

Objektorientierte Programmierung

Sortieren (Teil 4)

Imperativ!

SoSe 2018

Prof. Dr. Margarita Esponda

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.

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

Das C-Feld wird mit Nullen initialisiert.

	0	1	2	...				8
C	0	0	0	0	0	0	0	0

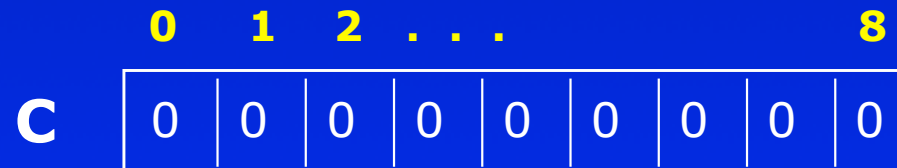
Wertebereich = (0-8)

Counting sort

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

	0	1	2	...				8
C	0	0	0	0	0	0	0	0

Counting sort



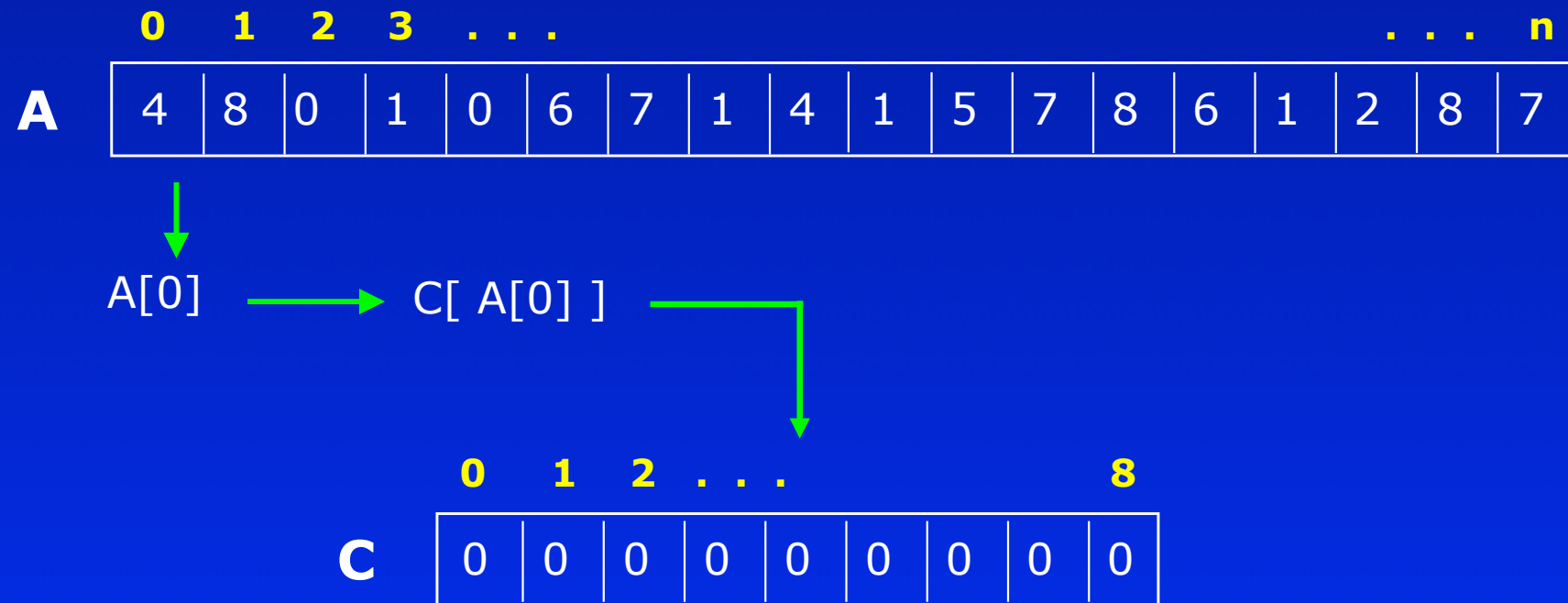
Counting sort

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

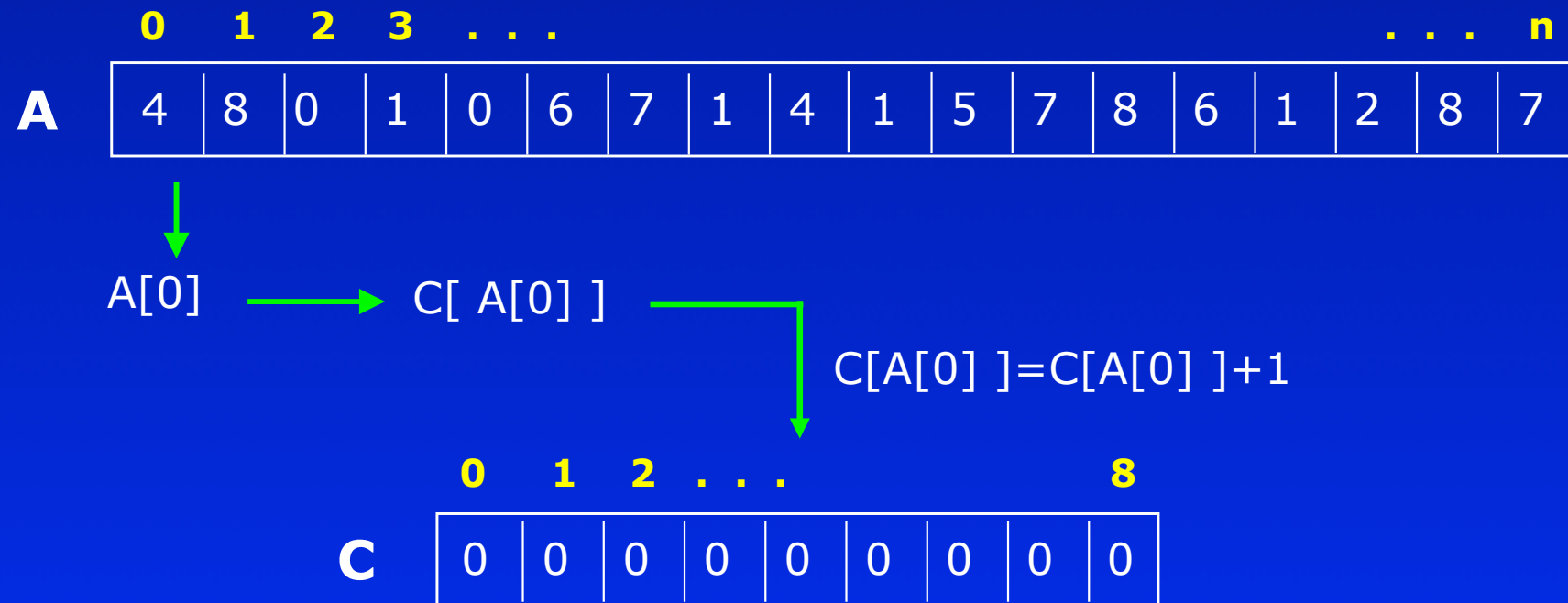
A[0] C[A[0]]

	0	1	2	...					8
C	0	0	0	0	0	0	0	0	0

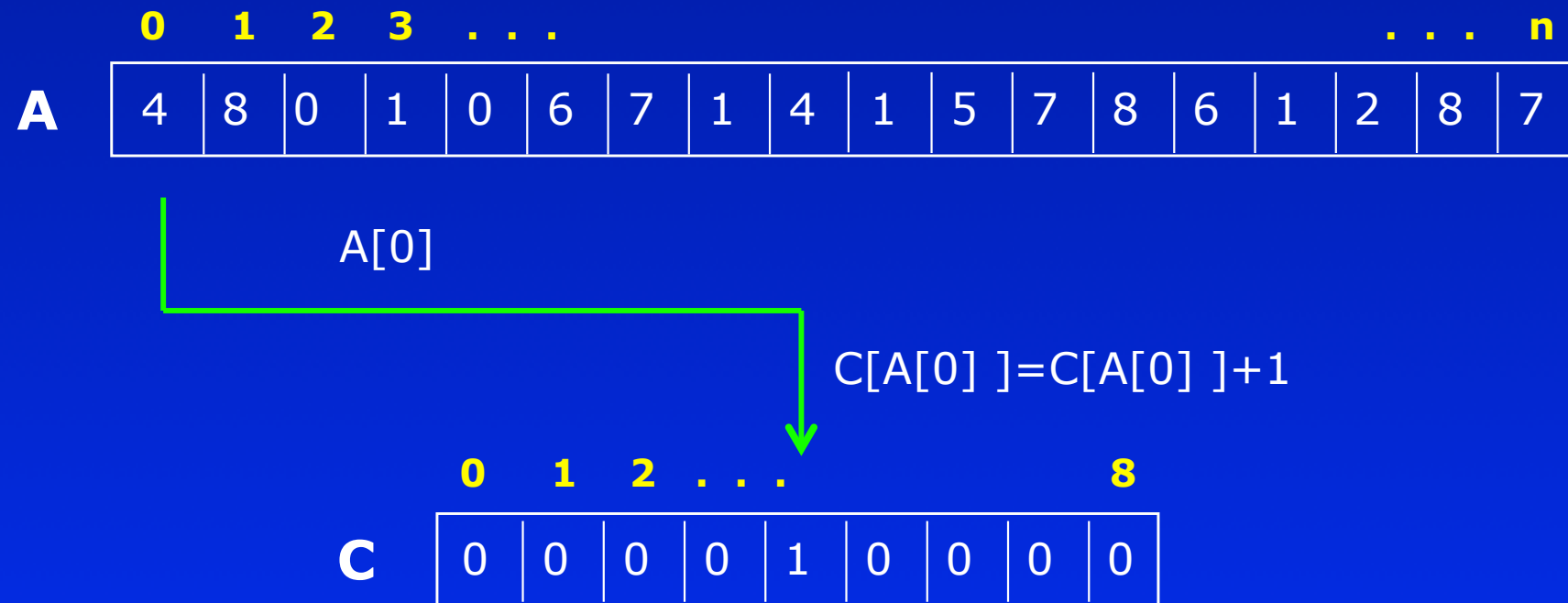
Counting sort



Counting sort



Counting sort



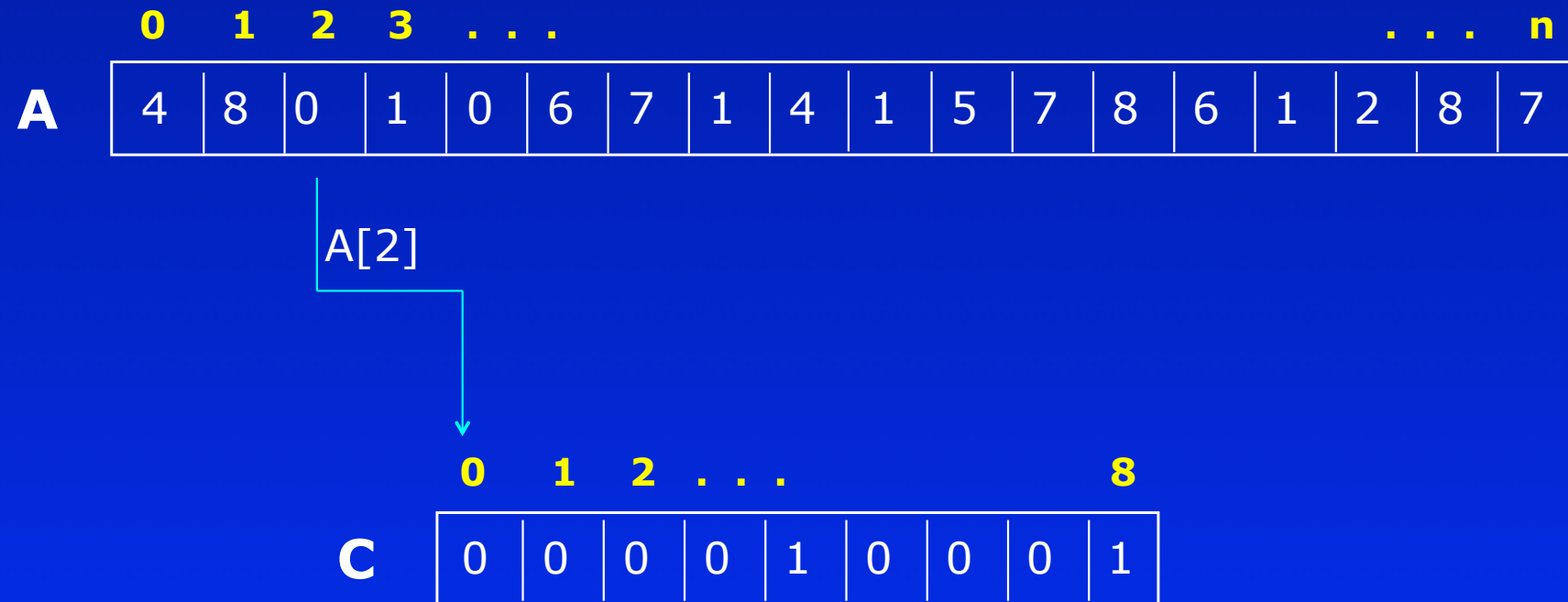
Counting sort



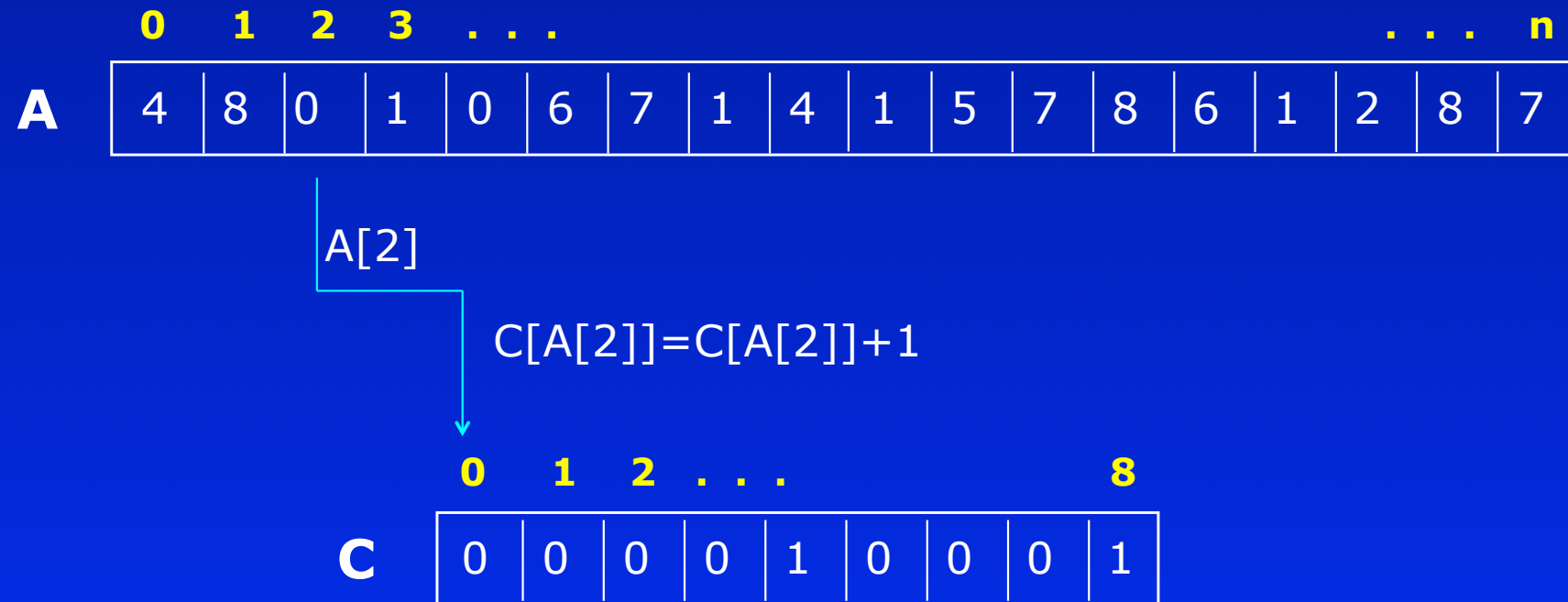
Counting sort



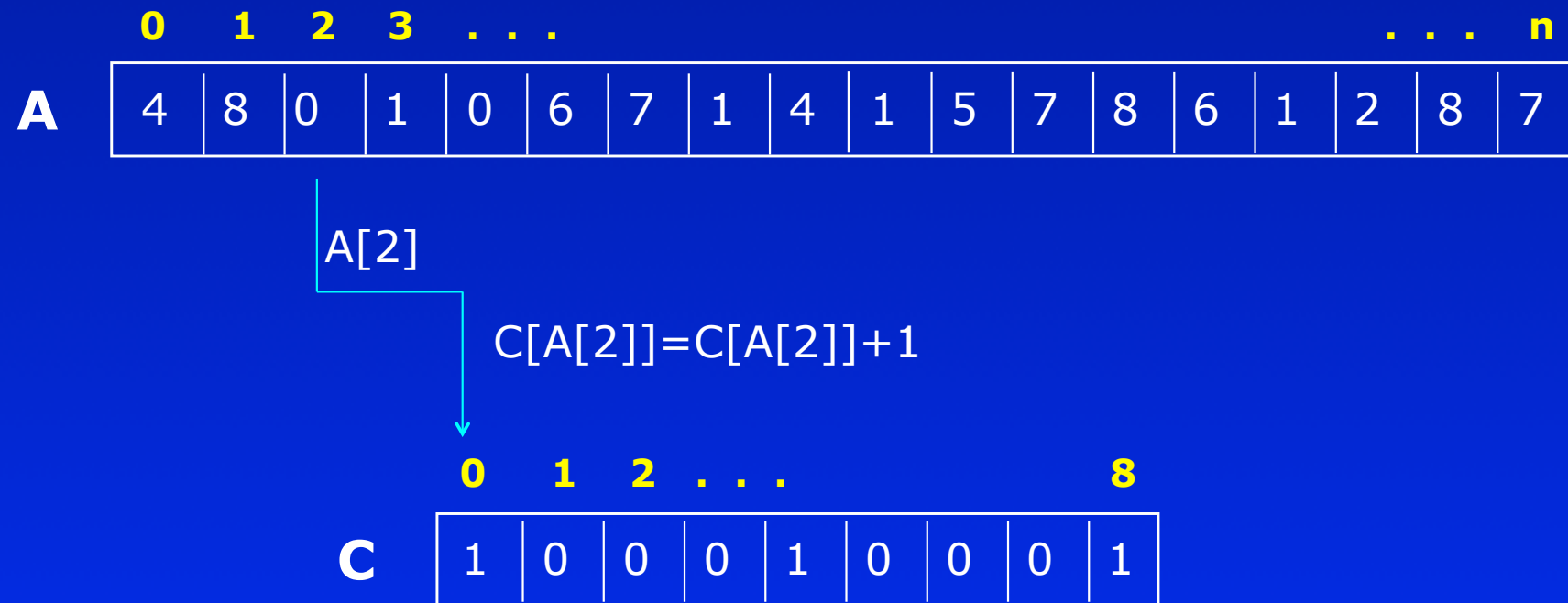
Counting sort



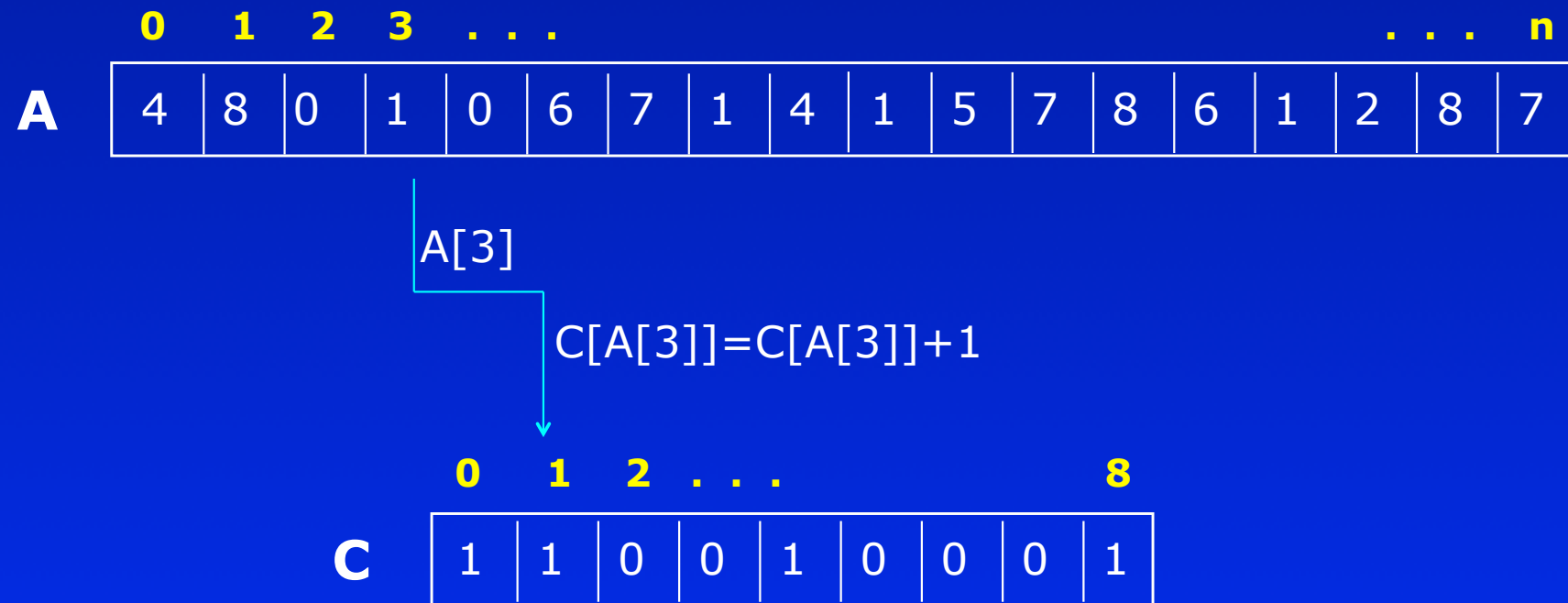
Counting sort



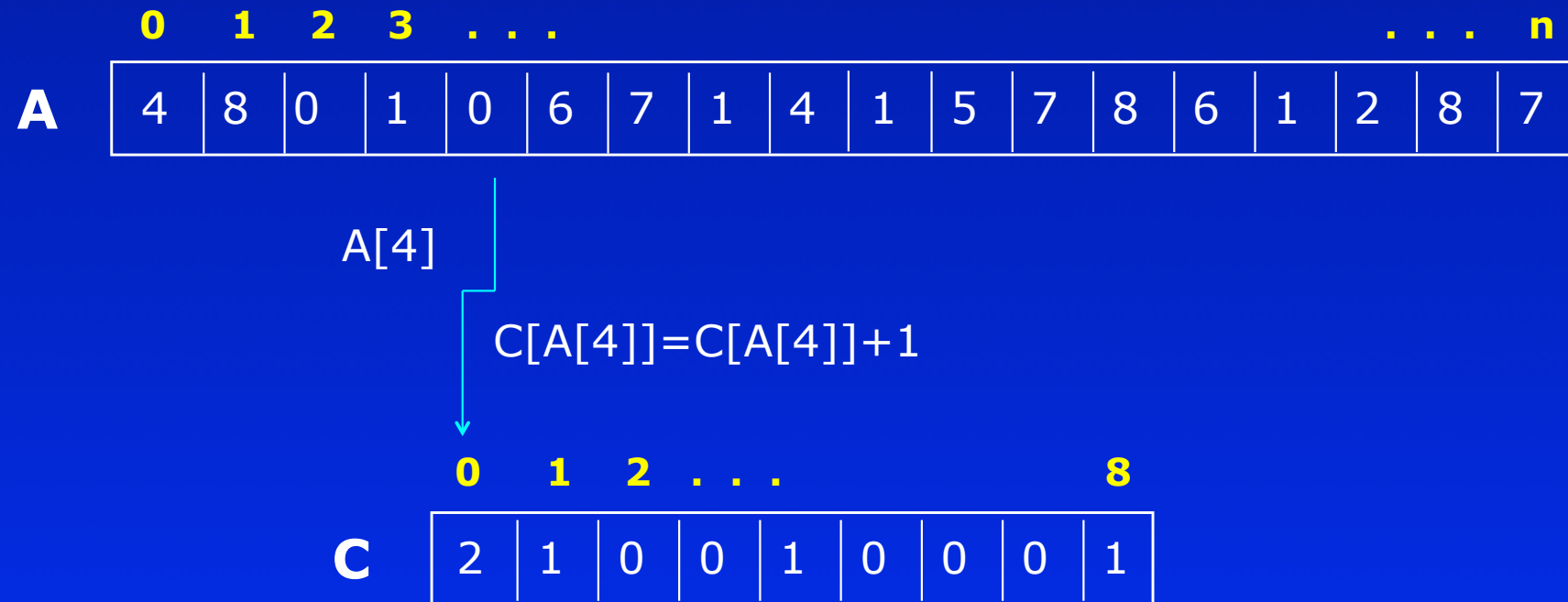
Counting sort



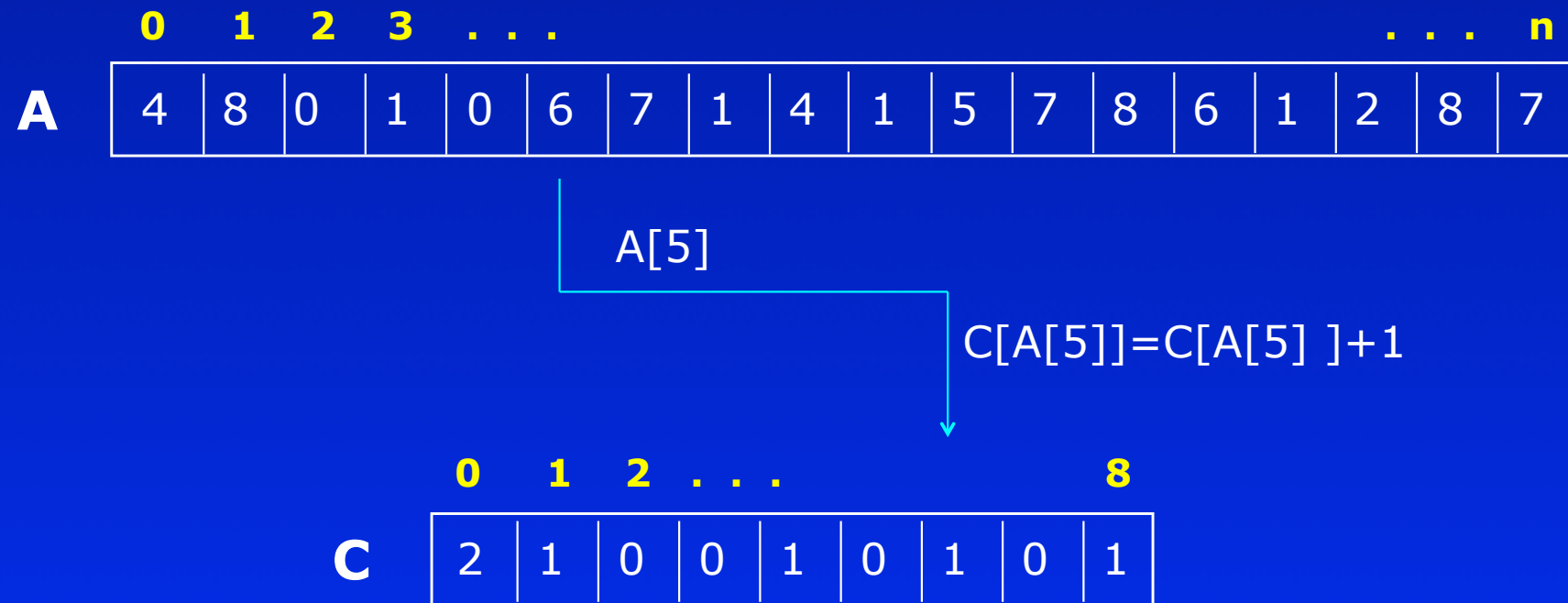
Counting sort



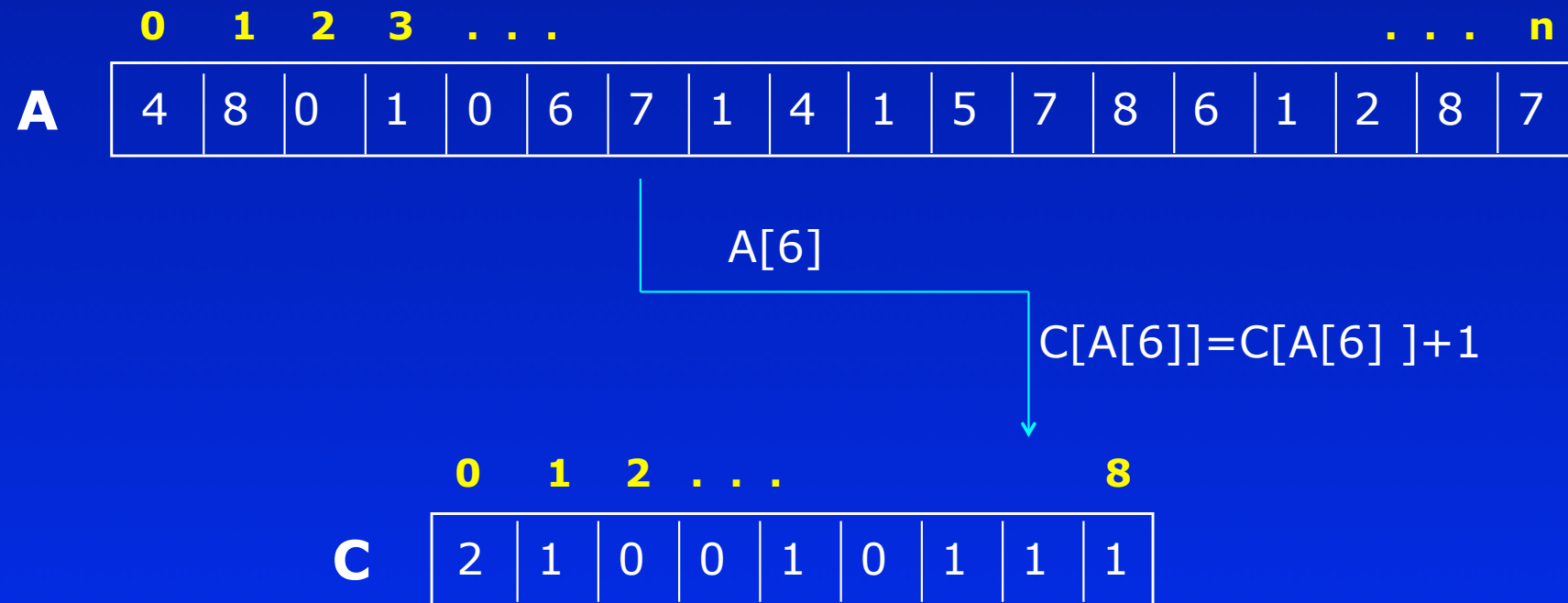
Counting sort



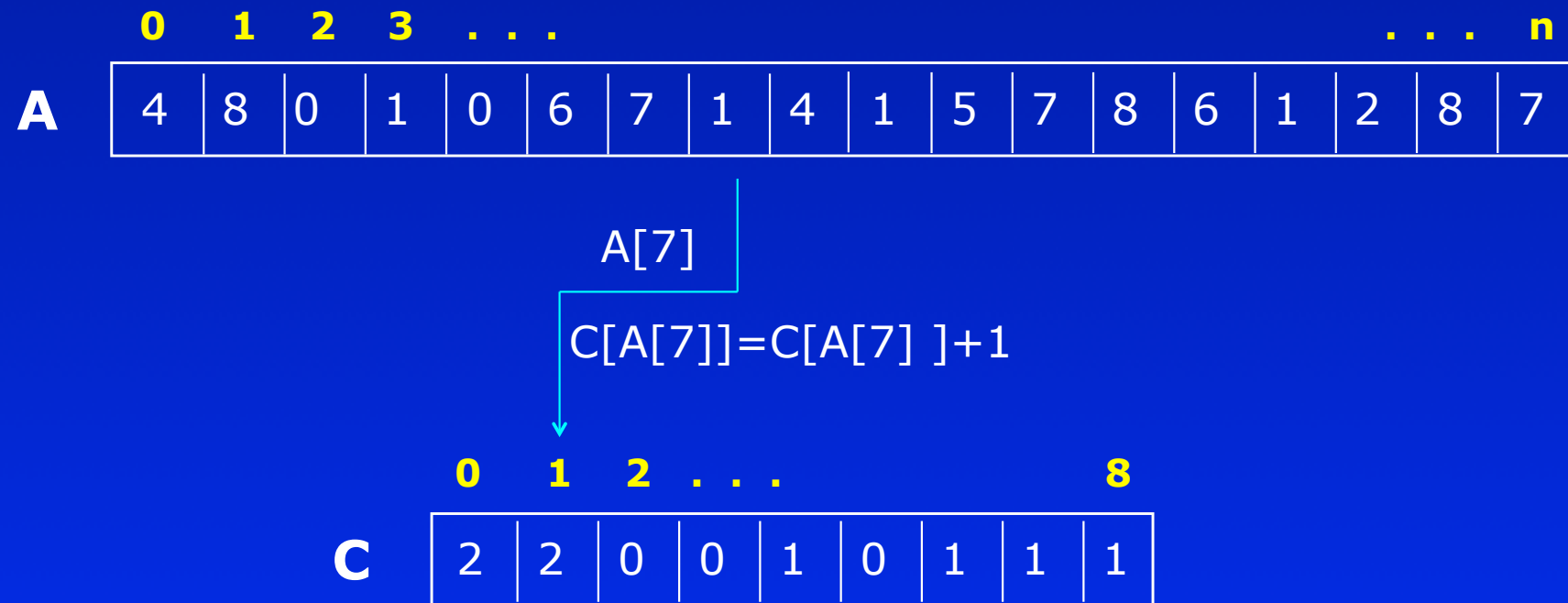
Counting sort



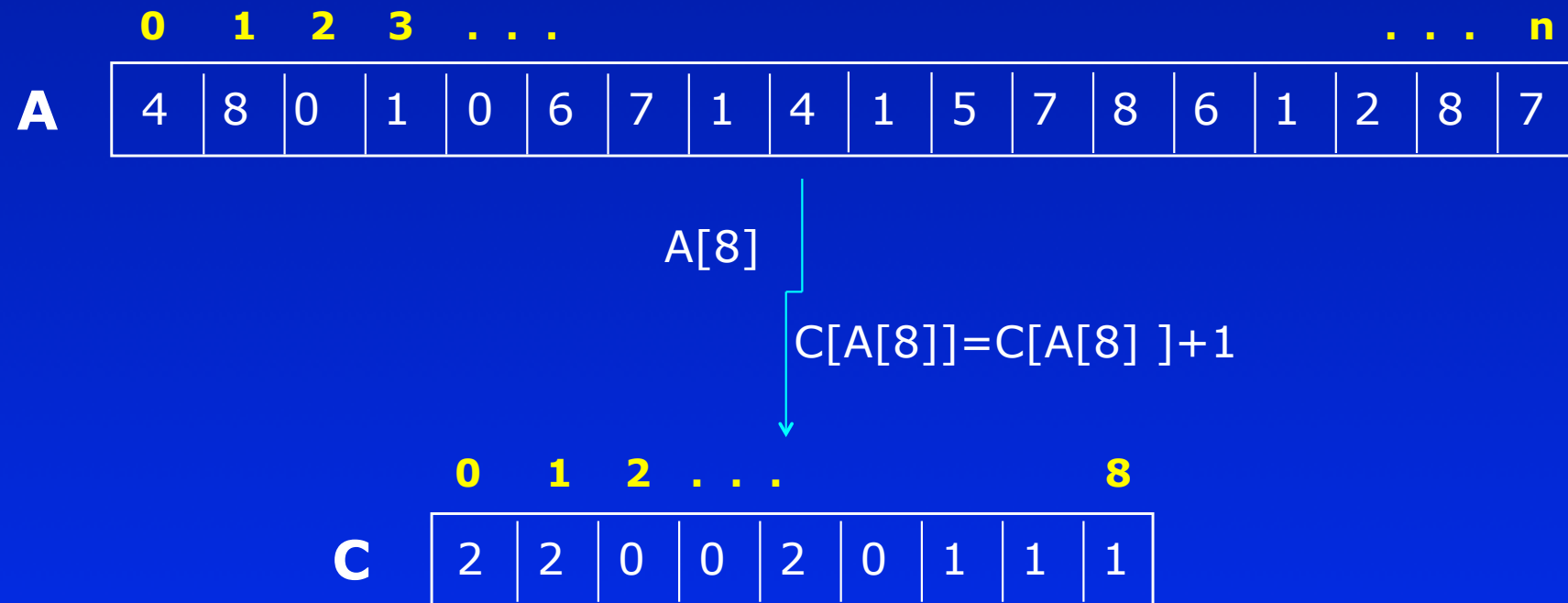
Counting sort



Counting sort



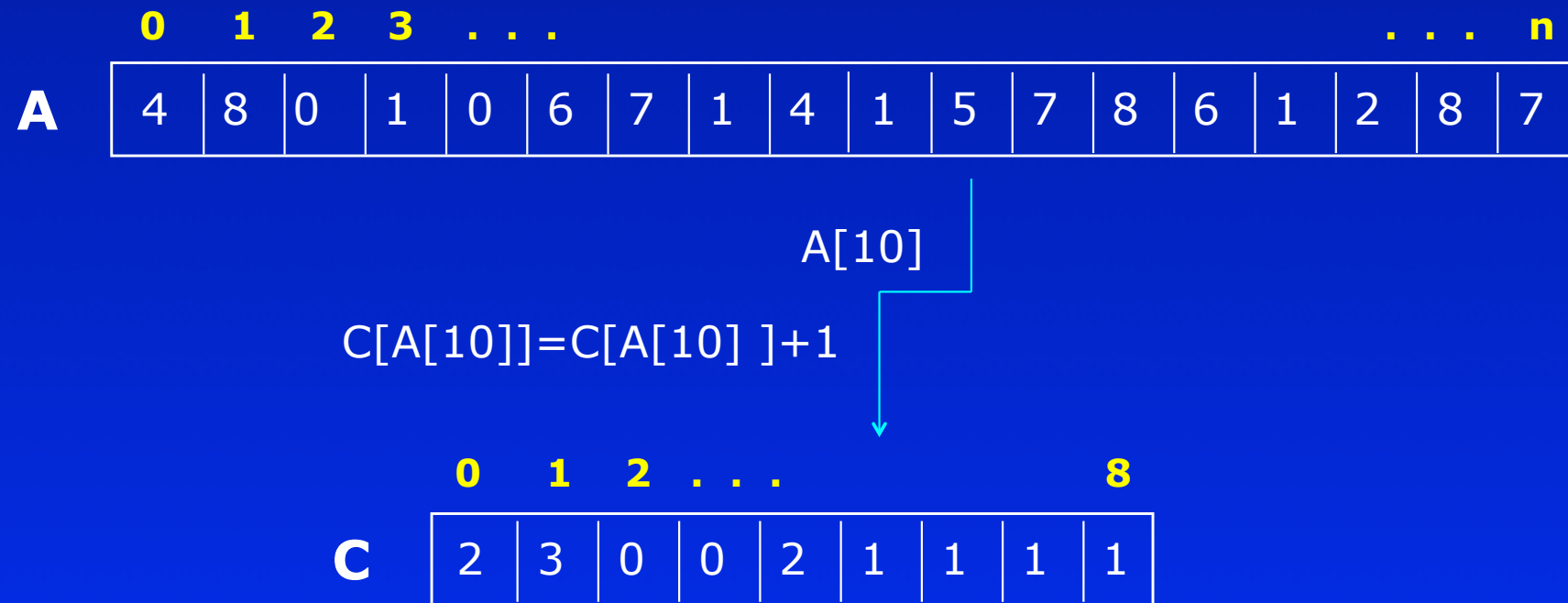
Counting sort



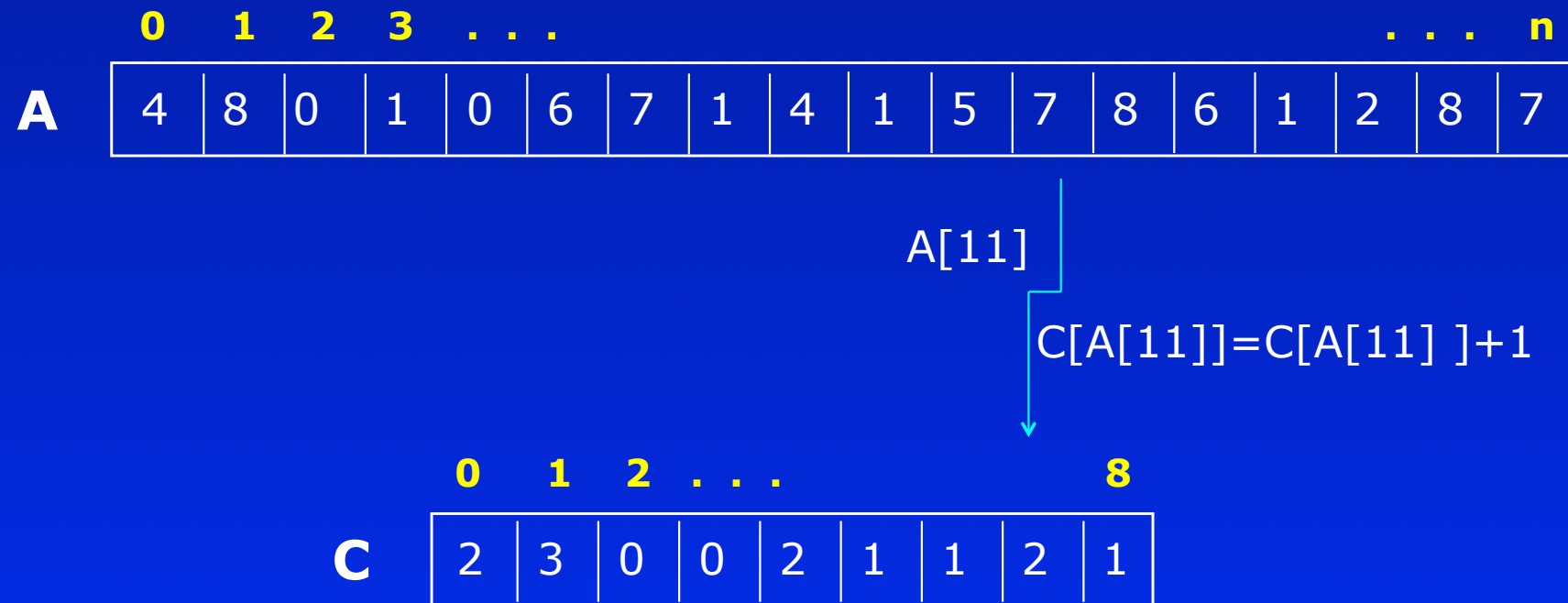
Counting sort



Counting sort



Counting sort



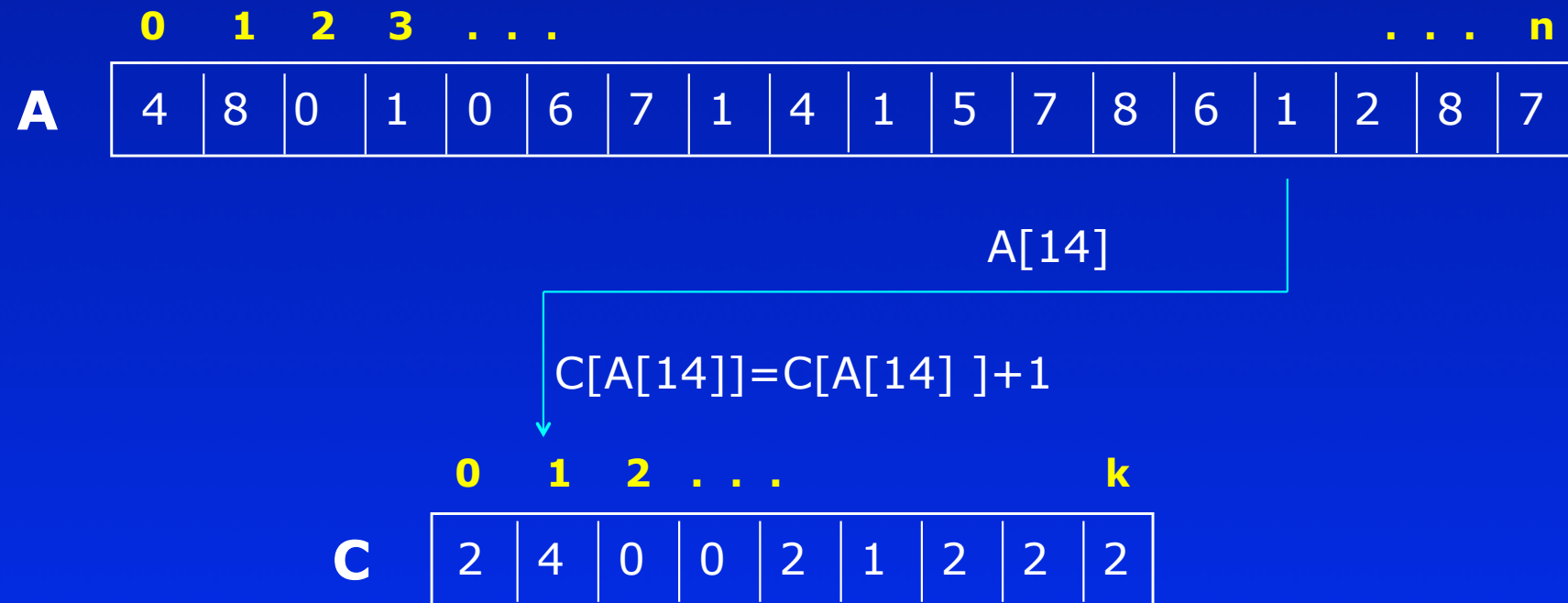
Counting sort



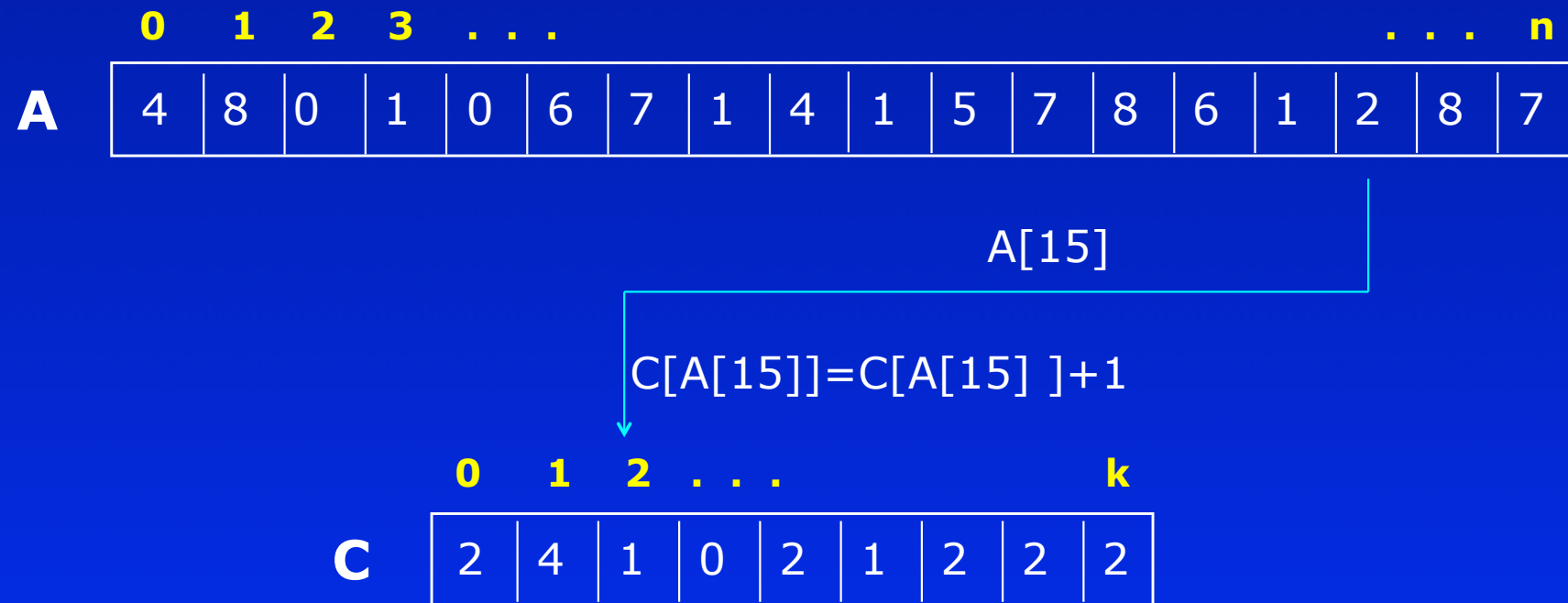
Counting sort



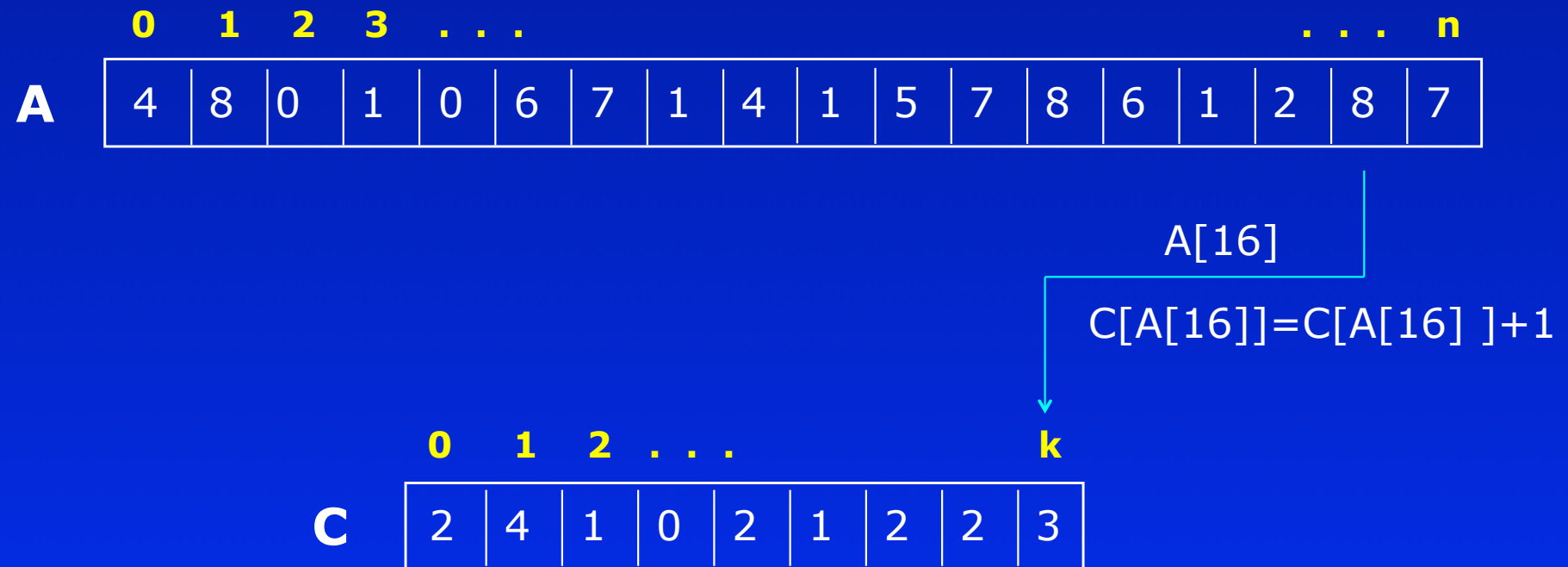
Counting sort



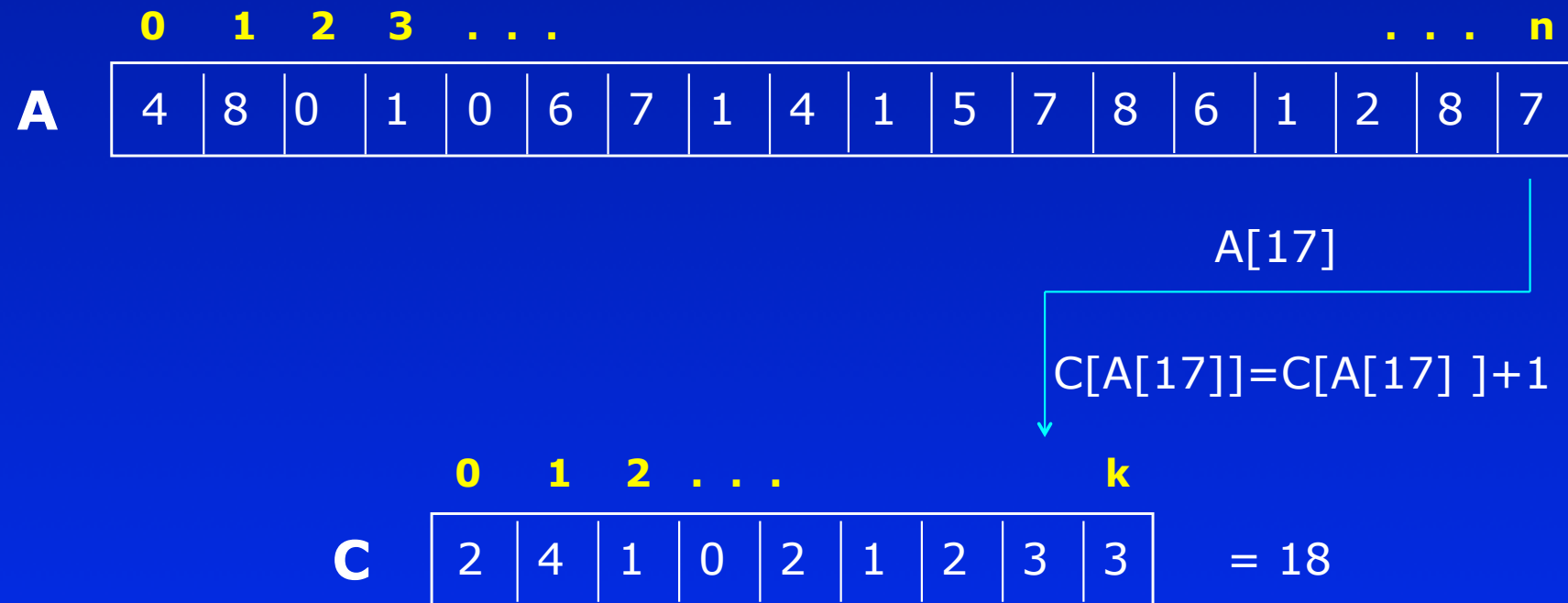
Counting sort



Counting sort



Counting sort



Counting sort


C

0	1	2	...					k
2	4	1	0	2	1	2	3	3

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

C

0	1	2	...					k
2	4	1	0	2	1	2	3	3



B


0	1	2	3	n

Counting sort

	0	1	2	...					k
C	2	4	1	0	2	1	2	3	3

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

	0	1	2	...				k	
C	2	6	1	0	2	1	2	3	3




	0	1	2	3	n
B																	

Counting sort

	0	1	2	...					k
C	2	4	1	0	2	1	2	3	3

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

	0	1	2	...					k
C	2	6	7	0	2	1	2	3	3



	0	1	2	3	n
B																	

Counting sort

	0	1	2	...					k
C	2	4	1	0	2	1	2	3	3

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

	0	1	2	...					k
C	2	6	7	7	2	1	2	3	3



	0	1	2	3	n
B																		

Counting sort

	0	1	2	...					k
C	2	4	1	0	2	1	2	3	3

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

	0	1	2	...					k
C	2	6	7	7	9	1	2	3	3



	0	1	2	3	n
B																		

Counting sort

	0	1	2	...					k
C	2	4	1	0	2	1	2	3	3

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

	0	1	2	. . .				k	
C	2	6	7	7	9	10	2	3	3



	0	1	2	3	n
B																	

Counting sort

	0	1	2	...					k
C	2	4	1	0	2	1	2	3	3

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

	0	1	2	. . .					k
C	2	6	7	7	9	10	12	3	3



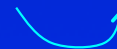
	0	1	2	3	n
B																	

Counting sort

	0	1	2	...					k
C	2	4	1	0	2	1	2	3	3

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

	0	1	2	...					k
C	2	6	7	7	9	10	12	15	3



	0	1	2	3	n
B																	

Counting sort

	0	1	2	...					k
C	2	4	1	0	2	1	2	3	3

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

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

	0	1	2	...					8
C	2	6	7	7	9	10	12	15	18

9 Zahlen sind kleiner oder gleich 4

	0	1	2	3	n
B																	

Counting sort

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

	0	1	2	...					k
C	2	6	7	7	9	10	12	15	18

0

1

2

3

...

...

n

B

Counting sort

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

	0	1	2	...				k	
C	2	6	7	7	9	10	12	15	18

$C[A[17]] = C[A[17]] - 1$

0

1

2

3

...

...

n

B

Counting sort

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

	0	1	2	...					k
C	2	6	7	7	9	10	12	14	18

$C[A[17]] = C[A[17]] - 1$

0

1

2

3

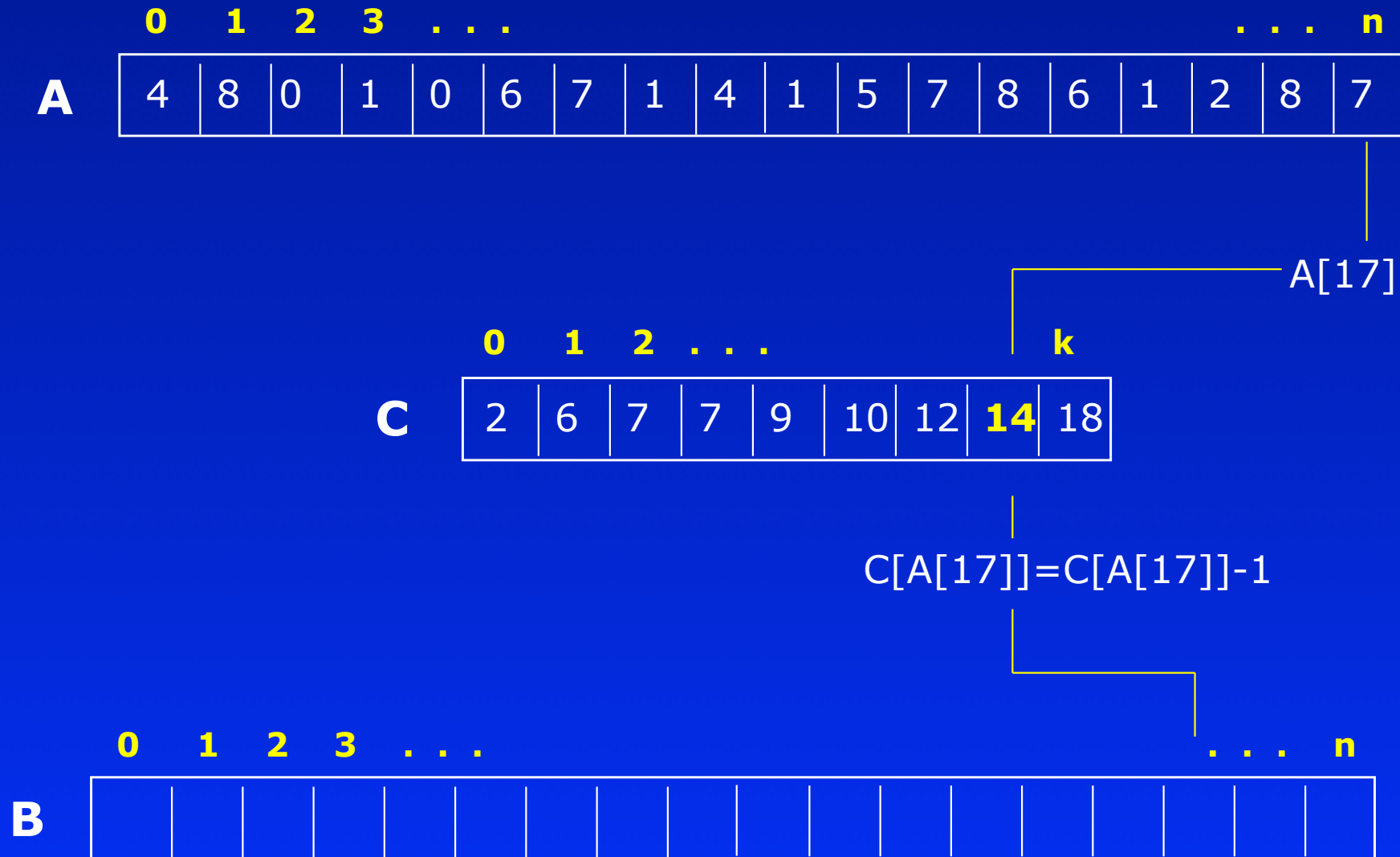
...

...

n

B

Counting sort



Counting sort

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

	0	1	2	...					k
C	2	6	7	7	9	10	12	14	18

$$C[A[17]] = C[A[17]] - 1$$

0

1

2

3

...

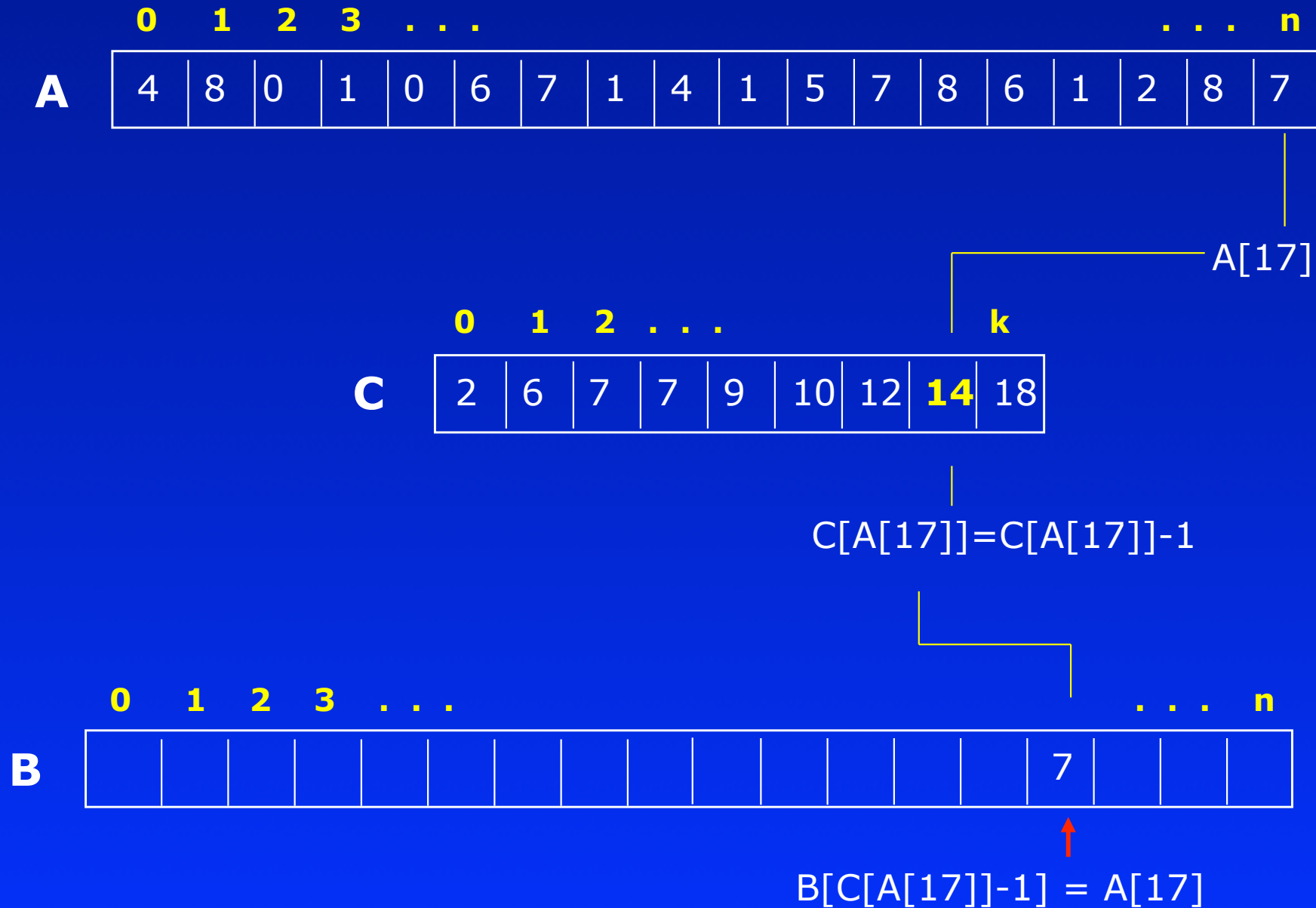
...

n

B

$$B[C[A[17]] - 1] = A[17]$$

Counting sort



Counting sort

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

	0	1	2	...					k
C	2	6	7	7	9	10	12	14	18

A[16]

	0	1	2	3	n	
B															7			

Counting sort

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

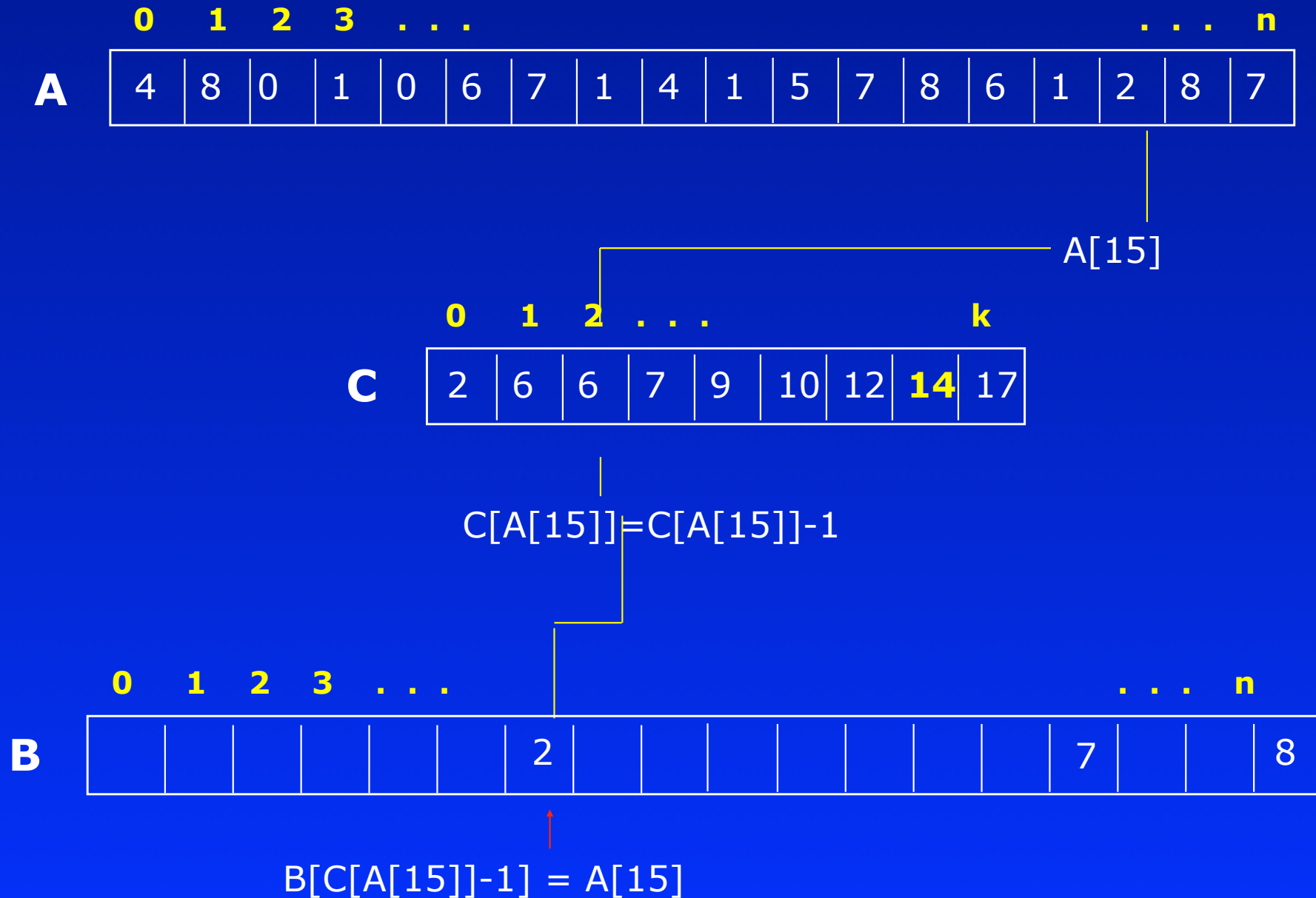
	0	1	2	...					k
C	2	6	7	7	9	10	12	14	17

$$C[A[16]] = C[A[16]] - 1$$

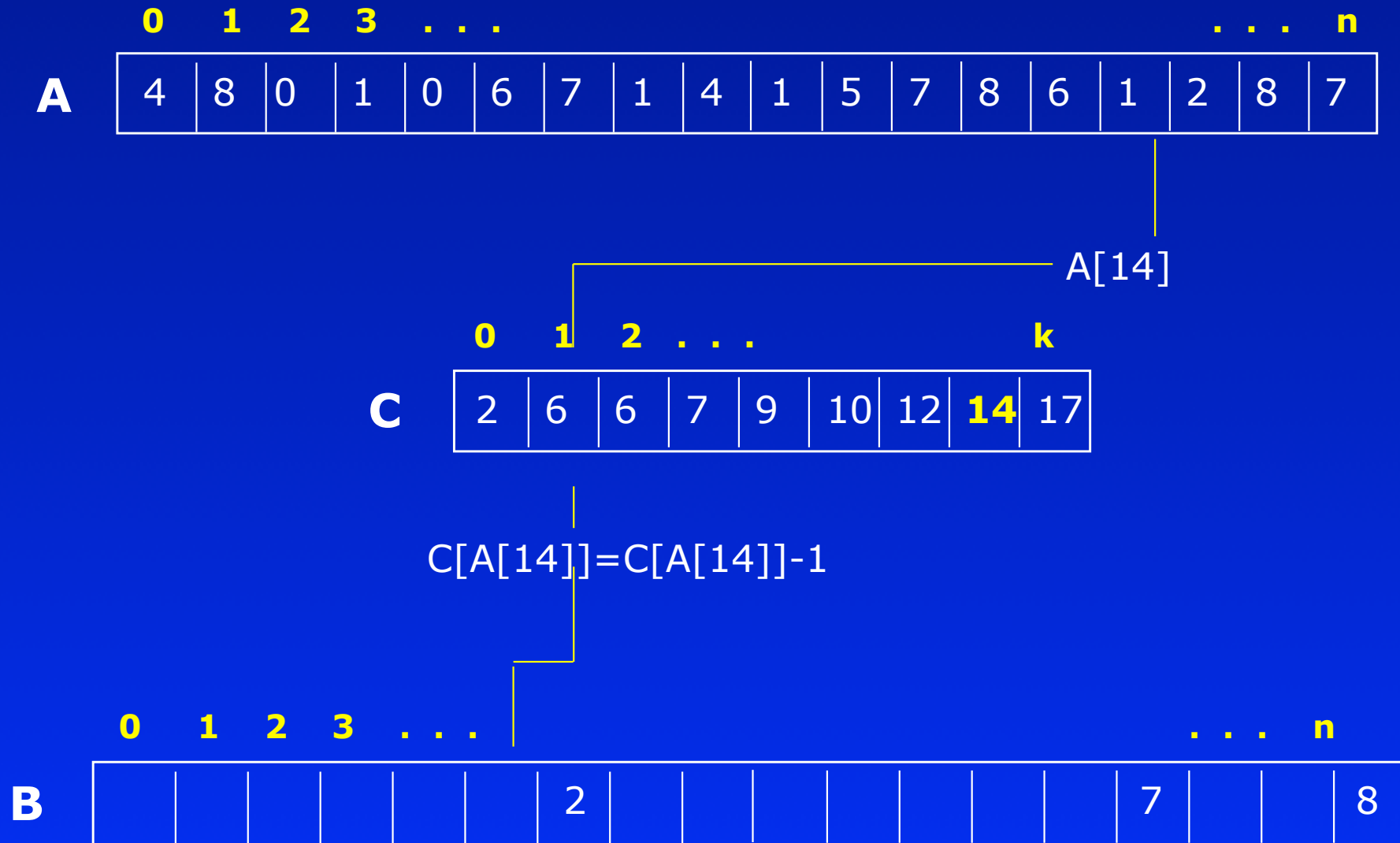
	0	1	2	3	n	
B															7			8

$$B[C[A[16]] - 1] = A[16]$$

Counting sort



Counting sort



Counting sort

0

1

2

3

...

...

n

A

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

C

2	5	6	7	9	10	12	14	17
---	---	---	---	---	----	----	-----------	----

0 1 2 . . . k

A[14]

$$C[A[14]] = C[A[14]] - 1$$

0

1

2

3

...

...

n

B

1

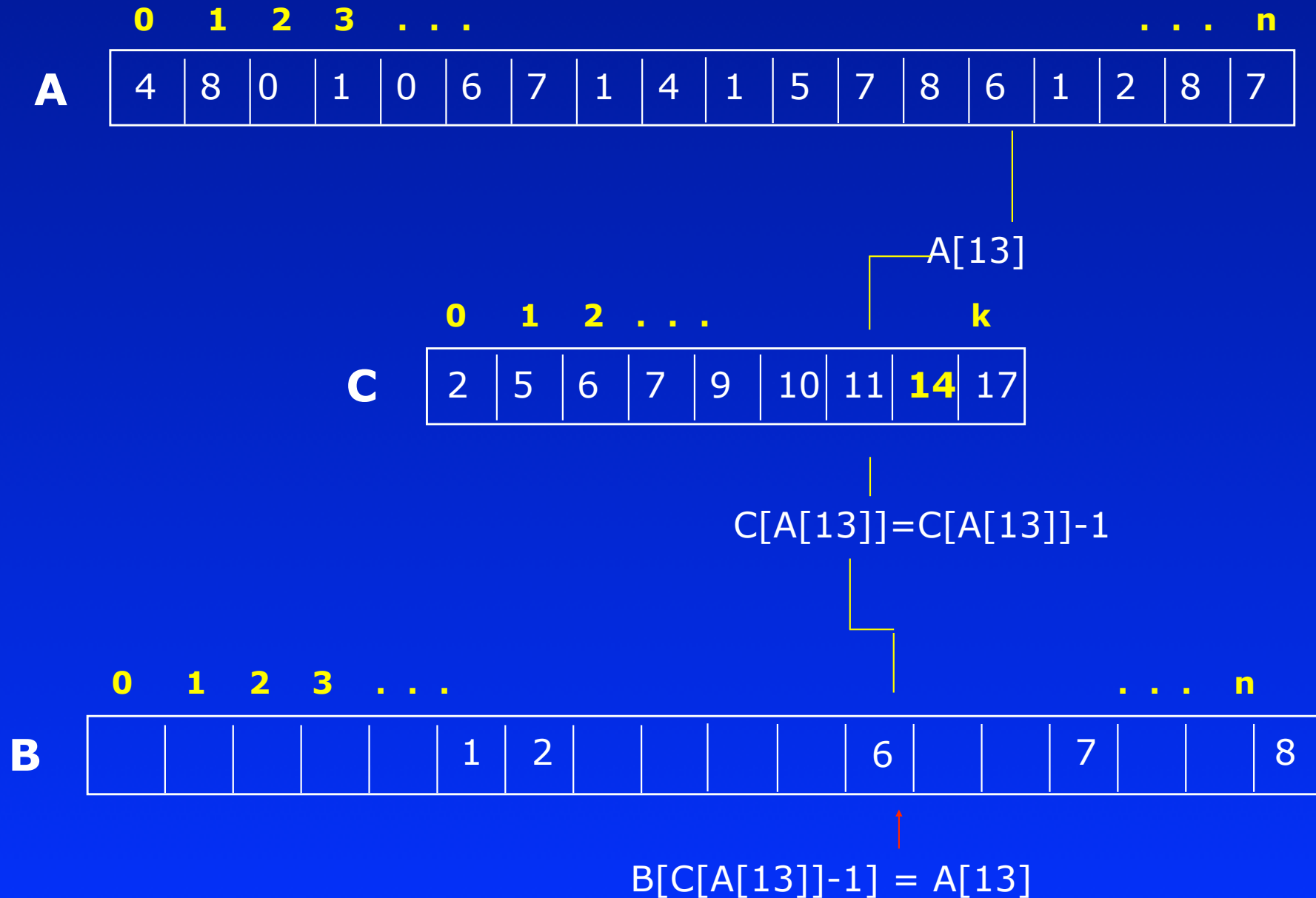
2

7

8

$$B[C[A[14]] - 1] = A[14]$$

Counting sort



Counting sort



Counting sort

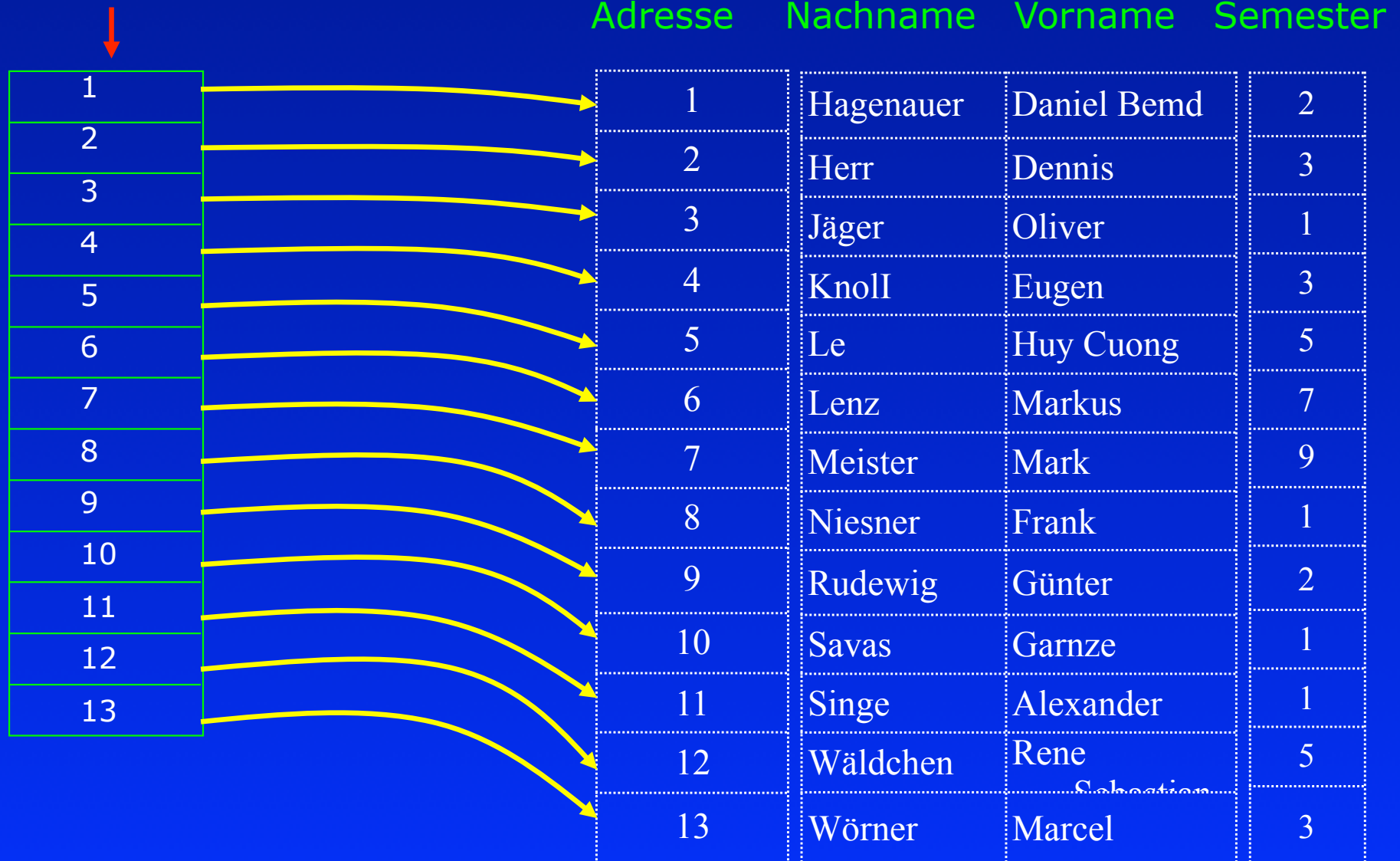
```
def counting_sort(A, k):  
    size = len(A)  
    B = [0 for i in range(0, size)]  
    C = [0 for i in range(0, k+1)]  
  
    for j in range(0, size):  
        C[A[j]] += 1  
    for i in range(1, k+1):  
        C[i] += C[i-1]  
    for j in range(size-1, -1, -1):  
        C[A[j]] -= 1  
 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

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

Counting sort

Im Array sind nur
die Adressen
(Referenzen)

Datenbank



Counting sort

Im Array sind
nur die
Adressen

↓

3
8
10
11
1
9
2
4
13
5
12
6
7

Datenbank

Adresse Nachname Vorname Semester

1	Hagenauer	Daniel Bemd	2
2	Herr	Dennis	3
3	Jäger	Oliver	1
4	Knoll	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

↓

3
8
10
11
1
9
2
4
13
5
12
6
7

Datenbank

Adresse Nachname Vorname Semester

1	Hagenauer	Daniel Bemd	2
2	Herr	Dennis	3
3	Jäger	Oliver	1
4	Knoll	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

3
8
10
11
1
9
2
4
13
5
12
6
7

Datenbank

Adresse Nachname Vorname Semester

1	Hagenauer	Daniel Bemd	2
2	Herr	Dennis	3
3	Jäger	Oliver	1
4	Knoll	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

Radix sort

1887 Entwickelt für das Sortieren von Lochkarten



```
radixsort ( A, d) {  
    for i in range(d):  
        stablesort(A, i)
```

- Zahlen werden ziffernweise sortiert
- die niedrigsten Stellenwerte zuerst
- alles nur mit einem stabilen Algorithmus (essentiell)
- auch gut für das Sortieren von zusammengesetzte Datenstrukturen

Beispiel: Sortieren nach Datum

↓
Countingsort

↓
 $T(n) = O(n)$

Bucket-Sort

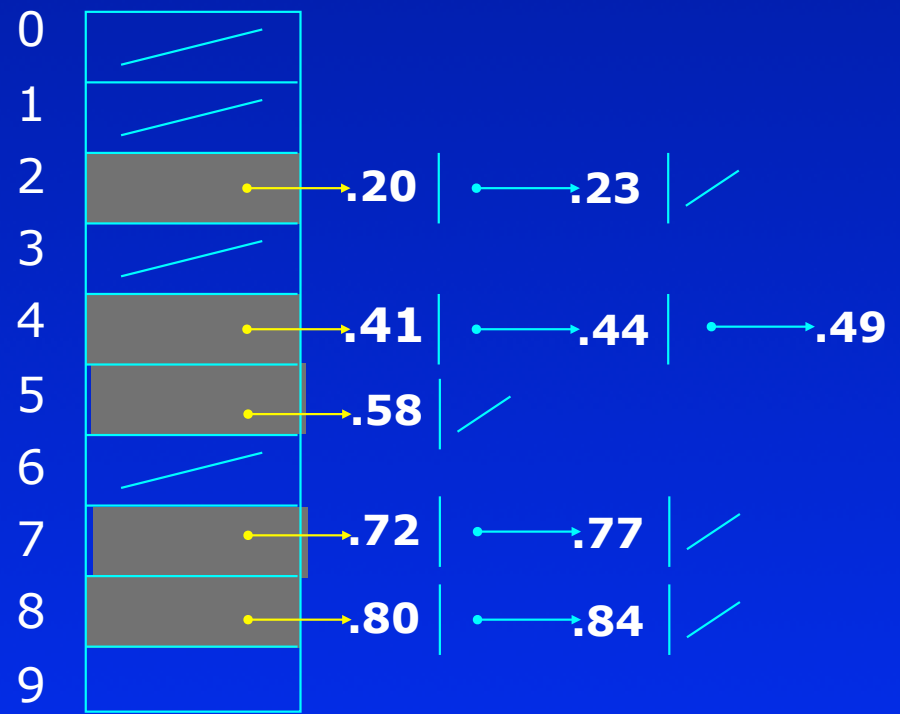
- Die zu sortierenden Daten müssen gleich verteilt über den Wertebereich $[0,1)$ sein.
- **Not-In-Place**
zusätzlicher Speicherplatz ($O(n)$) wird benötigt
- linearer Aufwand $O(n)$
- Grundidee ist den Wertebereich $[0,1)$ in m kleinere Wertebereiche zu teilen und *Buckets* dafür zu definieren.
- Die Zahlen werden in den dazugehörigen *Buckets* verteilt und innerhalb diesen sortiert.
- Zum Schluss werden die Zahlen der Reihe nach aus den *Buckets* ausgegeben.
- Eine Hilfsarray von verketteten Listen wird für die *Buckets* verwendet.

Bucket-Sort

A

.58
.23
.49
.72
.77
.41
.20
.44
.80
.84

B



Bucket-Sort

```
def bucketsort ( A ):
    n = len(A)
    B = [ [] for i in range(n) ]
    for i in range(n):
        B[math.floor((A[i])*10)].append(A[i])
    for i in range(0,n):
        insertsort(B[i])
    R = []
    for i in range(n):
        R = R+B[i]
    return R
```

Welches ist die längste Zeichenfolge, die sich wiederholt?

a a b c f d e s a b a b c d f e r f c s d e a e d a b c f d e s a b e d a

a a b c f d e s a b a b c d f e r f c s d e a e d a b c f d e s a b e d a

Anwendungen

- Linguistik
- Bioinformatik (DNA-Analyse)
- Datenkompression
- Untersuchung von Plagiaten
- Antiviren-Software
- Musik-Analyse
- usw.

Lösung mit Brute-Force



Für alle Startpositionen (i, j) müssen wir das längste Präfix finden.

Anzahl der i, j Kombinationen = $(n-1) + (n-2) + \dots + 2 + 1 = O(n^2)$

Lösung mit Sortieralgorithmen

```

0  a a c f d e b d a b c f d e b e d a
1  a c f d e b d a b c f d e b e d a
2  c f d e b d a b c f d e b e d a
3  f d e b d a b c f d e b e d a
4  d e b d a b c f d e b e d a
5  e b d a b c f d e b e d a
6  b d a b c f d e b e d a
7  d a b c f d e b e d a
8  a b c f d e b e d a
9  b c f d e b e d a
10 c f d e b e d a
11 f d e b e d a
12 d e b e d a
13 e b e d a
14 b e d a
15 e d a
16 d a
17 a

```

```

[ "a a c f d e b d a b c f d e b e d a",
  "a c f d e b d a b c f d e b e d a",
  "c f d e b d a b c f d e b e d a",
  "f d e b d a b c f d e b e d a",
  "d e b d a b c f d e b e d a",
  "e b d a b c f d e b e d a",
  "b d a b c f d e b e d a",
  "d a b c f d e b e d a",
  "a b c f d e b e d a",
  "b c f d e b e d a",
  "c f d e b e d a",
  "f d e b e d a",
  "d e b e d a",
  "e b e d a",
  "b e d a",
  "e d a",
  "d a",
  "a"]

```

Suffixen oder
Startpositionen innerhalb
der gesamten
Zeichenketten

Lösung mit Sortieralgorithmen

```
17  a
   0  a a c f d e b d a b c f d e b e d a
   8  a b c f d e b e d a
   1  a c f d e b d a b c f d e b e d a
   9  b c f d e b e d a
   6  b d a b c f d e b e d a
  14  b e d a
  10  c f d e b e d a
   2  c f d e b d a b c f d e b e d a
  16  d a
   7  d a b c f d e b e d a
   4  d e b d a b c f d e b e d a
  12  d e b e d a
   5  e b d a b c f d e b e d a
  13  e b e d a
  15  e d a
  11  f d e b e d a
   3  f d e b d a b c f d e b e d a
```

Die Suffixe werden sortiert

Lösung mit Sortieralgorithmen

17	a
0	a a c f d e b d a b c f d e b e d a
8	a b c f d e b e d a
1	a c f d e b d a b c f d e b e d a
9	b c f d e b e d a
6	b d a b c f d e b e d a
14	b e d a
10	c f d e b e d a
2	c f d e b d a b c f d e b e d a
16	d a
7	d a b c f d e b e d a
4	d e b d a b c f d e b e d a
12	d e b e d a
5	e b d a b c f d e b e d a
13	e b e d a
15	e d a
11	f d e b e d a
3	f d e b d a b c f d e b e d a

Die Prefix-Länge jeder zwei benachbarten Suffixe werden verglichen und der längste Prefix gesucht.

Lösung mit Sortieralgorithmen

17	a
0	a a c f d e b d a b c f d e b e d a
8	a b c f d e b e d a

1	a c f d e b d a b c f d e b e d a
9	b c f d e b e d a
6	b d a b c f d e b e d a
14	b e d a
10	c f d e b e d a
2	c f d e b d a b c f d e b e d a
16	d a
7	d a b c f d e b e d a
4	d e b d a b c f d e b e d a
12	d e b e d a
5	e b d a b c f d e b e d a
13	e b e d a
15	e d a
11	f d e b e d a
3	f d e b d a b c f d e b e d a

Die Prefix-Länge jeder zwei benachbarten Suffixe werden verglichen und der längste Prefix gesucht.

Lösung mit Sortieralgorithmen

```
17  a
0   a a c f d e b d a b c f d e b e d a
8   a b c f d e b e d a
1   a c f d e b d a b c f d e b e d a
9   b c f d e b e d a
6   b d a b c f d e b e d a
14  b e d a
10  c f d e b e d a
2   c f d e b d a b c f d e b e d a
16  d a
7   d a b c f d e b e d a
4   d e b d a b c f d e b e d a
12  d e b e d a
5   e b d a b c f d e b e d a
13  e b e d a
15  e d a
11  f d e b e d a
3   f d e b d a b c f d e b e d a
```

Die Prefix-Länge jeder zwei benachbarten Suffixe werden verglichen und der längste Prefix gesucht.

Lösung mit Sortieralgorithmen

```

17  a
0  a a c f d e b d a b c f d e b e d a
8  a b c f d e b e d a
1  a c f d e b d a b c f d e b e d a
9  b c f d e b e d a
6  b d a b c f d e b e d a
14 b e d a
10 c f d e b e d a
2  c f d e b d a b c f d e b e d a
16 d a
7  d a b c f d e b e d a
4  d e b d a b c f d e b e d a
12 d e b e d a
5  e b d a b c f d e b e d a
13 e b e d a
15 e d a
11 f d e b e d a
3  f d e b d a b c f d e b e d a

```

Die Prefix-Länge jeder zwei benachbarten Suffixe werden verglichen und der längste Prefix gesucht.

Implementierung

```
def generate_suffixes(A):  
    """ Find all suffixes of the String A """  
    n = len(A)  
    suffixes = [0 for i in range(len(A))]  
  
    for i in range(n):  
        suffixes[i] = A[i:n]  
  
    return suffixes
```

```
0 a a c f d e b d a b c f d e b e d a  
1 a c f d e b d a b c f d e b e d a  
2 c f d e b d a b c f d e b e d a  
3 f d e b d a b c f d e b e d a  
4 d e b d a b c f d e b e d a  
5 e b d a b c f d e b e d a  
6 b d a b c f d e b e d a  
7 d a b c f d e b e d a  
8 a b c f d e b e d a  
9 b c f d e b e d a  
10 c f d e b e d a  
11 f d e b e d a  
12 d e b e d a  
13 e b e d a  
14 b e d a  
15 e d a  
16 d a  
17 a
```

Implementierung

```
def LRS_Algorithmus(A):  
    B = generate_suffixes(A)  
    B.sort()  
    seq = find_longest_seq(B)  
    return seq
```


Implementierung

```
def find_longest_seq(B):  
    max = 0  
  
    for i in range(len(B)-1):  
        min_len = min(len(B[i]),len(B[i+1]))  
        j = 0  
        while j<min_len and B[i][j]==B[i+1][j]:  
            j += 1  
        if j > max:  
            max = j  
            if j == min_len:  
                seq = B[i][:j+1]  
            else:  
                seq = B[i][:j]  
  
    return seq
```