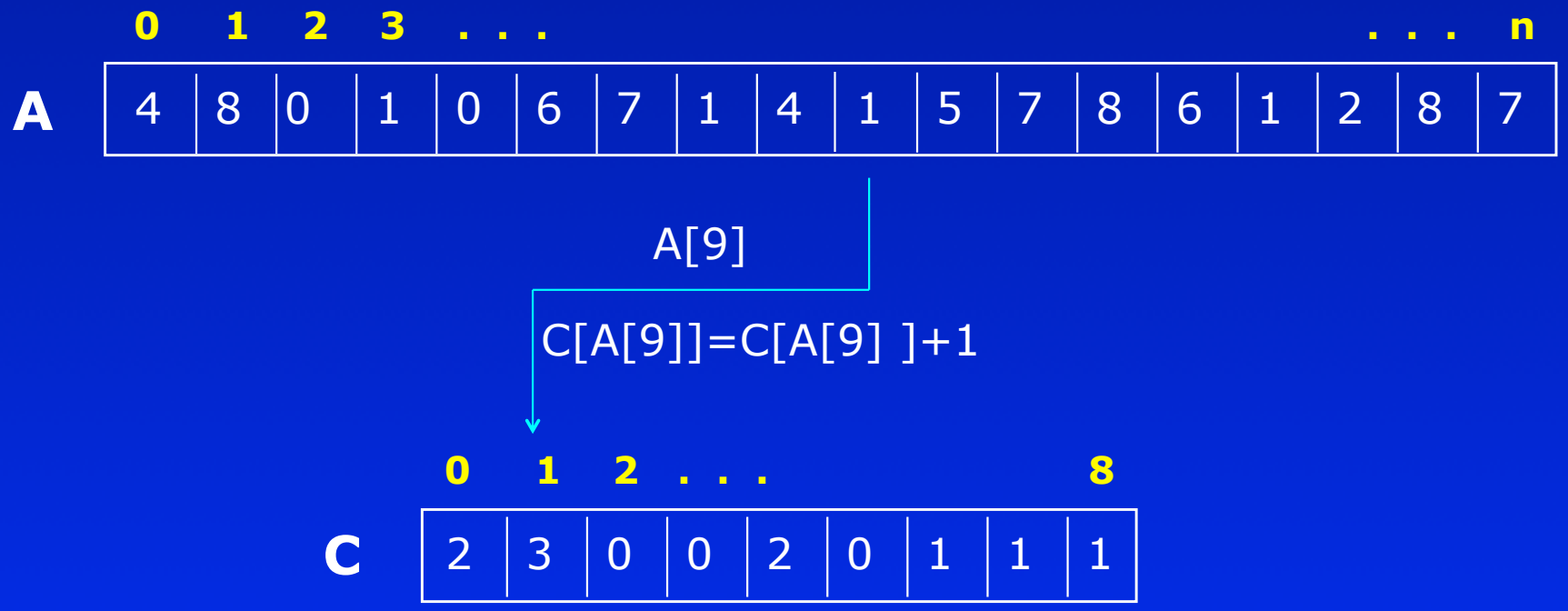
Counting sort



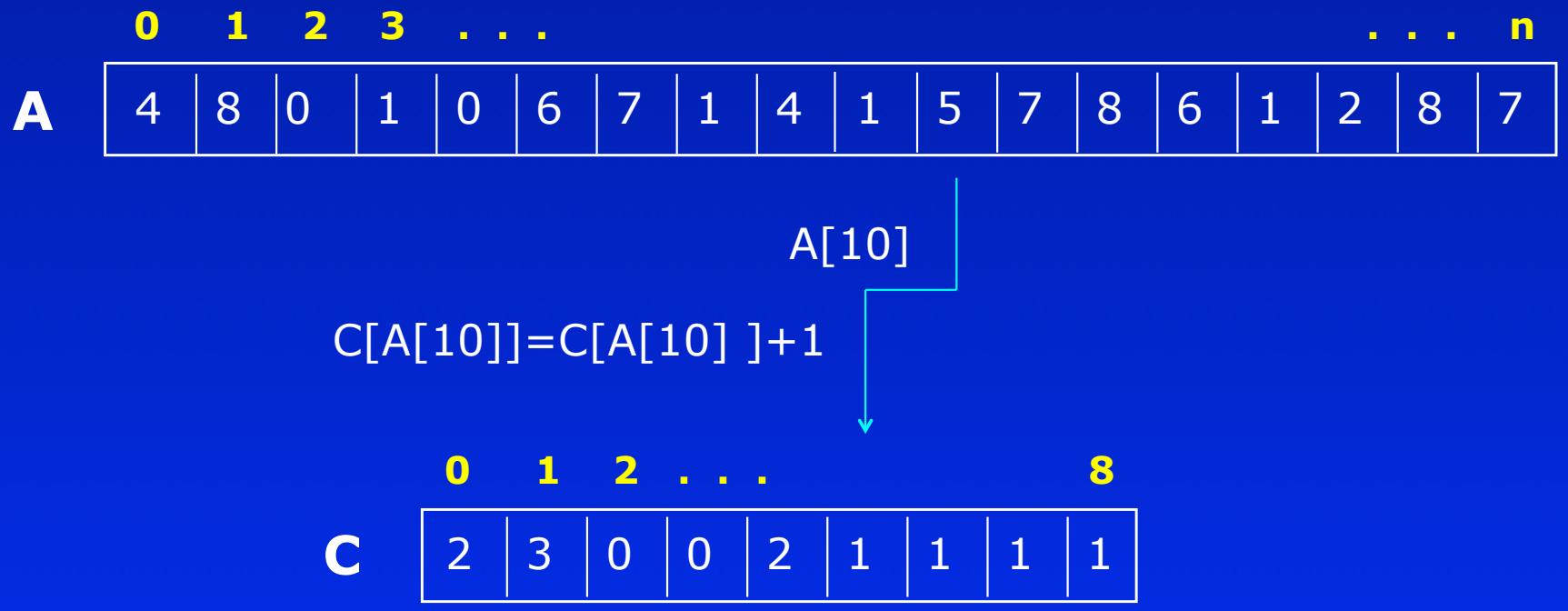
Counting sort



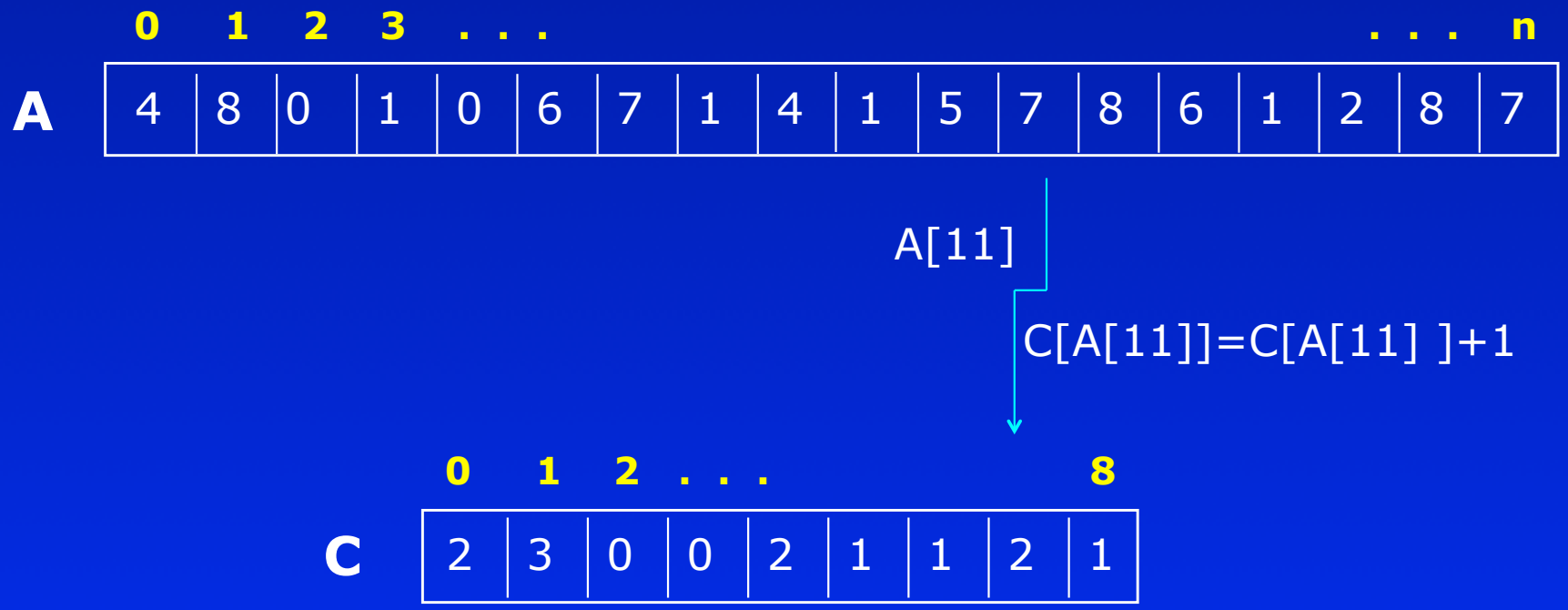
Counting sort



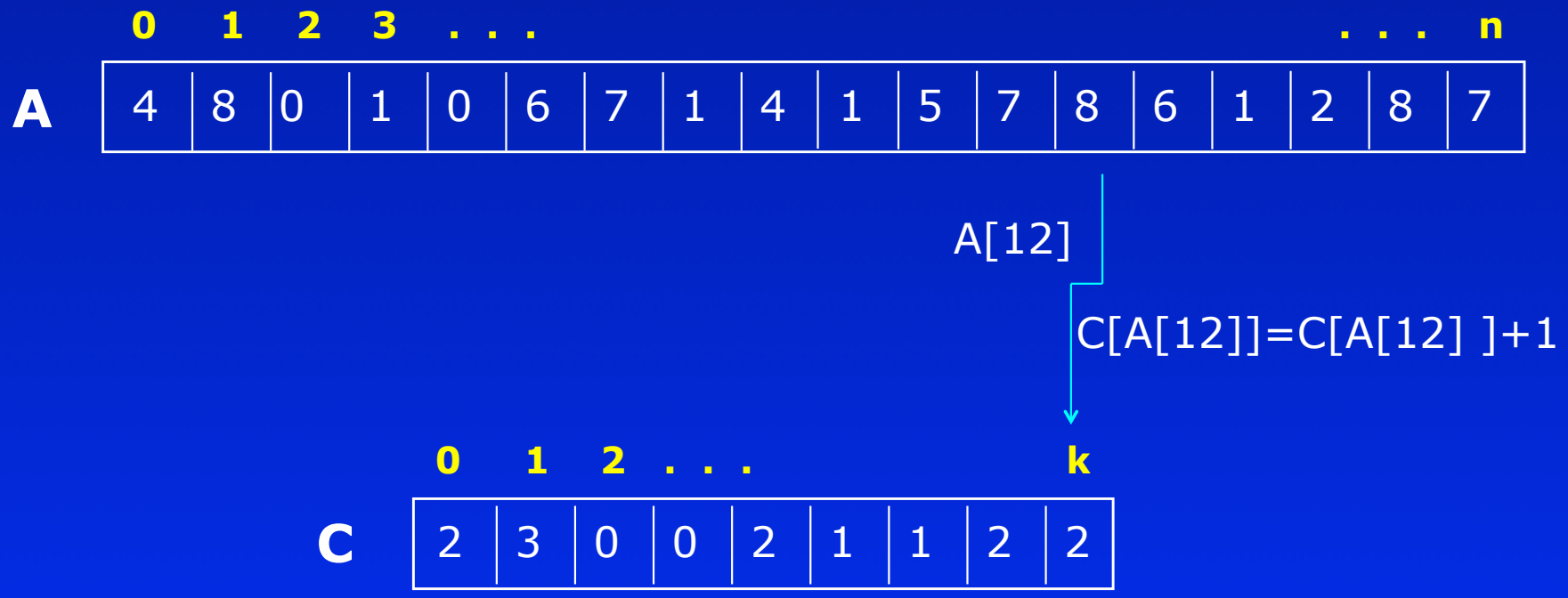
Counting sort



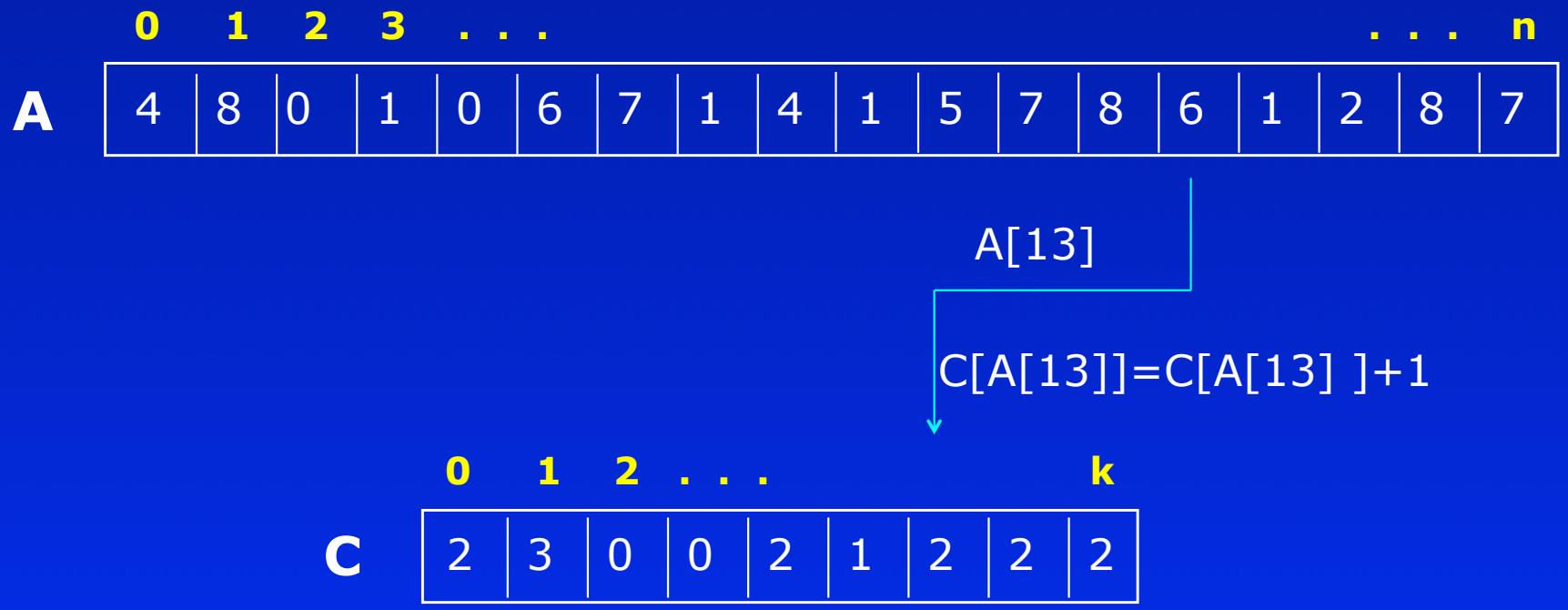
Counting sort



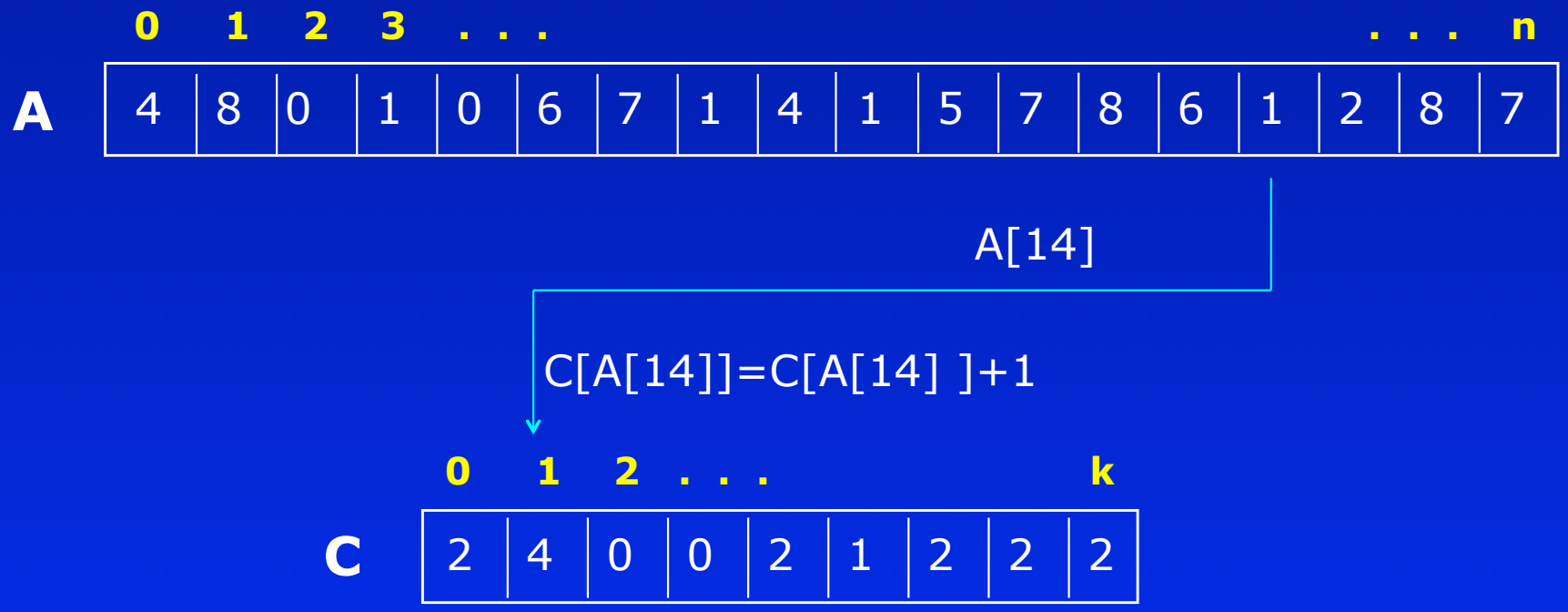
Counting sort



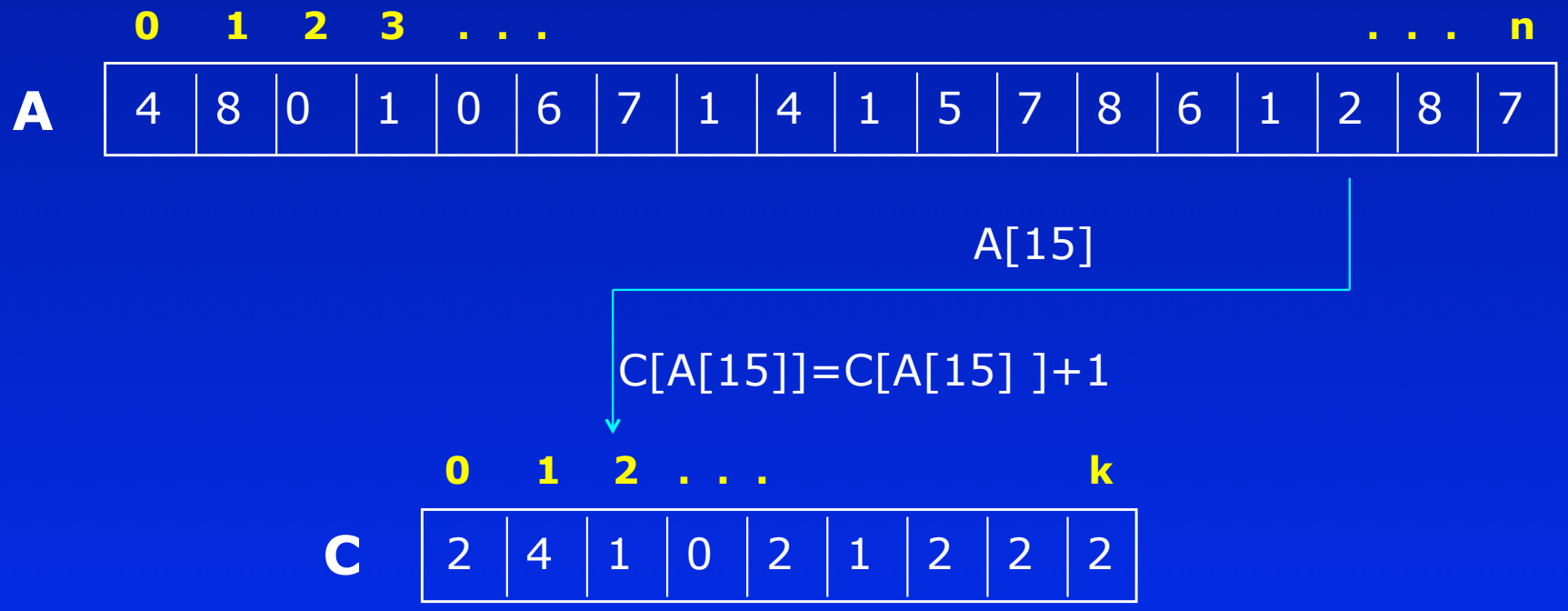
Counting sort



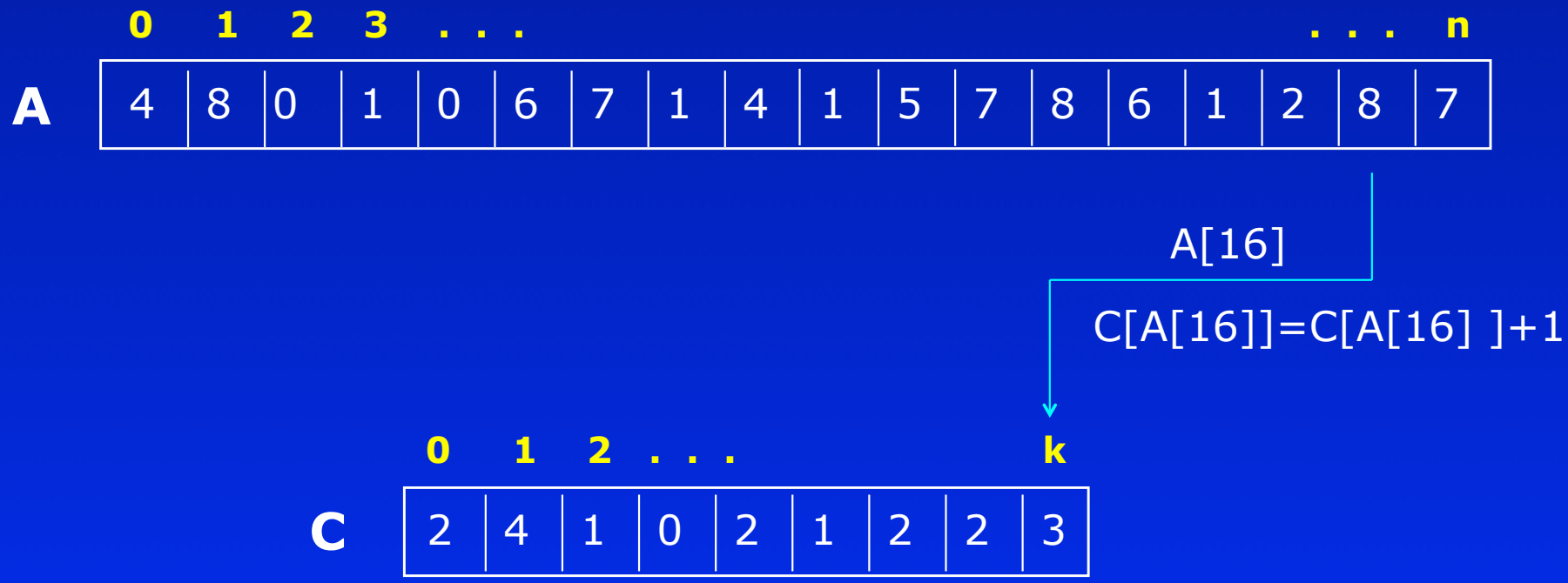
Counting sort



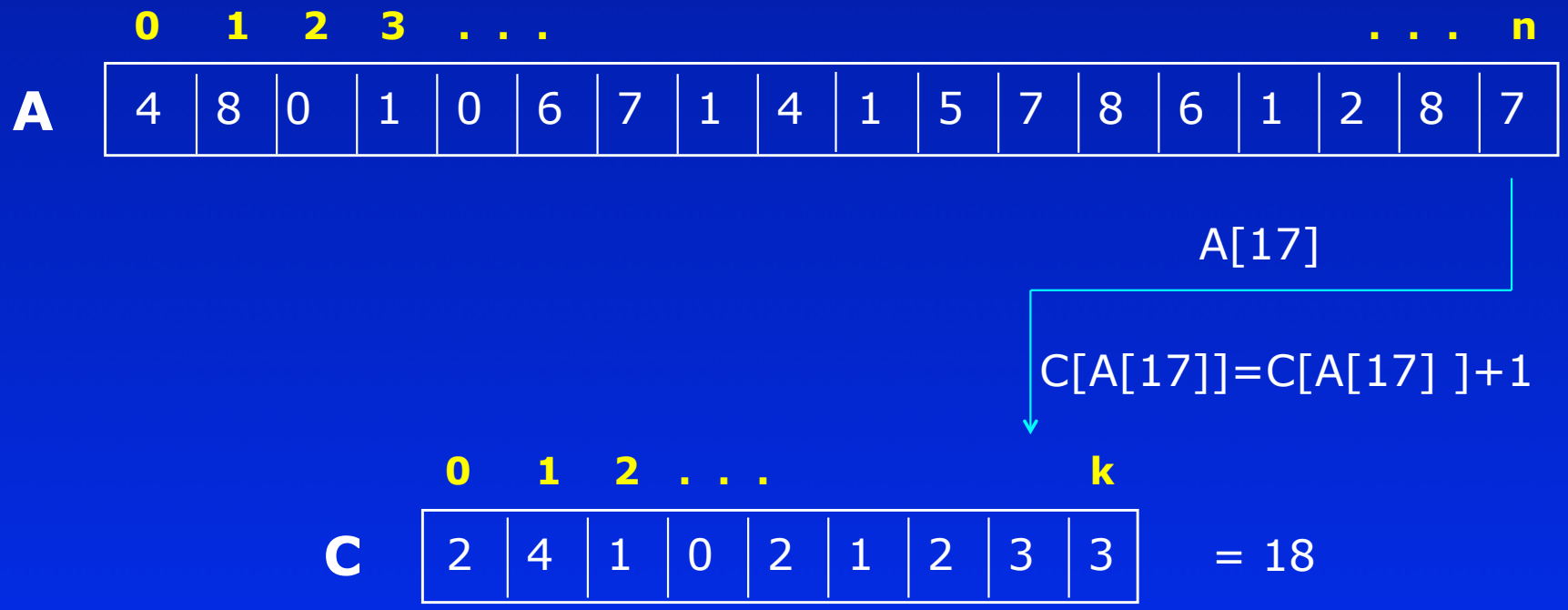
Counting sort



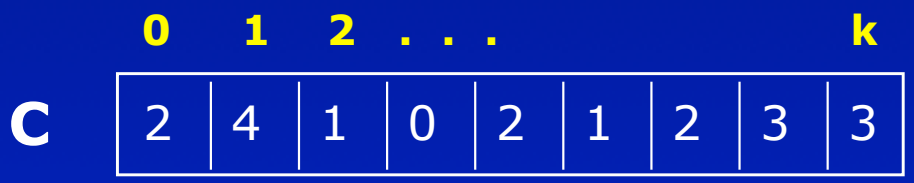
Counting sort



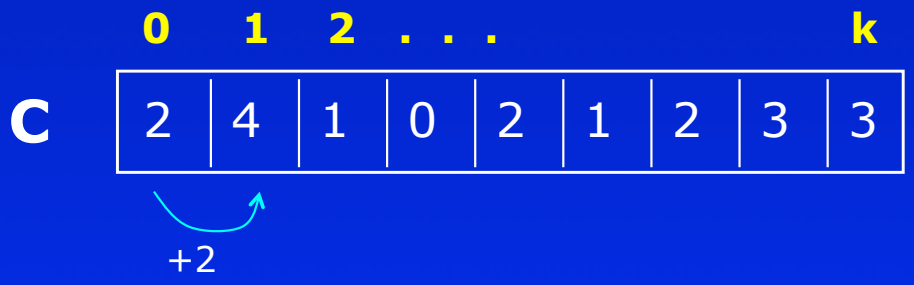
Counting sort



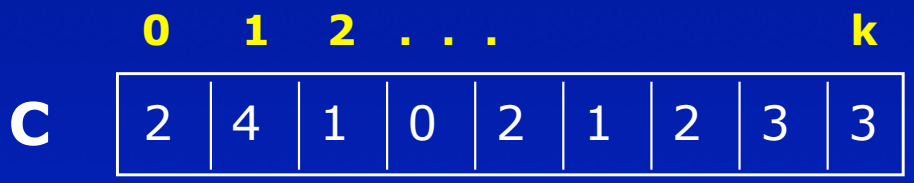
Counting sort



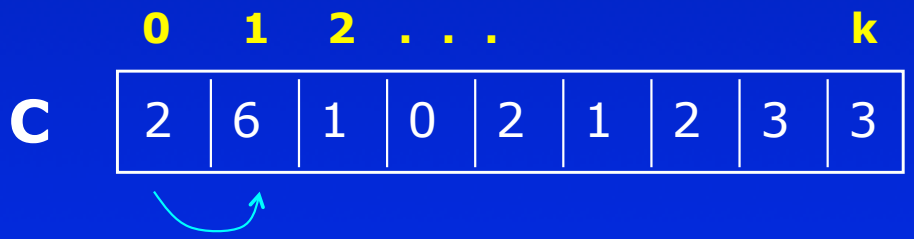
C[i] = C[i] + C[i-1]



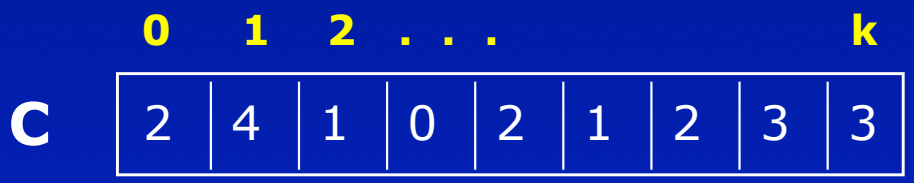
Counting sort



C[i] = C[i] + C[i-1]

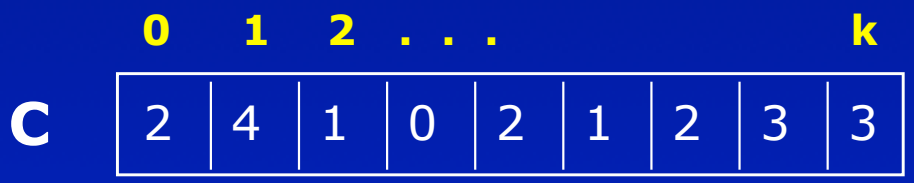


Counting sort

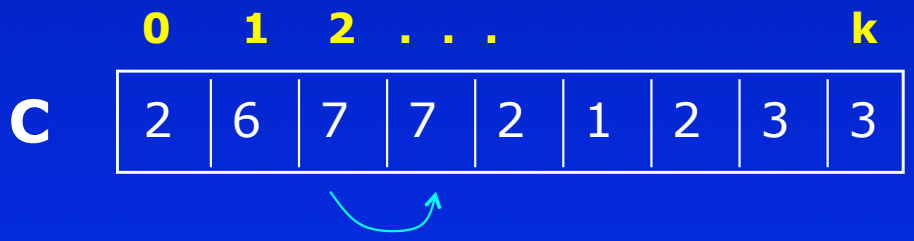


C[i] = C[i] + C[i-1]

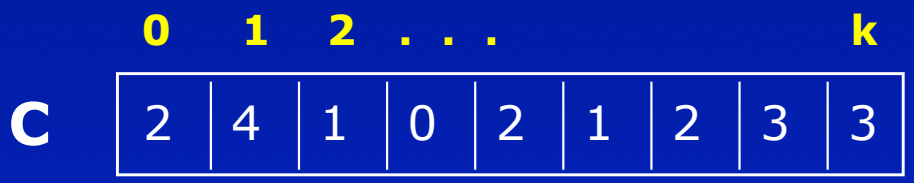
Counting sort



C[i] = C[i] + C[i-1]

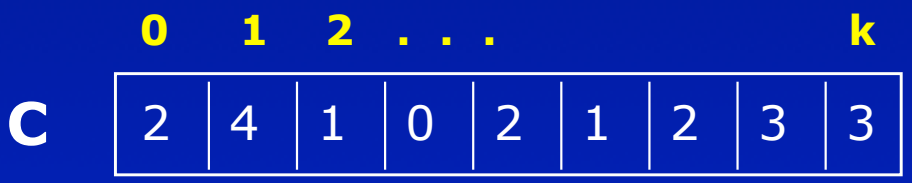


Counting sort

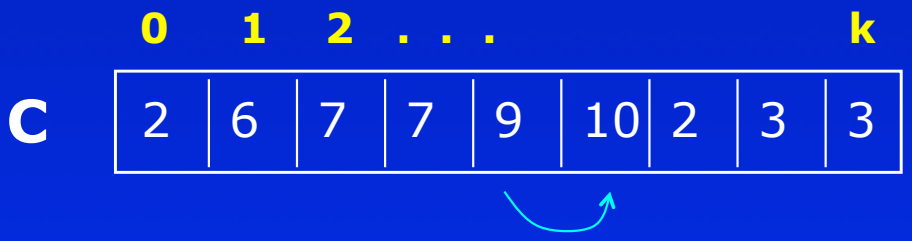


C[i] = C[i] + C[i-1]

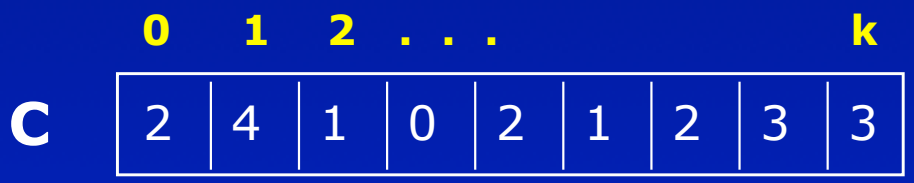
Counting sort



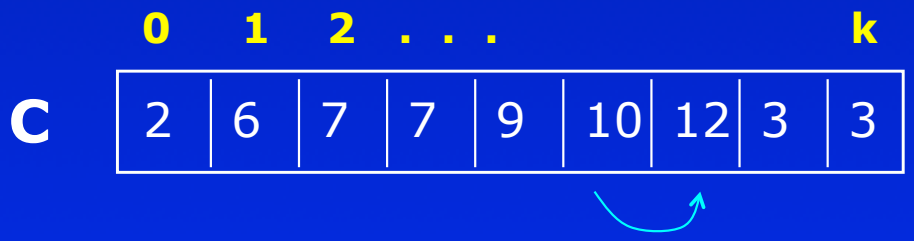
C[i] = C[i] + C[i-1]



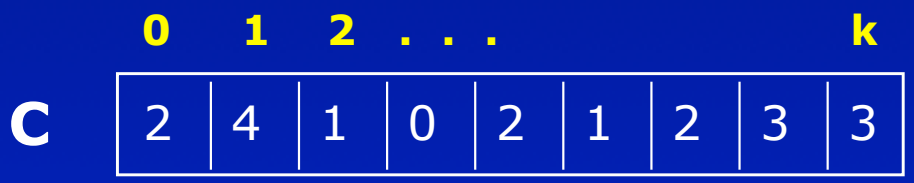
Counting sort



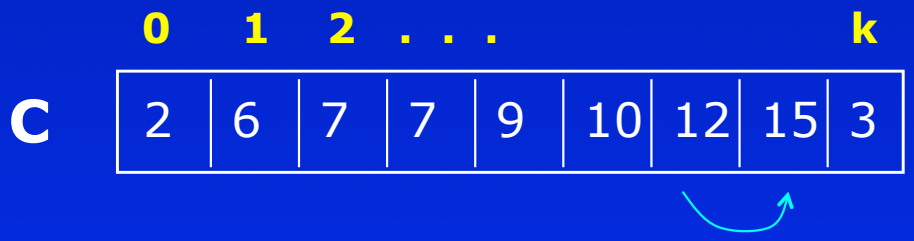
C[i] = C[i] + C[i-1]



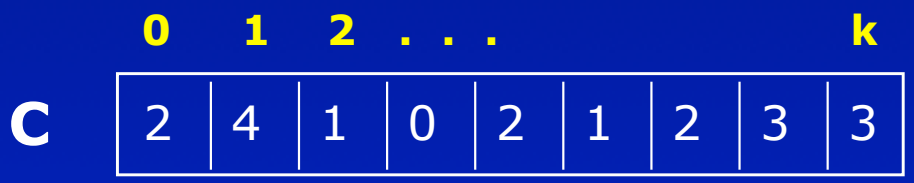
Counting sort



C[i] = C[i] + C[i-1]



Counting sort

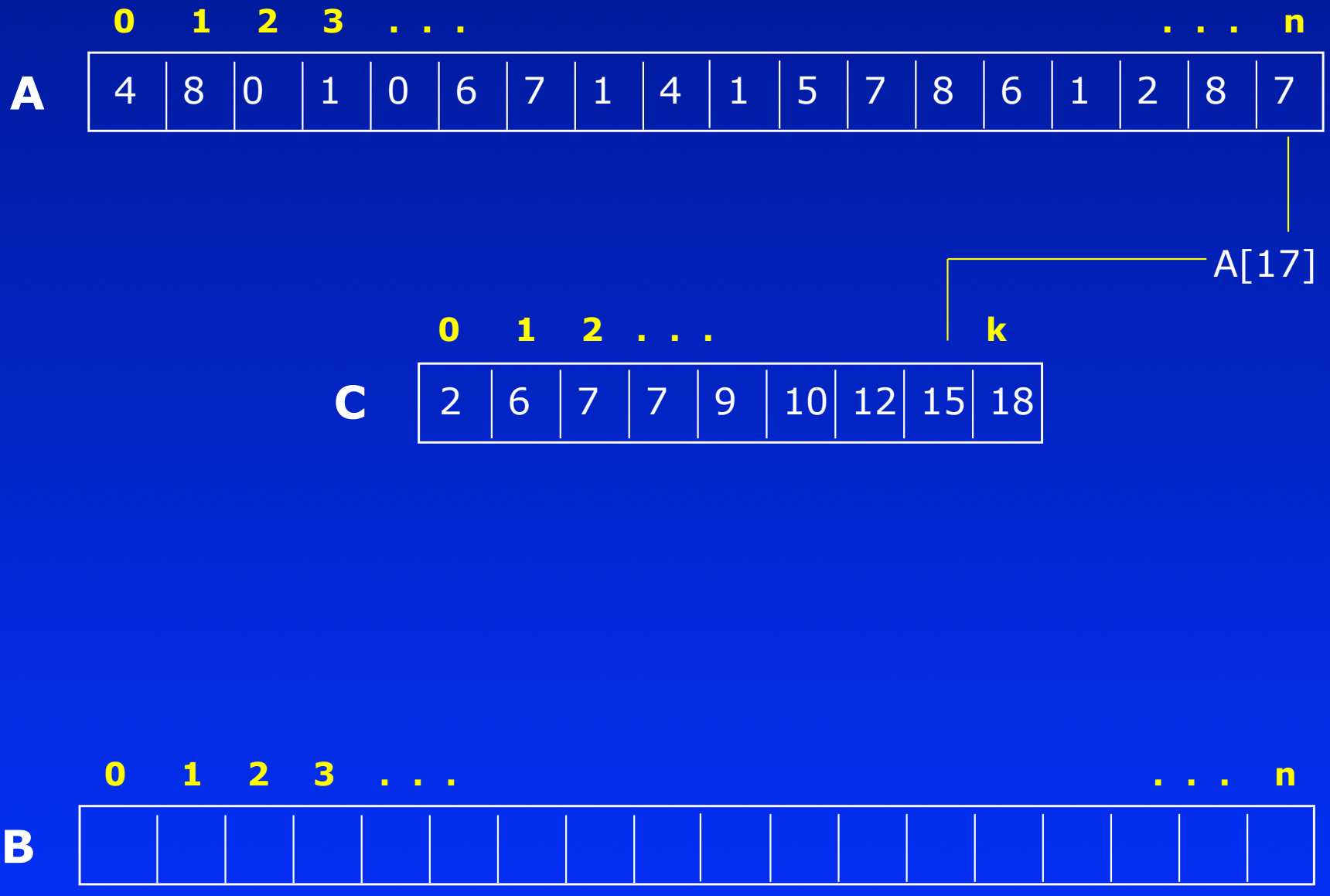


C[i] = C[i] + C[i-1]

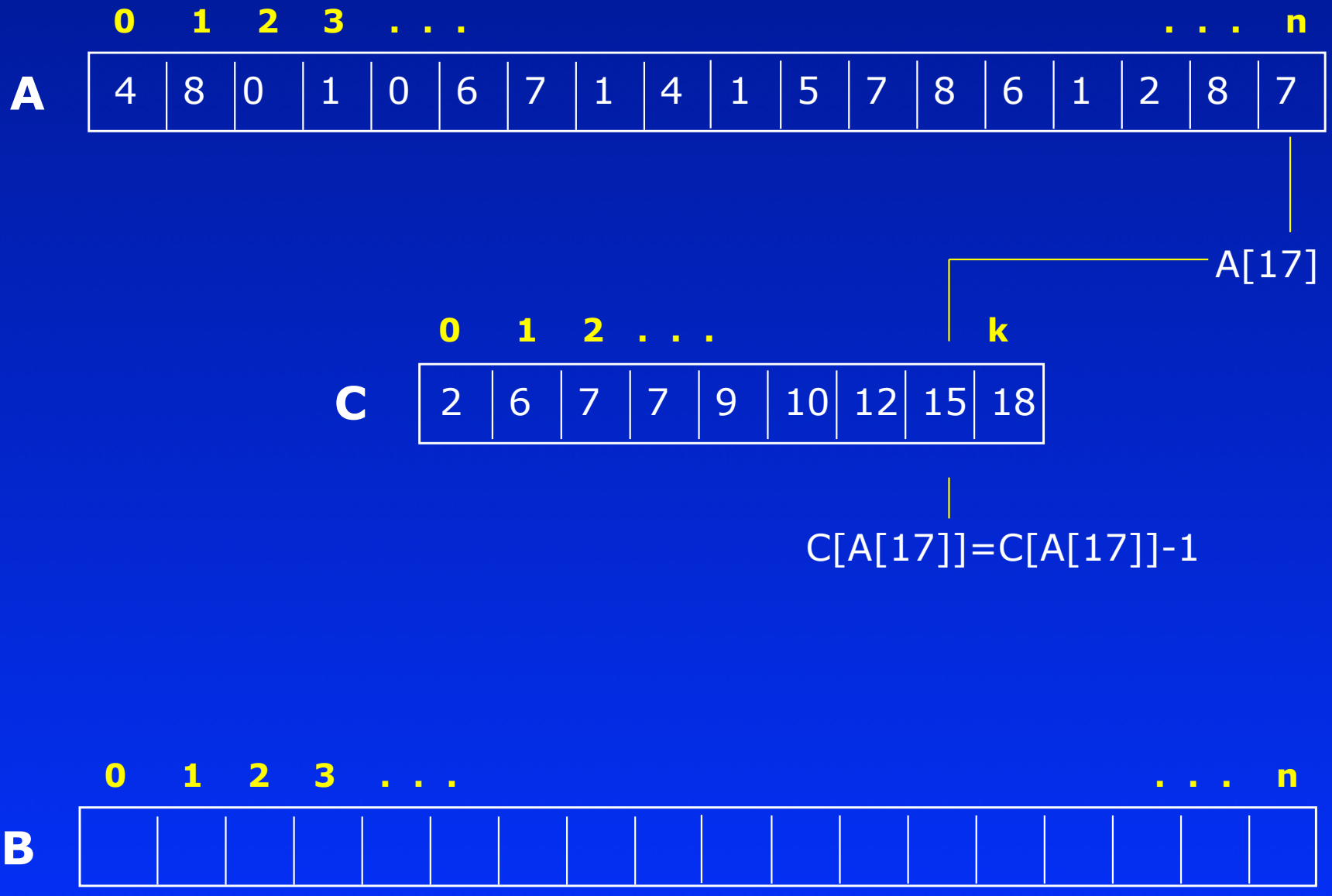
Jetzt beinhalte C[i] alle Elemente, die kleiner oder gleich **i** sind.

9 Zahlen sind kleiner oder gleich 4

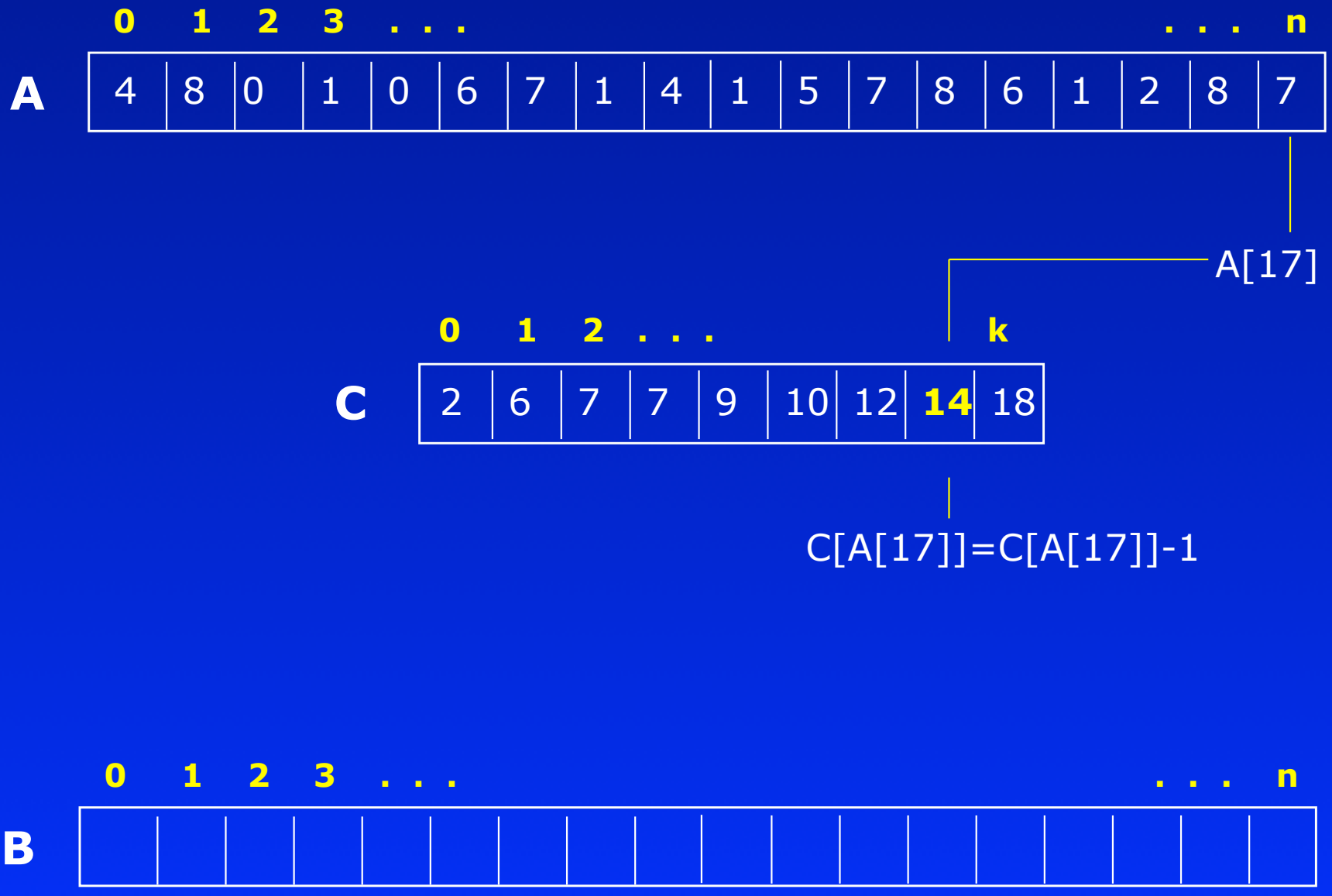
Counting sort



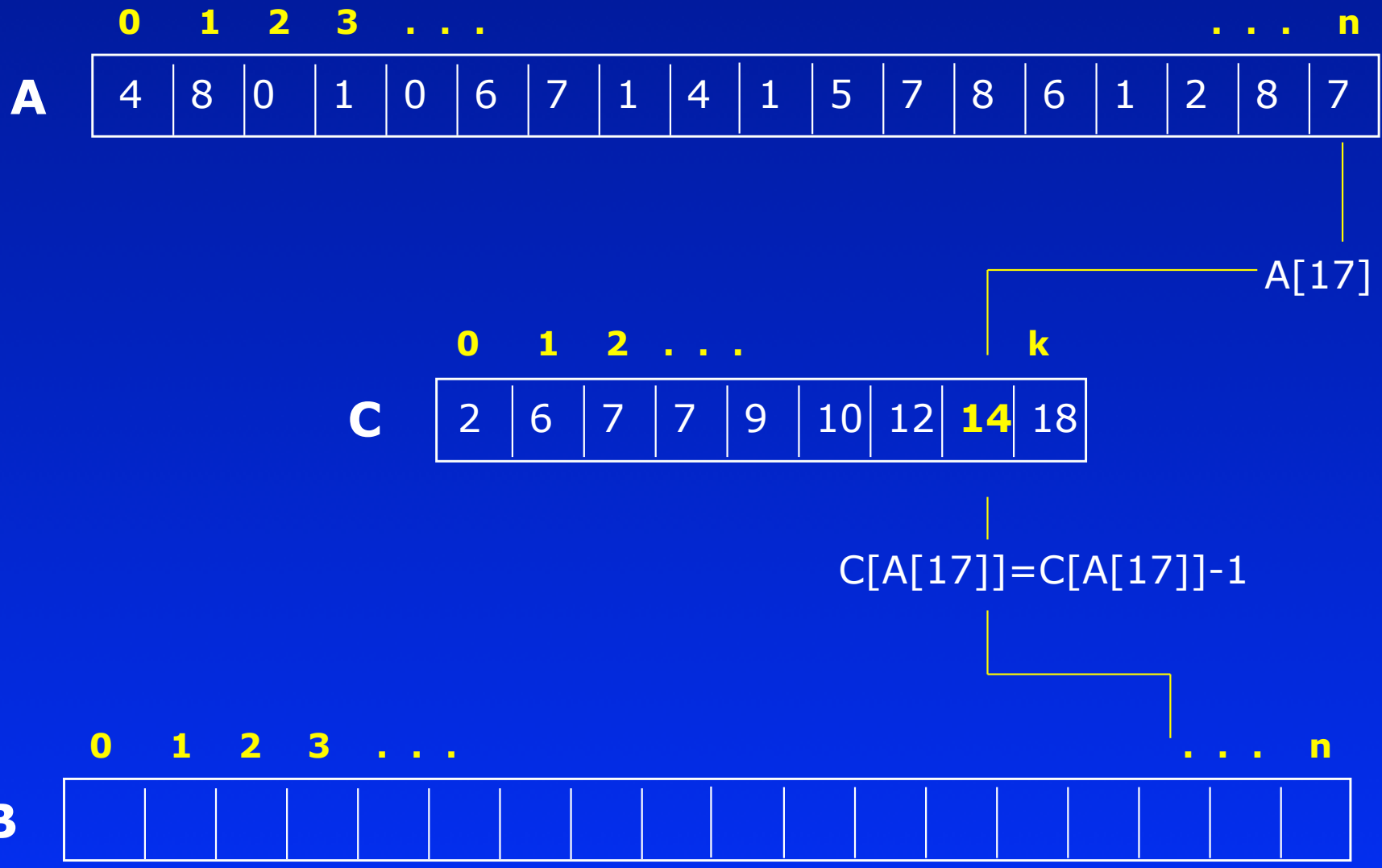
Counting sort



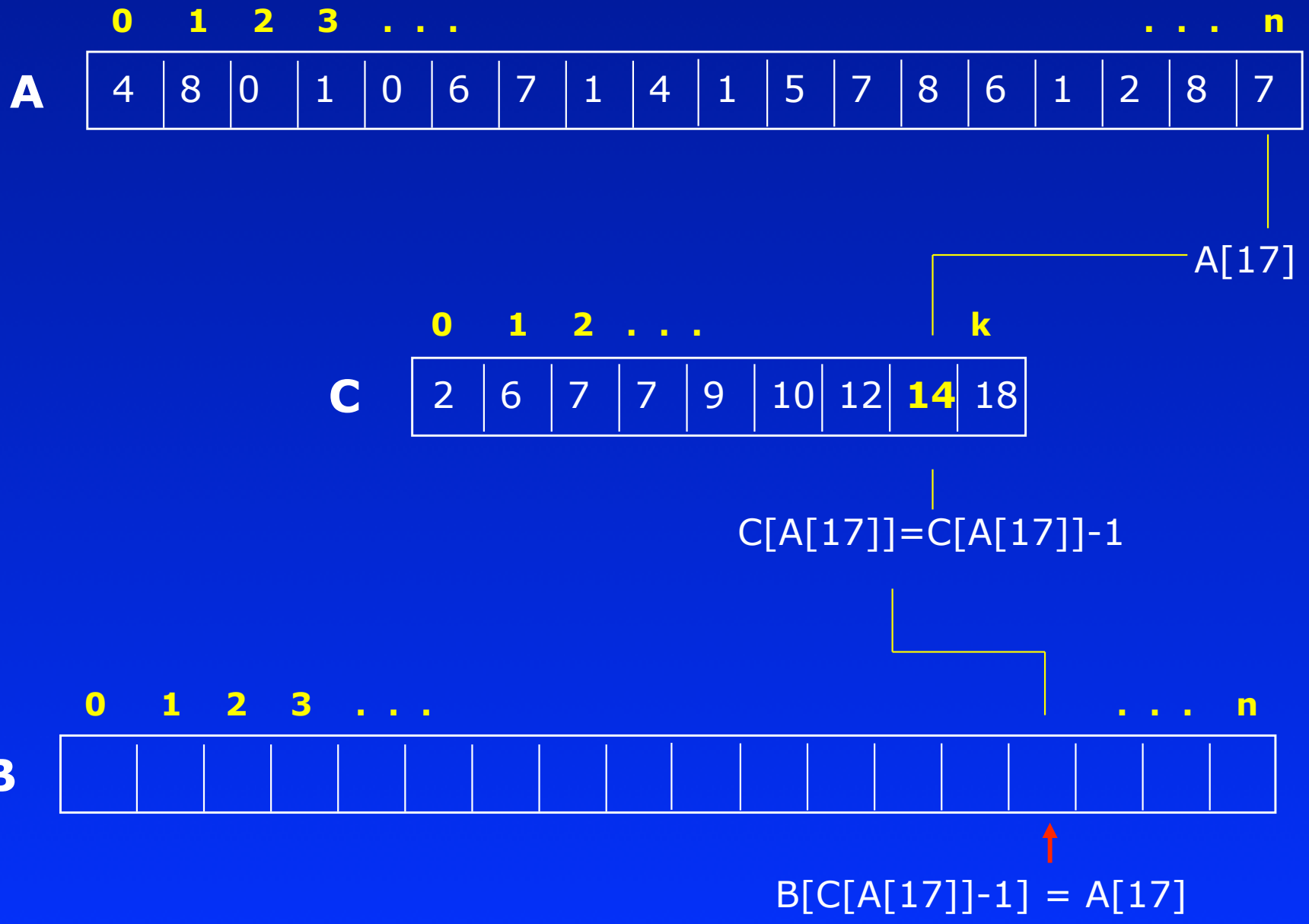
Counting sort



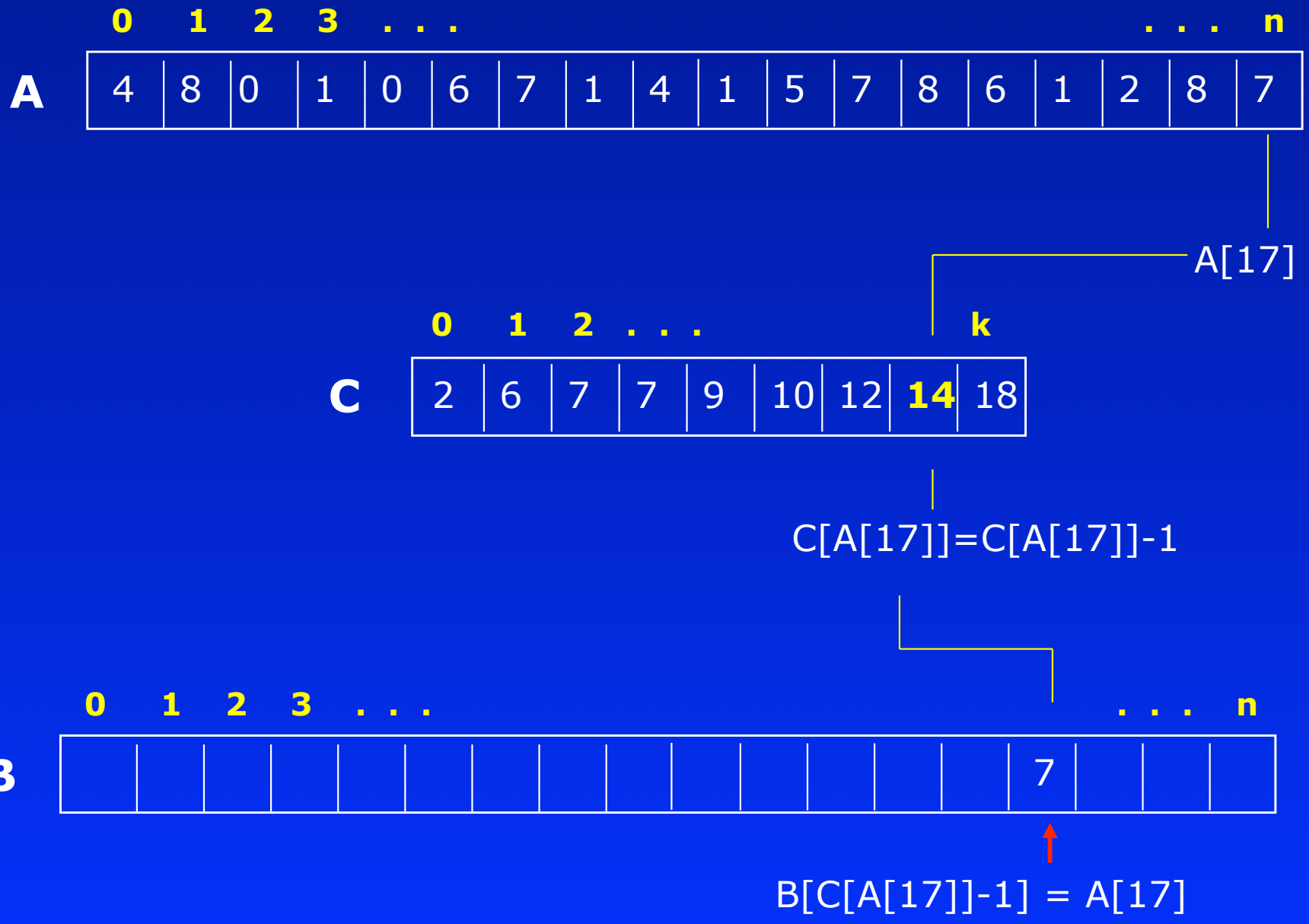
Counting sort



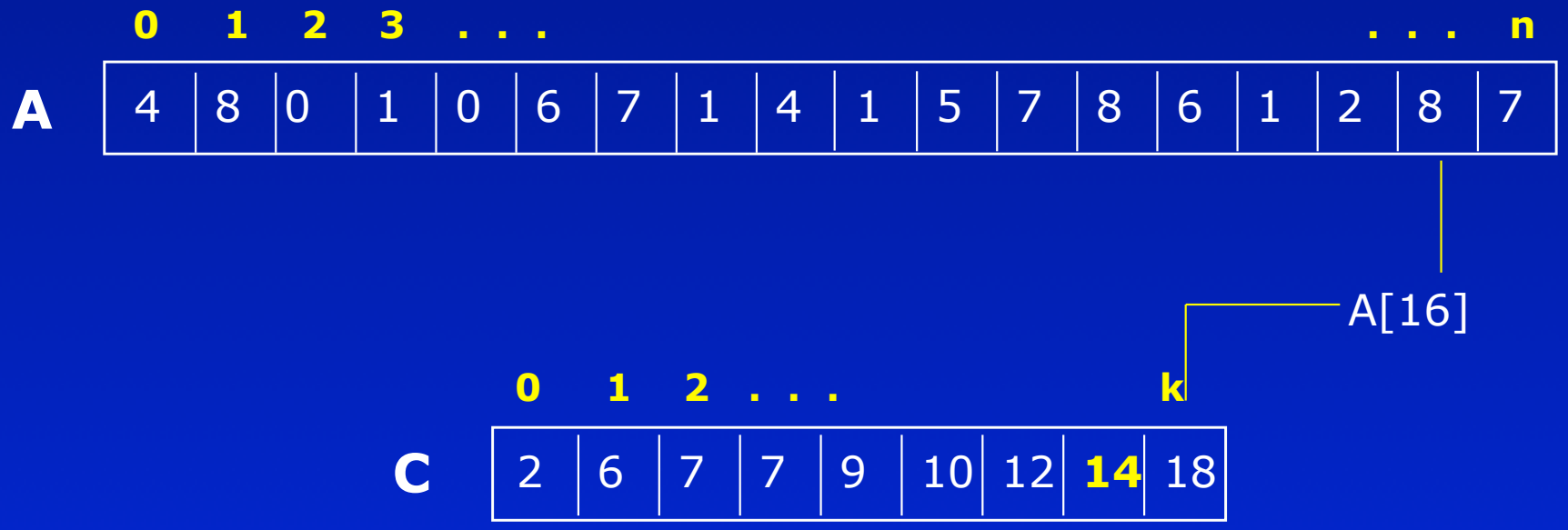
Counting sort



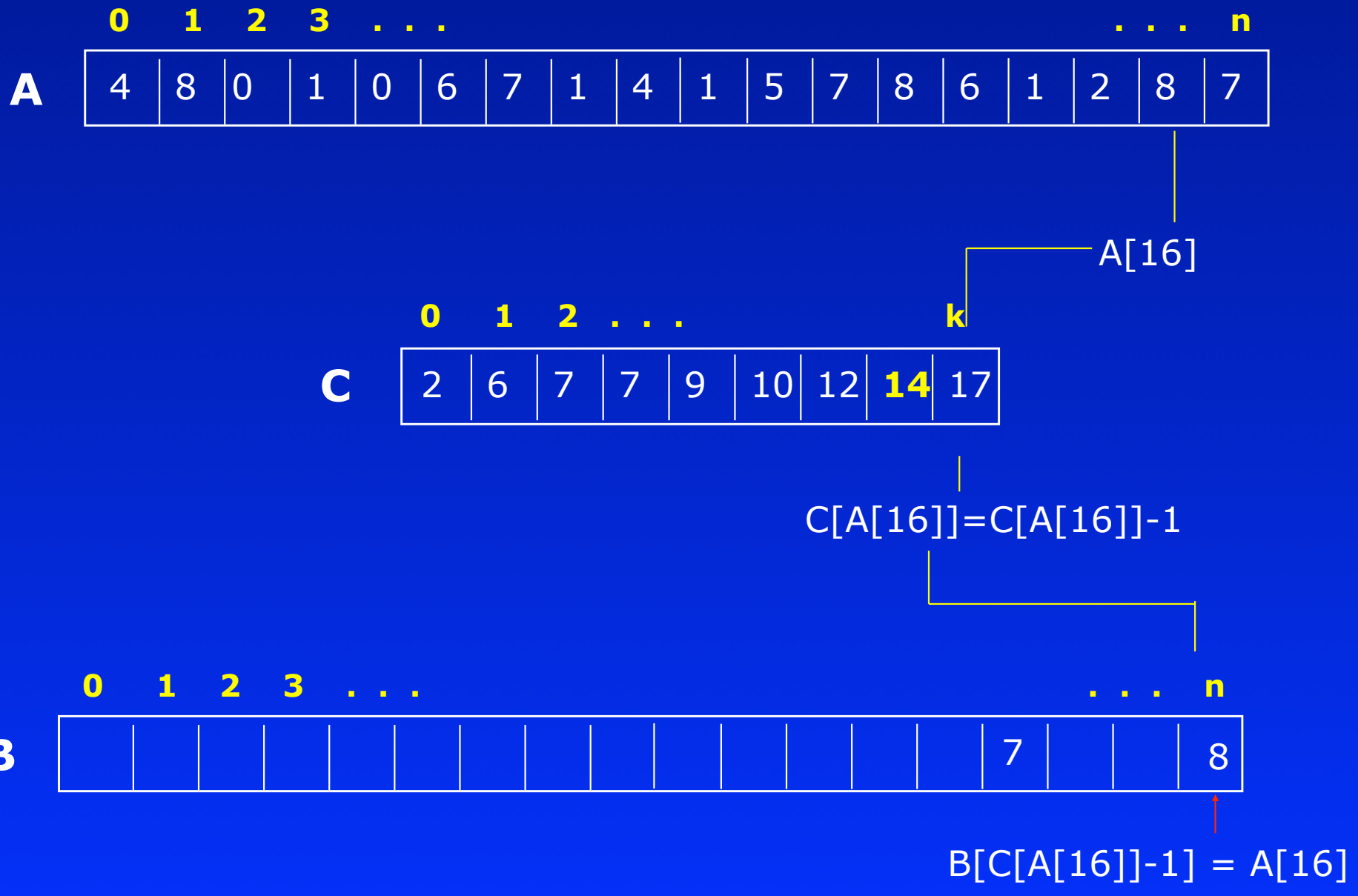
Counting sort



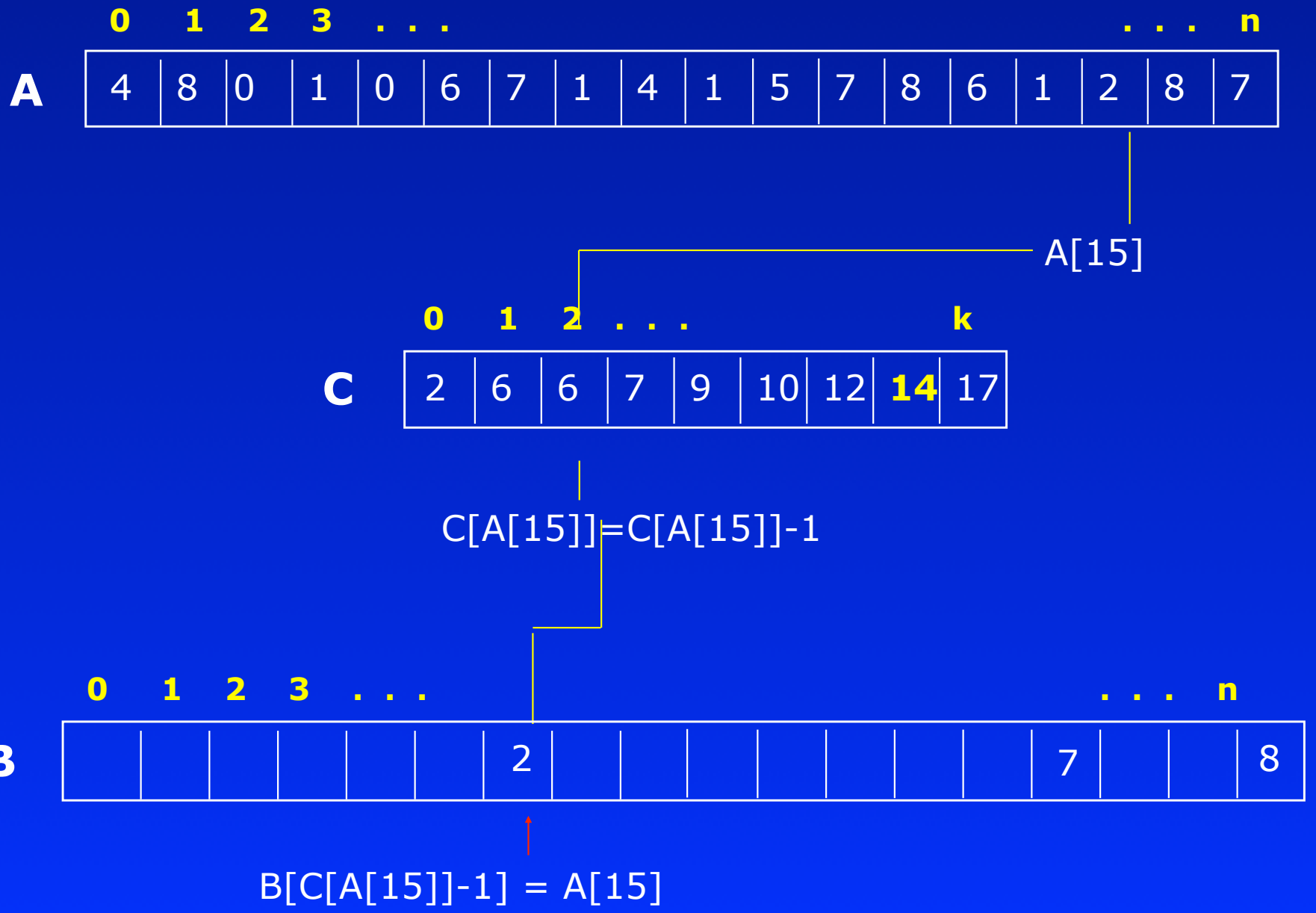
Counting sort



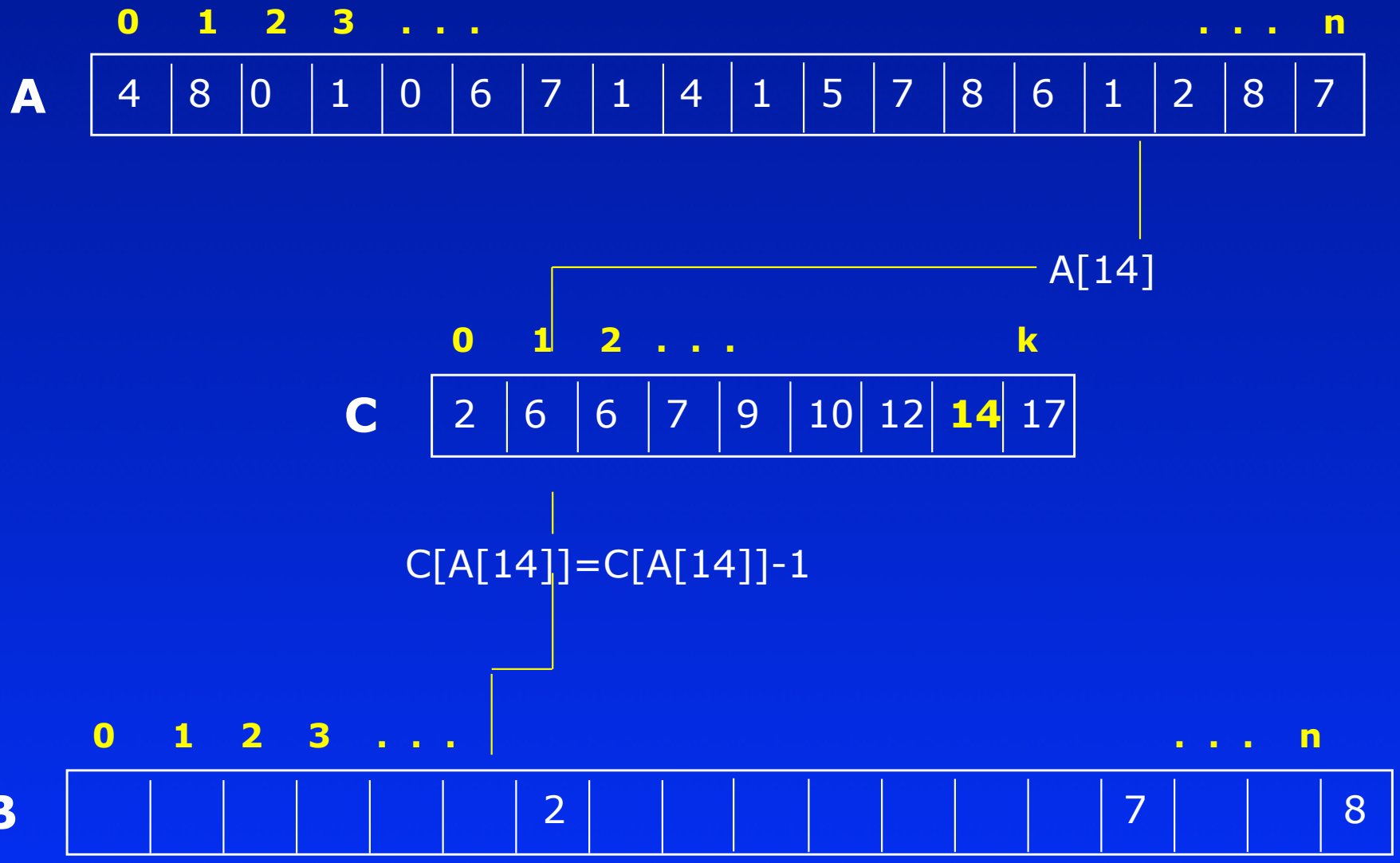
Counting sort



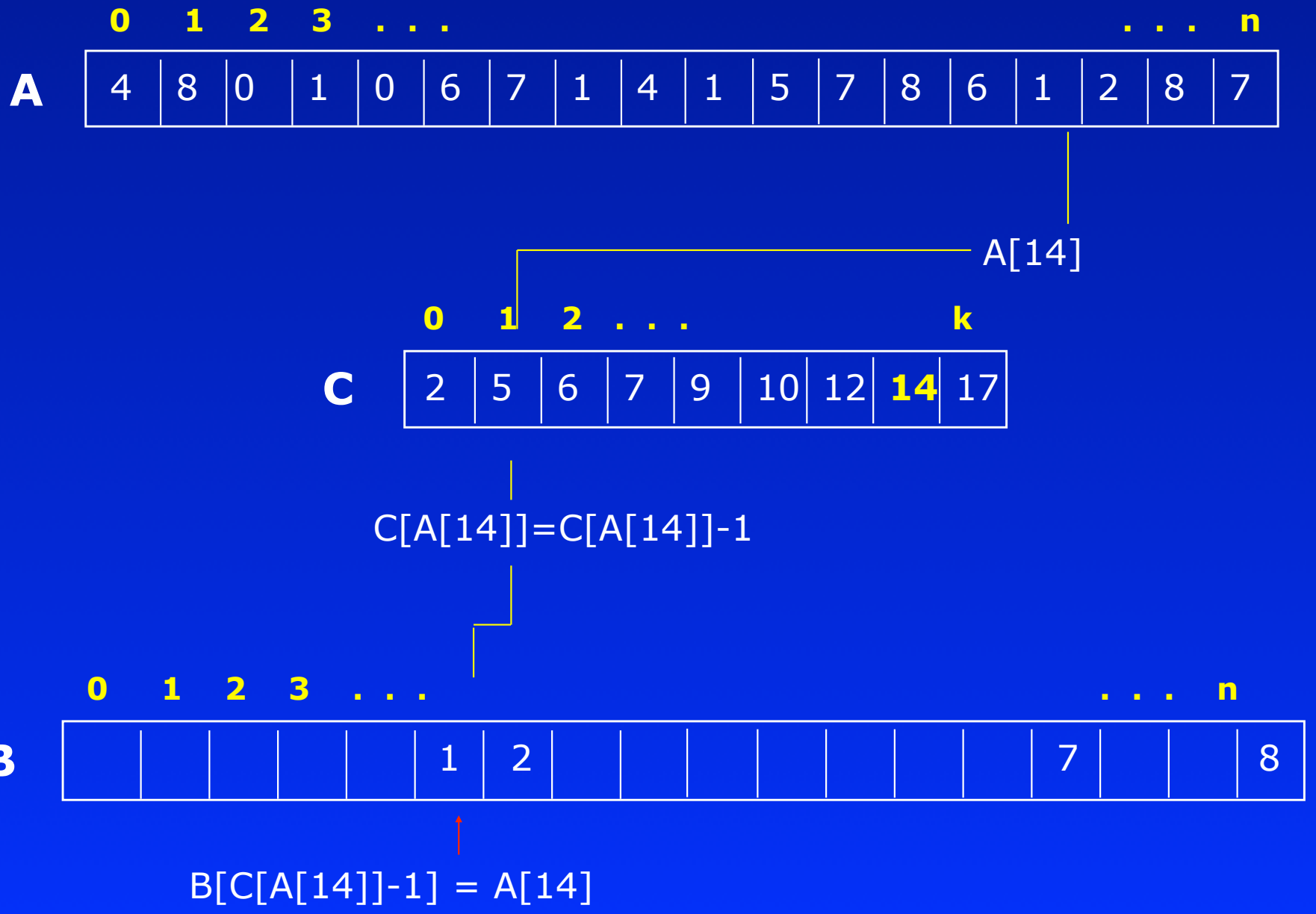
Counting sort



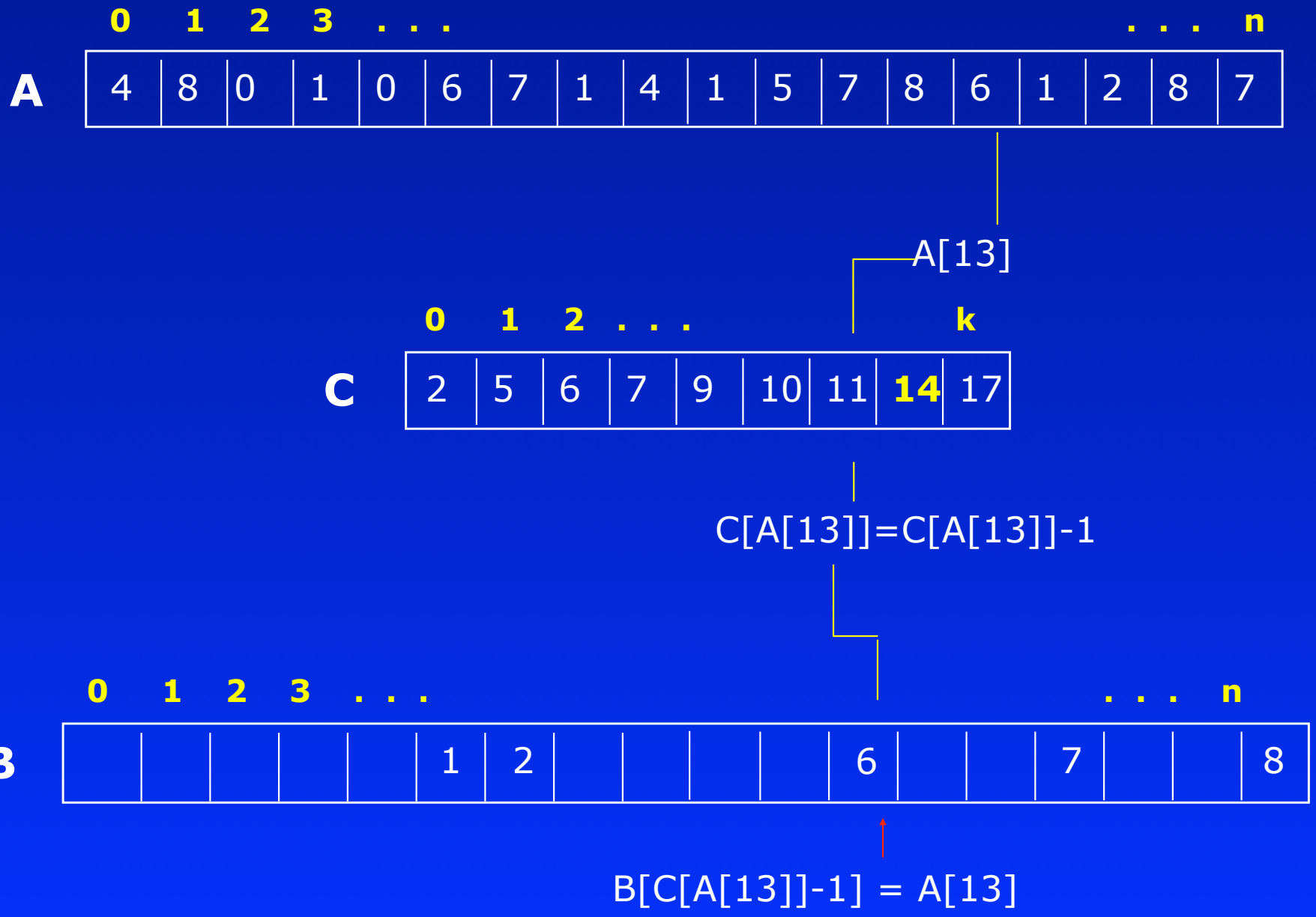
Counting sort



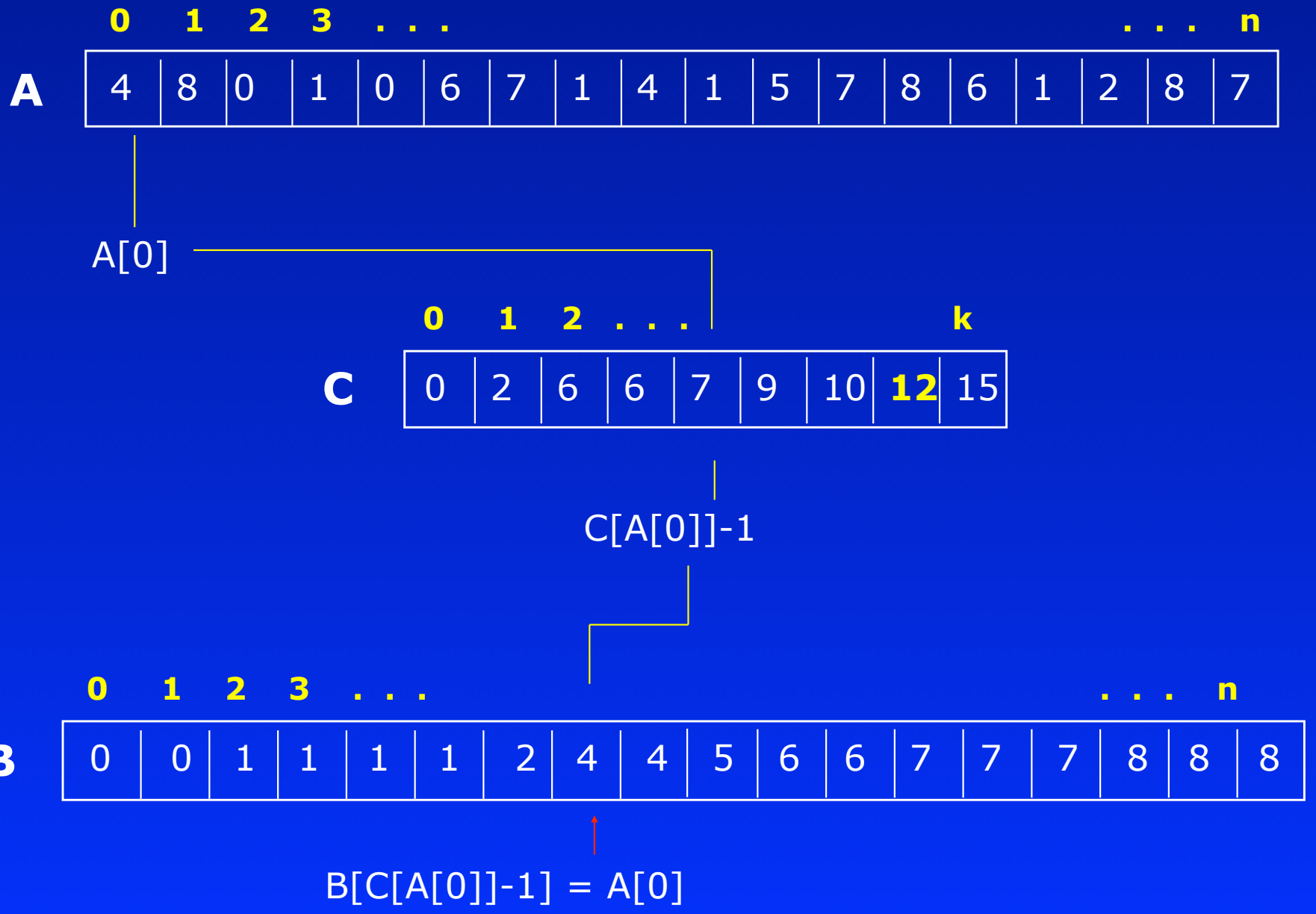
Counting sort



Counting sort



Counting sort



Counting sort

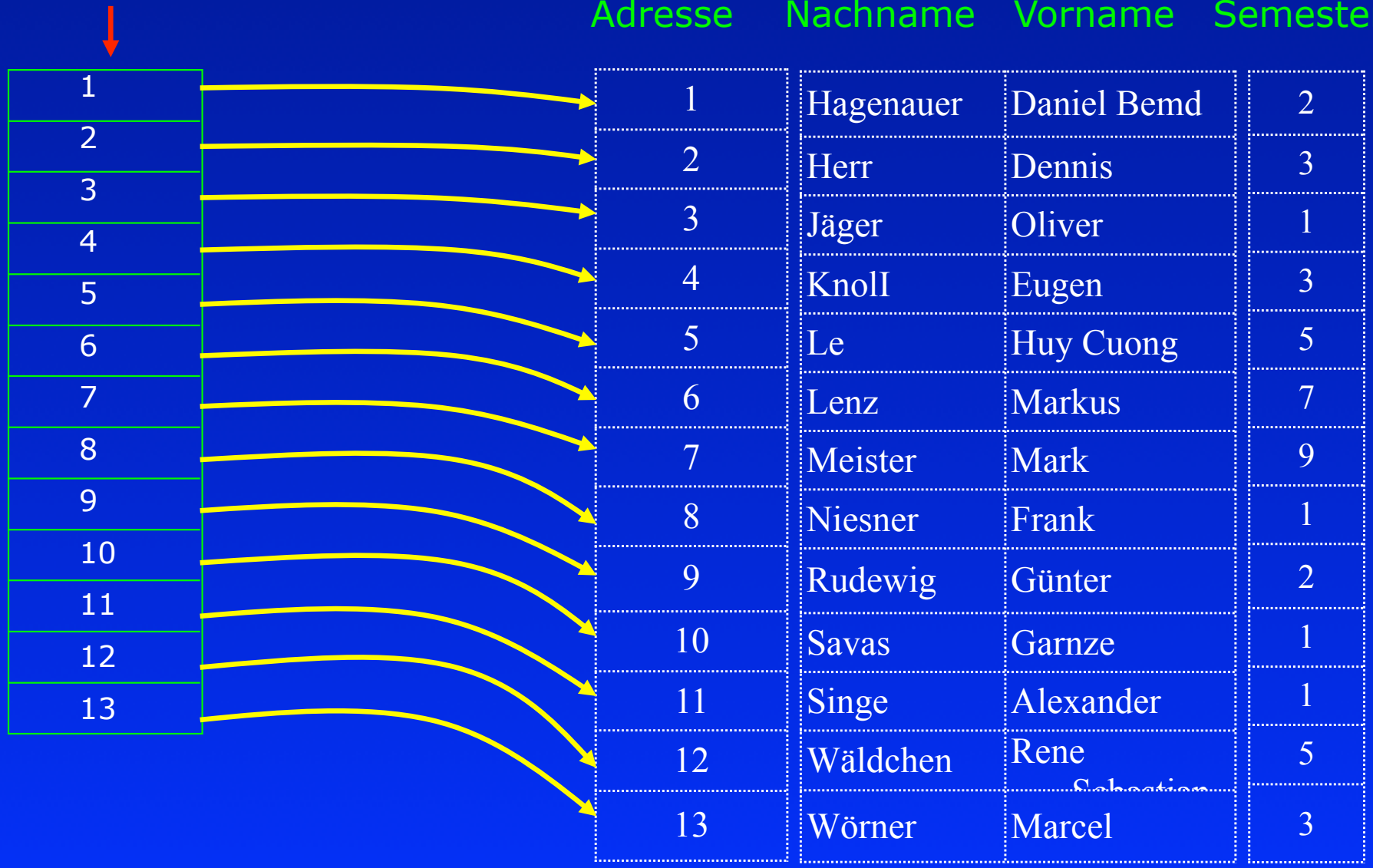
```
def counting_sort(A, k):
        size = len(A)
        B = [0 for i in range(0, size)]
        C = [0 for i in range(0, k+1)]

        for j in range(0, size):
                C[A[j]] += 1
        for i in range(1, k+1):
                C[i] += C[i-1]
        for j in range(size-1, -1, -1):
                C[A[j]] -= 1
                B[C[A[j]]] = A[j]
        return B
```

Eine sehr wichtige Eigenschaft des **Counting-Sort**-Algorithmus ist, dass er stabil ist.

Counting sort

Im Array sind nur die Adressen (Referenzen)

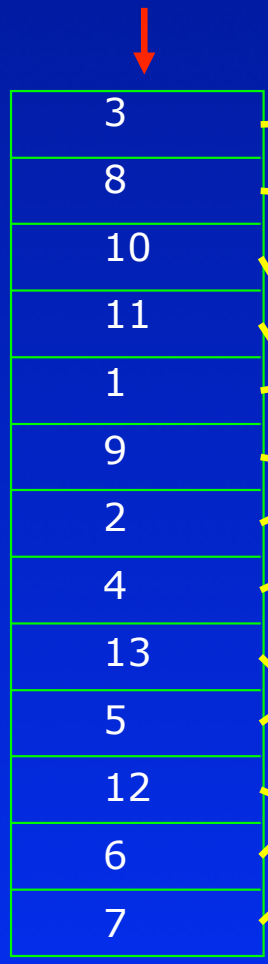
Datenbank



	Adresse	Nachname	Vorname	Semester
1	1	Hagenauer	Daniel Bemd	2
2	2	Herr	Dennis	3
3	3	Jäger	Oliver	1
4	4	KnolI	Eugen	3
5	5	Le	Huy Cuong	5
6	6	Lenz	Markus	7
7	7	Meister	Mark	9
8	8	Niesner	Frank	1
9	9	Rudewig	Günter	2
10	10	Savas	Garnze	1
11	11	Singe	Alexander	1
12	12	Wäldchen	Rene Sebastian	5
13	13	Wörner	Marcel	3

Counting sort

Im Array sind nur die Adressen



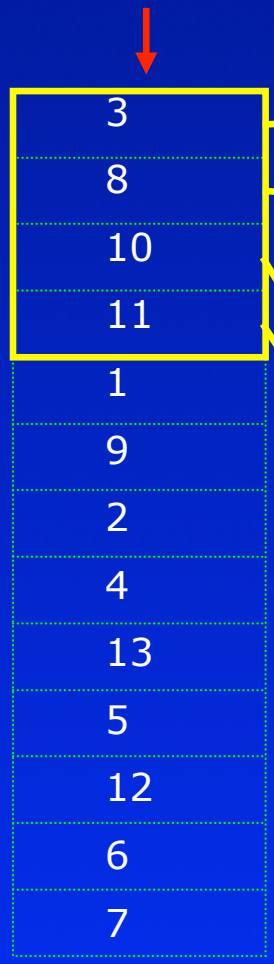
Datenbank

Adresse	Nachname	Vorname	Semester
1	Hagenauer	Daniel Bemd	2
2	Herr	Dennis	3
3	Jäger	Oliver	1
4	KnolI	Eugen	3
5	Le	Huy Cuong	5
6	Lenz	Markus	7
7	Meister	Mark	9
8	Niesner	Frank	1
9	Rudewig	Günter	2
10	Savas	Garnze	1
11	Singe	Alexander	1
12	Wäldchen	Rene	5
13	Wörner	Marcel	3

Sortiert nach Semester

Counting sort

Im Array sind nur die Adressen



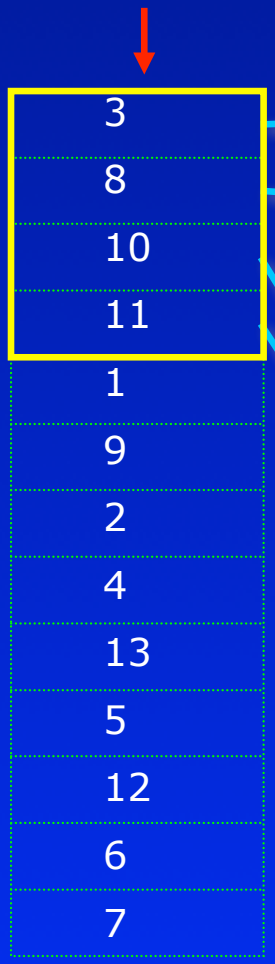
Datenbank

Adresse	Nachname	Vorname	Semester
1	Hagenauer	Daniel Bemd	2
2	Herr	Dennis	3
3	Jäger	Oliver	1
4	KnolI	Eugen	3
5	Le	Huy Cuong	5
6	Lenz	Markus	7
7	Meister	Mark	9
8	Niesner	Frank	1
9	Rudewig	Günter	2
10	Savas	Garnze	1
11	Singe	Alexander	1
12	Wäldchen	Rene Sebastian	5
13	Wörner	Marcel	3

Sortiert nach Semester

Counting sort

Im Array sind nur die Adressen



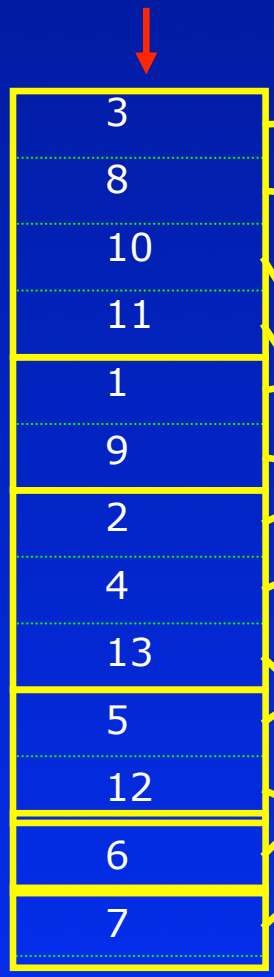
Datenbank

Adresse	Nachname	Vorname	Semester
1	Hagenauer	Daniel Bemd	2
2	Herr	Dennis	3
3	Jäger	Oliver	1
4	KnolI	Eugen	3
5	Le	Huy Cuong	5
6	Lenz	Markus	7
7	Meister	Mark	9
8	Niesner	Frank	1
9	Rudewig	Günter	2
10	Savas	Garnze	1
11	Singe	Alexander	1
12	Wäldchen	Rene Sebastian	5
13	Wörner	Marcel	3

Sortiert nach Semester

Counting sort

Im Array sind nur die Adressen



Datenbank

Adresse	Nachname	Vorname	Semester
1	Hagenauer	Daniel Bemd	2
2	Herr	Dennis	3
3	Jäger	Oliver	1
4	KnolI	Eugen	3
5	Le	Huy Cuong	5
6	Lenz	Markus	7
7	Meister	Mark	9
8	Niesner	Frank	1
9	Rudewig	Günter	2
10	Savas	Garnze	1
11	Singe	Alexander	1
12	Wäldchen	Rene	5
13	Wörner	Marcel	3

Sortiert nach Semester