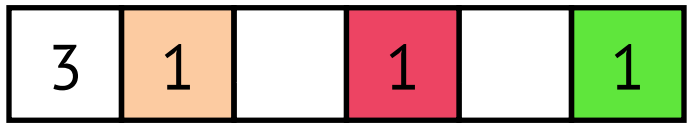
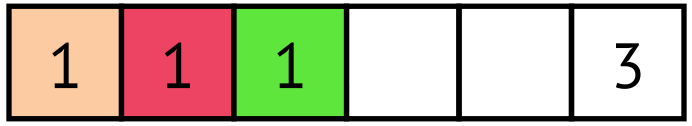
2.3 Sortieren

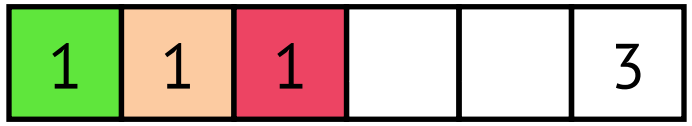
Stabilität von Sortieralgorithmen

- Ein Sortieralgorithmus heißt stabil, wenn sich die relative Reihenfolge von gleichen Elementen während des Sortierens nicht ändert.
- Beispiel:





stabil



nicht stabil

Counting-Sort

- Annahme: $R_1, R_2, \ldots, R_n \in \{1,\ldots,k\}$
- Idee: Bestimme zu jedem R_i die Zahl der Elemente $\leqslant R_i$ und sortiere R_i an die entsprechende Stelle

Counting-Sort: Algorithmus

- CountingSort(A[1..n],C[1..n])

```
for i ← 1 to k do
    B[ i ] ← 0
for i ← 1 to n do
    B[ A[ i ] ] ← B[ A[ i ] ]+1
for i ← 2 to k do
    B[ i ] ← B[ i ] + B[ i−1 ]
for i ← n downto 1 do
    C[ B[ A[ i ] ] ] ← A[ i ]
    B[ A[ i ] ] ← B[ A[ i ] ]−1
```

Counting-Sort: Algorithmus

- CountingSort(A[1..n],C[1..n])

```
for i ← 1 to k do
    B[ i ] ← 0
for i ← 1 to n do
    B[ A[ i ] ] ← B[ A[ i ] ]+1
for i ← 2 to k do
    B[ i ] ← B[ i ] + B[ i−1 ]
for i ← n downto 1 do
    C[ B[ A[ i ] ] ] ← A[ i ]
    B[ A[ i ] ] ← B[ A[ i ] ]−1
```

Eingabe: A[1..n]
Ausgabe: C[1..n]

Counting-Sort: Algorithmus

- CountingSort(A[1..n],C[1..n])

```
for i ← 1 to k do
    B[ i ] ← 0
for i ← 1 to n do
    B[ A[ i ] ] ← B[ A[ i ] ]+1
for i ← 2 to k do
    B[ i ] ← B[ i ] + B[ i−1 ]
for i ← n downto 1 do
    C[ B[ A[ i ] ] ] ← A[ i ]
    B[ A[ i ] ] ← B[ A[ i ] ]−1
```

B ist mit 0en initialisiert

Counting-Sort: Algorithmus

- CountingSort(A[1..n],C[1..n])

```
for i ← 1 to k do
    B[ i ] ← 0
for i ← 1 to n do
    B[ A[ i ] ] ← B[ A[ i ] ]+1
for i ← 2 to k do
    B[ i ] ← B[ i ] + B[ i−1 ]
for i ← n downto 1 do
    C[ B[ A[ i ] ] ] ← A[ i ]
    B[ A[ i ] ] ← B[ A[ i ] ]−1
```

B[j] enthält die Anzahl der Elemente = j

Counting-Sort: Algorithmus

- CountingSort(A[1..n],C[1..n])

```
for i ← 1 to k do
    B[ i ] ← 0
for i ← 1 to n do
    B[ A[ i ] ] ← B[ A[ i ] ]+1
for i ← 2 to k do
    B[ i ] ← B[ i ] + B[ i−1 ]
for i ← n downto 1 do
    C[ B[ A[ i ] ] ] ← A[ i ]
    B[ A[ i ] ] ← B[ A[ i ] ]−1
```

B[j] enthält die Anzahl der Elemente ⩽ j

Counting-Sort: Algorithmus

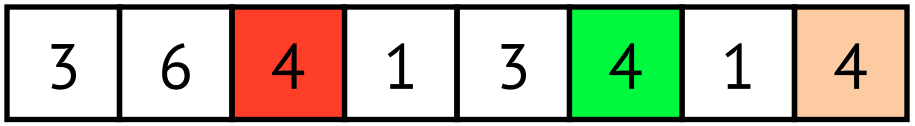
- CountingSort(A[1..n],C[1..n])

```
for i ← 1 to k do
    B[ i ] ← 0
for i ← 1 to n do
    B[ A[ i ] ] ← B[ A[ i ] ]+1
for i ← 2 to k do
    B[ i ] ← B[ i ] + B[ i−1 ]
for i ← n downto 1 do
    C[ B[ A[ i ] ] ] ← A[ i ]
    B[ A[ i ] ] ← B[ A[ i ] ]−1
```

B[j] enthält das j-te Element

Counting-Sort: Beispiel

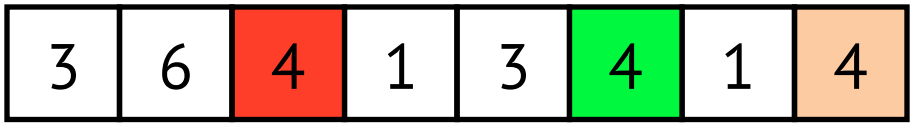
A



n=8, k=6

Counting-Sort: Beispiel

A

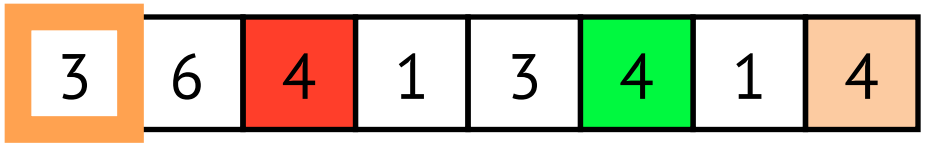


n=8, k=6

	1	2	3	4	5	6
B	0	0	0	0	0	0

Counting-Sort: Beispiel

A

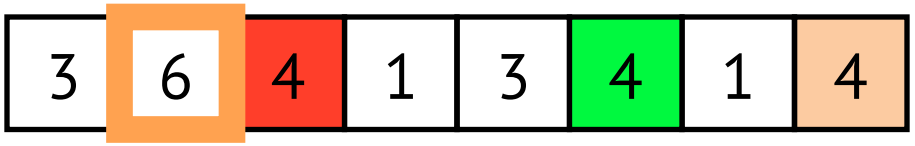


n=8, k=6

	1	2	3	4	5	6
B	0	0	1	0	0	0

Counting-Sort: Beispiel

A

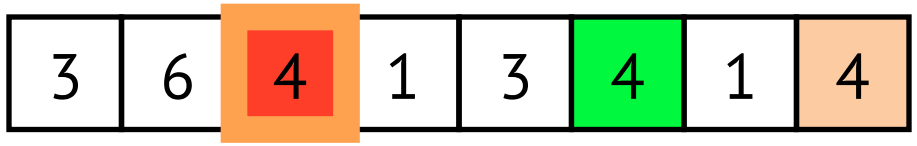


n=8, k=6

	1	2	3	4	5	6
B	0	0	1	0	0	1

Counting-Sort: Beispiel

A

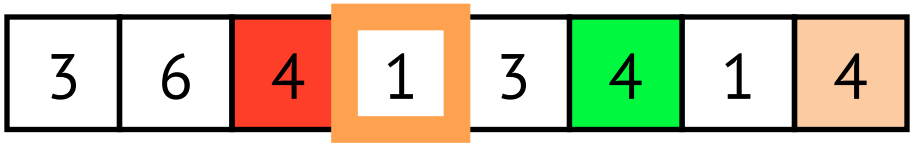


n=8, k=6

	1	2	3	4	5	6
B	0	0	1	1	0	1

Counting-Sort: Beispiel

A



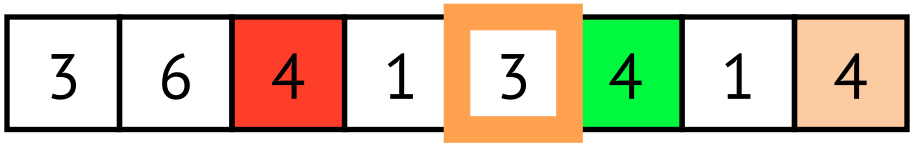
n=8, k=6

B

1	2	3	4	5	6
1	0	1	1	0	1

Counting-Sort: Beispiel

A

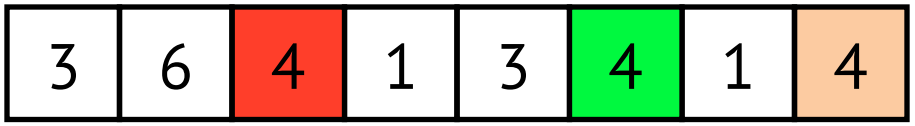


n=8, k=6

	1	2	3	4	5	6
B	1	0	2	1	0	1

Counting-Sort: Beispiel

A



n=8, k=6

	1	2	3	4	5	6
B	2	0	2	3	0	1

Counting-Sort: Beispiel

A

3	6	4	1	3	4	1	4

n=8, k=6

B

1	2	3	4	5	6
2	0	2	3	0	1

Counting-Sort: Beispiel

A

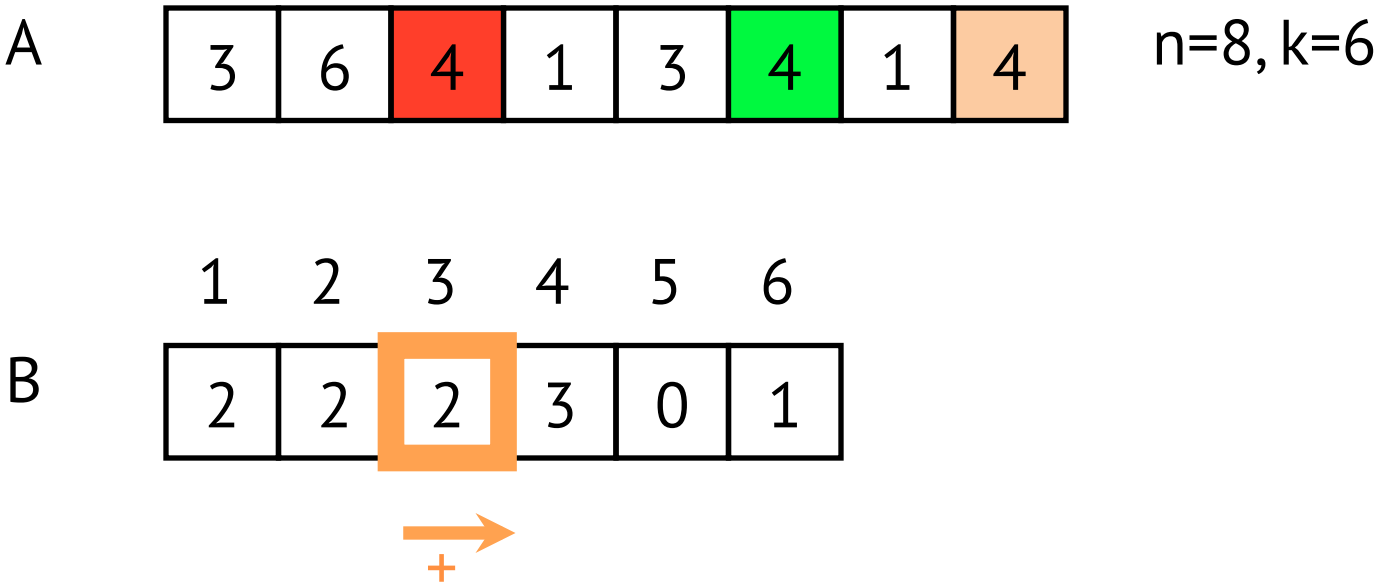
3	6	4	1	3	4	1	4

n=8, k=6

B

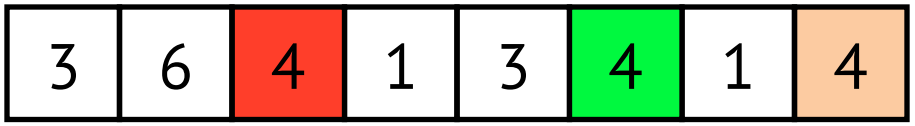
1	2	3	4	5	6
2	2	2	3	0	1

Counting-Sort: Beispiel



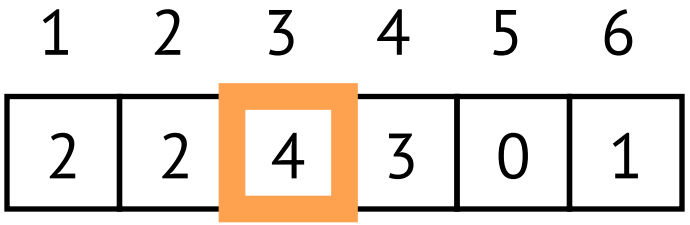
Counting-Sort: Beispiel

A



n=8, k=6

B



Counting-Sort: Beispiel

A

3	6	4	1	3	4	1	4

n=8, k=6

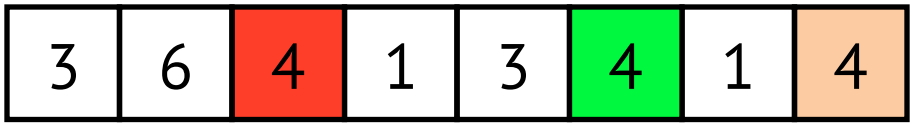
B

1	2	3	4	5	6
2	2	4	3	0	1

+

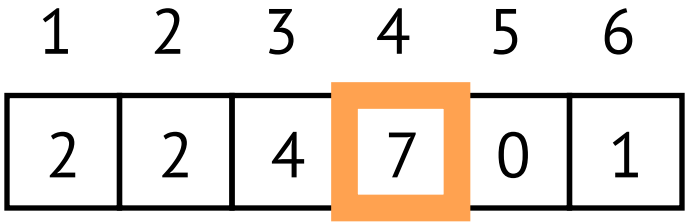
Counting-Sort: Beispiel

A



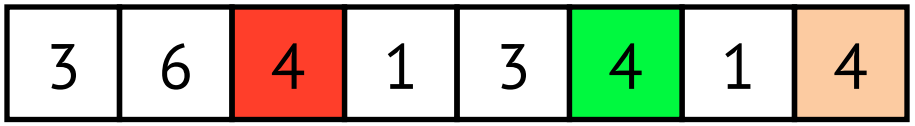
n=8, k=6

B



Counting-Sort: Beispiel

A



n=8, k=6

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

Counting-Sort: Beispiel

A

3	6	4	1	3	4	1	4

n=8, k=6

B

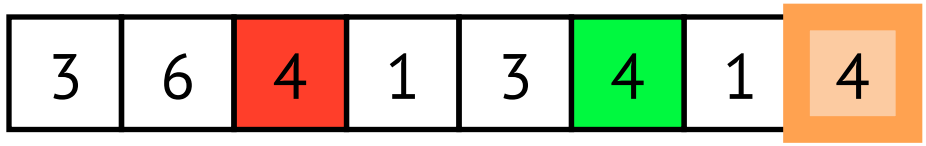
1	2	3	4	5	6
2	2	4	7	7	8

C

1	2	3	4	5	6	7	8

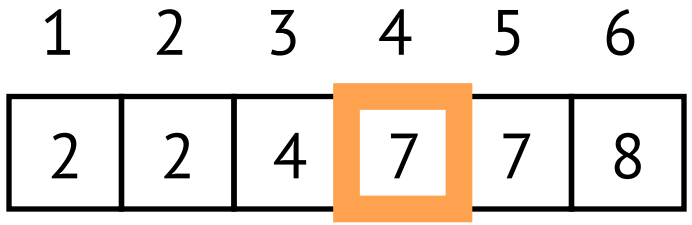
Counting-Sort: Beispiel

A

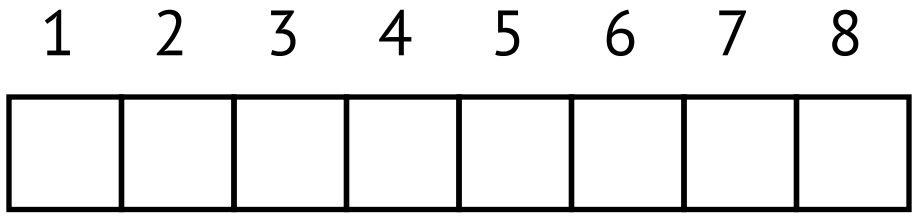


n=8, k=6

B

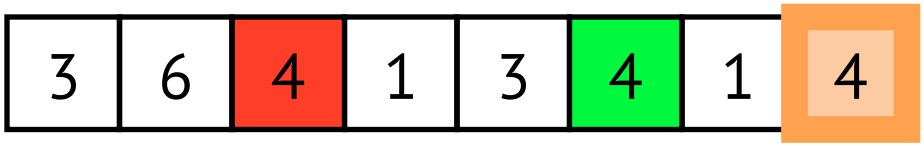


C



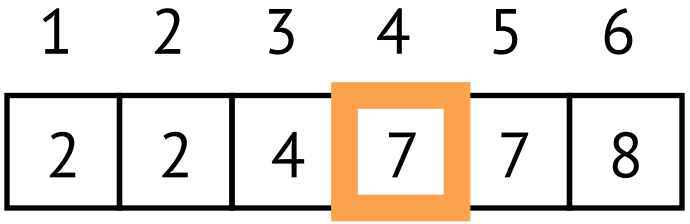
Counting-Sort: Beispiel

A

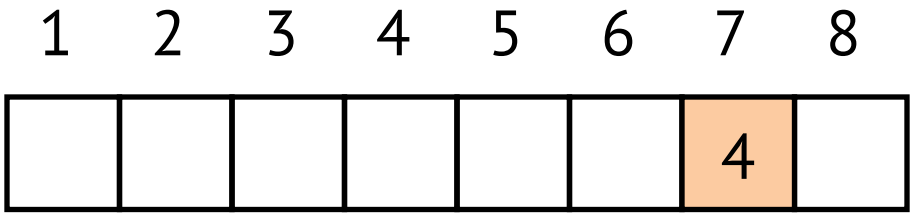


n=8, k=6

B

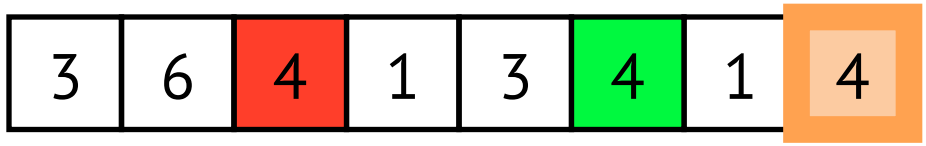


C



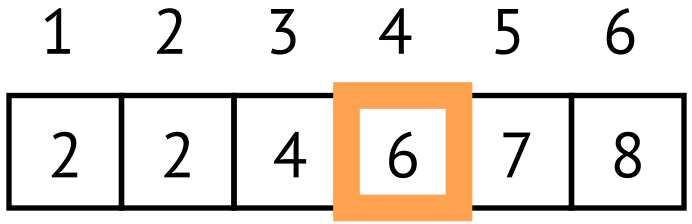
Counting-Sort: Beispiel

A

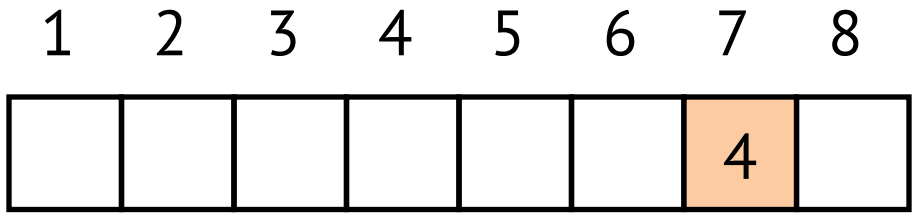


n=8, k=6

B

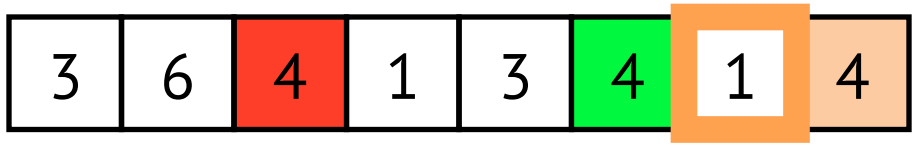


C



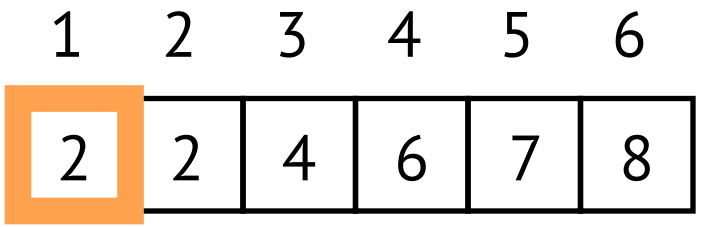
Counting-Sort: Beispiel

A

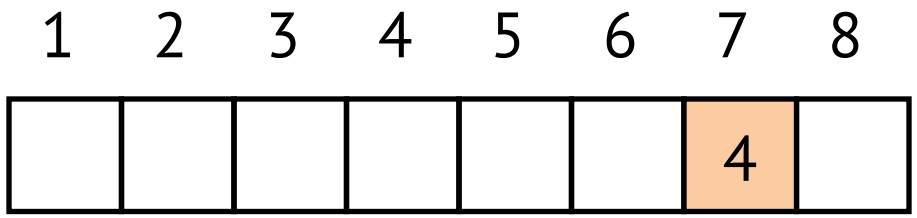


n=8, k=6

B

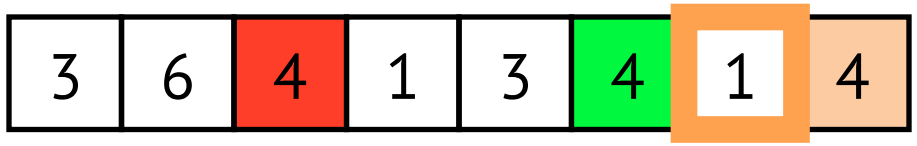


C



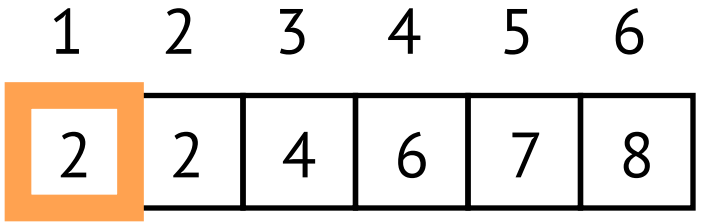
Counting-Sort: Beispiel

A

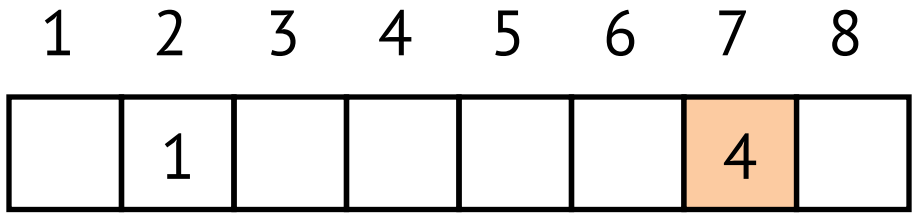


n=8, k=6

B

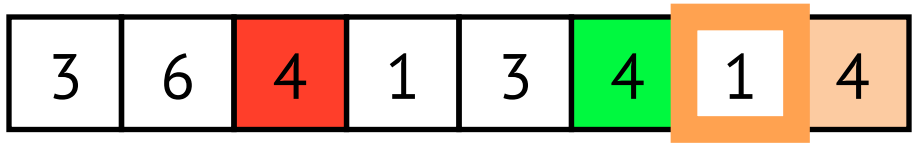


C



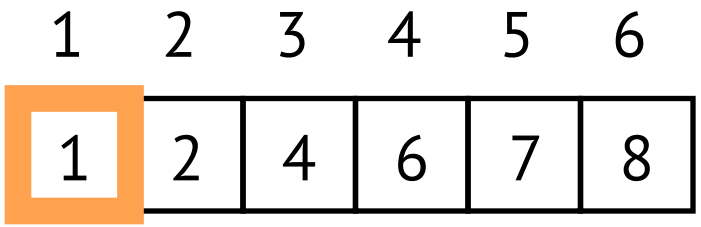
Counting-Sort: Beispiel

A

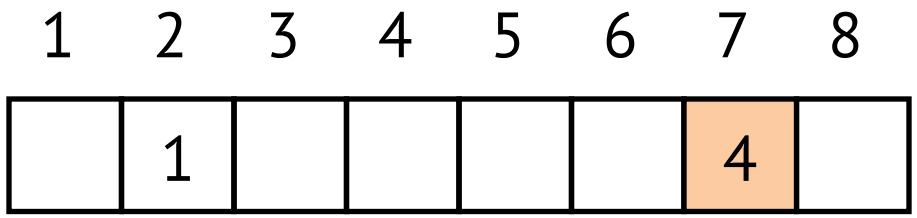


n=8, k=6

B

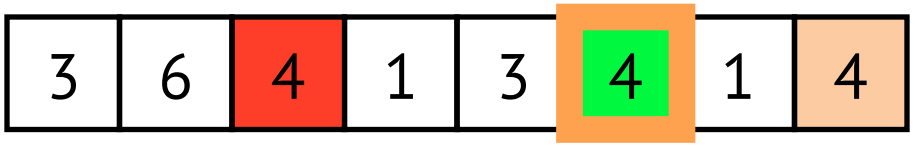


C



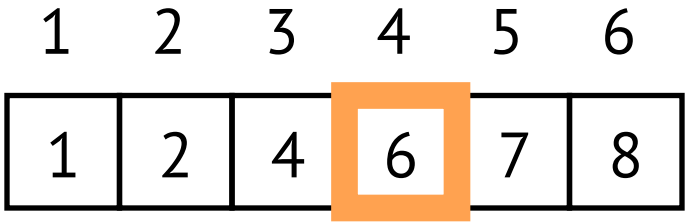
Counting-Sort: Beispiel

A

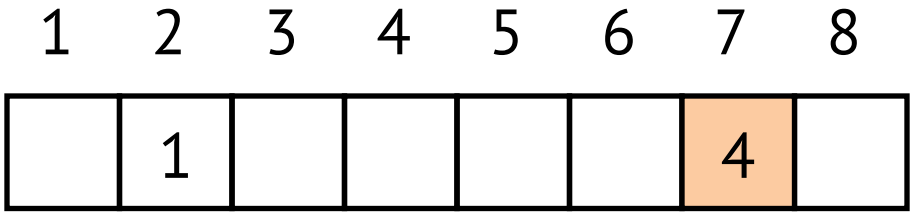


n=8, k=6

B

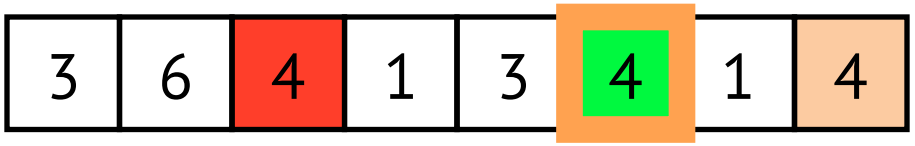


C



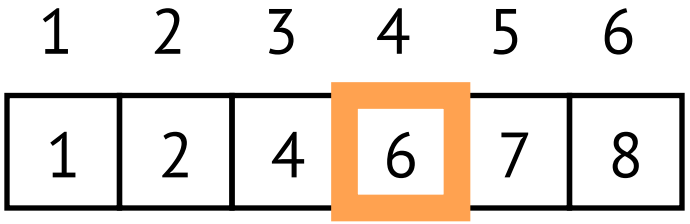
Counting-Sort: Beispiel

A

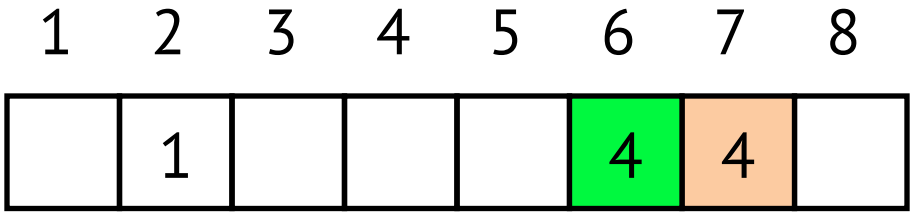


n=8, k=6

B

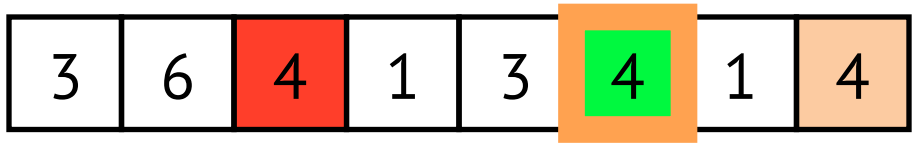


C



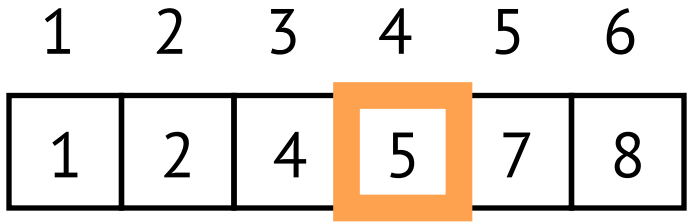
Counting-Sort: Beispiel

A

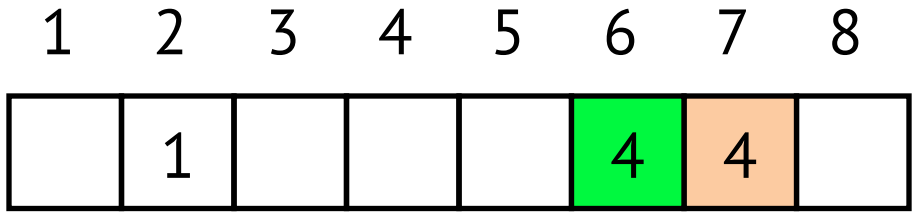


n=8, k=6

B

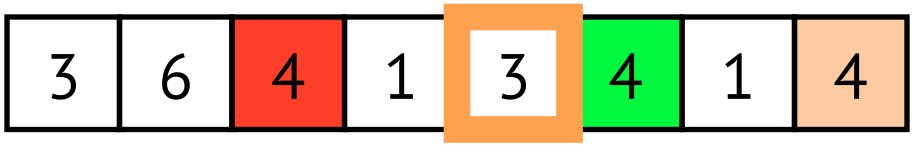


C



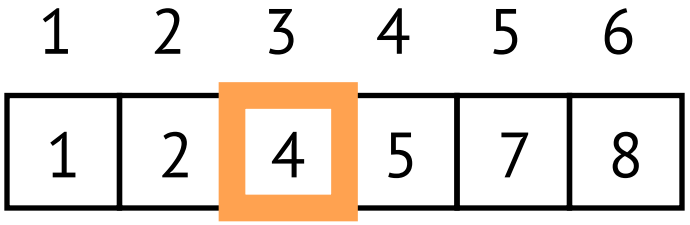
Counting-Sort: Beispiel

A

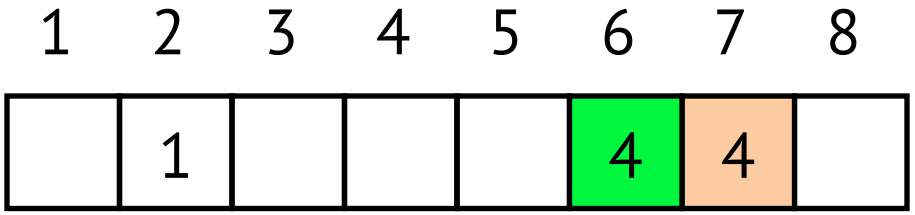


n=8, k=6

B

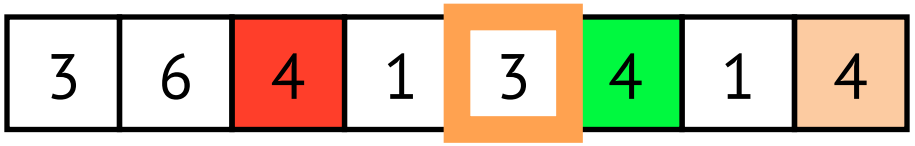


C



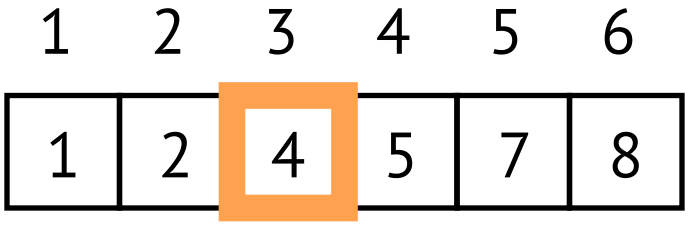
Counting-Sort: Beispiel

A

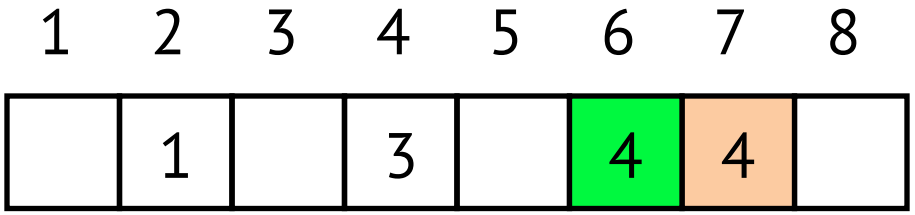


n=8, k=6

B

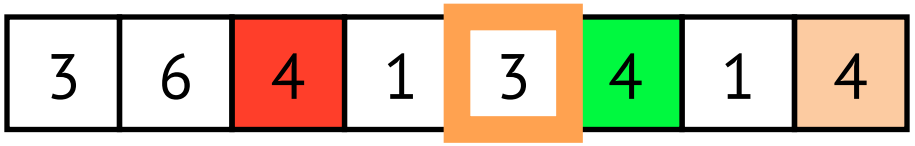


C



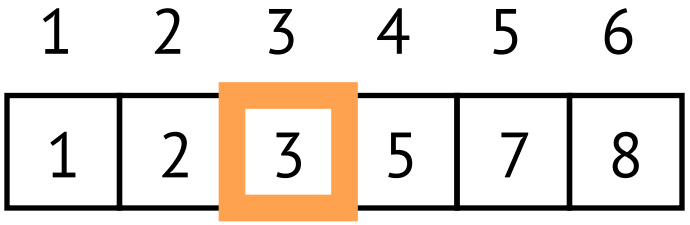
Counting-Sort: Beispiel

A

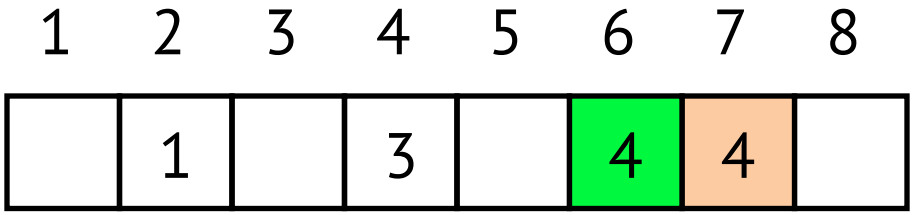


n=8, k=6

B

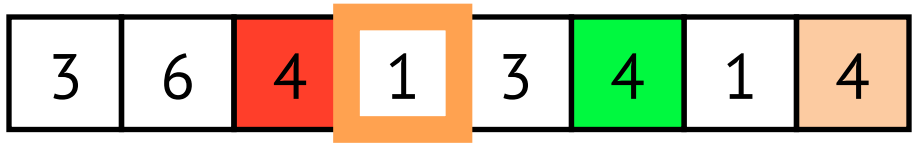


C



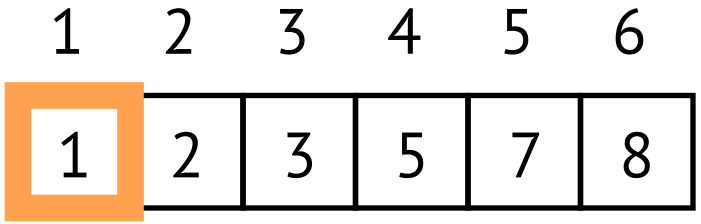
Counting-Sort: Beispiel

A

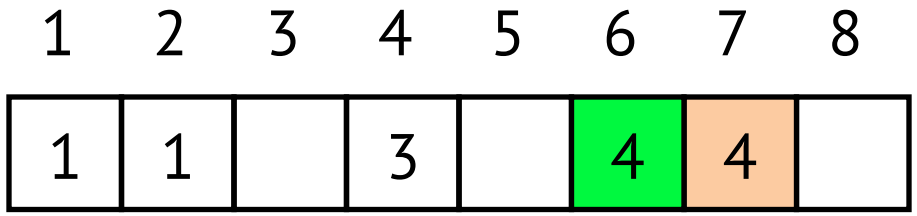


n=8, k=6

B

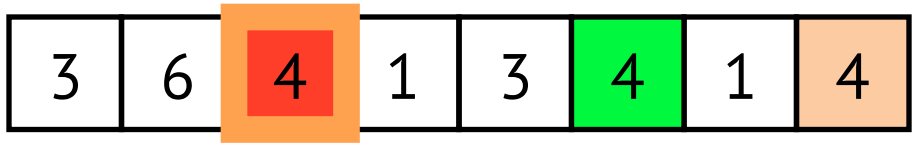


C



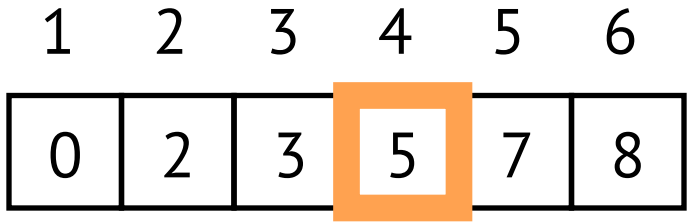
Counting-Sort: Beispiel

A

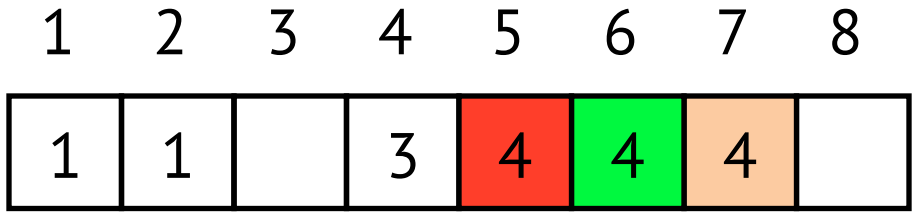


n=8, k=6

B

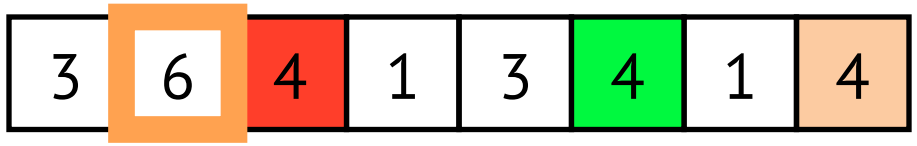


C



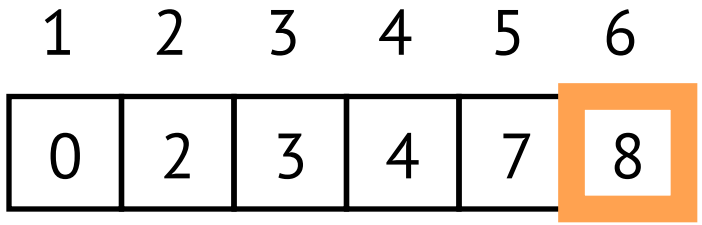
Counting-Sort: Beispiel

A

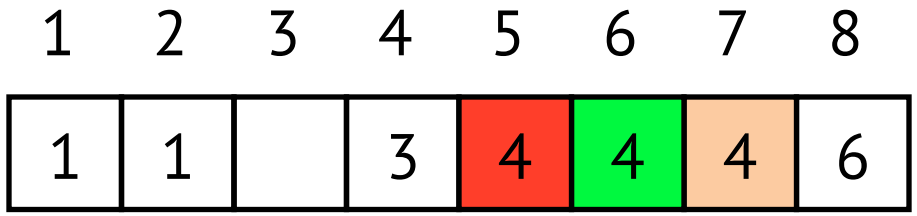


n=8, k=6

B

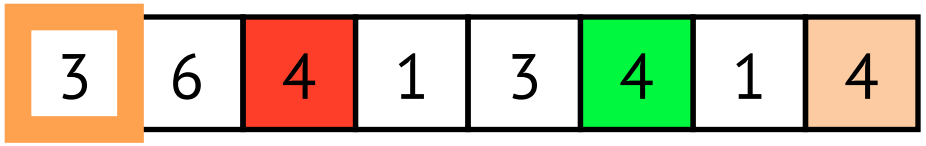


C



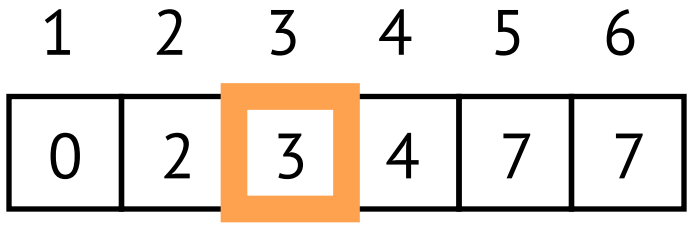
Counting-Sort: Beispiel

A

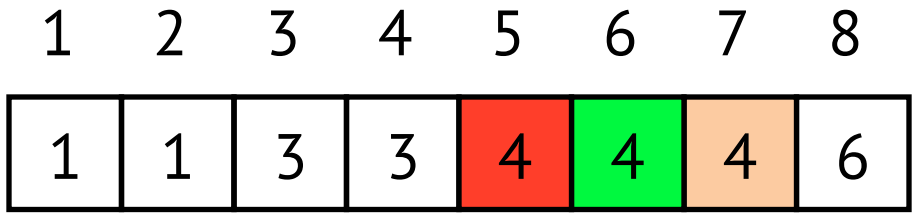


n=8, k=6

B



C



Counting-Sort: Beispiel

A

3	6	4	1	3	4	1	4

n=8, k=6

B

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

C

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

Counting-Sort: Aufwand

- CountingSort(A[1..n],C[1..n])

```
for i ← 1 to k do                          O(k)
    B[ i ] ← 0
for i ← 1 to n do                          O(n)
    B[ A[ i ] ] ← B[ A[ i ] ]+1
for i ← 2 to k do                          O(k)
    B[ i ] ← B[ i ] + B[ i−1 ]
for i ← n downto 1 do                      O(n)
    C[ B[ A[ i ] ] ] ← A[ i ]
    B[ A[ i ] ] ← B[ A[ i ] ]−1            ———————
                                           O(k+n)
```

Counting-Sort: Aufwand

- Counting-Sort ist nur sinnvoll, wenn $k = O(n)$ und damit
  $$T(n) = O(n)$$
- Counting-Sort verwendet keine Vergleiche
- Counting-Sort ist stabil

Radix-Sort

- Annahme: $R_1, R_2, \ldots, R_n \in \{0,\ldots,k^d-1\}$
- „d-stellige Zahlen zur Basis k“
- „Wörter der Länge d aus einem Alphabet der Größe k“

Radix-Sort

- RadixSort(A[1..n])
  for i ← 1 to d do
    „sortiere A stabil nach Stelle i“

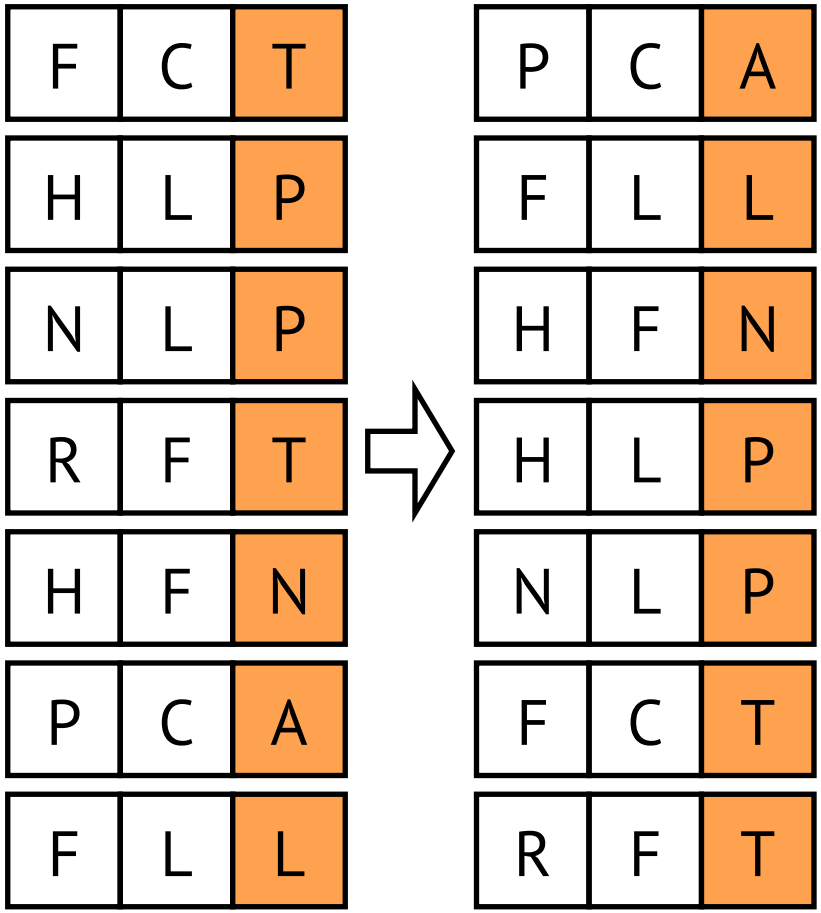
Radix-Sort: Beispiel

F	C	T
H	L	P
N	L	P
R	F	T
H	F	N
P	C	A
F	L	L

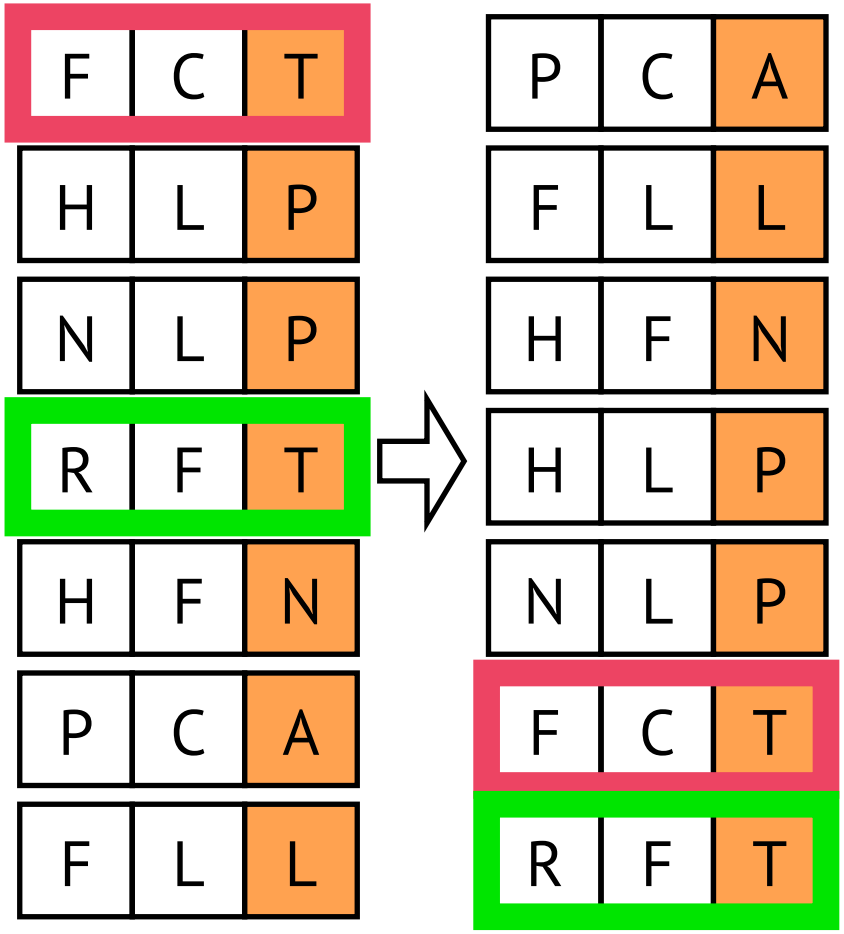
Radix-Sort: Beispiel

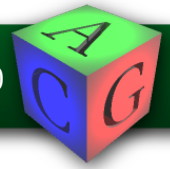
F	C	T
H	L	P
N	L	P
R	F	T
H	F	N
P	C	A
F	L	L

Radix-Sort: Beispiel

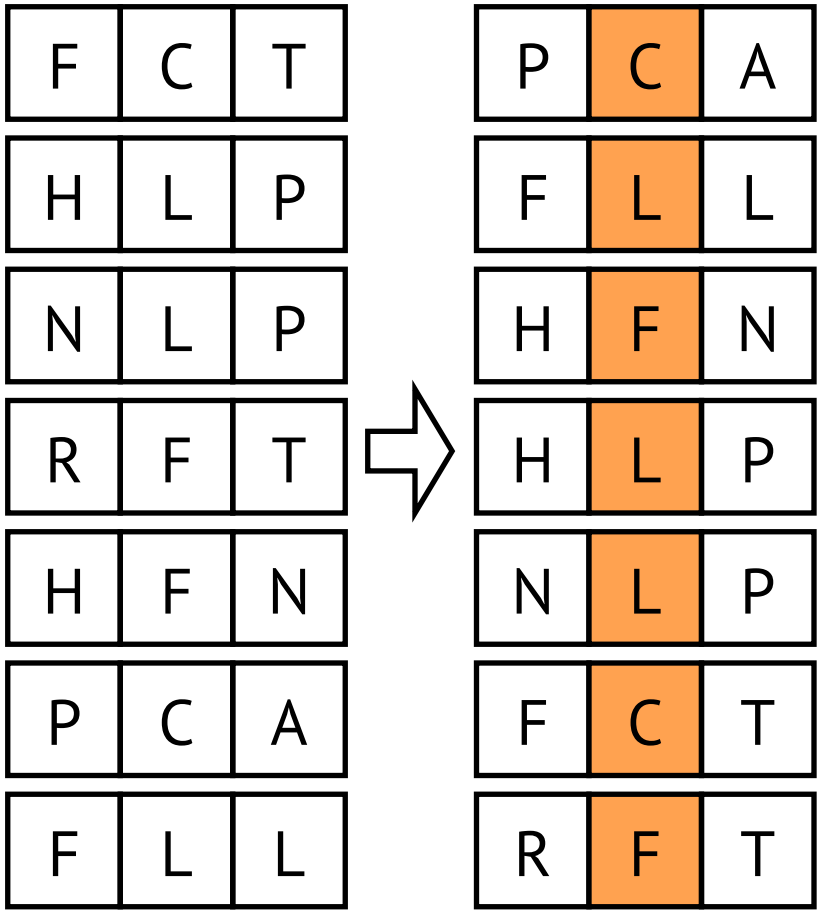


Radix-Sort: Beispiel

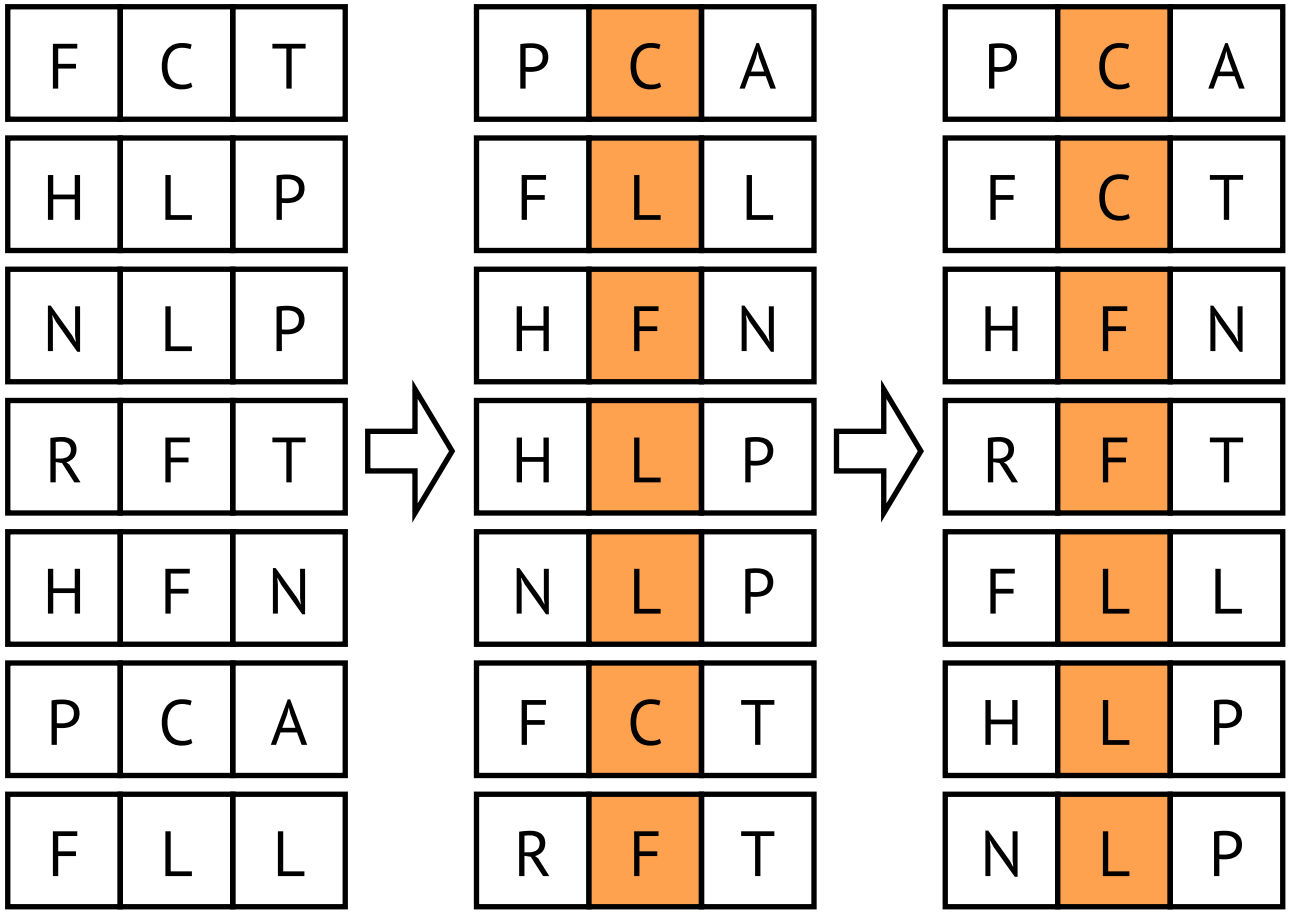




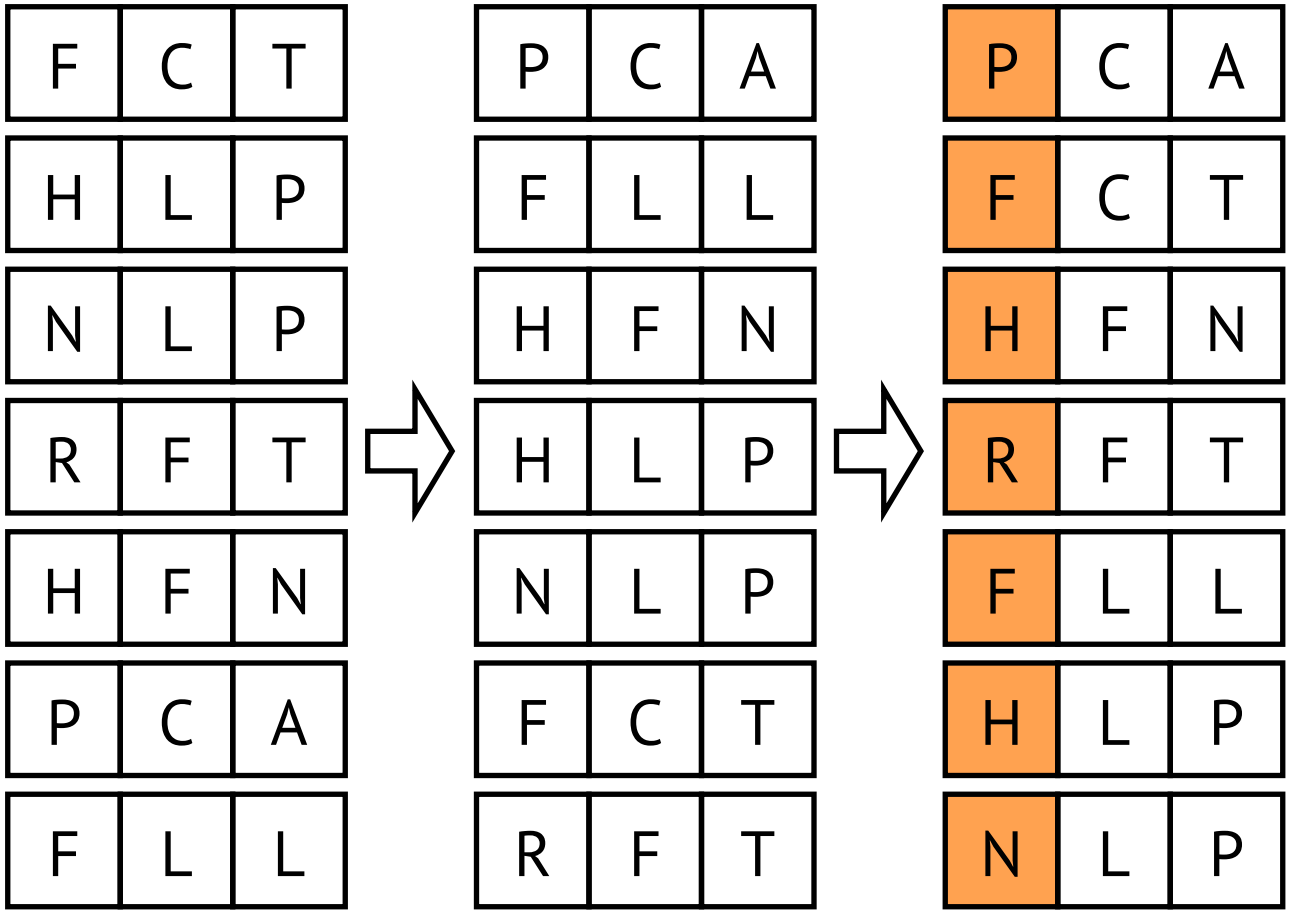
Radix-Sort: Beispiel



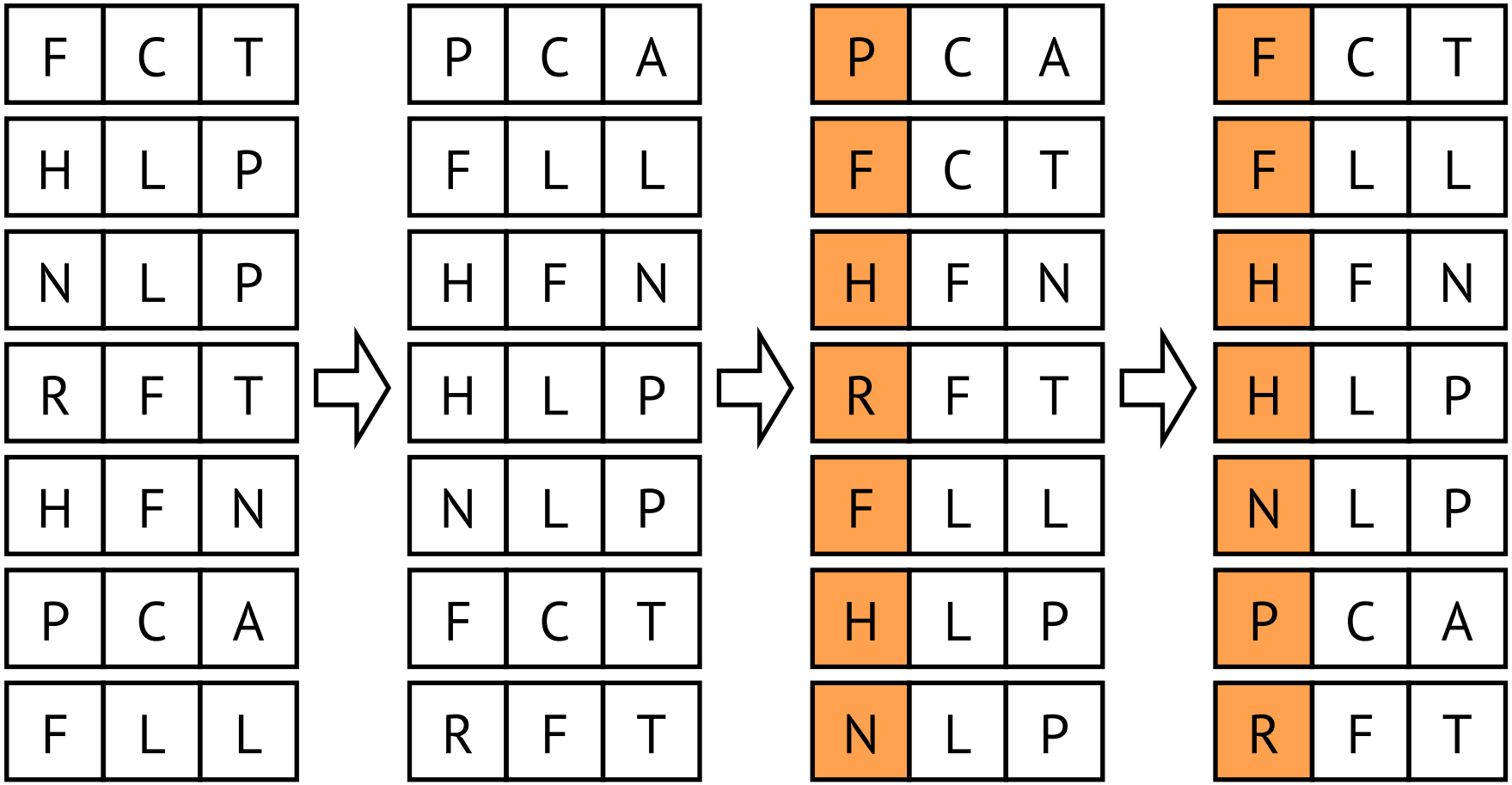
Radix-Sort: Beispiel



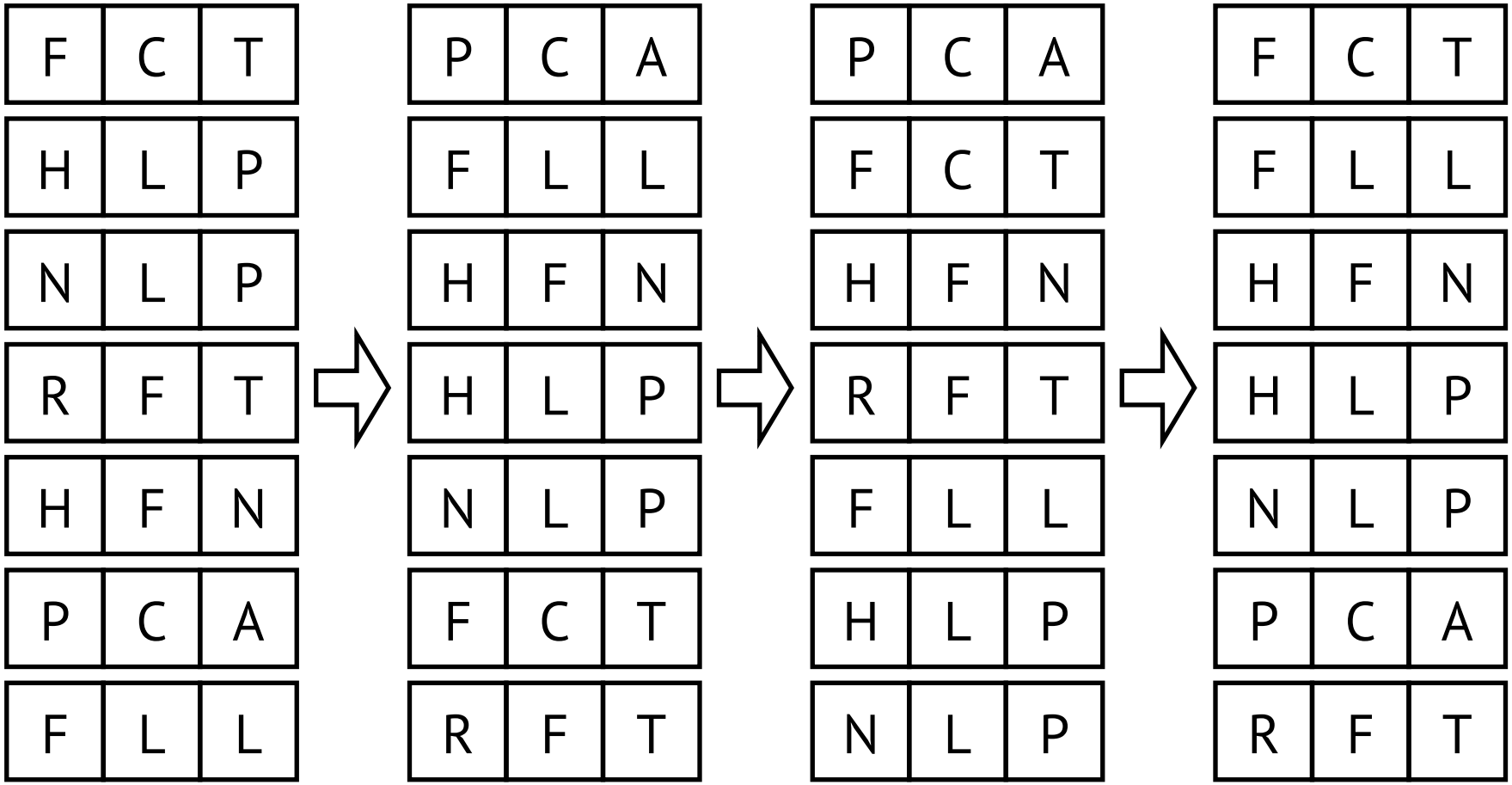
Radix-Sort: Beispiel



Radix-Sort: Beispiel



Radix-Sort: Beispiel



Radix-Sort: Aufwand

- RadixSort(A[1..n])
  for i ← 1 to d do
    „sortiere A stabil nach Stelle i“

Radix-Sort: Aufwand

- RadixSort(A[1..n])
  for i ← 1 to d do
    „sortiere A stabil mit
     Counting-Sort nach Stelle i“

Radix-Sort: Aufwand

- RadixSort(A[1..n])
  for i ← 1 to d do
    „sortiere A stabil mit
     Counting-Sort nach Stelle i“

$$T(k,d,n)=O(d\times(n+k))$$

Radix-Sort: Aufwand

- Beachte: Ist k gegeben, so benötigt man zur Darstellung von n verschiedenen Elementen mindestens $d \geqslant \log_k(n)$ Stellen

$$\rightarrow \; T(n) = \Omega(n \times \log n)$$

Bucket-Sort

- Annahme:
  $R_1, R_2, \ldots, R_n$ gleichverteilt aus $[0,1)$
- Idee:
  1. Unterteile $[0,1)$ in n „Buckets“
     $[0,1/n), [1/n,2/n), \ldots, [(n-1)/n,1)$
  2. Füge die R_i in die Buckets ein (erwartet: ein Element pro Bucket)
  3. Sortiere die Buckets
  4. Hänge die Buckets aneinander

Bucket-Sort: Algorithmus

- BucketSort(A[1..n])

```
new B[0..n−1]
for i ← 1 to n do
    push(B[⌊n×A[ i ]⌋], A[ i ])
for i ← 0 to n−1 do
    sort(B[ i ])
c ← 1
for i ← 0 to n−1 do
    for j ← 1 to size(B[ i ]) do
        A[c] ← B[ i ][ j ]
        c ← c+1
```

Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

0 [0.0,0.1)
1 [0.1,0.2)
2 [0.2,0.3)
3 [0.3,0.4)
4 [0.4,0.5)
5 [0.5,0.6)
6 [0.6,0.7)
7 [0.7,0.8)
8 [0.8,0.9)
9 [0.9,1.0)

n=10

Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

n=10

0	[0.0,0.1)
1	[0.1,0.2)
2	[0.2,0.3)
3	[0.3,0.4)
4	[0.4,0.5)
5	[0.5,0.6) → 0.59
6	[0.6,0.7)
7	[0.7,0.8)
8	[0.8,0.9)
9	[0.9,1.0)

Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

n=10

0	[0.0,0.1)	
1	[0.1,0.2)	
2	[0.2,0.3)	
3	[0.3,0.4)	
4	[0.4,0.5)	
5	[0.5,0.6)	→ 0.59
6	[0.6,0.7)	
7	[0.7,0.8)	→ 0.73
8	[0.8,0.9)	
9	[0.9,1.0)	

Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

n=10

0	[0.0,0.1)	
1	[0.1,0.2)	
2	[0.2,0.3)	→ 0.24
3	[0.3,0.4)	
4	[0.4,0.5)	
5	[0.5,0.6)	→ 0.59
6	[0.6,0.7)	
7	[0.7,0.8)	→ 0.73
8	[0.8,0.9)	
9	[0.9,1.0)	

Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

n=10

0 [0.0,0.1)
1 [0.1,0.2)
2 [0.2,0.3) → 0.24
3 [0.3,0.4)
4 [0.4,0.5)
5 [0.5,0.6) → 0.59
6 [0.6,0.7)
7 [0.7,0.8) → 0.73 → 0.75
8 [0.8,0.9)
9 [0.9,1.0)

Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

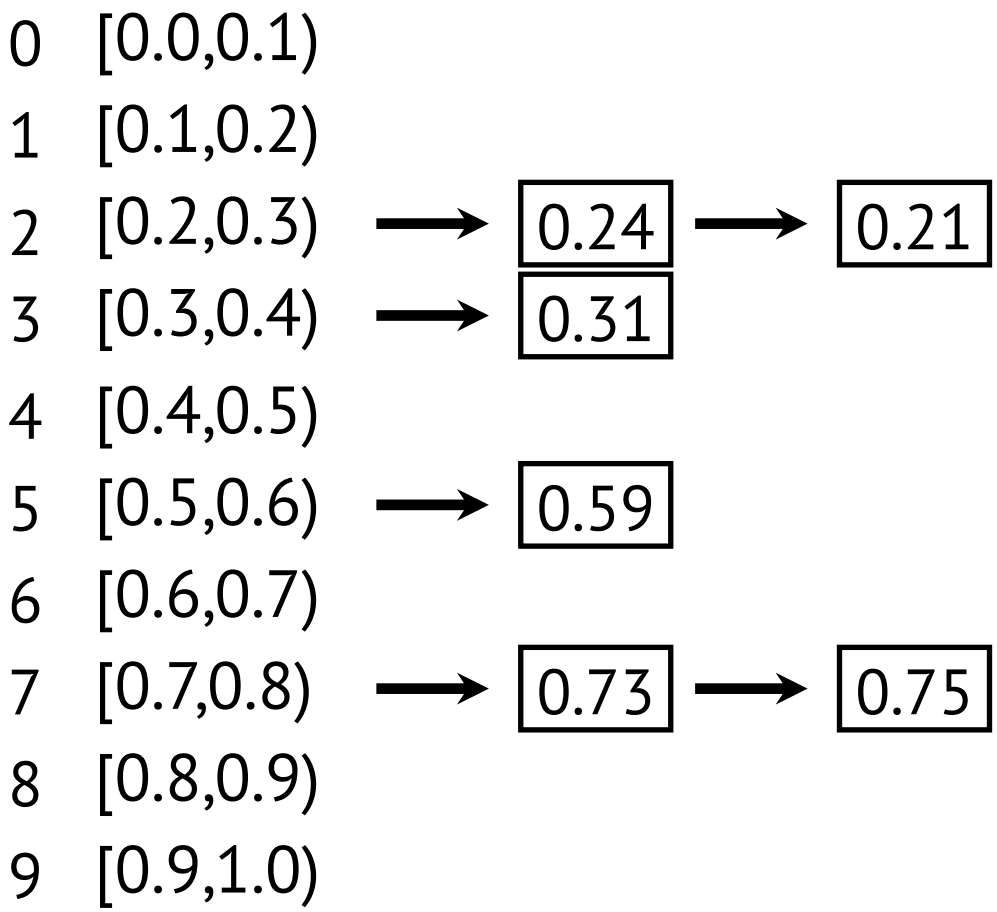
n=10

0	[0.0,0.1)	
1	[0.1,0.2)	
2	[0.2,0.3)	→ 0.24
3	[0.3,0.4)	→ 0.31
4	[0.4,0.5)	
5	[0.5,0.6)	→ 0.59
6	[0.6,0.7)	
7	[0.7,0.8)	→ 0.73 → 0.75
8	[0.8,0.9)	
9	[0.9,1.0)	

Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

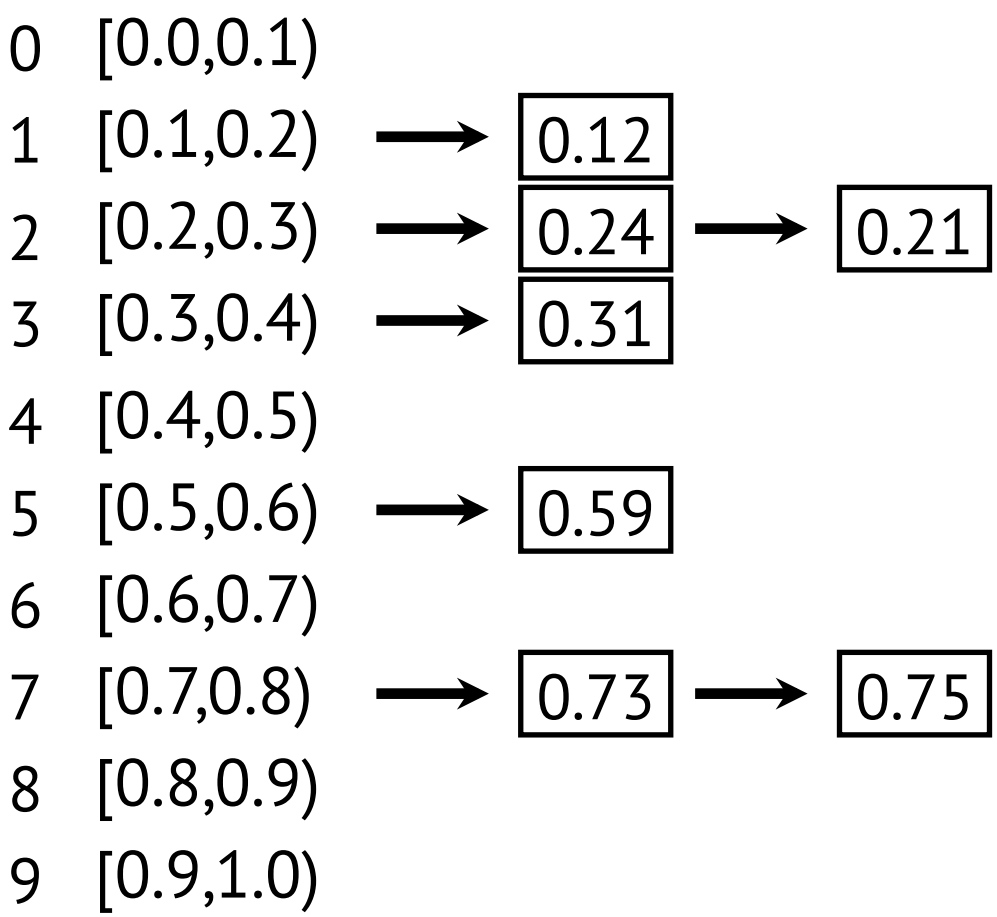
n=10



Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

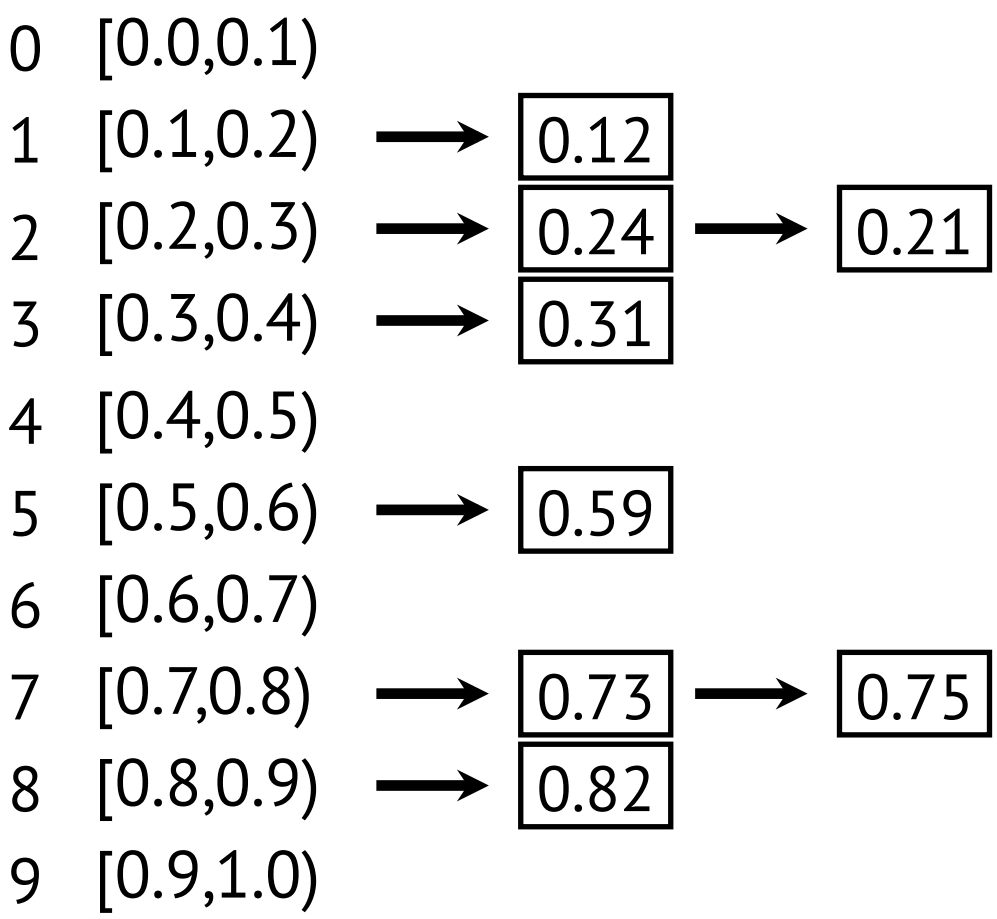
n=10



Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

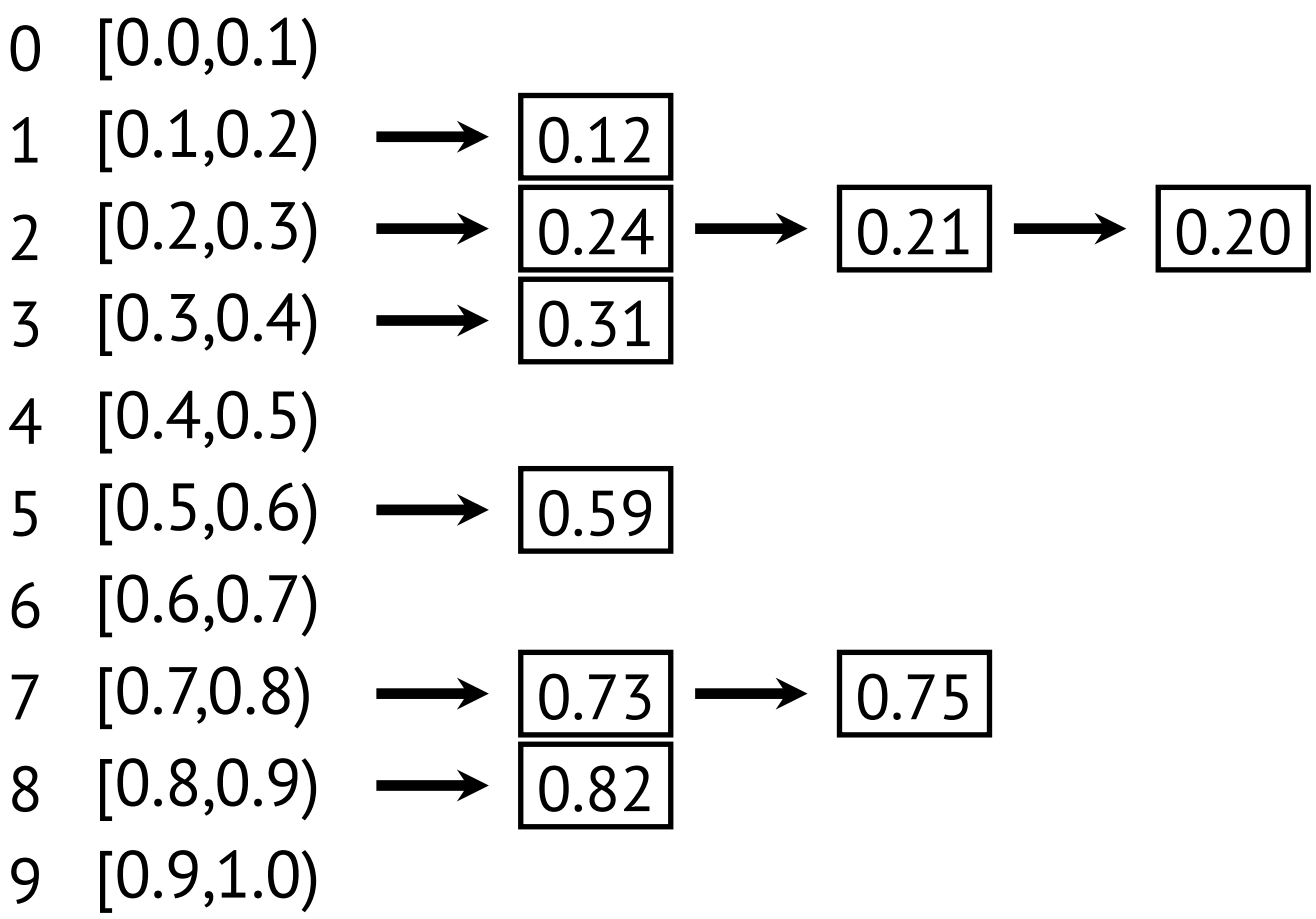
n=10



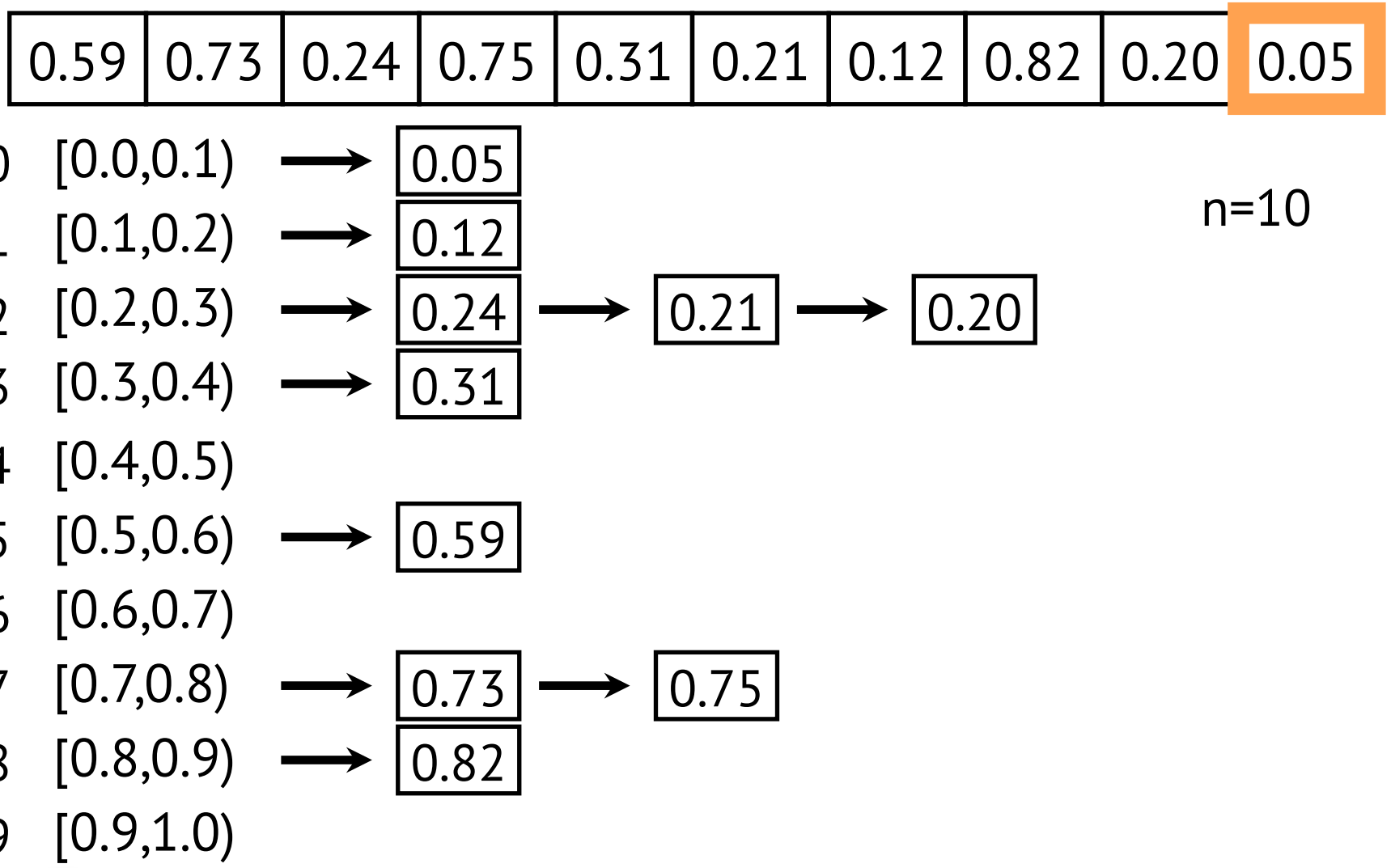
Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

n=10

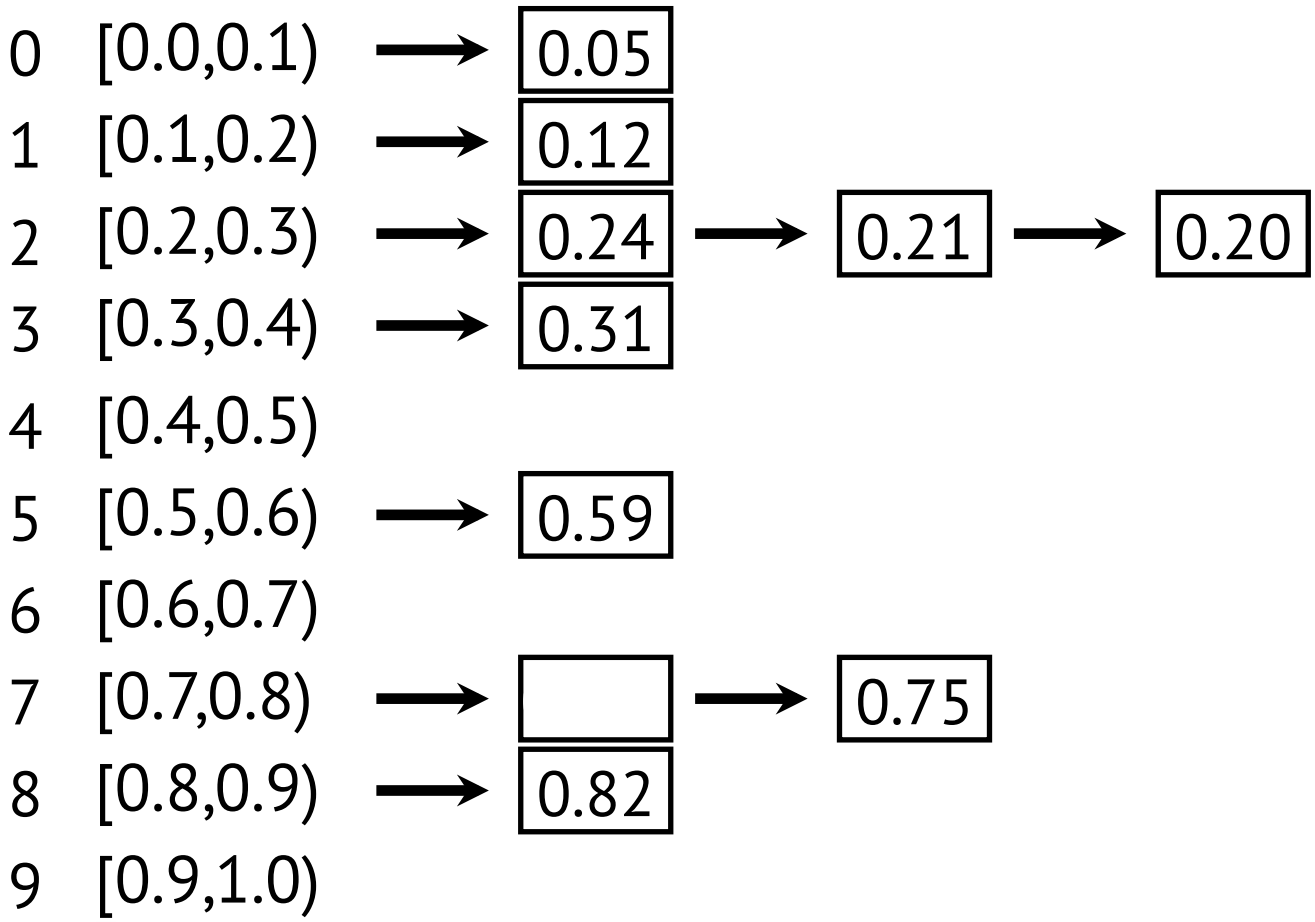


Bucket-Sort: Beispiel



Bucket-Sort: Beispiel

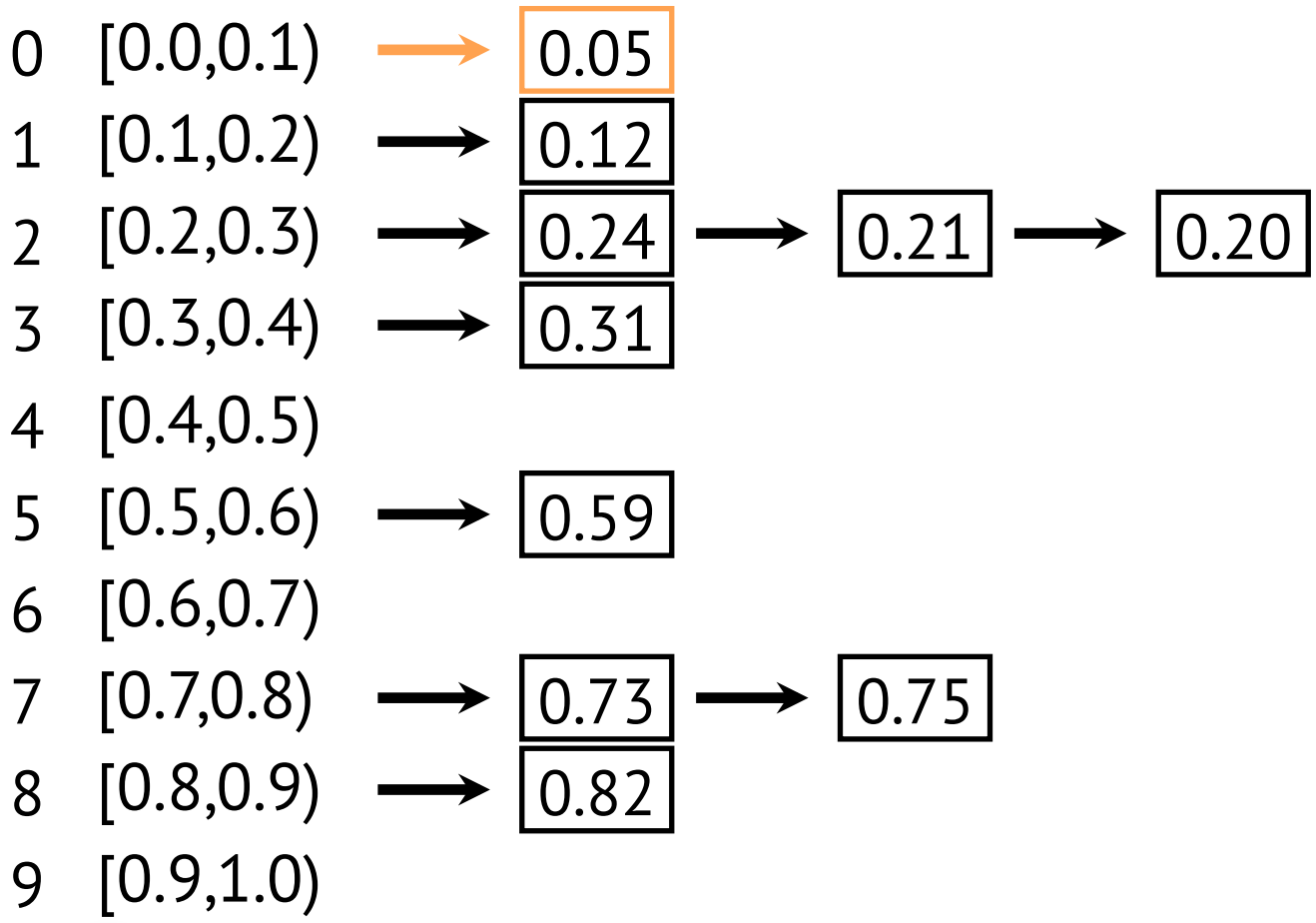
0.59		0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05



n=10

Bucket-Sort: Beispiel

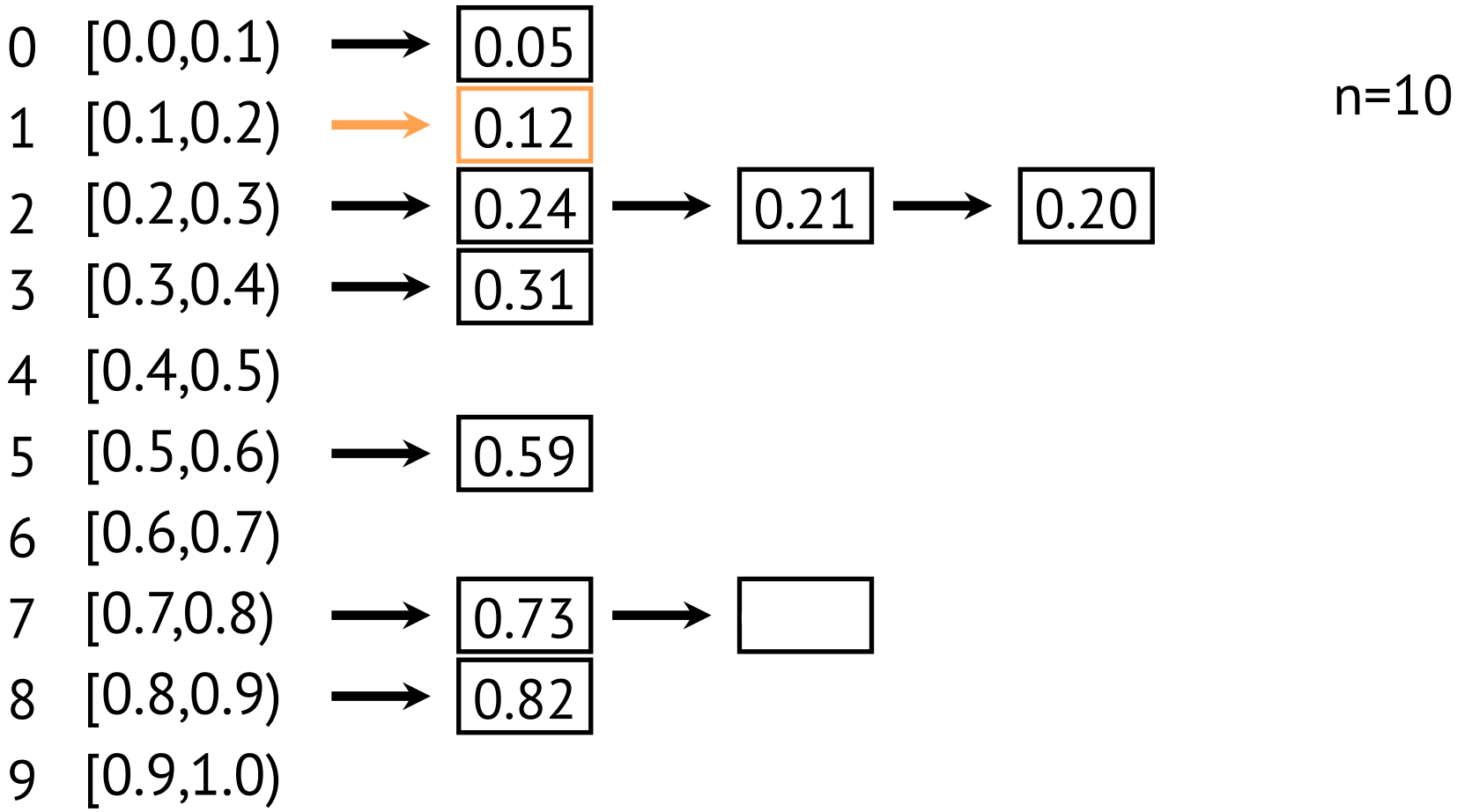
0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05



n=10

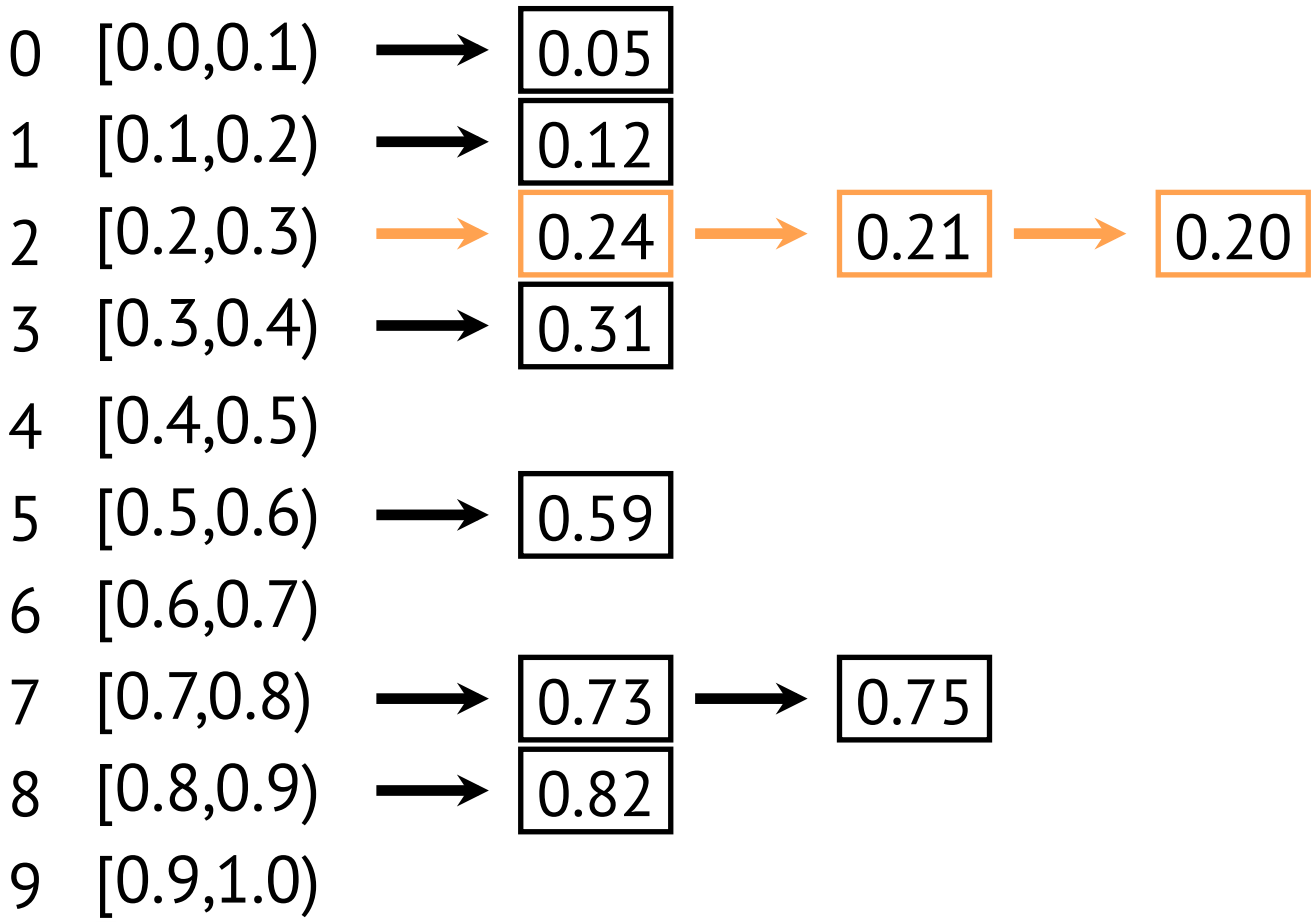
Bucket-Sort: Beispiel

0.59	0.73	0.24		0.31	0.21	0.12	0.82	0.20	0.05



Bucket-Sort: Beispiel

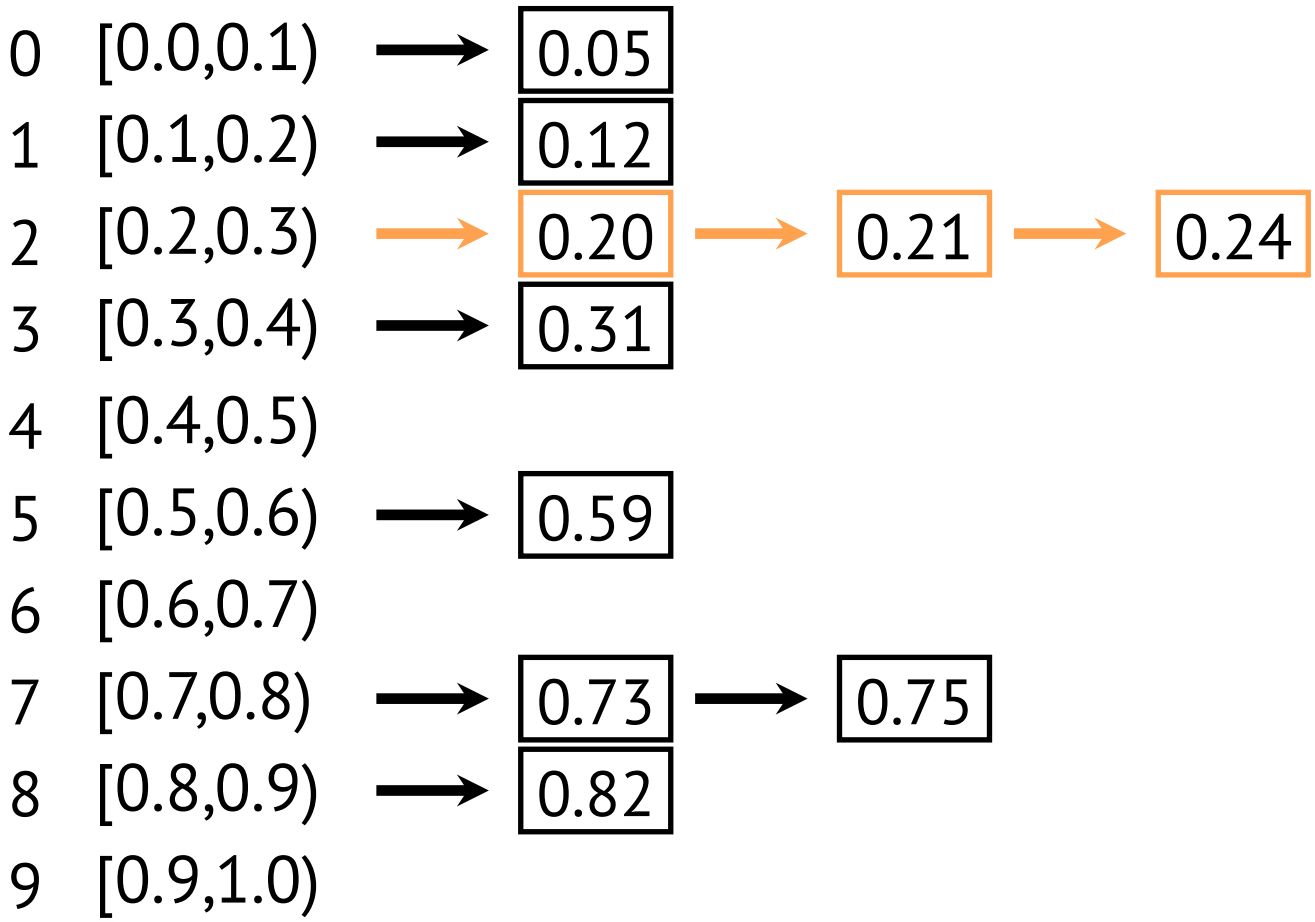
0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05



n=10

Bucket-Sort: Beispiel

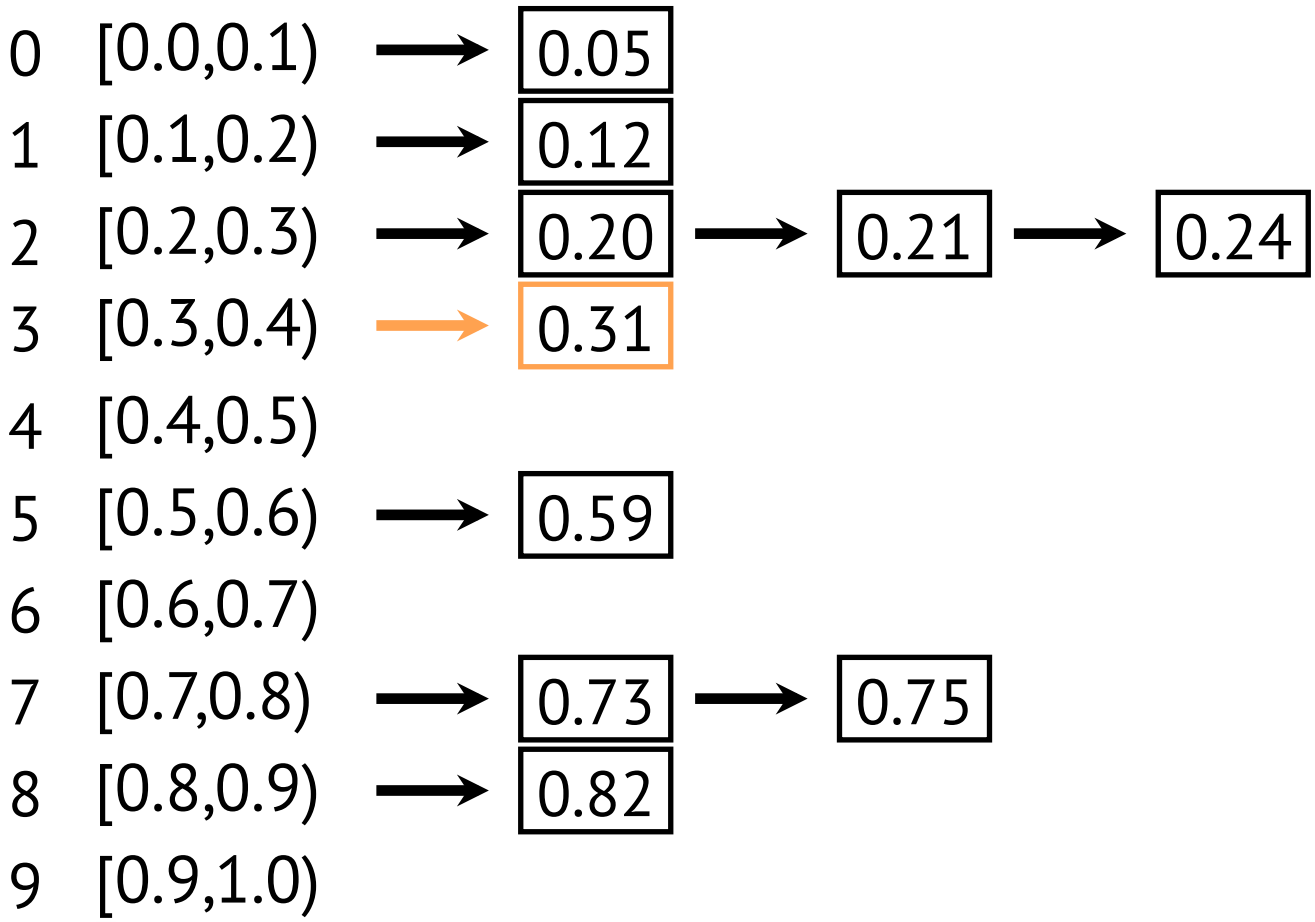
0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05



n=10

Bucket-Sort: Beispiel

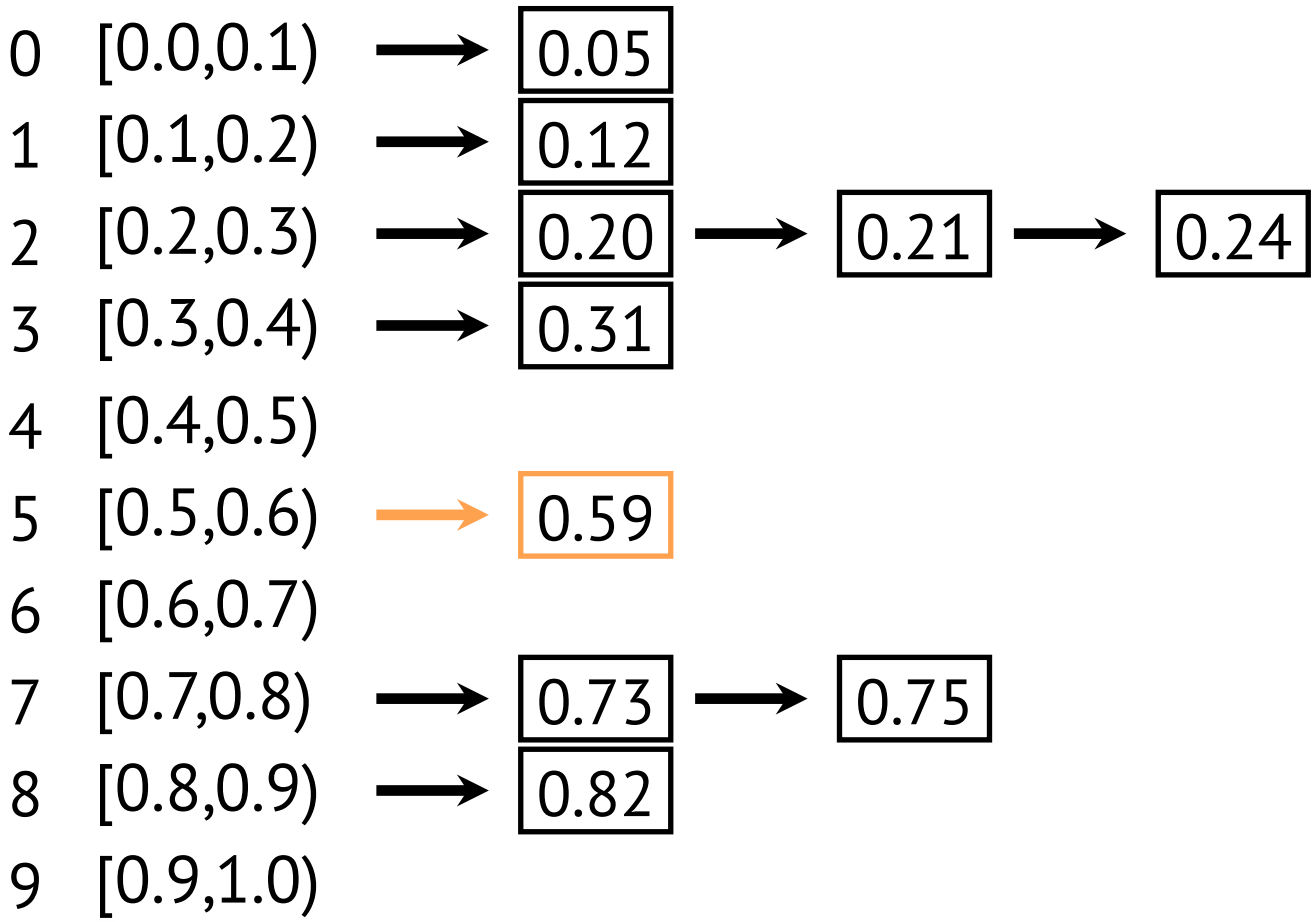
0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05



n=10

Bucket-Sort: Beispiel

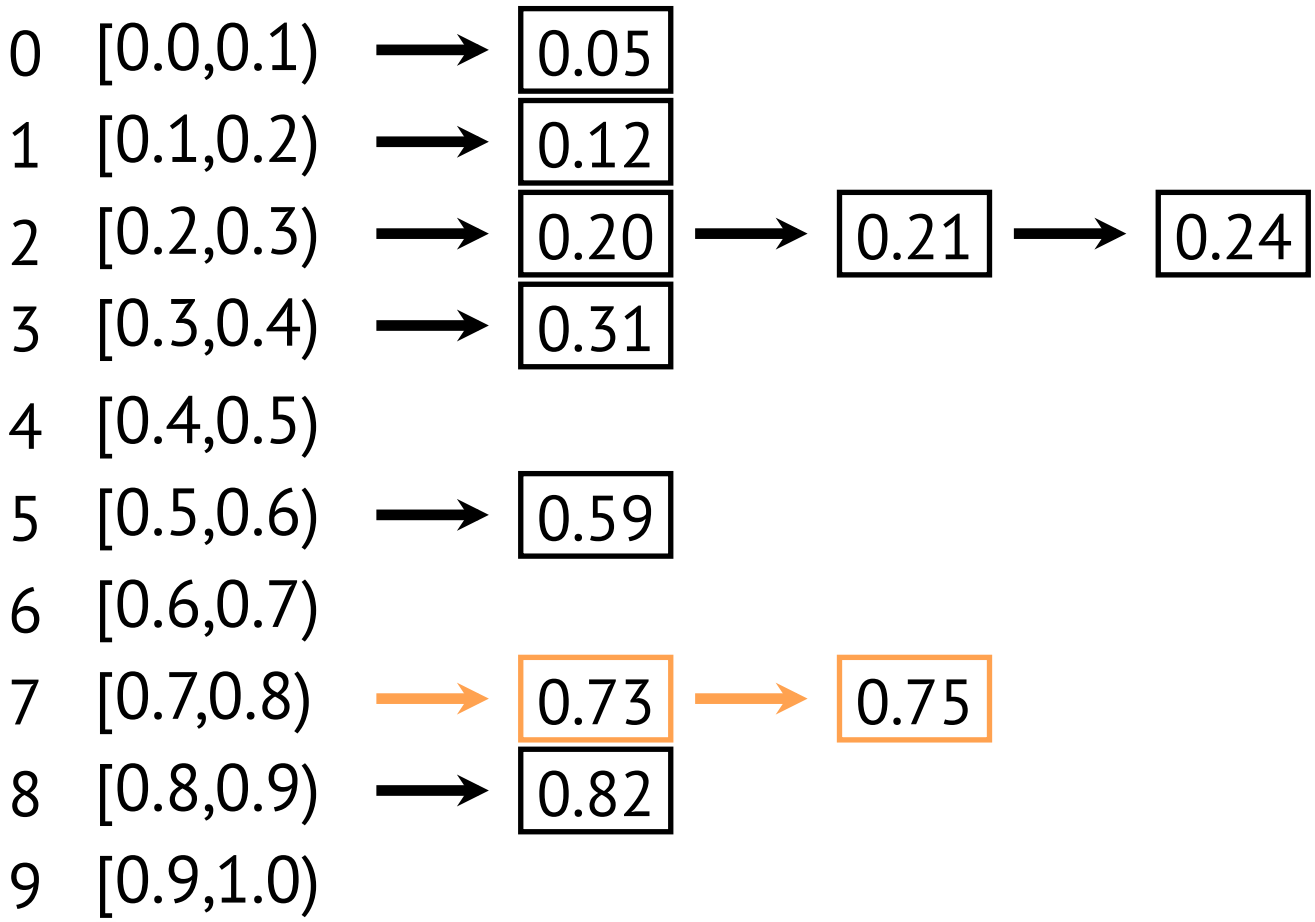
0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05



n=10

Bucket-Sort: Beispiel

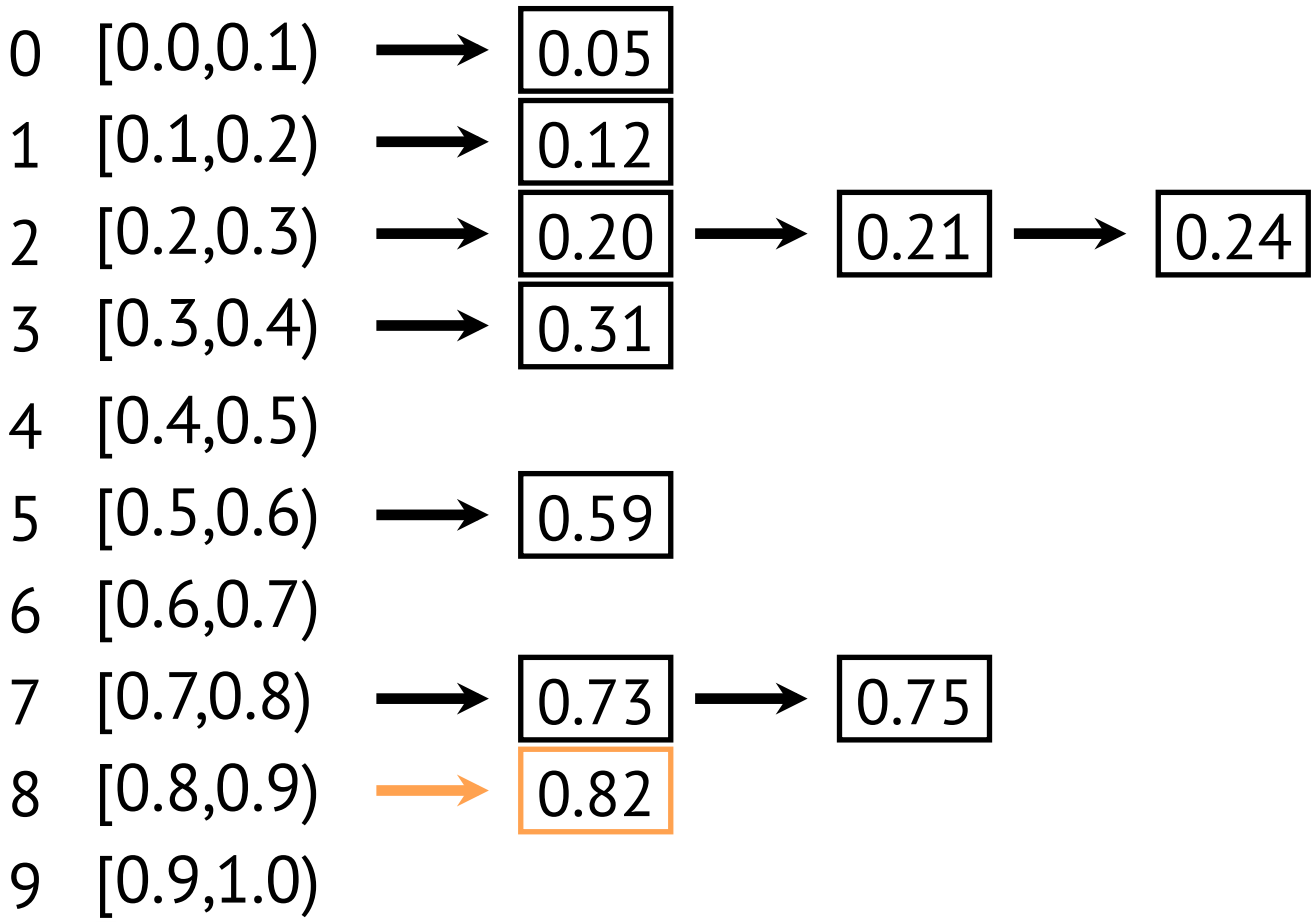
0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05



n=10

Bucket-Sort: Beispiel

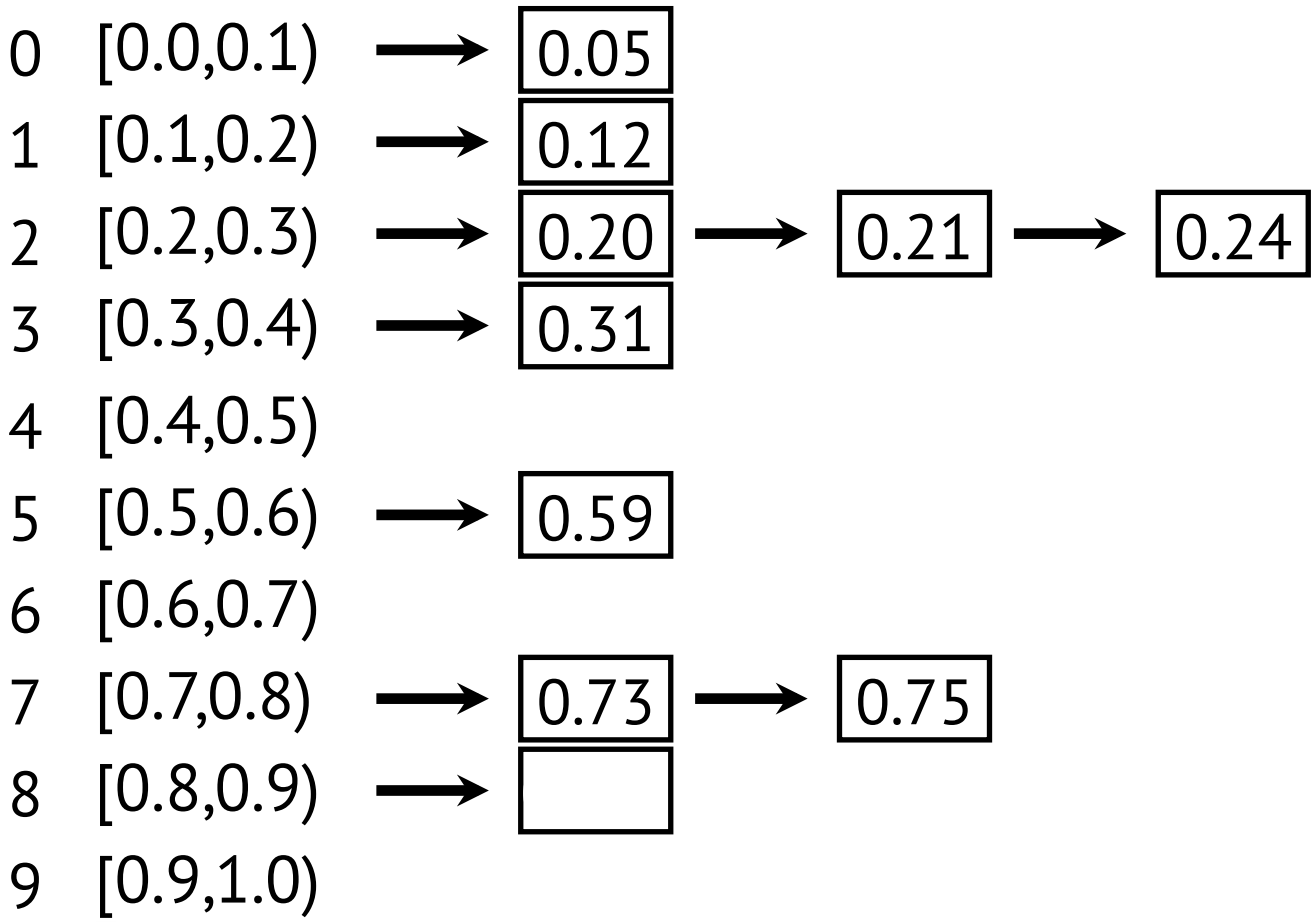
0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05



n=10

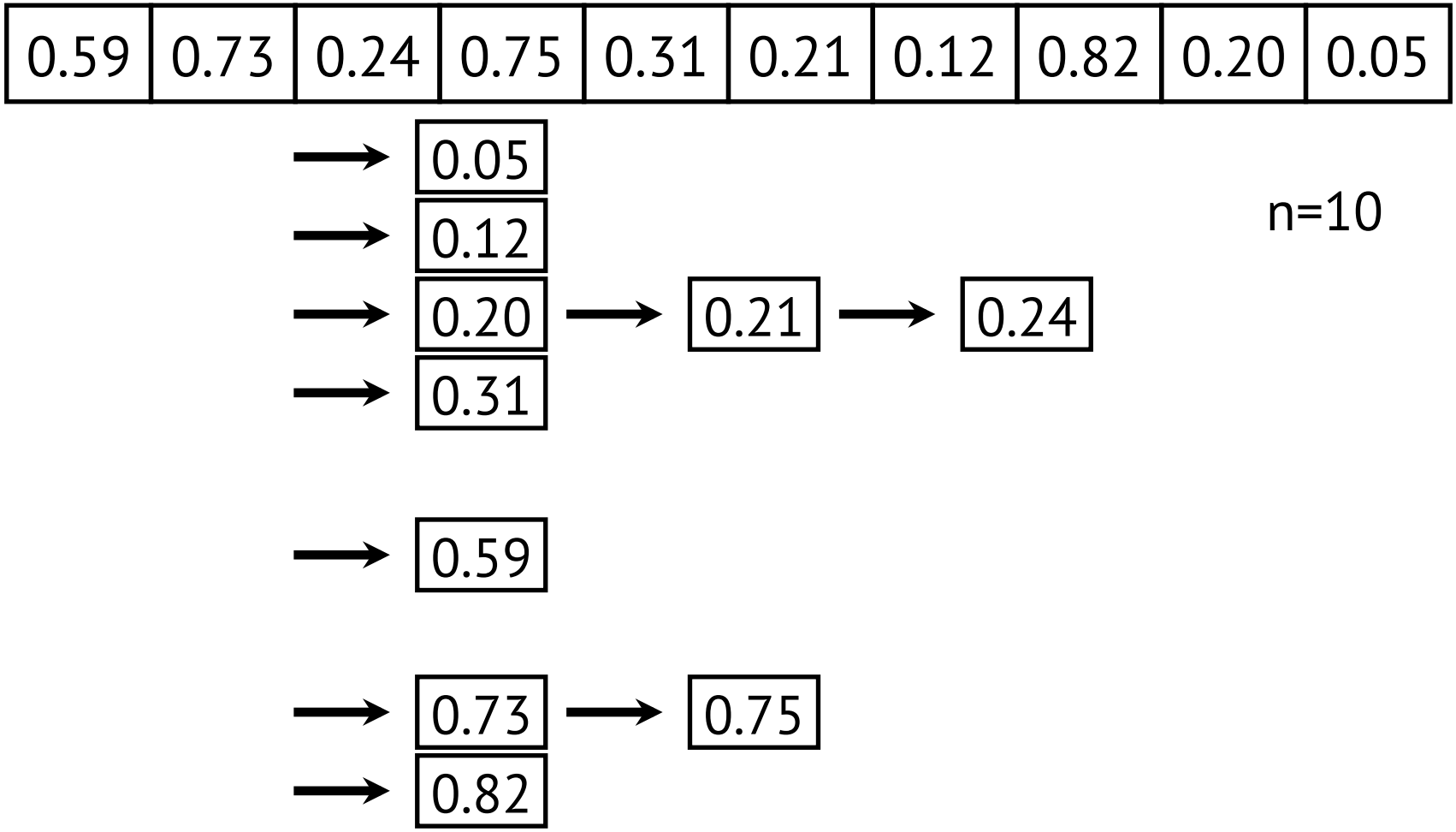
Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12		0.20	0.05



n=10

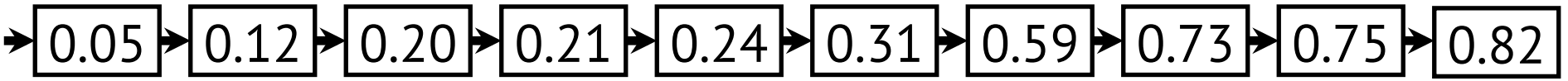
Bucket-Sort: Beispiel



Bucket-Sort: Beispiel

0.59	0.73	0.24	0.75	0.31	0.21	0.12	0.82	0.20	0.05

n=10



Bucket-Sort: Analyse

- Diskrete Zufallsvariable

$$X \in \{x_1, x_2, x_3, \ldots\}$$

- Erwartungswert

$$E(X) = \mu = \sum_{i=1}^{\infty} x_i P(X = x_i)$$

- Varianz

$$\begin{aligned} V(X) = \sigma^2 &= E((X - \mu)^2) \\ &= E(X^2) - E^2(X) \end{aligned}$$

Bucket-Sort: Analyse

- X=1 mit 50 % Wahrscheinlichkeit
  X=3 mit 50 % Wahrscheinlichkeit
- Erwartungswert:
  $E(X) = 1 \times 0.5 + 3 \times 0.5 = 2.0$
- X=1 mit 75 % Wahrscheinlichkeit
  X=3 mit 25 % Wahrscheinlichkeit
- Erwartungswert:
  $E(X) = 1 \times 0.75 + 3 \times 0.25 = 1.5$

Bucket-Sort: Analyse

- X=1 mit 50 % Wahrscheinlichkeit
  X=3 mit 50 % Wahrscheinlichkeit
- Varianz:
  $V(X) = (1-2)^2 \times 0.5 + (3-2)^2 \times 0.5 = 1.0$
- X=1 mit 75 % Wahrscheinlichkeit
  X=3 mit 25 % Wahrscheinlichkeit
- Varianz:
  $V(X) = (1-1.5)^2 \times 0.75 + (3-1.5)^2 \times 0.25 = 0.75$

Bucket-Sort: Analyse

- Sei n_i die Zahl der Elemente in Bucket i
- Wahrscheinlichkeit, dass ein Element in Bucket i fällt ist $p=1/n$
- Wahrscheinlichkeit, dass genau k Elemente in Bucket i fallen ist

$$P(n_i = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

"Binomialverteilung"

Bucket-Sort: Analyse

- Erwartungswert

$$E(n_i) = np = 1$$

- Varianz

$$V(n_i) = np(1-p) = 1 - \frac{1}{n}$$

- Sortieraufwand, z.B. Insertion-Sort

$$T(l) = O(l^2) \Rightarrow \exists c > 0 : T(l) \leq cl^2$$

Bucket-Sort: Analyse

- Erwarteter Sortieraufwand für Bucket i

$$
\begin{aligned}
E(T(n_i)) &= \sum_{k=0}^{n} T(k) P(n_i = k) \\
&= \sum_{k=0}^{n} ck^2 P(n_i = k) \\
&= c \sum_{k=0}^{n} k^2 P(n_i = k) \\
&= cE(n_i^2)
\end{aligned}
$$

Bucket-Sort: Analyse

- Erwarteter Sortieraufwand für Bucket i

$$
\begin{aligned}
E(T(n_i)) &= \cdots \\
&= cE(n_i^2) \\
&= c\left(V(n_i) + E^2(n_i)\right) \\
&= c\left(1 - \tfrac{1}{n} + 1\right) \\
&\leq 2c \\
&= O(1)
\end{aligned}
$$

Bucket-Sort: Analyse

- Erwarteter Sortieraufwand für alle Buckets:

$$E\left(\sum_{i=0}^{n-1} T(n_i)\right) = \sum_{i=0}^{n-1} \underbrace{E(T(n_i))}_{O(1)}$$

$$= O(n)$$

Bucket-Sort: Analyse

- BucketSort(A[1..n])

```
new B[0..n−1]
for i ← 1 to n do                                   O(n)
    push(B[⌊n×A[ i ]⌋], A[ i ])
for i ← 0 to n−1 do                                 O(n)
    sort(B[ i ])
c ← 1
for i ← 0 to n−1 do                                 O(n) erwartet
    for j ← 1 to size(B[ i ]) do                    O(n) (!)
        A[c] ← B[ i ][ j ]                          ---------------
        c ← c+1                                     O(n) erwartet
```

Spezielle Sortierverfahren

- Counting-Sort: $O(n+k)$
- Radix-Sort: $O(d \times (n+k))$
- Bucket-Sort: $O(n)$ erwartet