Forelesning 4

Vi får ofte bedre løsninger ved å styrke krav til input eller svekke krav til output. Sammenligningsbasert sortering er et klassisk eksempel: I verste tilfelle må vi bruke $\lg n!$ sammenligninger, men om vi *antar mer* om elementene eller *bare sorterer noen* av dem så kan vi gjøre det bedre.

Pensum

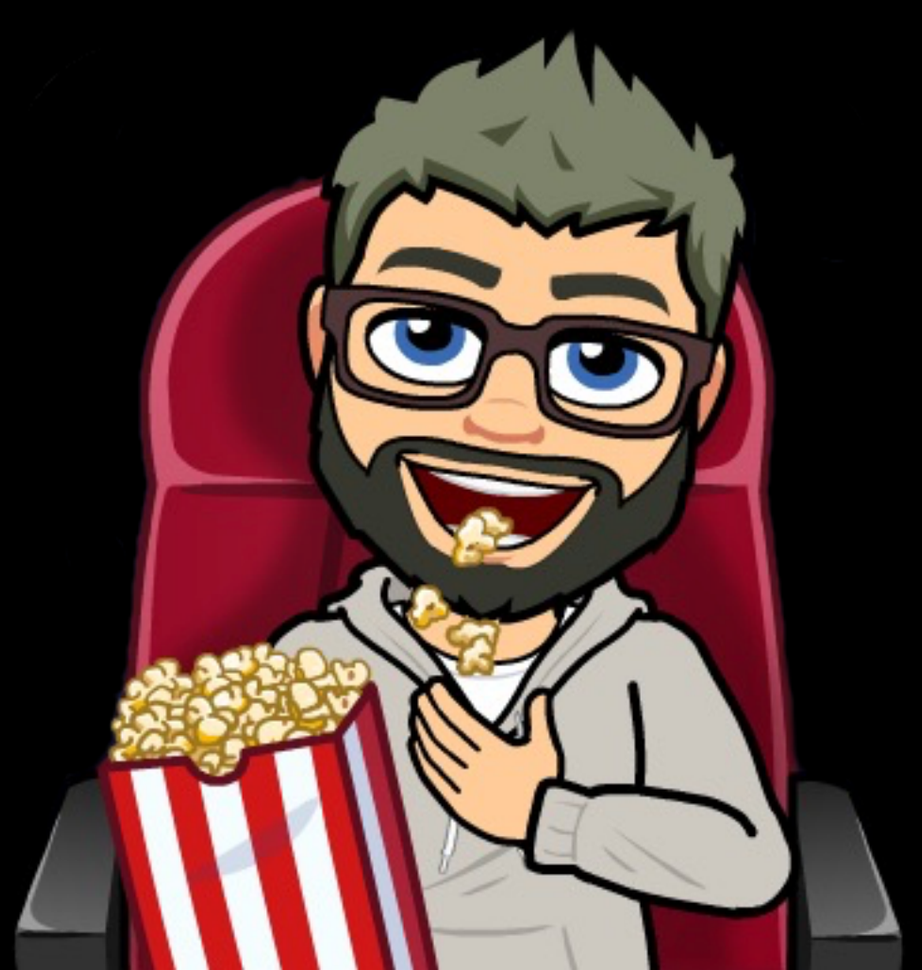
- ☐ Kap. 8. Sorting in linear time
- ☐ Kap. 9. Medians and order statistics

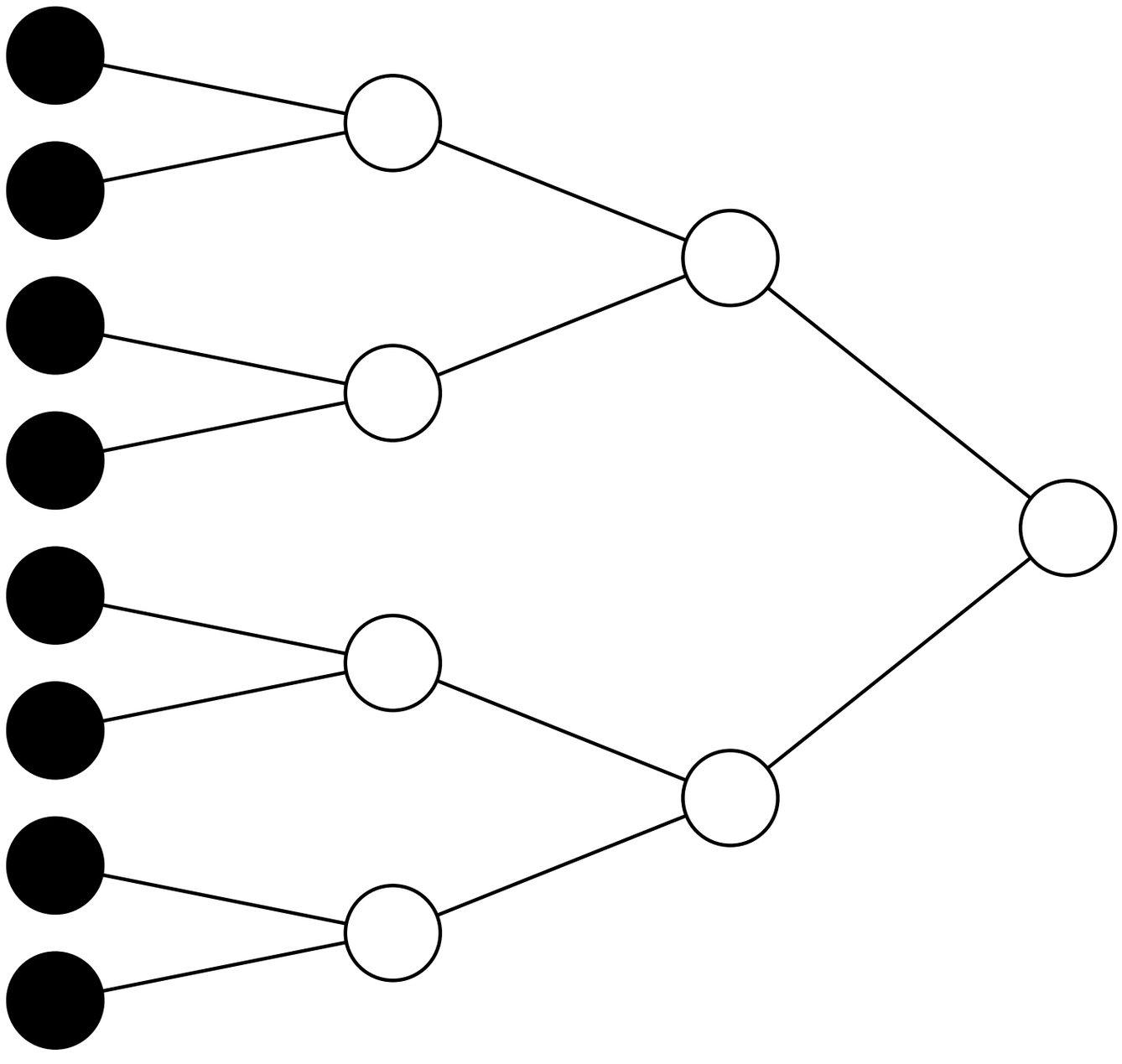
Læringsmål

- [D$_1$] Forstå hvorfor *sammenlignings-basert sortering* har en *worst-case* på $\Omega(n \lg n)$
- [D$_2$] Vite hva en *stabil sorteringsalgoritme* er
- [D$_3$] Forstå Counting-Sort, og hvorfor den er stabil
- [D$_4$] Forstå Radix-Sort, og hvorfor den trenger en stabil subrutine
- [D$_5$] Forstå Bucket-Sort
- [D$_6$] Forstå Randomized-Select
- [D$_7$] Kjenne til Select; grundig forståelse kreves ikke

mlh@ntnu.no

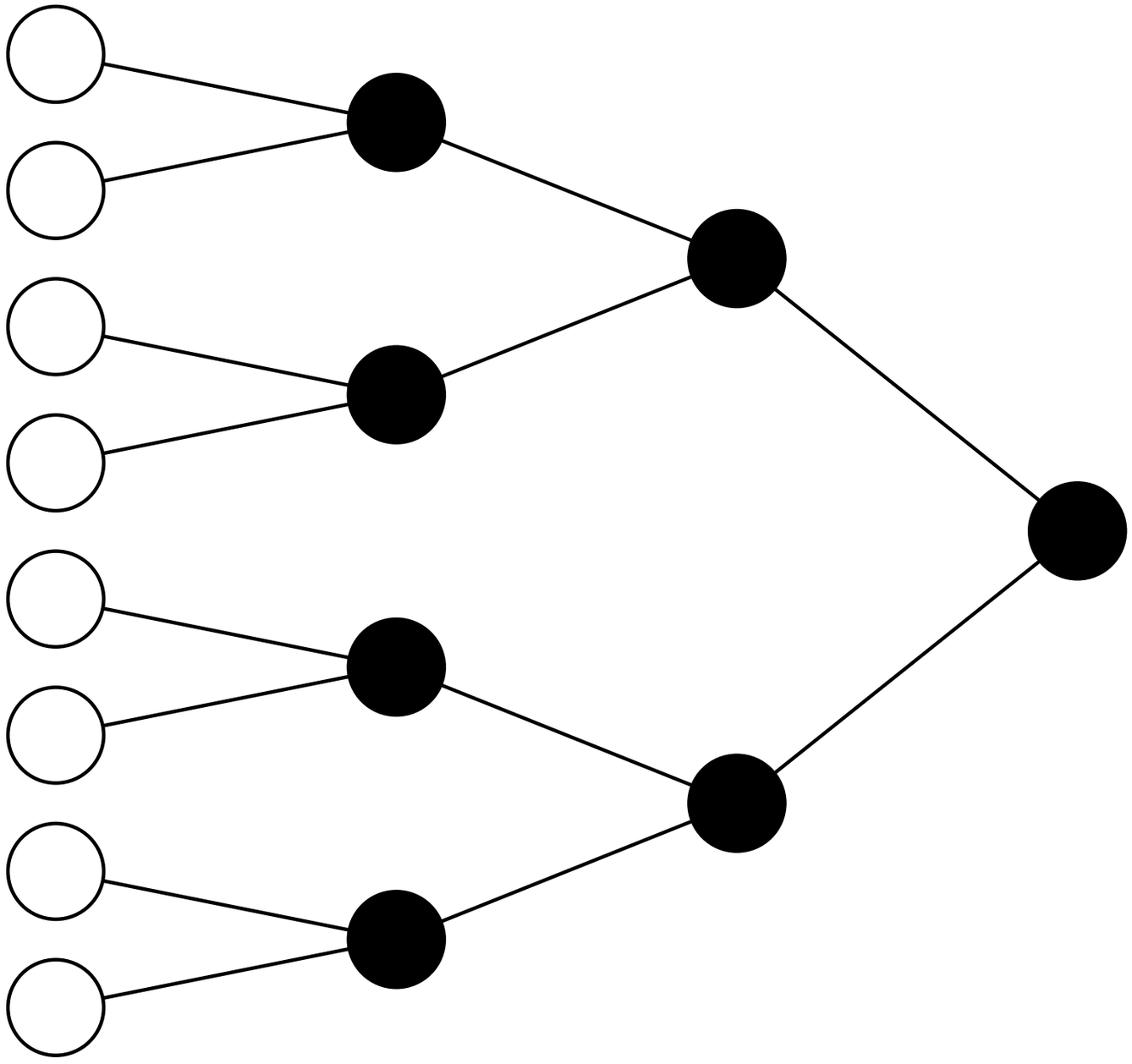
Forelesningen filmes





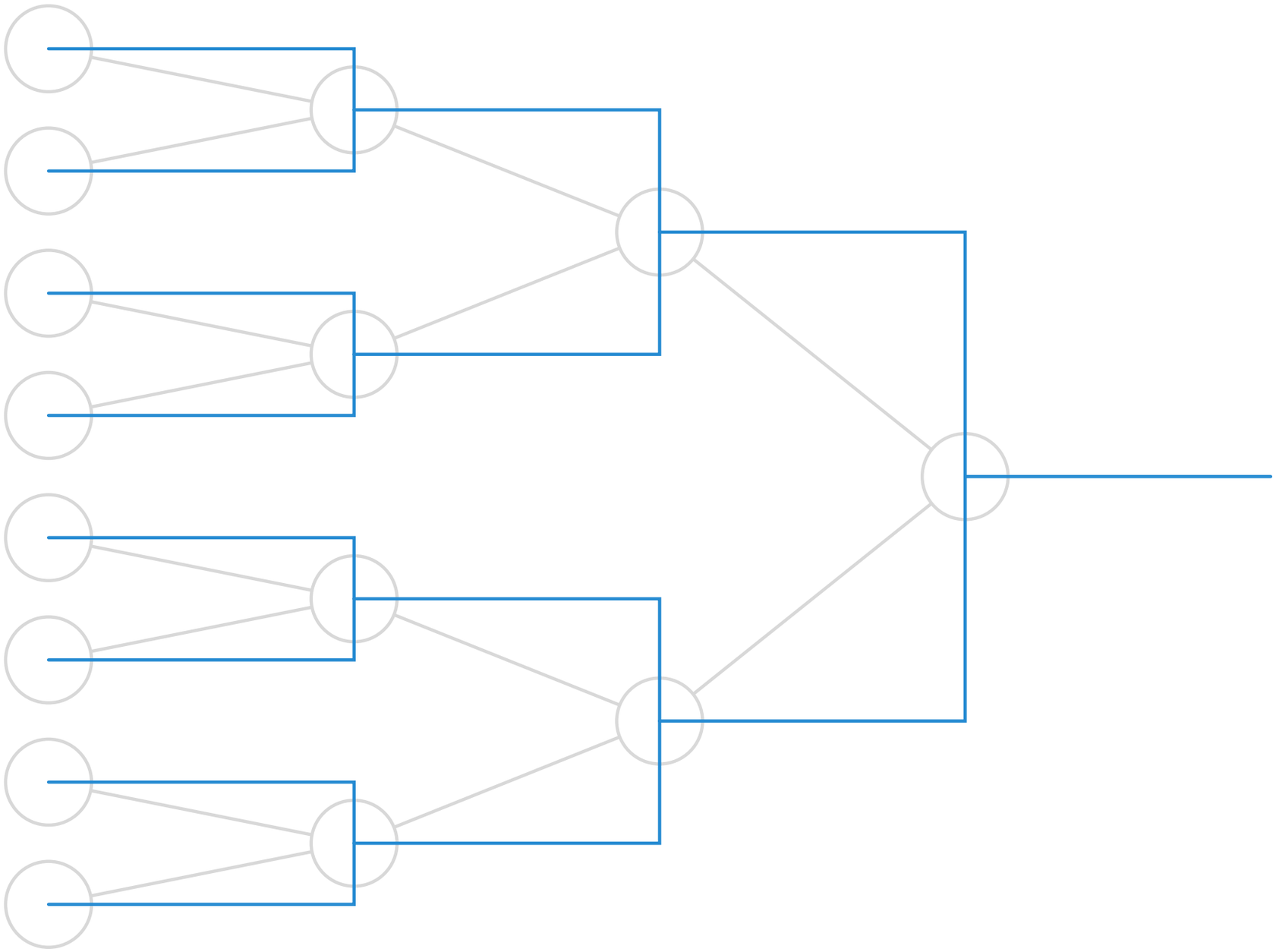
Binært tre med n løvnoder

mlh@ntnu.no



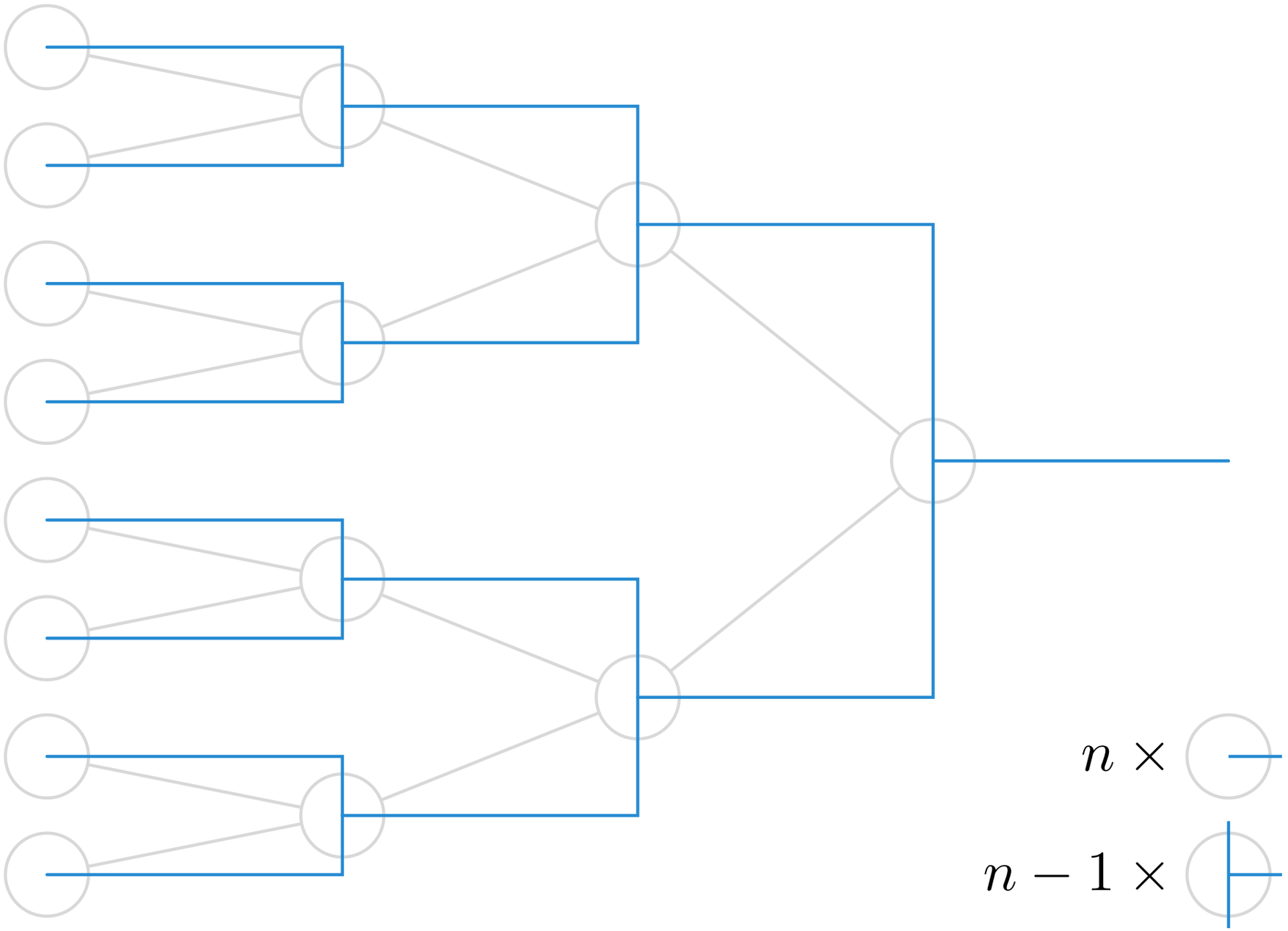
Hvorfor $n - 1$ interne noder?

mlh@ntnu.no



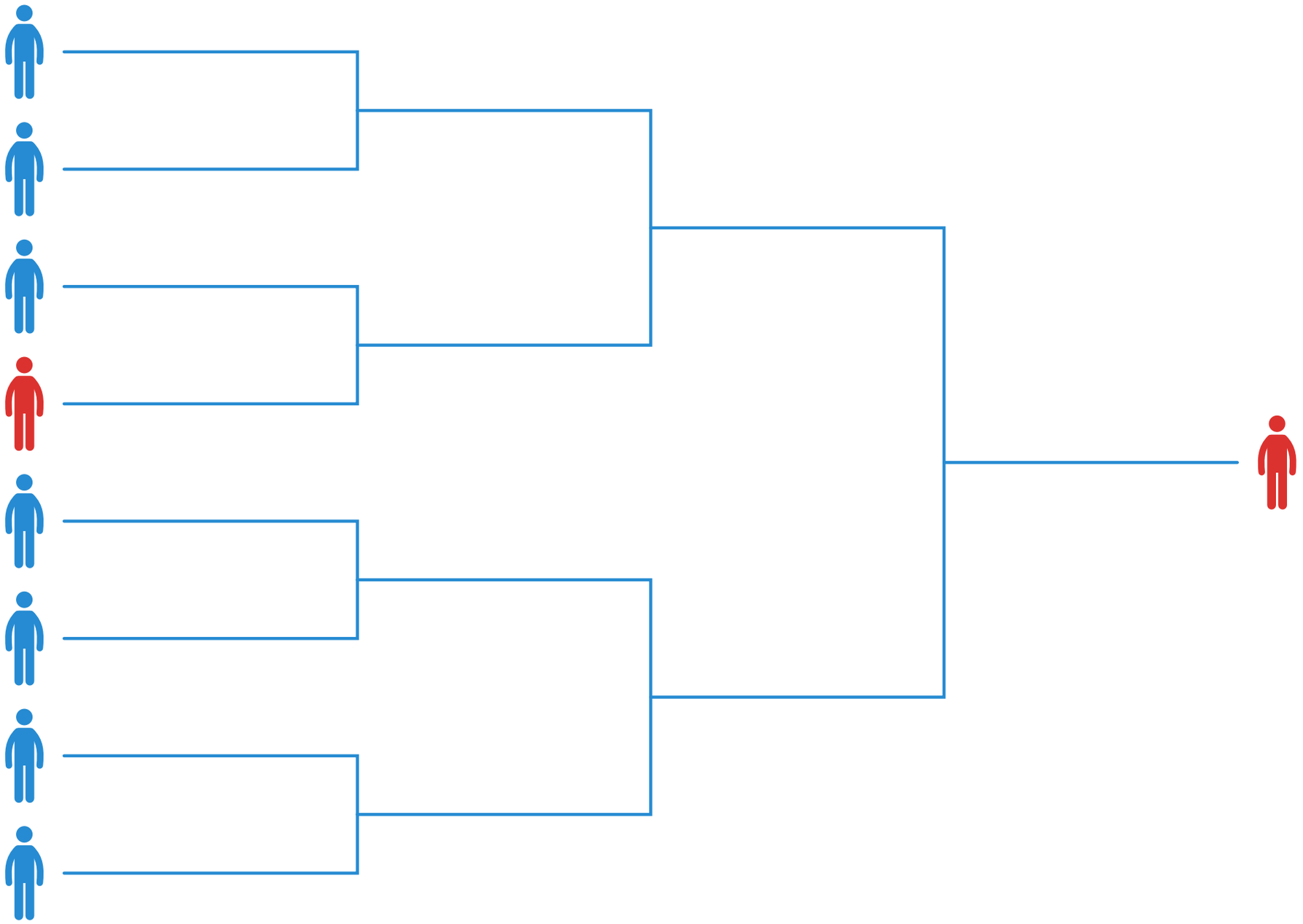
Tenk på det som en knockout-turnering

mlh@ntnu.no



$n - 1$ matcher slår ut alle unntatt vinneren

mlh@ntnu.no



$n - 1$ matcher slår ut alle unntatt vinneren

mlh@ntnu.no

Forelesning 4

Rangering i lineær tid

1. **Sorteringsgrensen**
2. **Tellesortering**
3. **Radikssortering**
4. **Bøttesortering**
5. **Randomized Select**
6. **Select**

0.6

Variabelskifte

Å bytte ut variable er en teknikk som også brukes i f.eks. kalkulus.

Se f.eks. https://en.wikipedia.org/wiki/Change_of_variables

$$\mathrm{T}(\sqrt{n}) = \lg n$$

$$\mathrm{T}(\sqrt{n}) = \lg n$$

$$\mathrm{T}(n^{\frac{1}{2}}) = \lg n$$

$$m \overset{\text{def}}{=} \lg n$$

For å bli kvitt logaritmen: Gi $\lg n$ et nytt navn

$$m \overset{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^{\frac{m}{2}}) = m$$

Potensen er fortsatt problematisk

$$m \stackrel{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^{\frac{m}{2}}) = m$$

$$\mathrm{S}(m) \stackrel{\text{def}}{=} \mathrm{T}(n) = \mathrm{T}(2^m)$$

Gi $\mathrm{T}(2^m)$ et nytt navn!

$$m \stackrel{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^{\frac{m}{2}}) = m$$

$$\mathrm{S}(m) \stackrel{\text{def}}{=} \mathrm{T}(n) = \mathrm{T}(2^m)$$

$$\mathrm{T}(2^{\frac{m}{2}}) = \mathrm{S}(m/2)$$

Hvis $\mathrm{S}(m) = \mathrm{T}(2^m)$ så må $\mathrm{S}(m/2)$ være $\mathrm{T}(2^{\frac{m}{2}})$

$$m \stackrel{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^{\frac{m}{2}}) = m$$

$$\mathrm{S}(m) \stackrel{\text{def}}{=} \mathrm{T}(n) = \mathrm{T}(2^m)$$

$$\mathrm{T}(2^{\frac{m}{2}}) = \mathrm{S}(m/2)$$

$$\mathrm{S}(m/2) = m$$

Slik ser ligningen vår ut med nye navn

$$m \overset{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^{\frac{m}{2}}) = m$$

$$\mathrm{S}(m) \overset{\text{def}}{=} \mathrm{T}(n) = \mathrm{T}(2^m)$$

$$\mathrm{T}(2^{\frac{m}{2}}) = \mathrm{S}(m/2)$$

$$\mathrm{S}(m/2) = m$$

$$\mathrm{S}(m) = 2m$$

(Evt. enda et skifte: $x = m/2 \implies \mathrm{S}(x) = 2x \implies \mathrm{S}(m) = 2m$)

$$m \stackrel{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^{\frac{m}{2}}) = m$$

$$\mathrm{S}(m) \stackrel{\text{def}}{=} \mathrm{T}(n) = \mathrm{T}(2^m)$$

$$\mathrm{T}(2^{\frac{m}{2}}) = \mathrm{S}(m/2)$$

$$\mathrm{S}(m/2) = m$$

$$\mathrm{S}(m) = 2m$$

$$\mathrm{T}(n) = 2 \lg n$$

Her bruker vi bare definisjonene våre: $\mathrm{S}(m) = \mathrm{T}(n)$ og $m = \lg n$

$$\mathrm{T}(n) = 2\mathrm{T}(\sqrt{n}) + \lg n$$

$$\mathrm{T}(n) = 2\mathrm{T}(\sqrt{n}) + \lg n$$

$$\mathrm{T}(n) = 2\mathrm{T}(n^{\frac{1}{2}}) + \lg n$$

$$m \overset{\text{def}}{=} \lg n$$

For å bli kvitt logaritmen: Gi $\lg n$ et nytt navn

$$m \overset{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^m) = \mathrm{T}(2^{\frac{m}{2}}) + m$$

Potensen er fortsatt problematisk

$$m \overset{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^m) = \mathrm{T}(2^{\frac{m}{2}}) + m$$

$$\mathrm{S}(m) \overset{\text{def}}{=} \mathrm{T}(n) = \mathrm{T}(2^m)$$

Gi $\mathrm{T}(2^m)$ et nytt navn!

$$m \overset{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^m) = \mathrm{T}(2^{\frac{m}{2}}) + m$$

$$\mathrm{S}(m) \overset{\text{def}}{=} \mathrm{T}(n) = \mathrm{T}(2^m)$$

$$\mathrm{T}(2^{\frac{m}{2}}) = \mathrm{S}(m/2)$$

Hvis $\mathrm{S}(m) = \mathrm{T}(2^m)$ så må $\mathrm{S}(m/2)$ være $\mathrm{T}(2^{\frac{m}{2}})$

$$m \stackrel{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^m) = \mathrm{T}(2^{\frac{m}{2}}) + m$$

$$\mathrm{S}(m) \stackrel{\text{def}}{=} \mathrm{T}(n) = \mathrm{T}(2^m)$$

$$\mathrm{T}(2^{\frac{m}{2}}) = \mathrm{S}(m/2)$$

$$\mathrm{S}(m) = 2\mathrm{S}(m/2) + m$$

Slik ser ligningen vår ut med nye navn

$$m \stackrel{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^m) = \mathrm{T}(2^{\frac{m}{2}}) + m$$

$$\mathrm{S}(m) \stackrel{\text{def}}{=} \mathrm{T}(n) = \mathrm{T}(2^m)$$

$$\mathrm{T}(2^{\frac{m}{2}}) = \mathrm{S}(m/2)$$

$$\mathrm{S}(m) = 2\mathrm{S}(m/2) + m$$

$$\mathrm{S}(m) = m \lg m + m$$

Standard rekurrensløsning, som tidligere i dag

$$m \overset{\text{def}}{=} \lg n$$

$$\mathrm{T}(2^m) = \mathrm{T}(2^{\frac{m}{2}}) + m$$

$$\mathrm{S}(m) \overset{\text{def}}{=} \mathrm{T}(n) = \mathrm{T}(2^m)$$

$$\mathrm{T}(2^{\frac{m}{2}}) = \mathrm{S}(m/2)$$

$$\mathrm{S}(m) = 2\mathrm{S}(m/2) + m$$

$$\mathrm{S}(m) = m \lg m + m$$

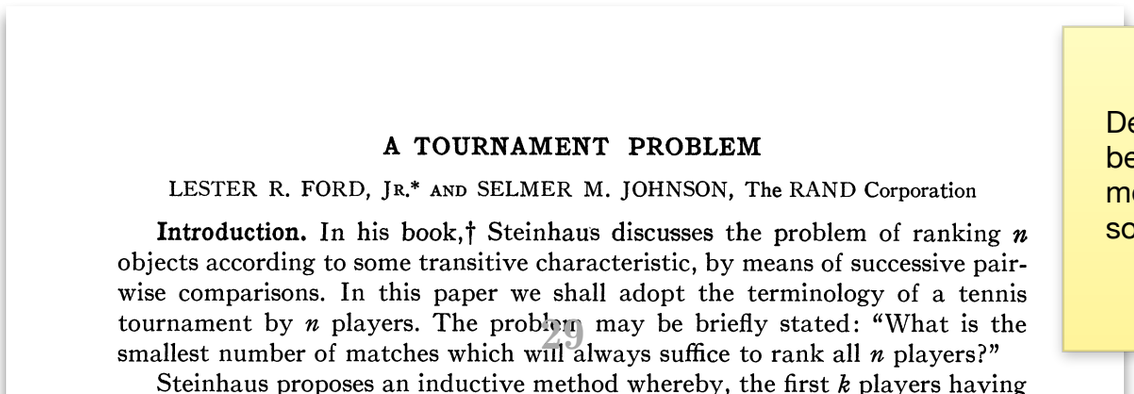
$$\mathrm{T}(n) = \lg n \lg \lg n + \lg n$$

Her bruker vi bare definisjonene våre: $\mathrm{S}(m) = \mathrm{T}(n)$ og $m = \lg n$

Hvor fort kan vi sortere n elementer om den eneste informasjonen vi får kommer fra parvise sammenligninger (såkalt sammenligningsbasert sortering)?

1:6

Sorteringsgrensen



A TOURNAMENT PROBLEM

LESTER R. FORD, JR.* AND SELMER M. JOHNSON, The RAND Corporation

Introduction. In his book,† Steinhaus discusses the problem of ranking n objects according to some transitive characteristic, by means of successive pairwise comparisons. In this paper we shall adopt the terminology of a tennis tournament by n players. The problem may be briefly stated: "What is the smallest number of matches which will always suffice to rank all n players?"

Steinhaus proposes an inductive method whereby, the first k players having

Denne artikkelen, fra 1959, beskriver beslutningstre-modellen for analyse av sortering og rangering.

«Tenk på en permutasjon»

… av n! mulige

Trenger maks lg n! ja-nei-spørsmål

Worst-case: Tenk deg at en kjiping kjenner algoritmen din og velger input for deg.

«Tenk på en permutasjon»

… av n! mulige

Trenger lg n! ja-nei-spørsmål

$$\geqslant n!$$

Vi må skille mellom $n!$ sekvenser med ja/nei-spørsmål

$$2^{\mathrm{T}(n)} \geqslant n!$$

Hvert spørsmål dobler hva vi kan skille mellom

$$\mathrm{T}(n) \geqslant \lg n!$$

$\lg n!$ er antall halveringer fra $n!$ til 1

$$\mathrm{T}(n) \geqslant \lg n!$$

Om vi *ikke har flaks* kan vi ikke gjøre det bedre

$$\mathrm{T}(n) \geqslant \lg n!$$

Dette er en *nedre grense* for *verste tilfelle*

$$\mathrm{T_W}(n) \geqslant \lg n!$$

Dette er en *nedre grense* for *verste tilfelle*

$$\mathrm{T_W}(n) = \Omega(\lg n!)$$

Dette er en *nedre grense* for *verste tilfelle*

$$\mathrm{T_W}(n) = \Omega(\lg n!)$$

Fra Stirlings approksimasjon: $n! \geqslant (n/e)^n$

$$\mathrm{T_W}(n) = \Omega(\lg n!)$$

Fra Stirlings approksimasjon: $n! \geqslant (n/e)^n \implies \lg n! \geqslant$

$$\mathrm{T_W}(n) = \Omega(\lg n!)$$

Fra Stirlings approksimasjon: $n! \geqslant (n/e)^n \implies \lg n! \geqslant n \lg n - n \lg e$

$$\mathrm{T_W}(n) = \Omega(\lg n!)$$

$$= \Omega(n \lg n)$$

Fra Stirlings approksimasjon: $n! \geqslant (n/e)^n \implies \lg n! \geqslant n \lg n - n \lg e$

$$\mathrm{T}_{\mathrm{A}}(n) = \Omega(\lg n!)$$

$$= \Omega(n \lg n)$$

$\lg n!$ er også informasjonsinnholdet i en tilfeldig permutasjon

$$\mathrm{T_W}(n) =$$

$$\mathrm{T_W}(n) =$$

$$\mathrm{T_W}(n) = \Omega(n \lg n)$$

Dette har vi nettopp etablert

$$\mathrm{T_W}(n) = \mathbf{O}(\infty)$$

$$\mathrm{T_W}(n) =$$

$$\mathrm{T_W}(n) = \mathbf{\Omega}(n \lg n)$$

Algoritmer kan bli vilkårlig dårlige

$$\mathrm{T_W}(n) = \mathbf{O}(\infty)$$

$$\mathrm{T_W}(n) = \mathbf{\Theta}(?)$$

$$\mathrm{T_W}(n) = \mathbf{\Omega}(n \lg n)$$

Vi har ingen generell felles øvre og nedre grense

$$\mathrm{T}_{\mathrm{A}}(n) = \mathbf{O}(\infty)$$

$$\mathrm{T}_{\mathrm{A}}(n) = \mathbf{\Theta}(?)$$

$$\mathrm{T}_{\mathrm{A}}(n) = \mathbf{\Omega}(n \lg n)$$

Det samme gjelder forventet kjøretid

$$\mathrm{T_B}(n) = \mathbf{O}(\infty)$$

$$\mathrm{T_B}(n) = \mathbf{\Theta}(?)$$

$$\mathrm{T_B}(n) = \mathbf{\Omega}(n)$$

For å garantere rett svar må hele sekvensen behandles

Bryt grensen

… vha. ekstra antagelser

Vi antar at elementene består av ett siffer!
(Eller at de er verdier i et begrenset verdiområde 0…k)

2.6

Tellesortering

Fra 1954

COUNTING-SORT(A, B, k)

A input
B output
k maks

A, B, k: input, output, maksverdi ($A[i] \in \{0, \ldots, k\}$)

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
```

A input
B output
k maks
C antall

Antall forekomster av hver mulig verdi

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
```

A input
B output
k maks
C antall

For hver mulig verdi i ...

mlh@ntnu.no

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3          C[i] = 0
```

A input
B output
k maks
C antall

Ingen forekomster funnet av i ennå

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3        C[i] = 0
 4  for j = 1 to A.length
```

A input
B output
k maks
C antall
j pos. i A

For hvert posisjon i input-tabellen ...

mlh@ntnu.no

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3        C[i] = 0
 4  for j = 1 to A.length
 5        C[A[j]] = C[A[j]] + 1
```

A input
B output
k maks
C antall
j pos. i A

Vi har nå sett enda en forekomst av A[j]

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3        C[i] = 0
 4  for j = 1 to A.length
 5        C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
```

A input
B output
k maks
C antall
j pos. i A

For hver av tellingene unntatt 0 ...

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3        C[i] = 0
 4  for j = 1 to A.length
 5        C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7        C[i] = C[i] + C[i − 1]
```

A input
B output
k maks
C antall
j pos. i A
i pos. i C

Gjør C *kumulativ*: C[i] = Hvor skal den siste i-en være?

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3        C[i] = 0
 4  for j = 1 to A.length
 5        C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7        C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
```

A input
B output
k maks
C antall
j pos. i A
i pos. i C

Vil ikke bytte like verdier; C angir *siste* forekomster!

mlh@ntnu.no

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3        C[i] = 0
 4  for j = 1 to A.length
 5        C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7        C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9        B[C[A[j]]] = A[j]
```

A input
B output
k maks
C antall
j pos. i A
i pos. i C

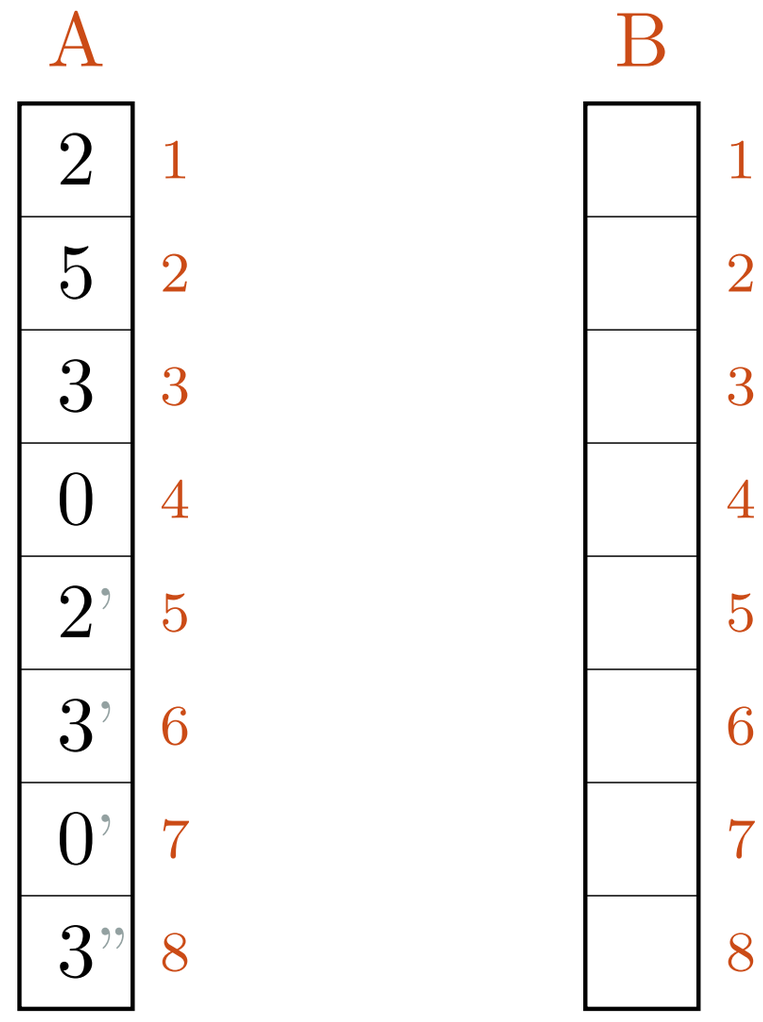
C[i] = Hvor skal den siste i-en være? (i = A[j])

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3        C[i] = 0
 4  for j = 1 to A.length
 5        C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7        C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9        B[C[A[j]]] = A[j]
10        C[A[j]] = C[A[j]] − 1
```

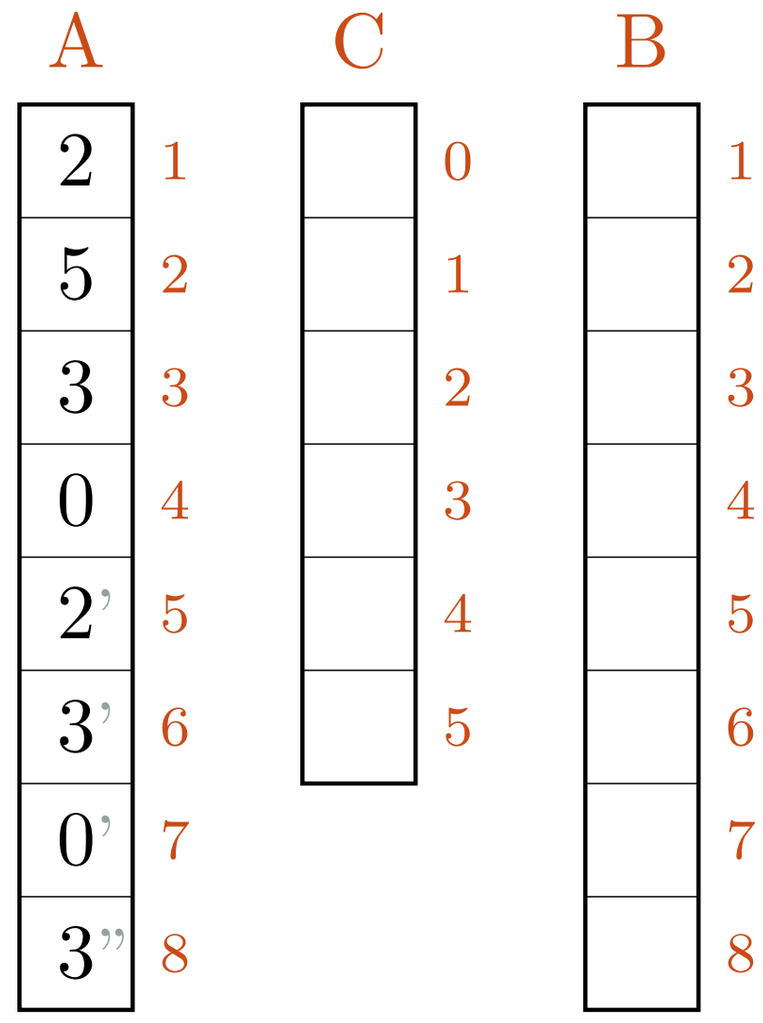
A input
B output
k maks
C antall
j pos. i A
i pos. i C

Den neste i-en skal være ett hakk til venstre

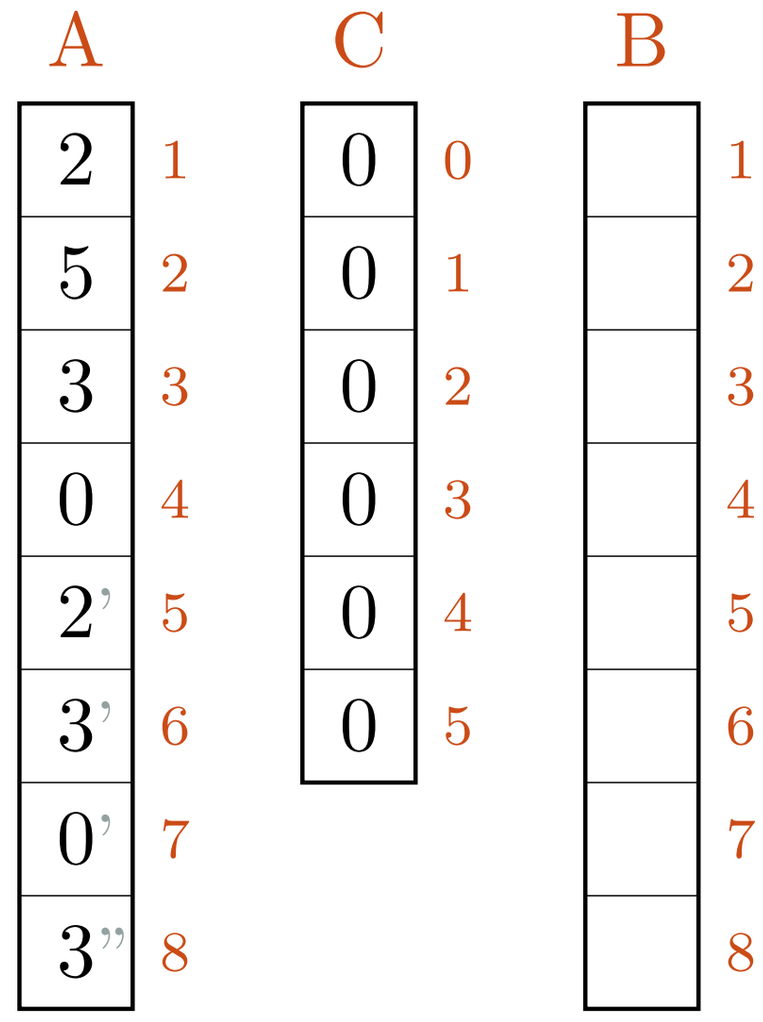
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



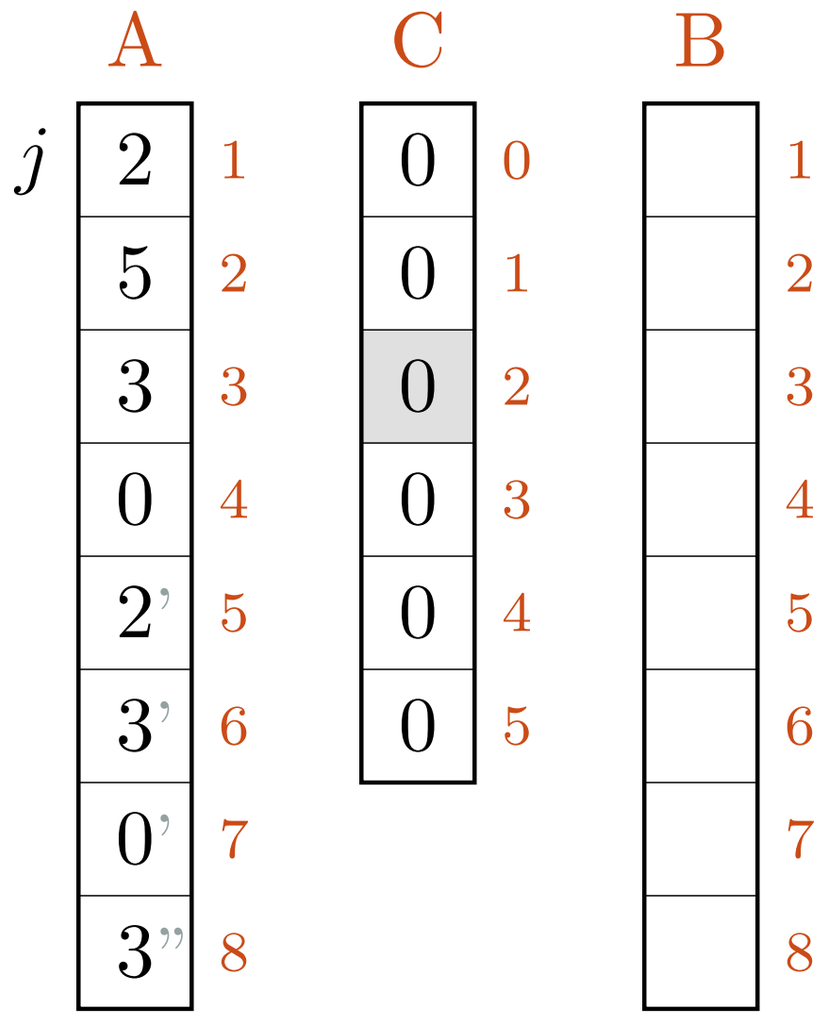
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



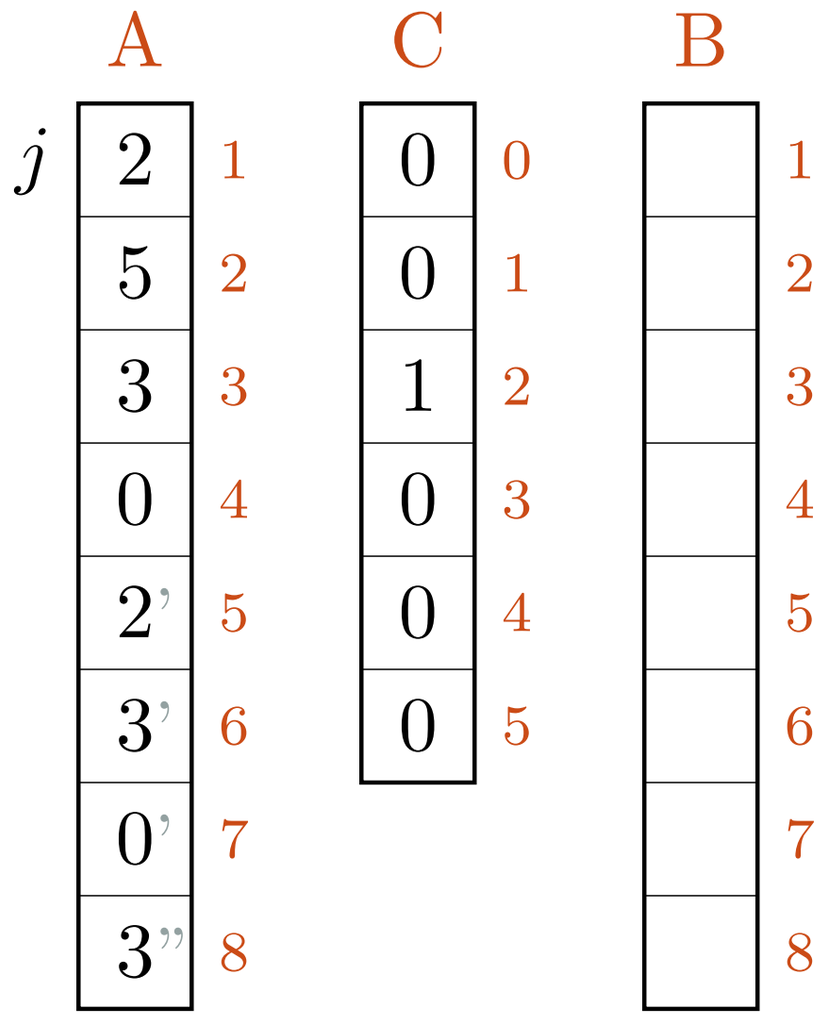
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



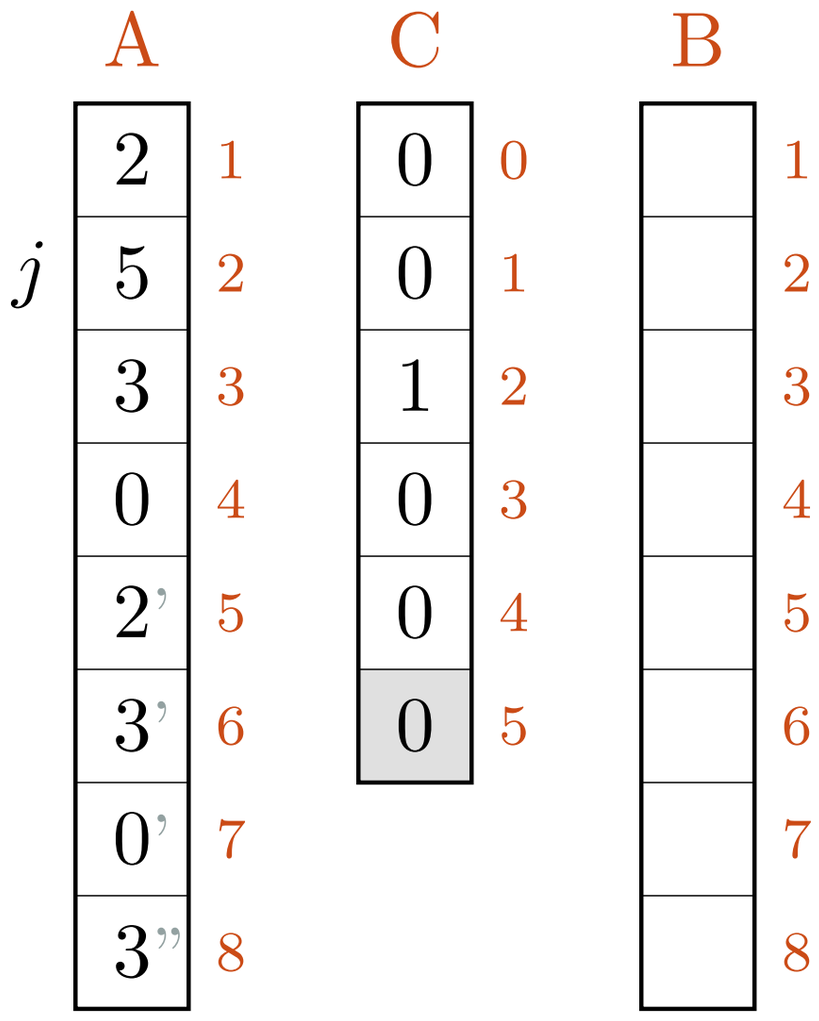
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



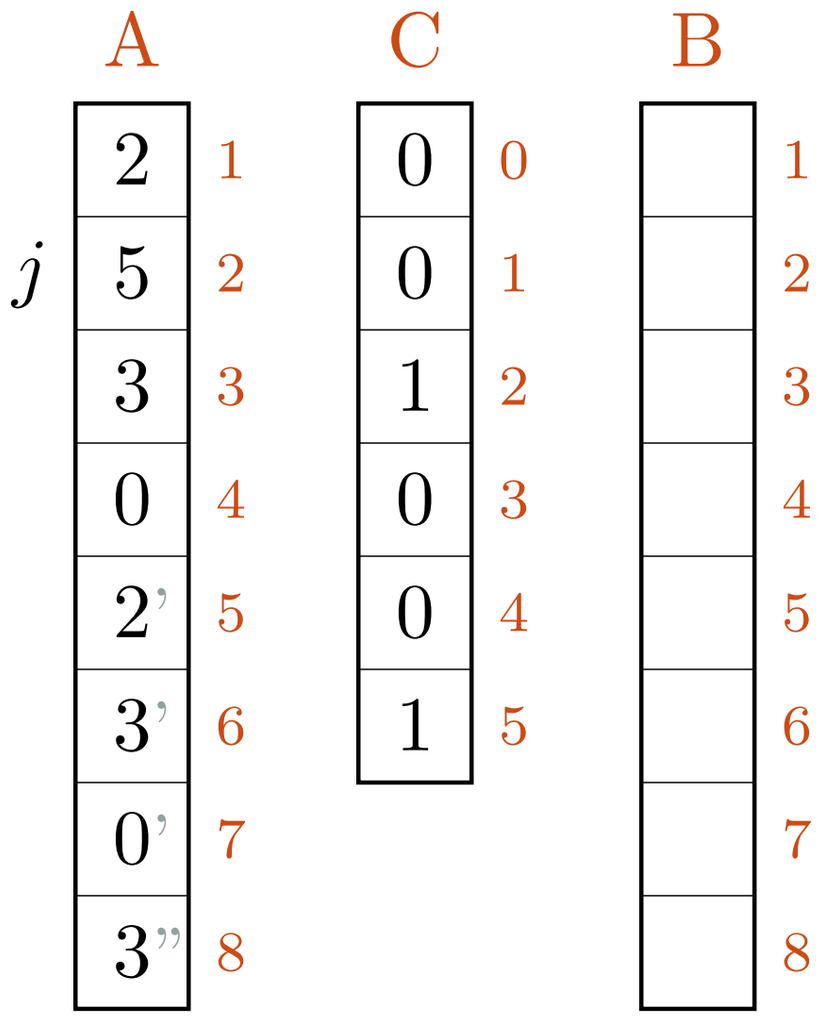
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



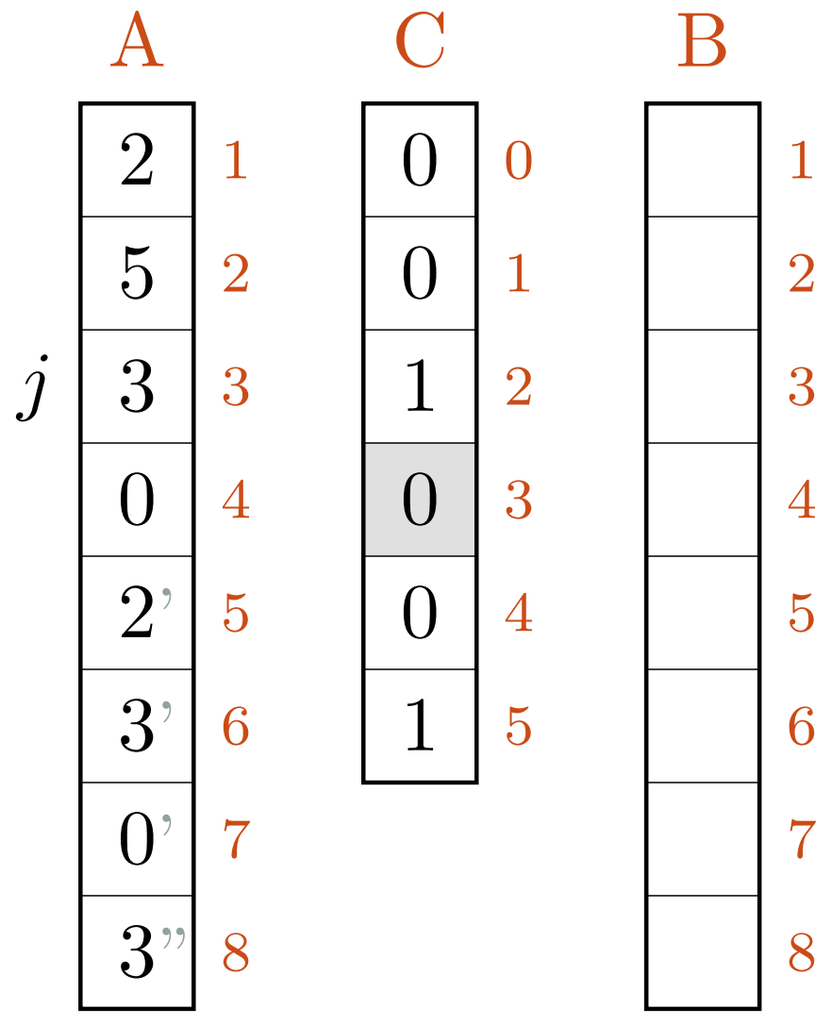
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



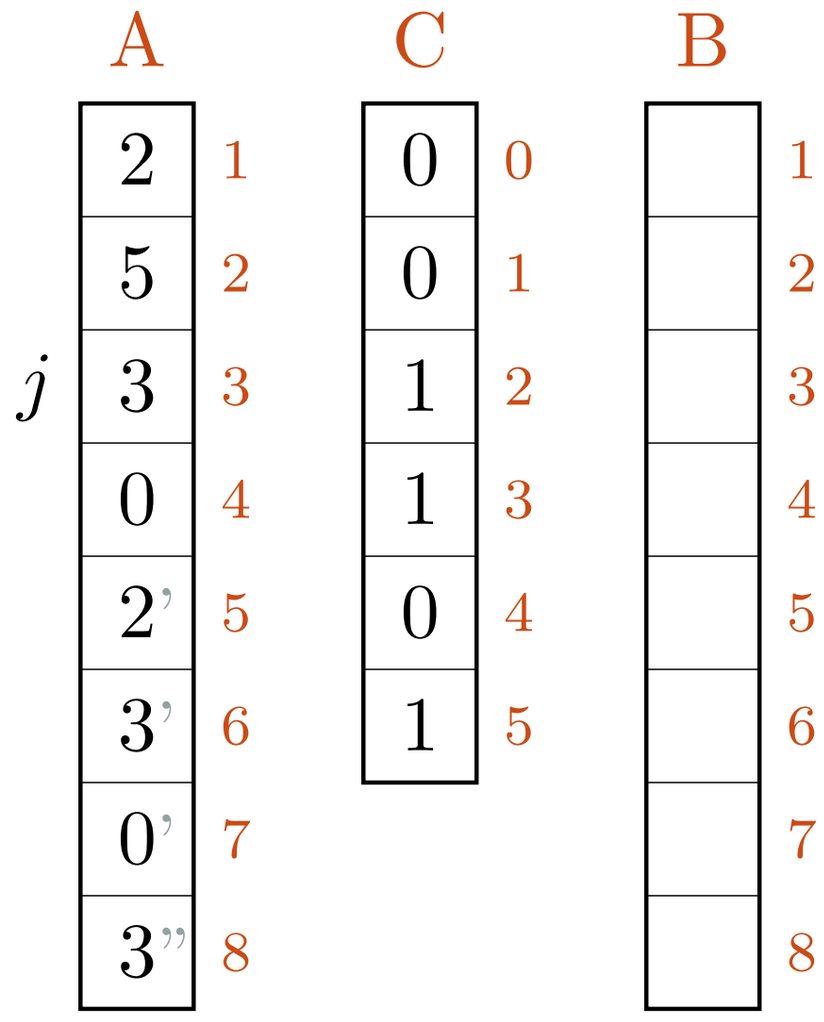
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



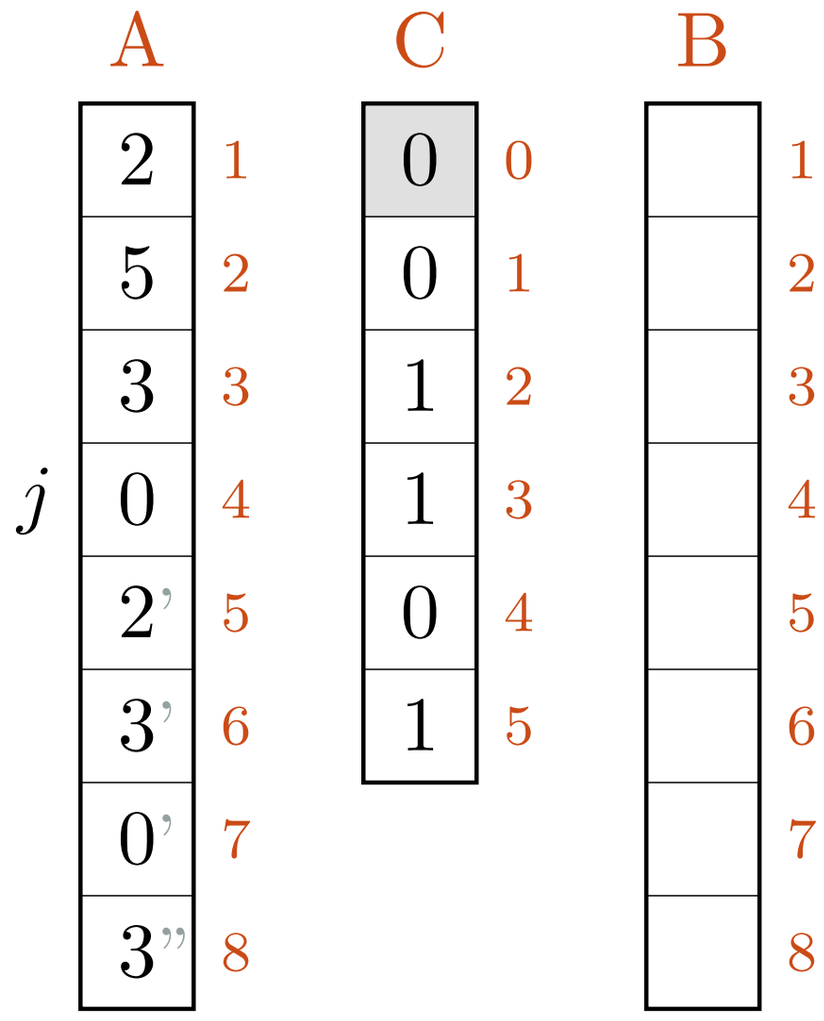
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



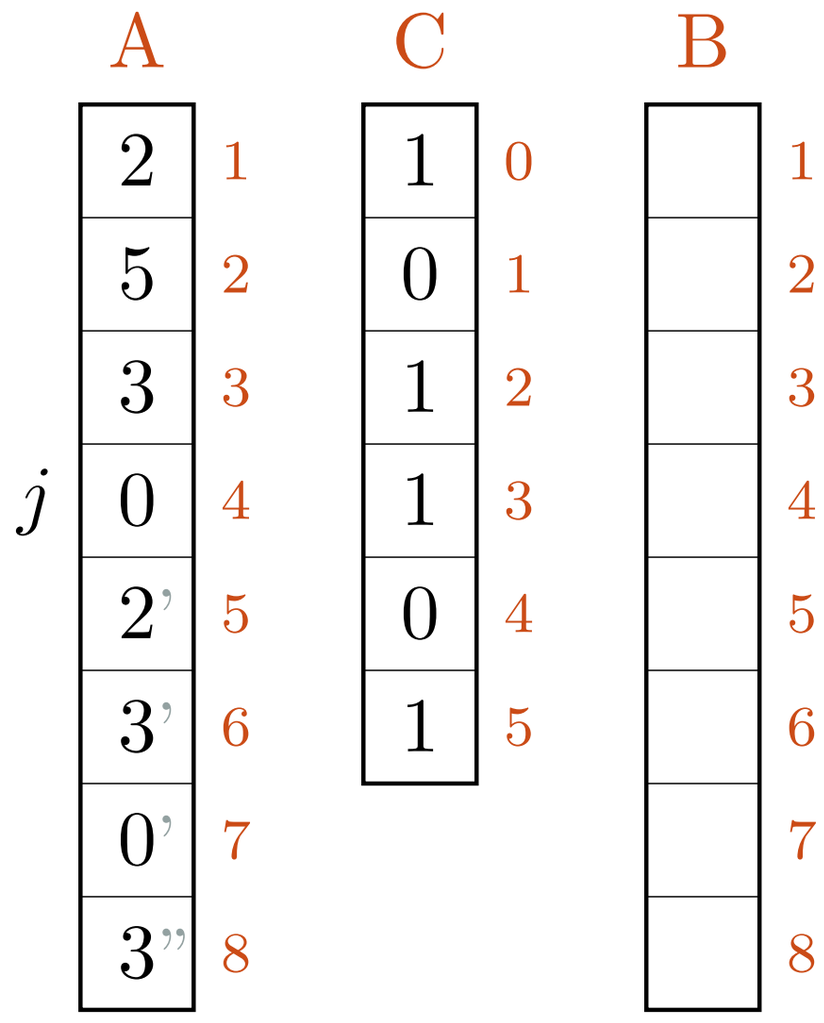
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



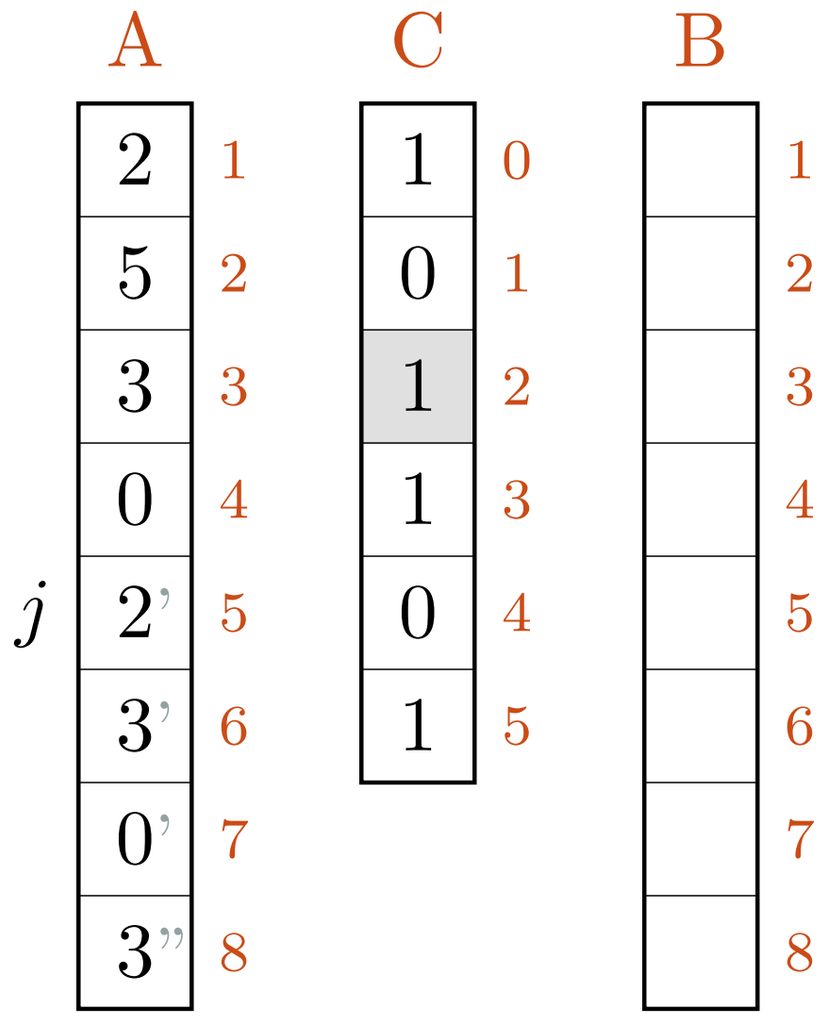
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



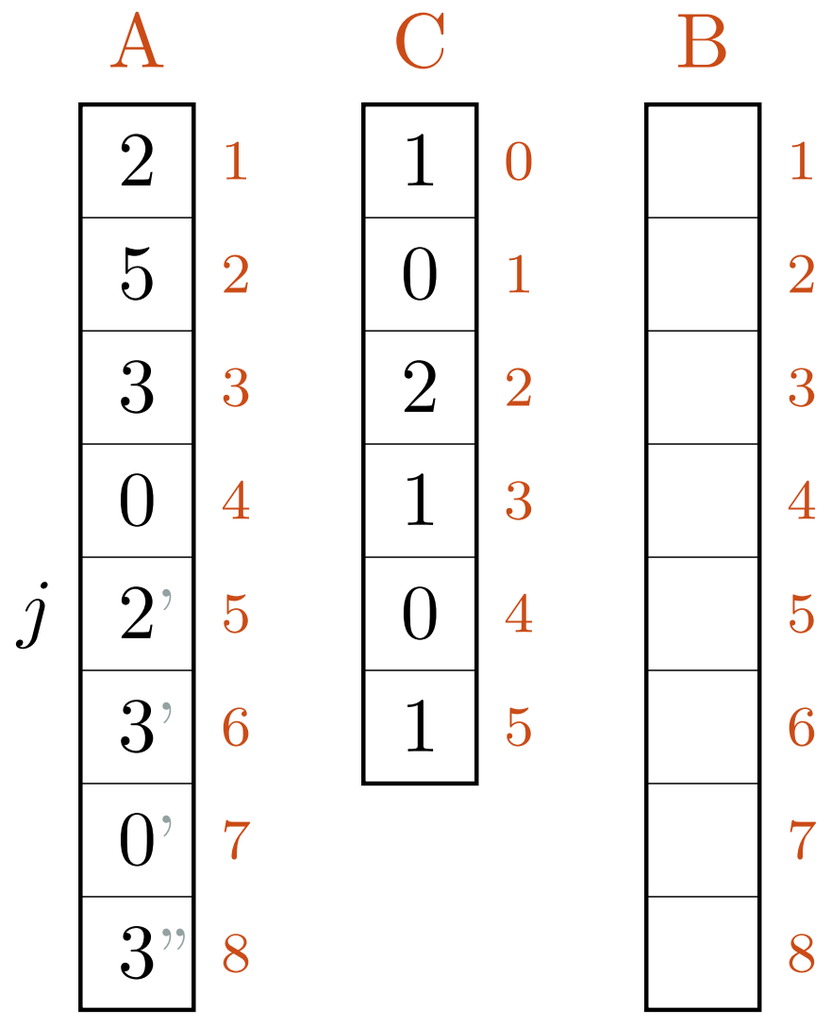
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



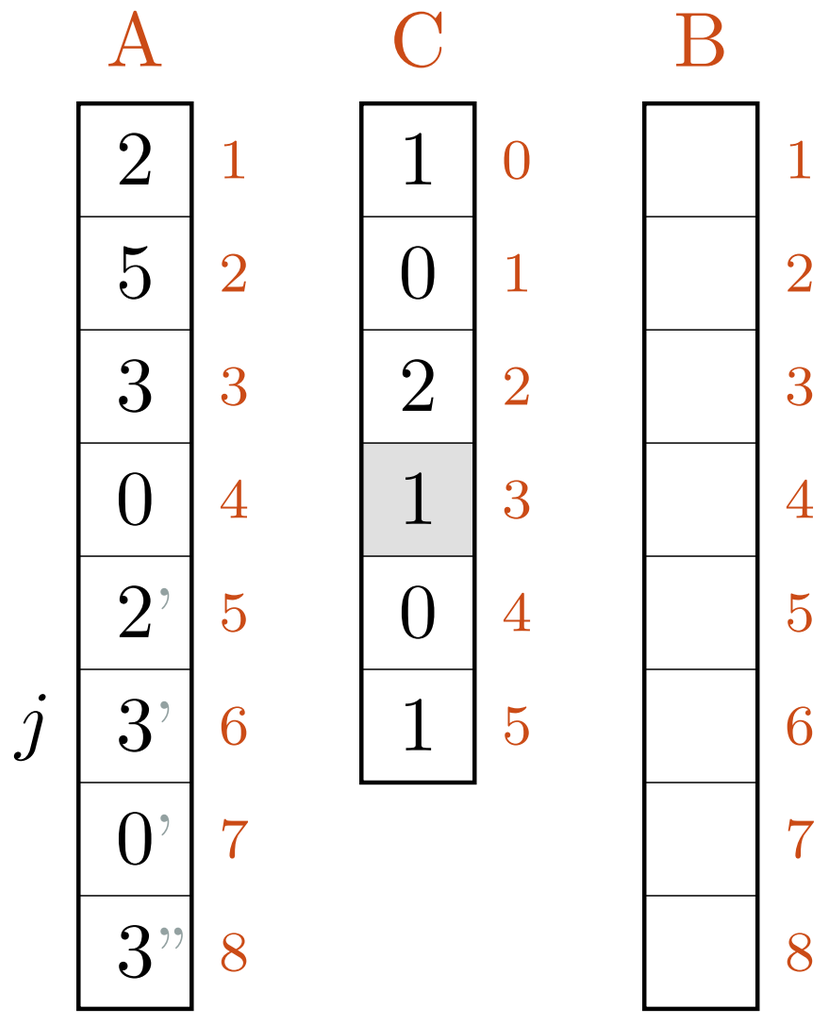
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



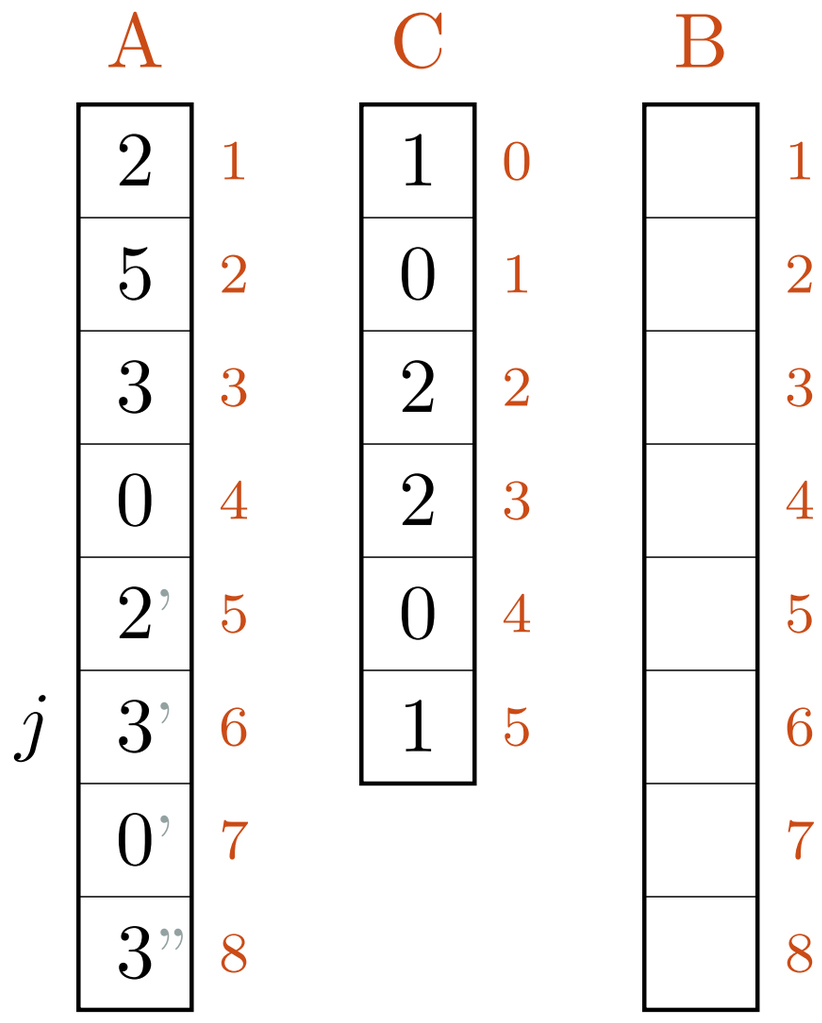
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



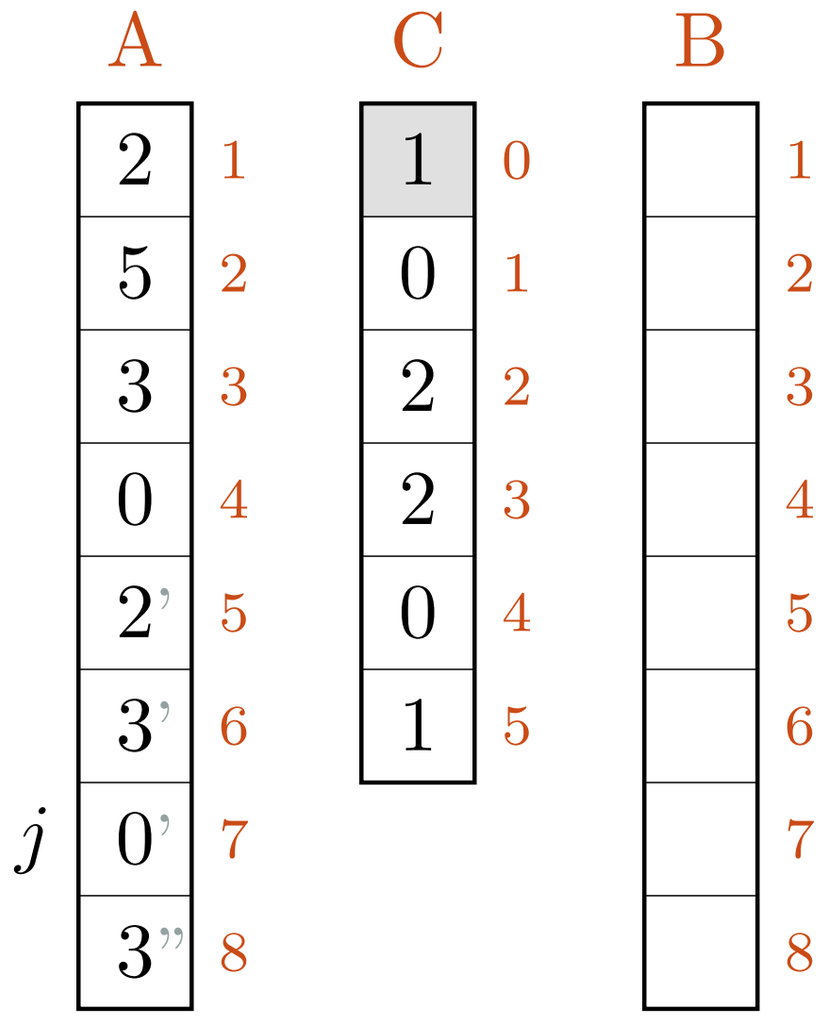
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



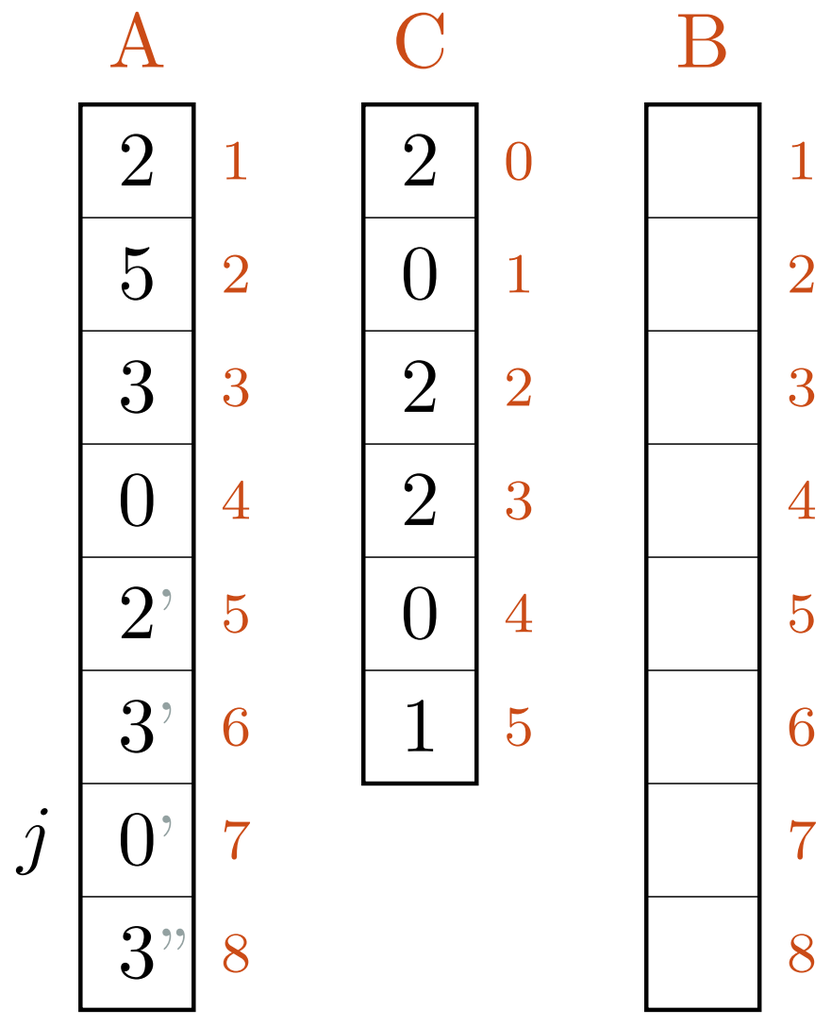
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



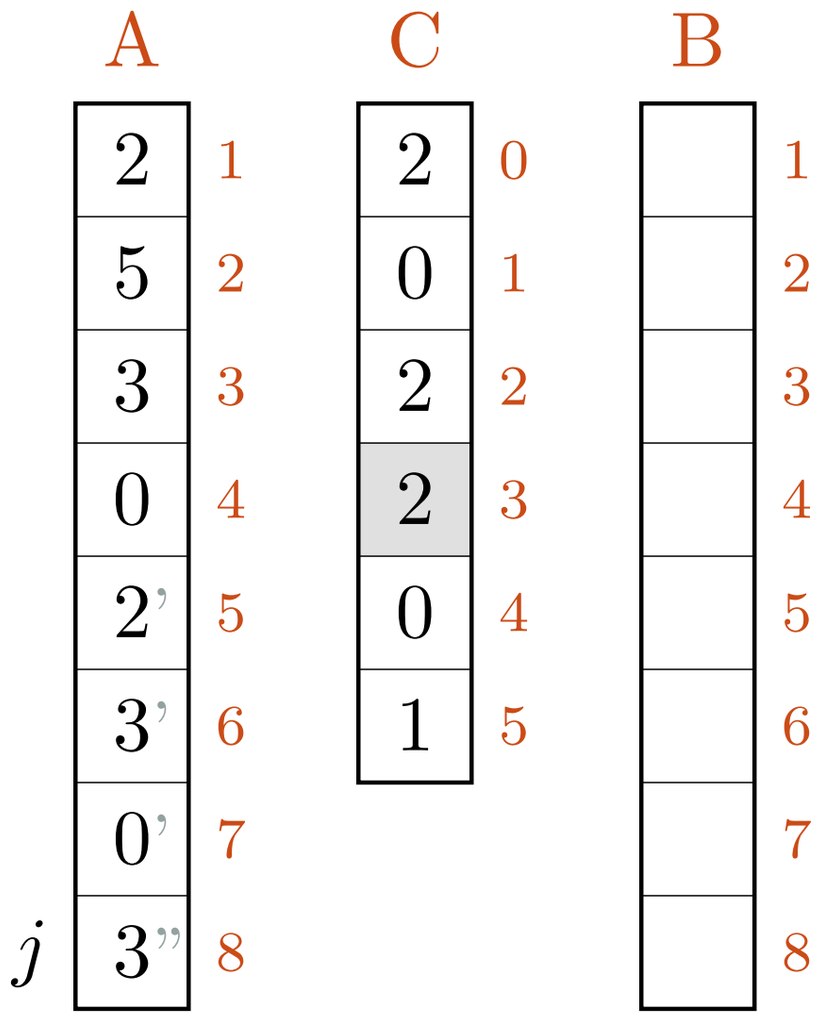
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



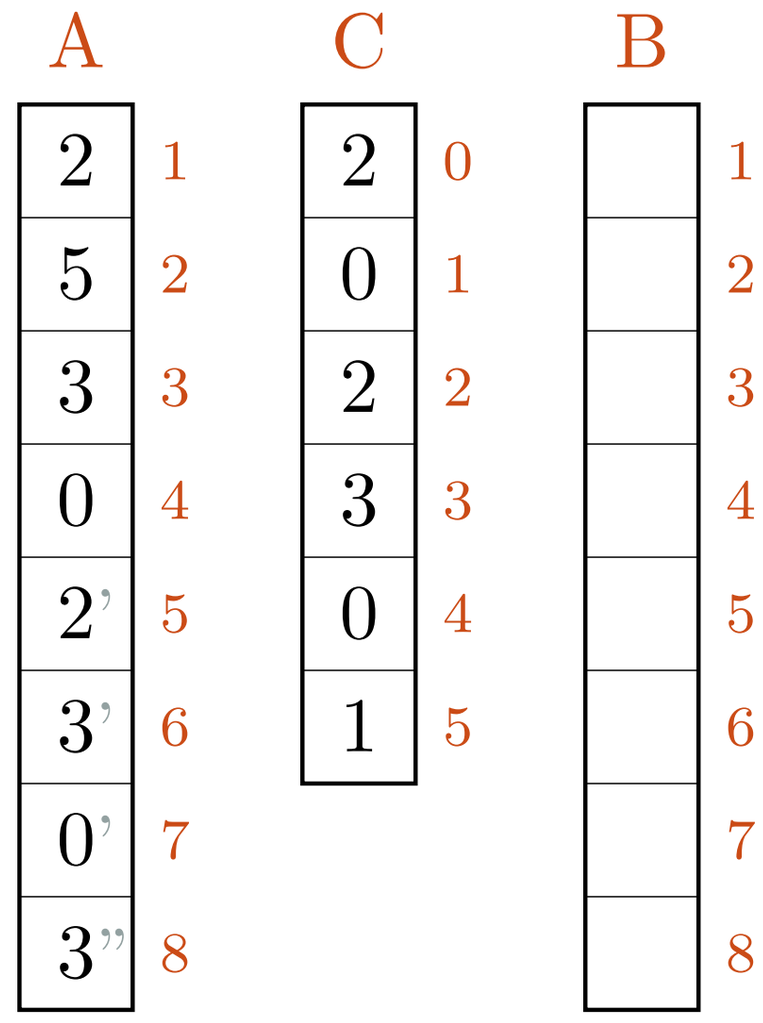
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



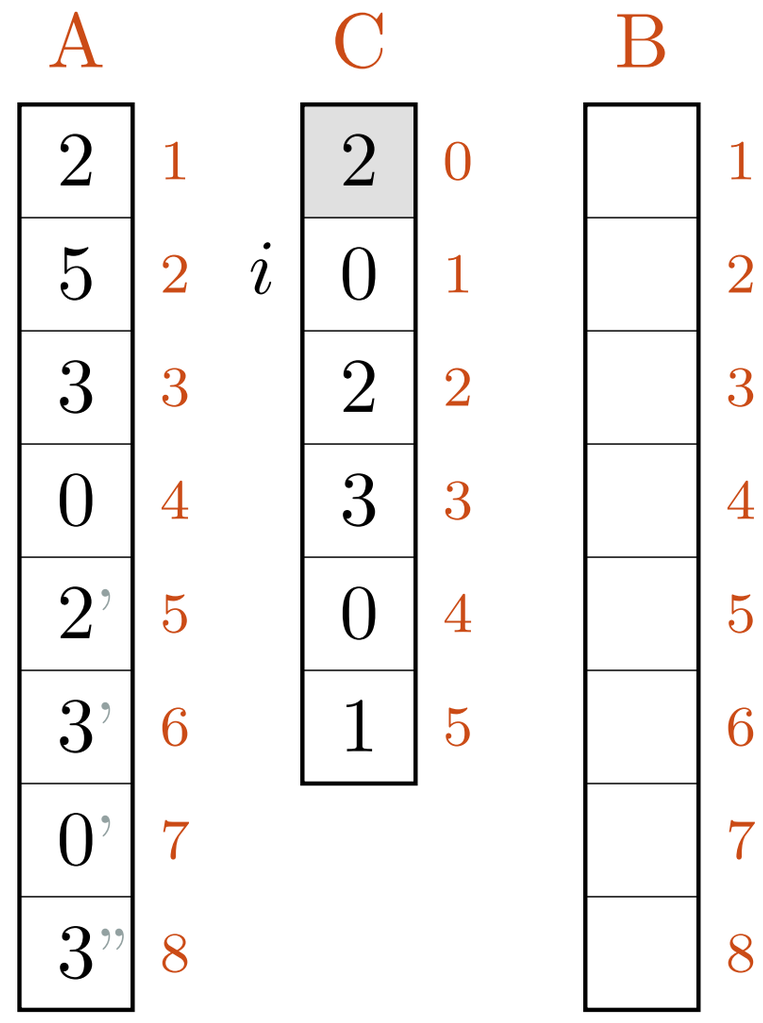
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



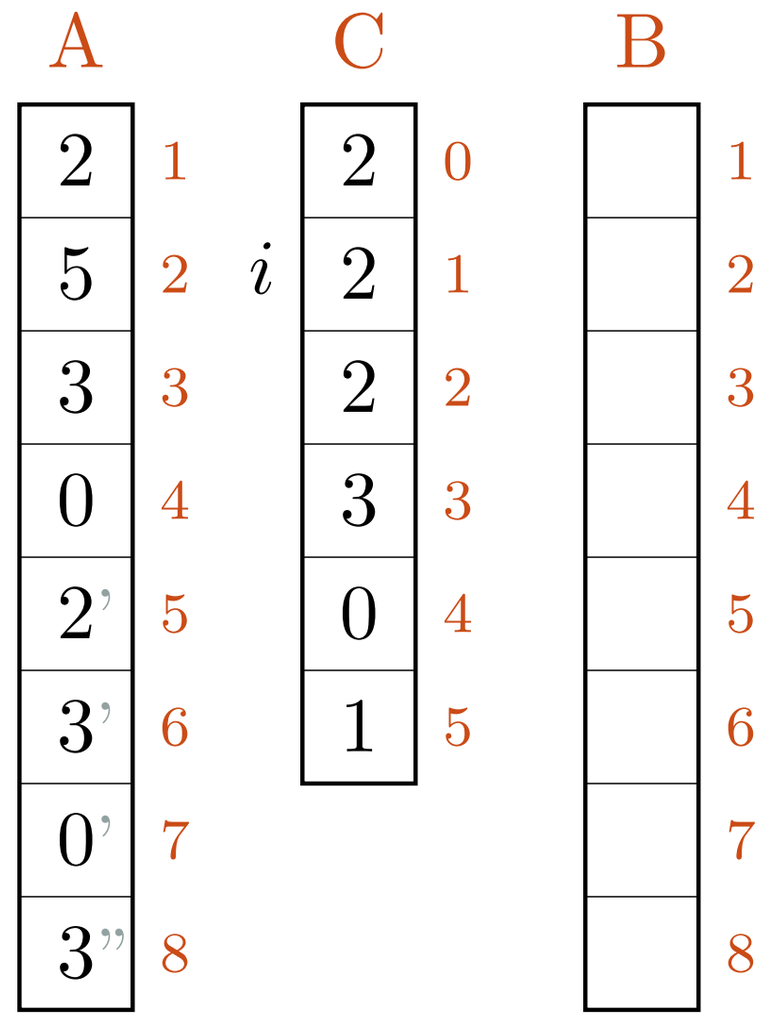
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



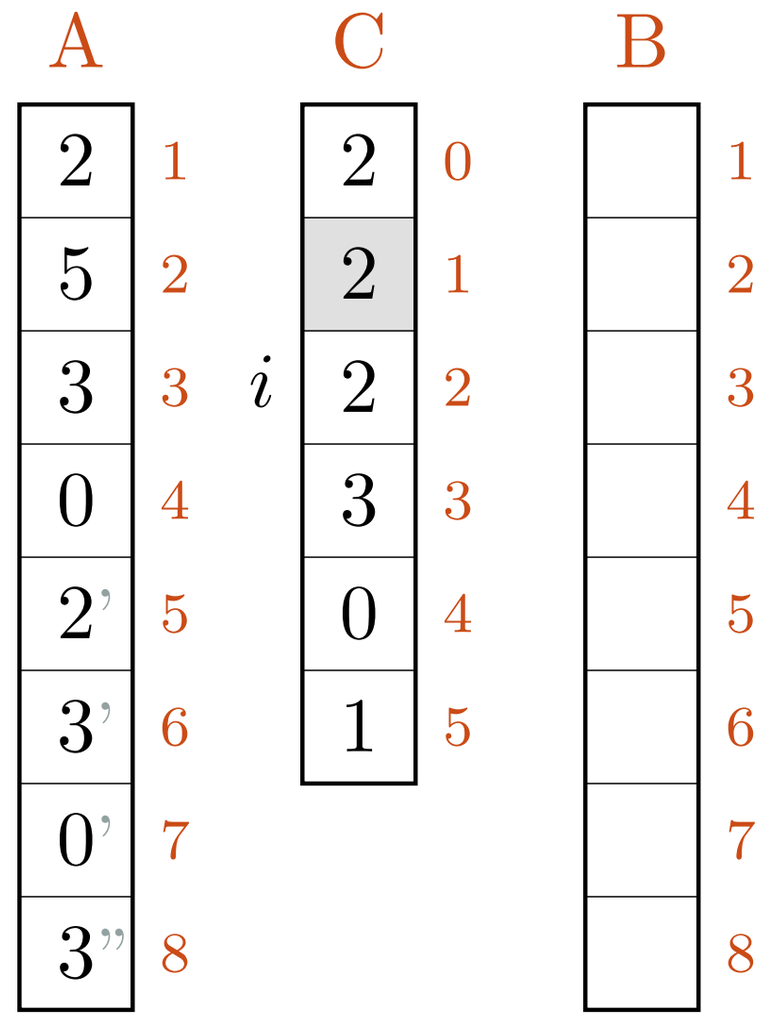
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



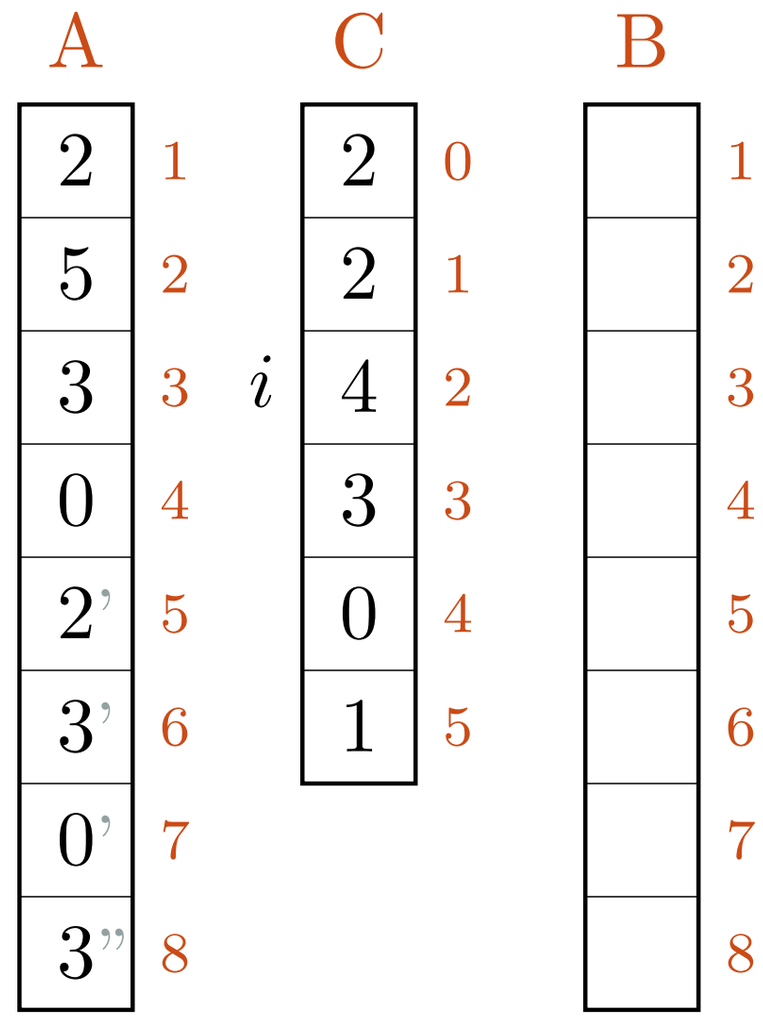
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



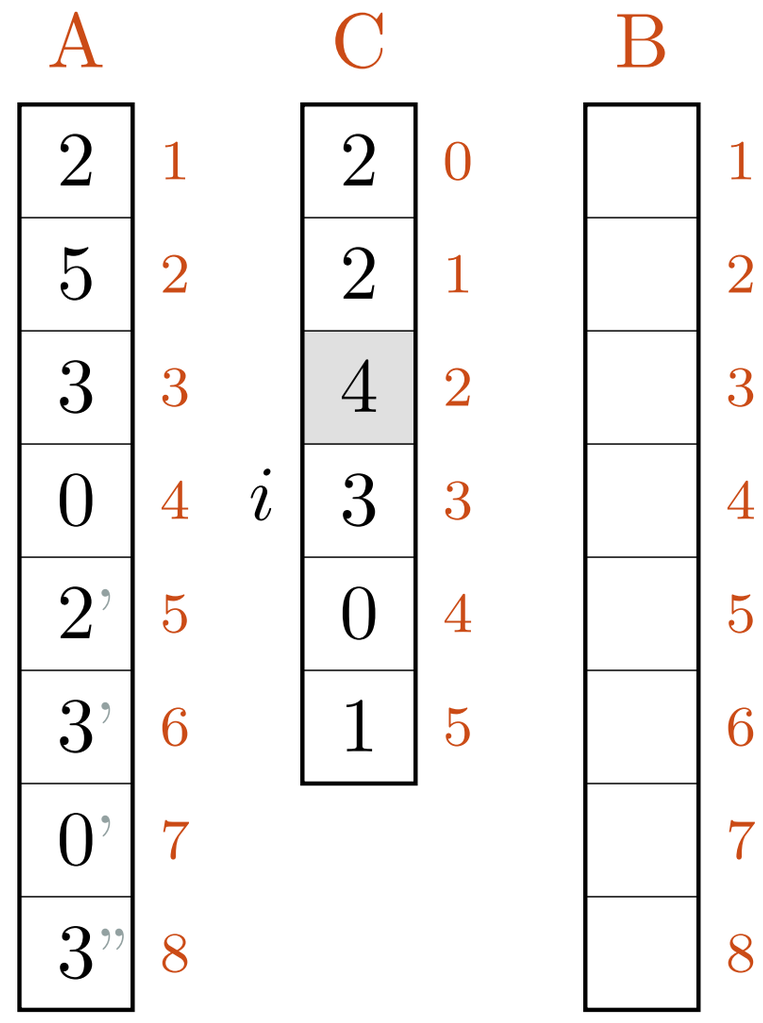
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



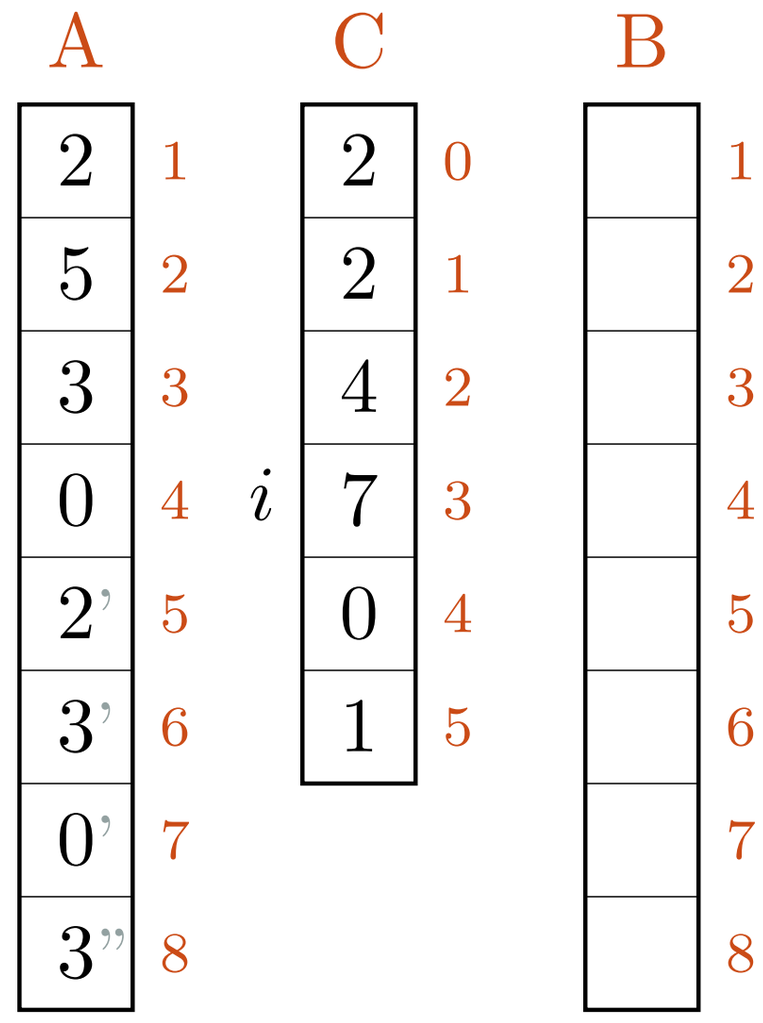
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



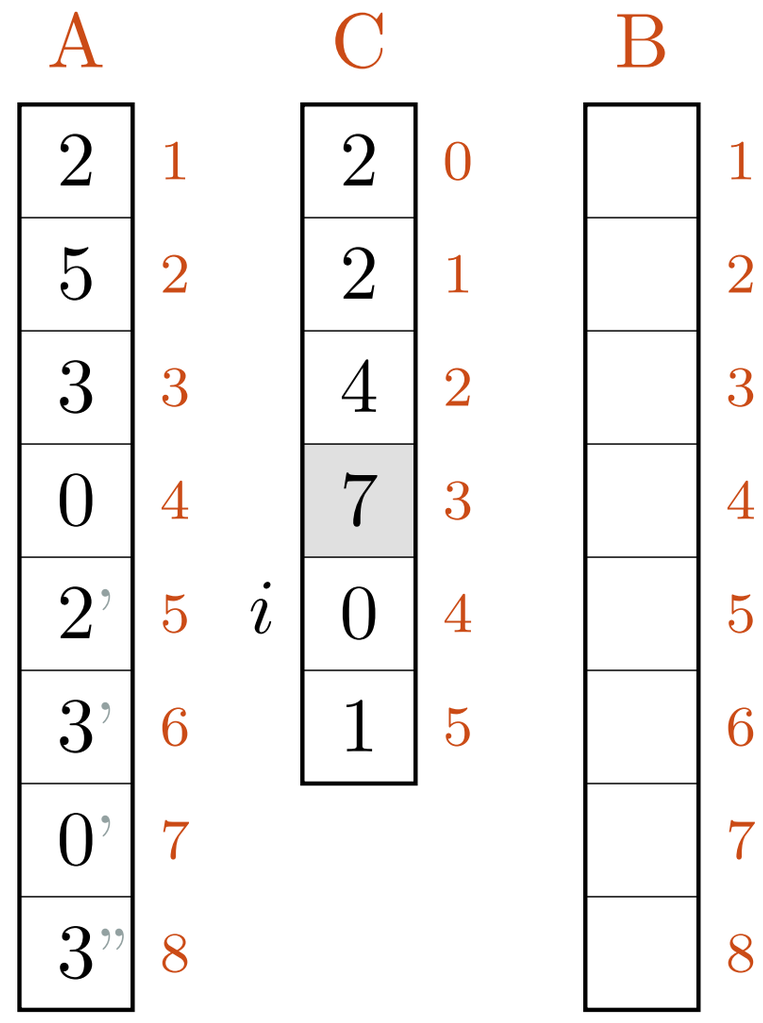
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



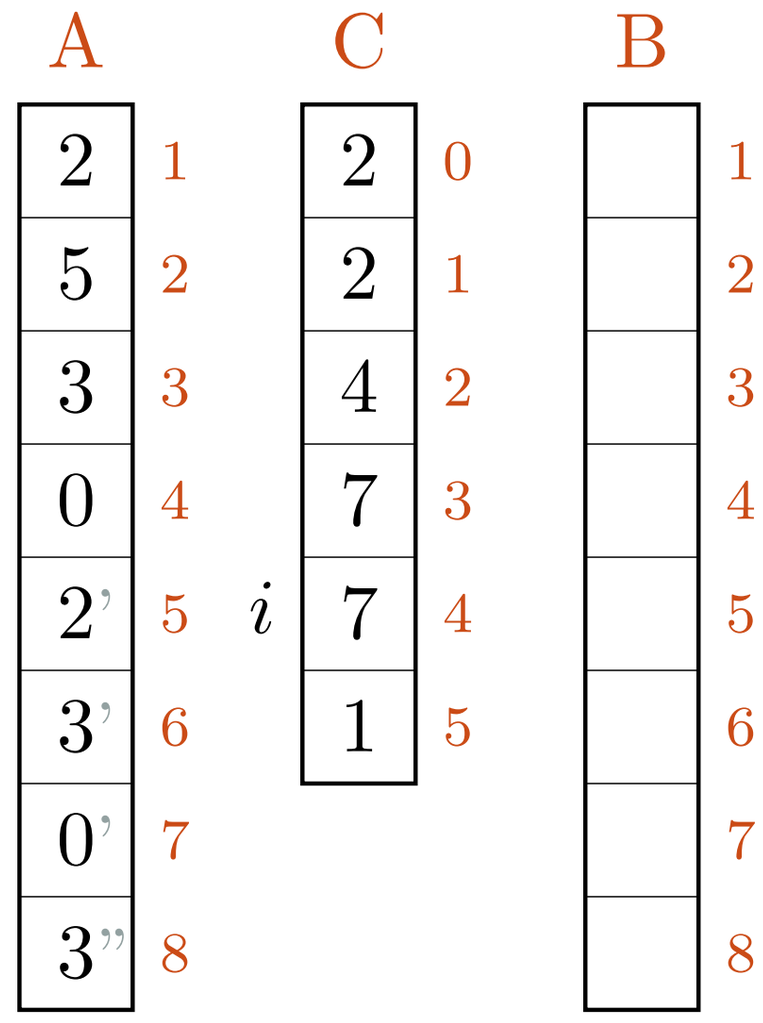
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



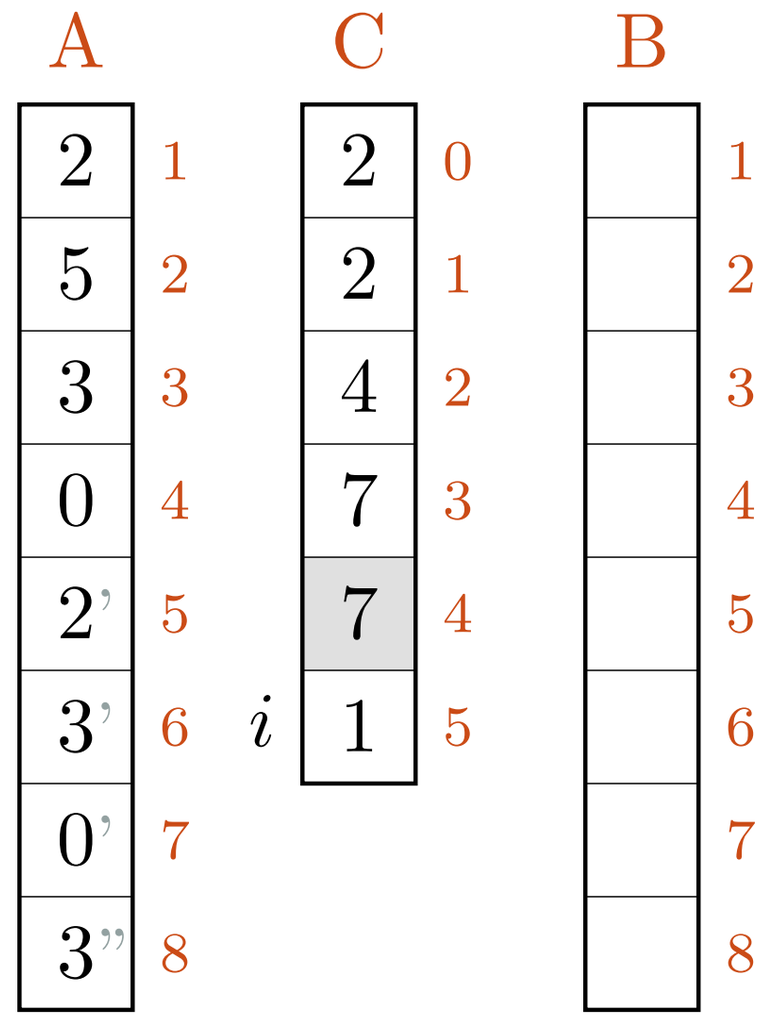
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



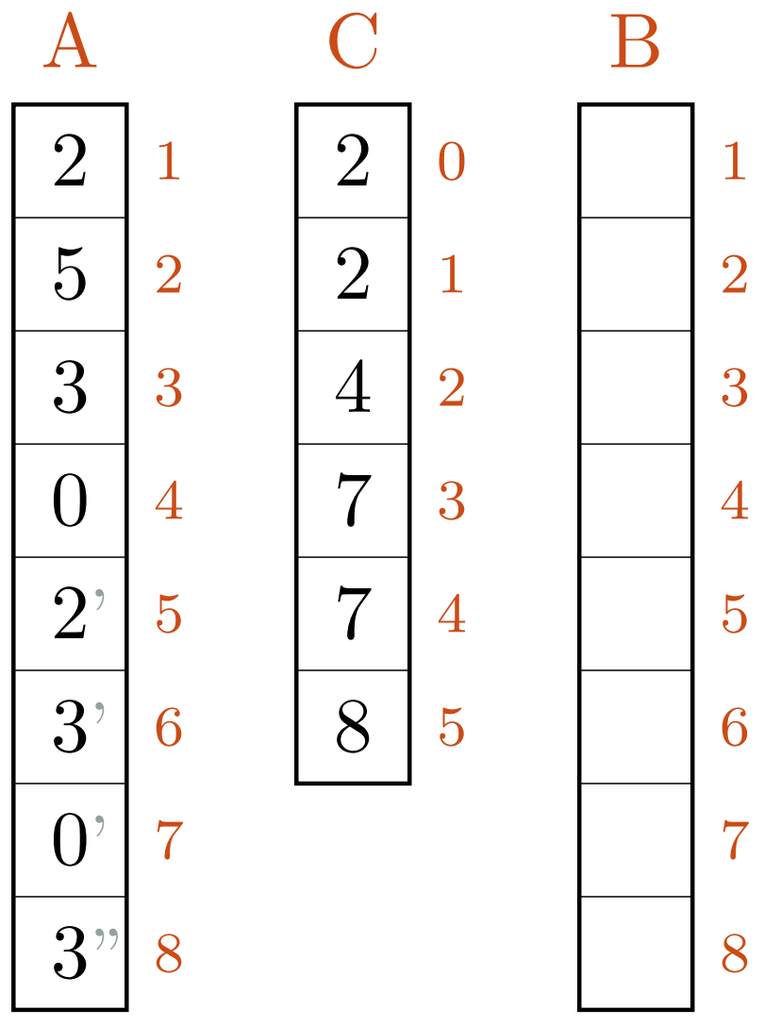
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



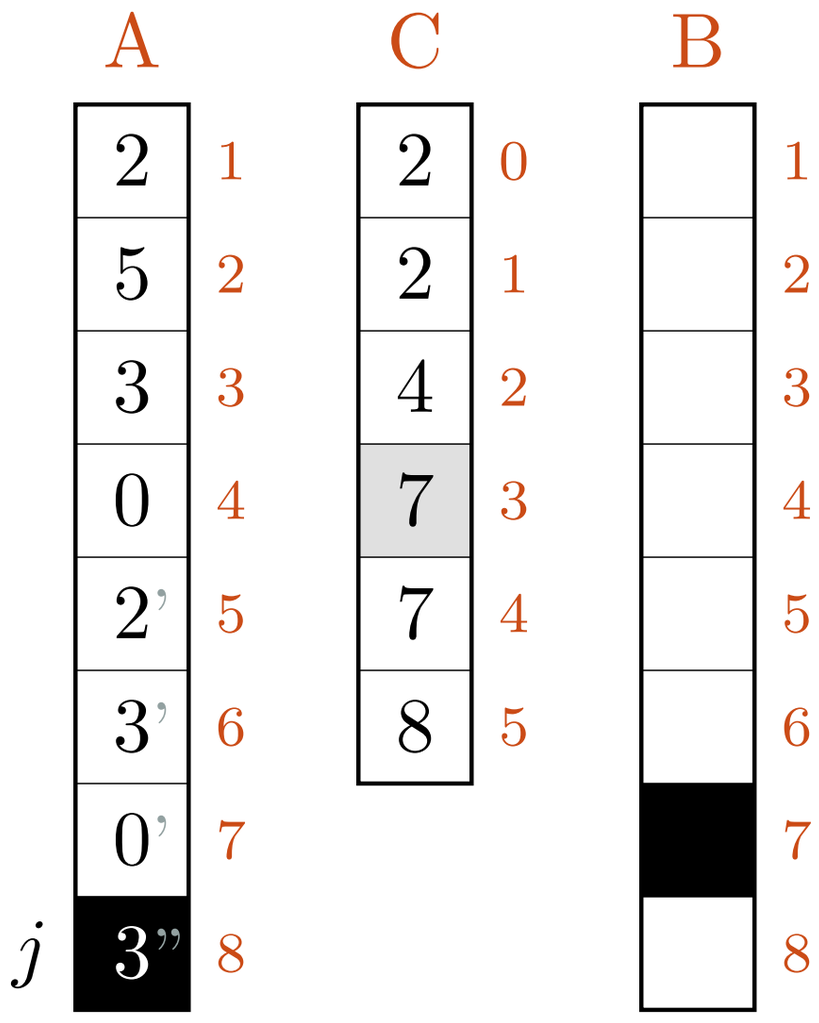
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



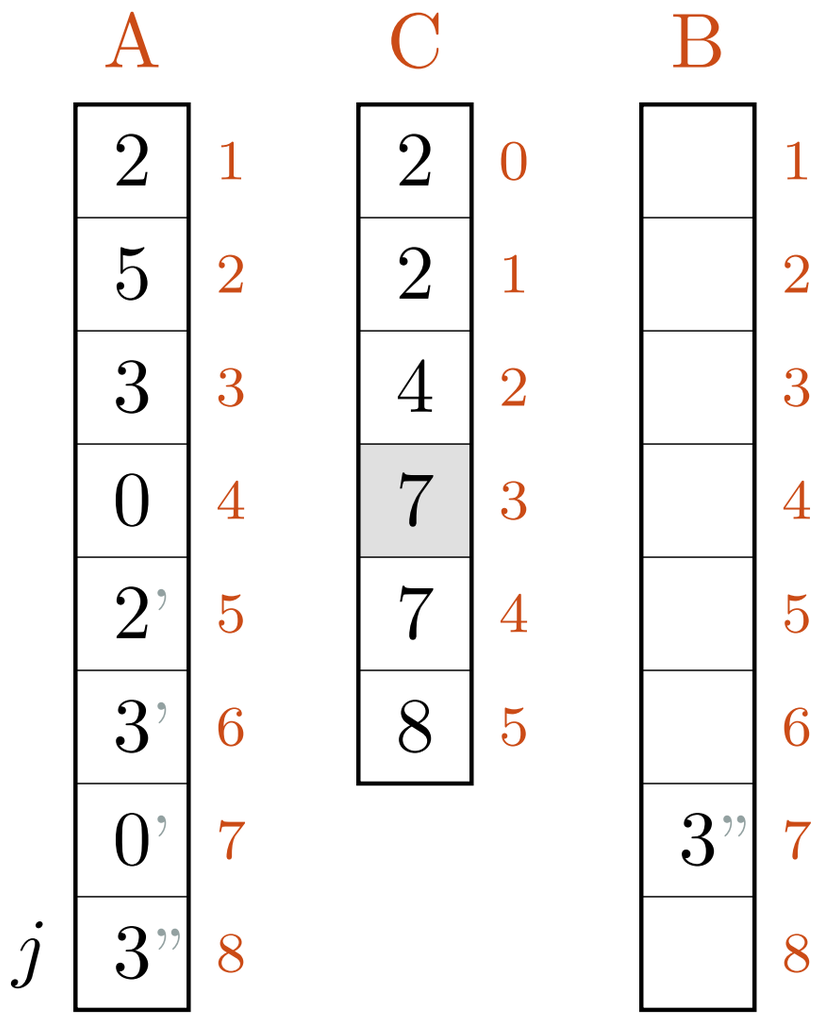
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



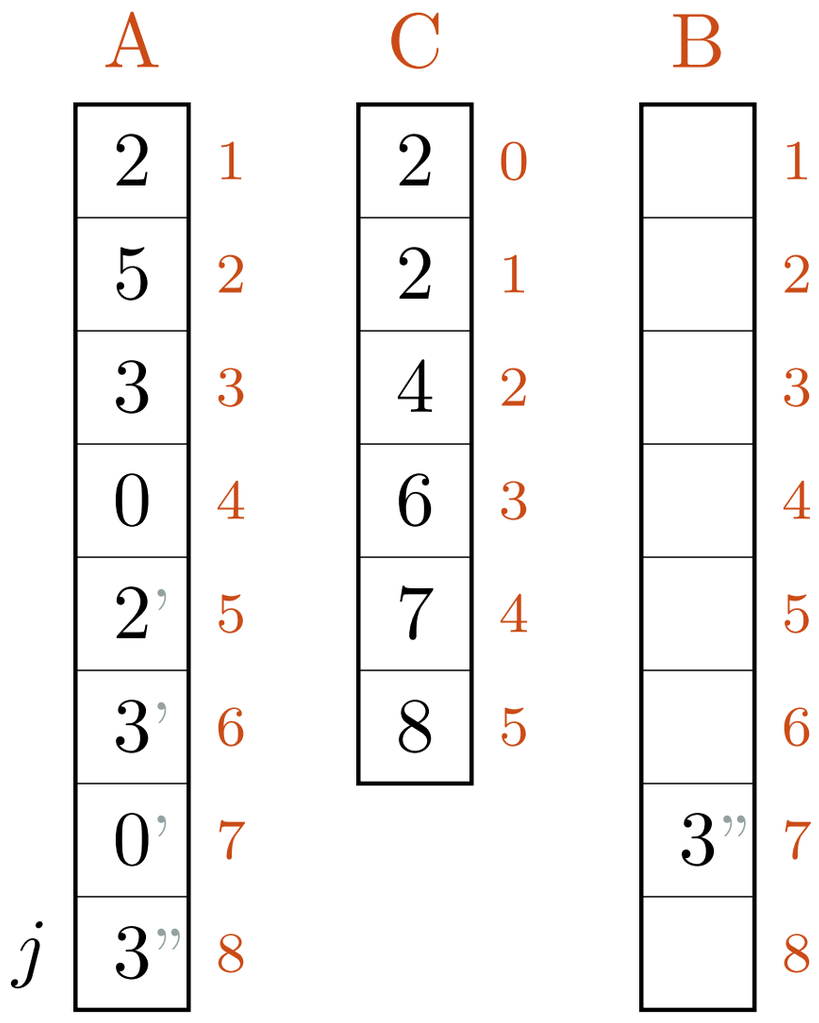
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



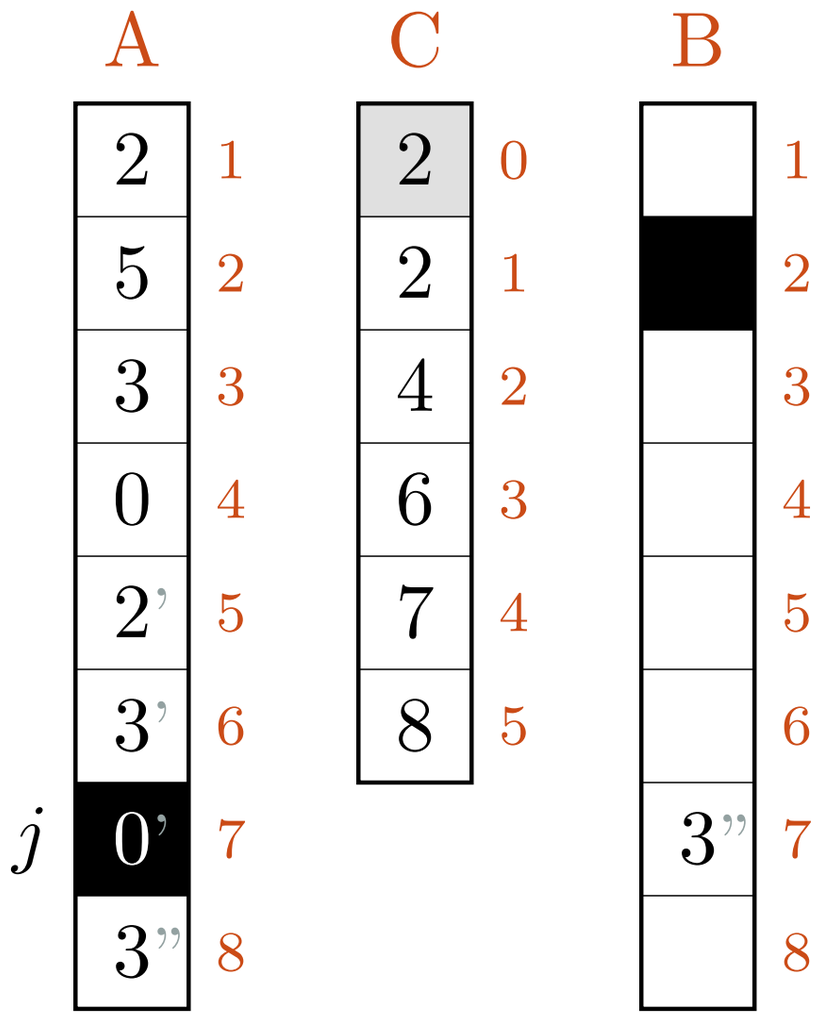
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



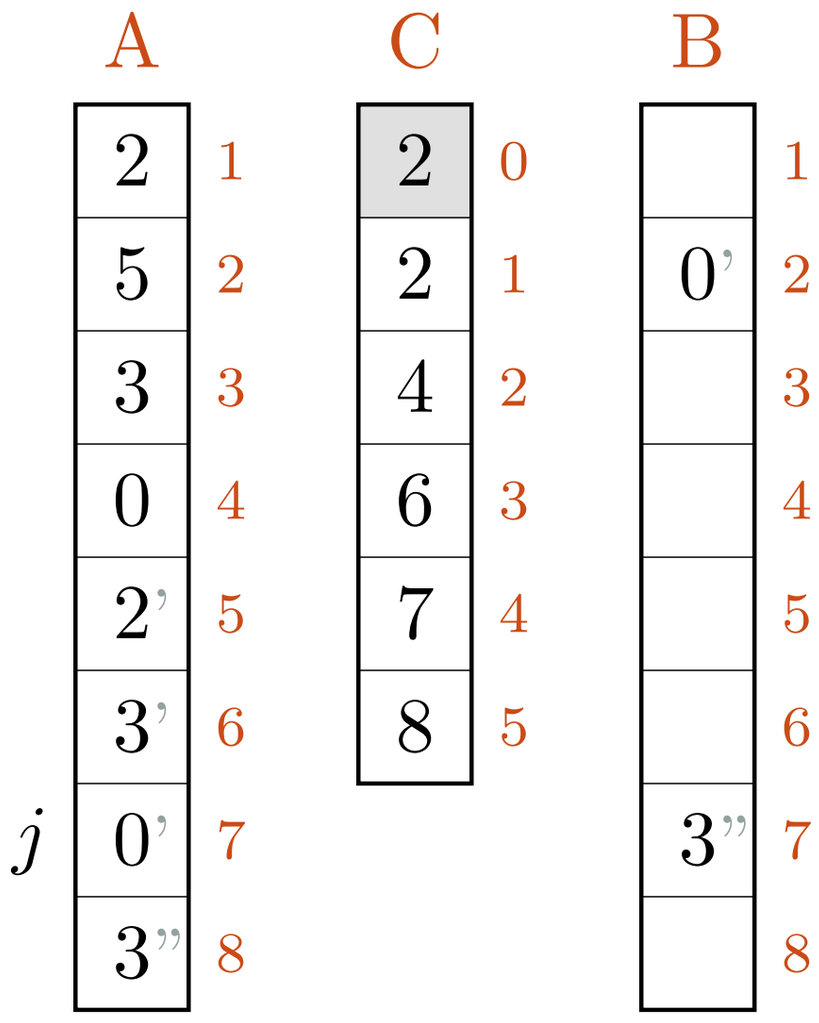
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



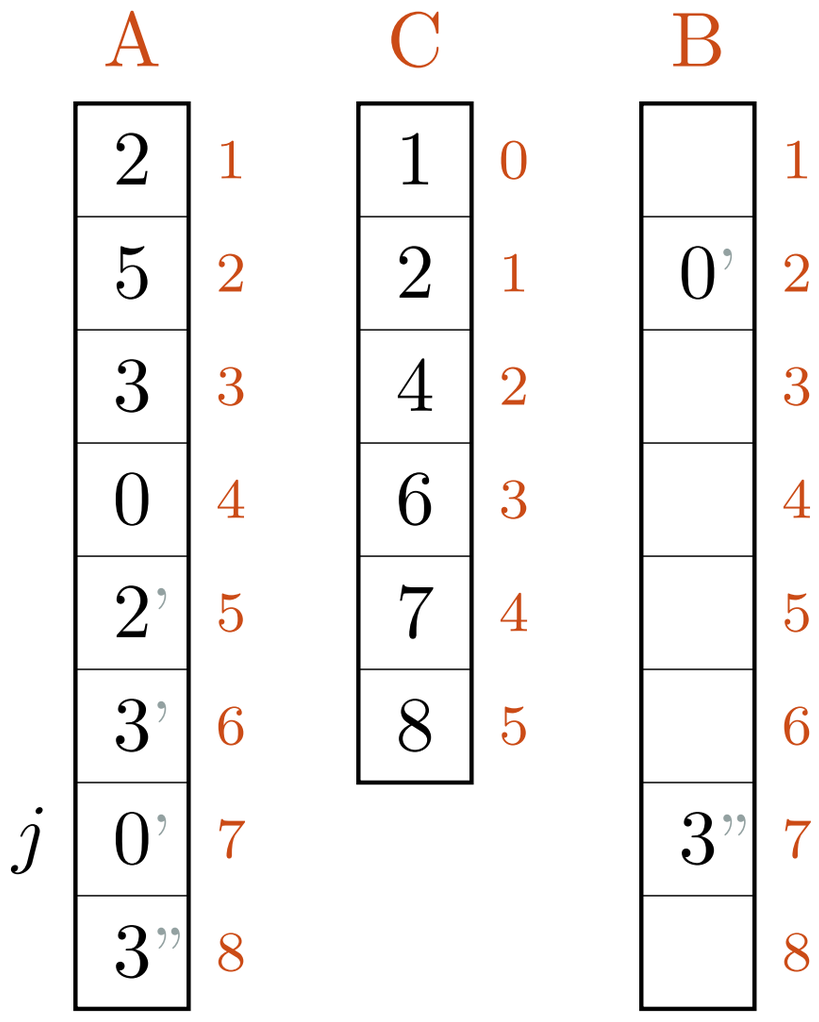
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



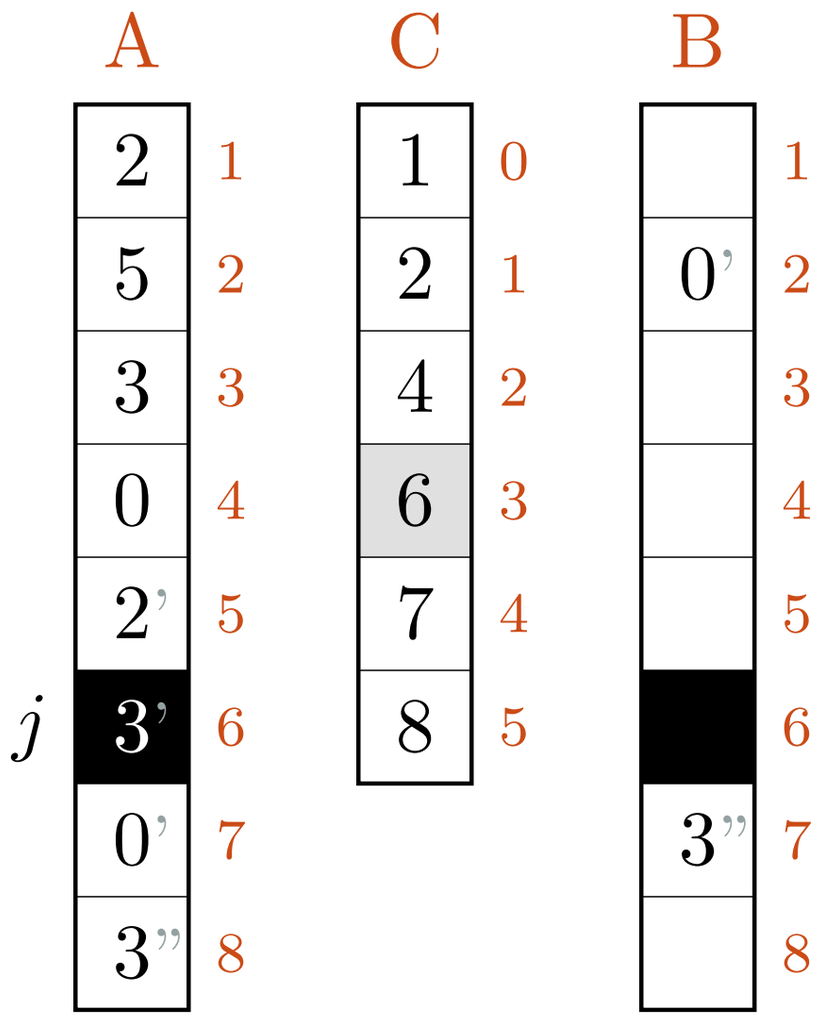
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



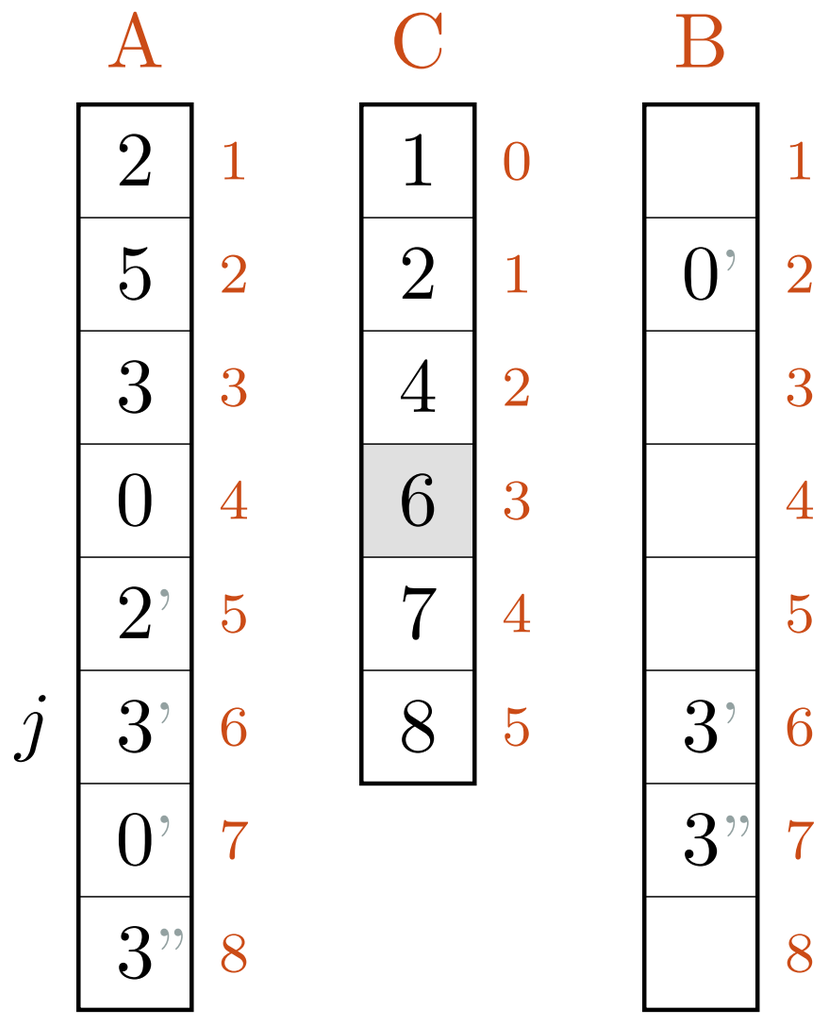
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



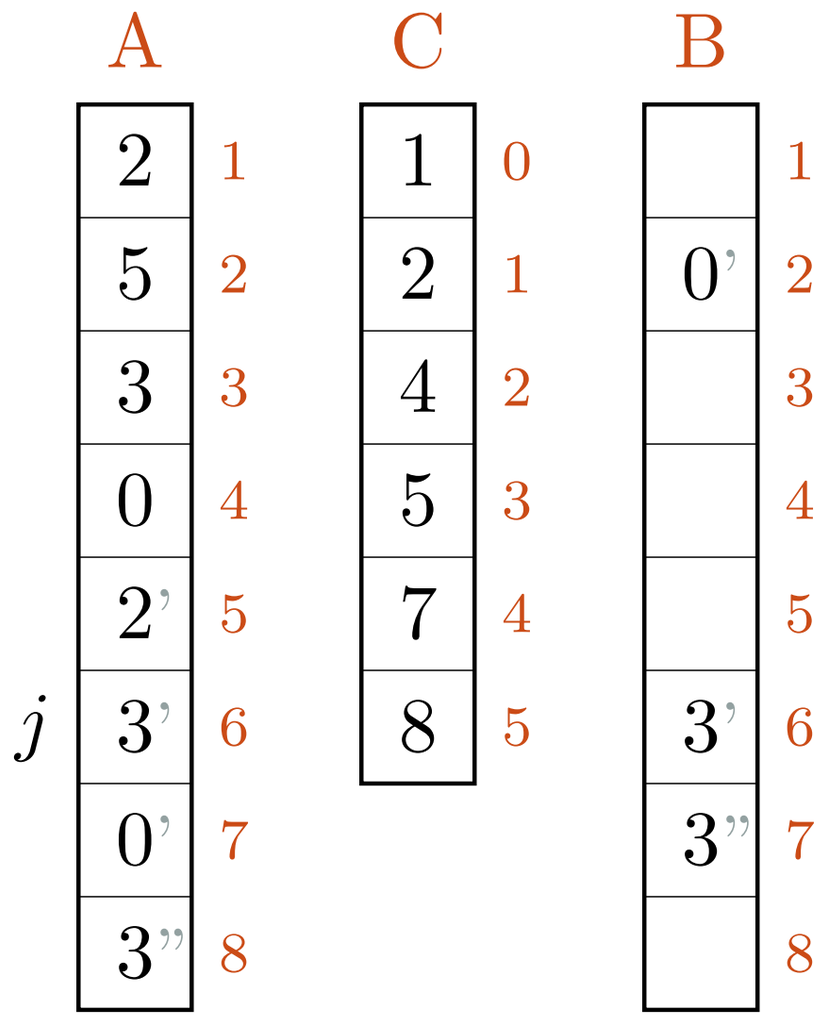
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



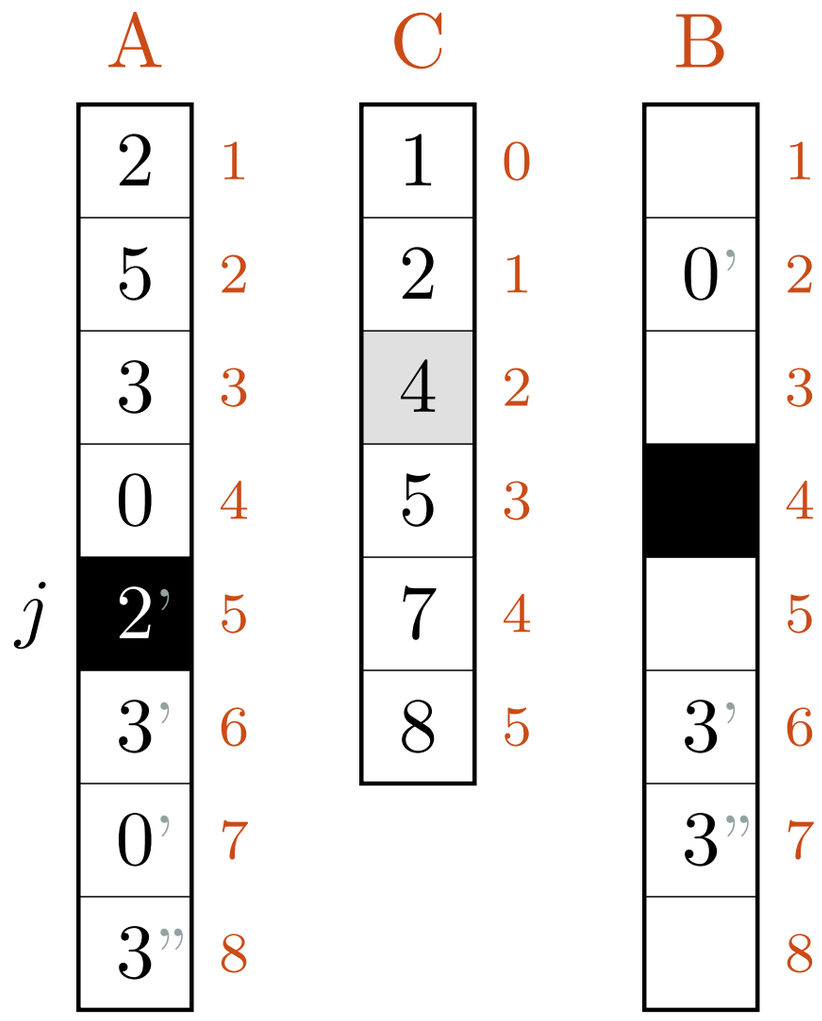
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



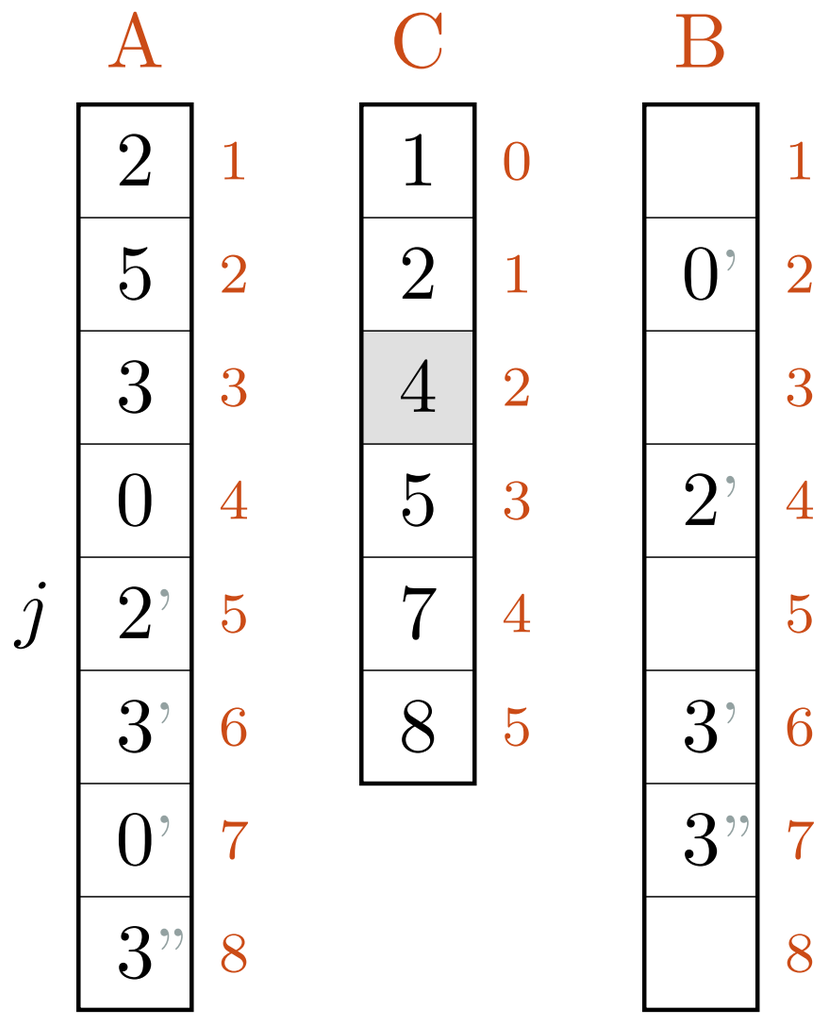
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



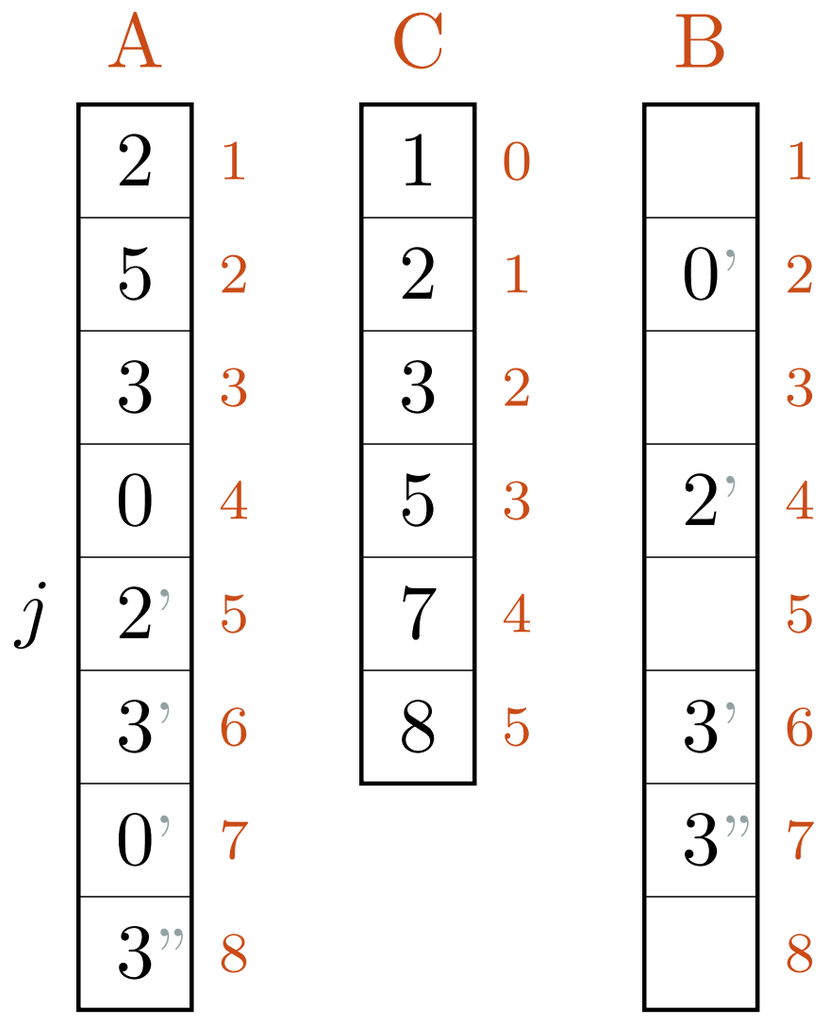
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



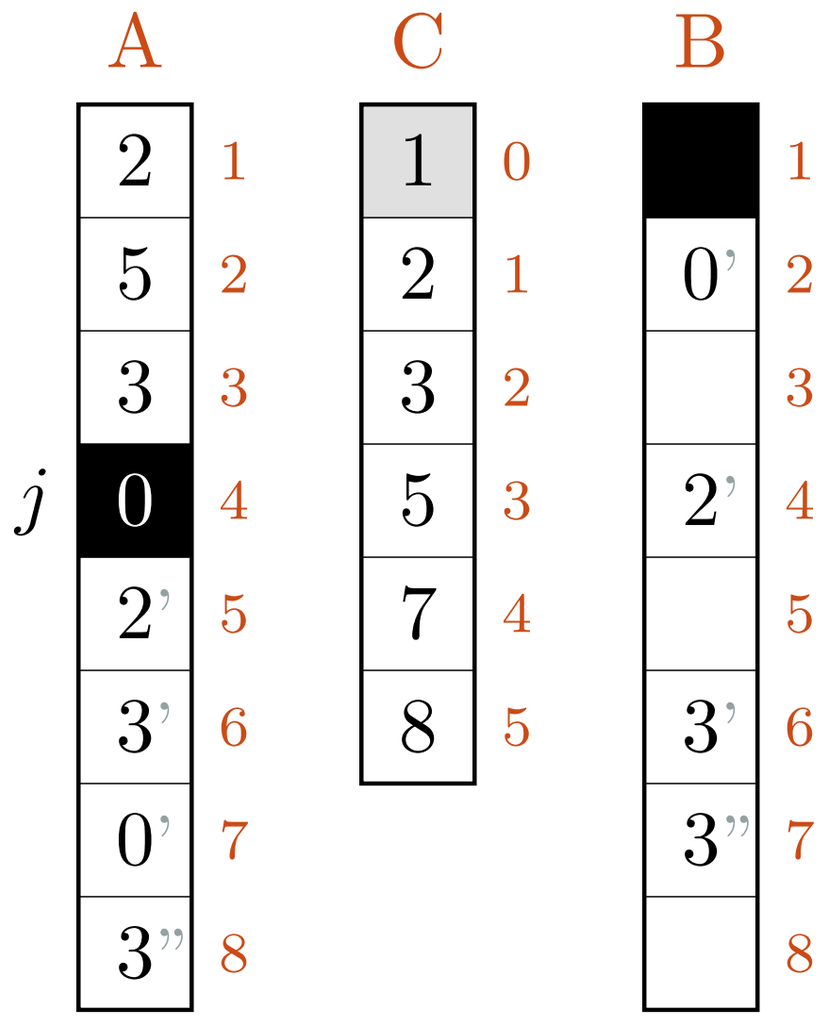
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



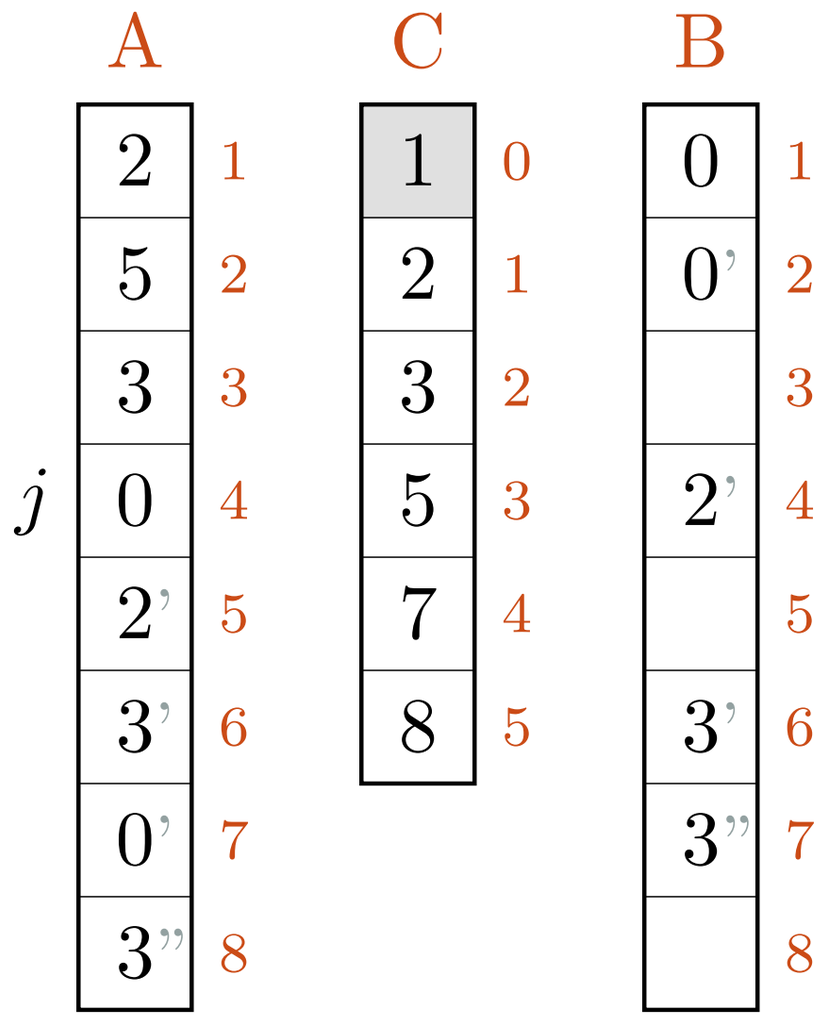
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



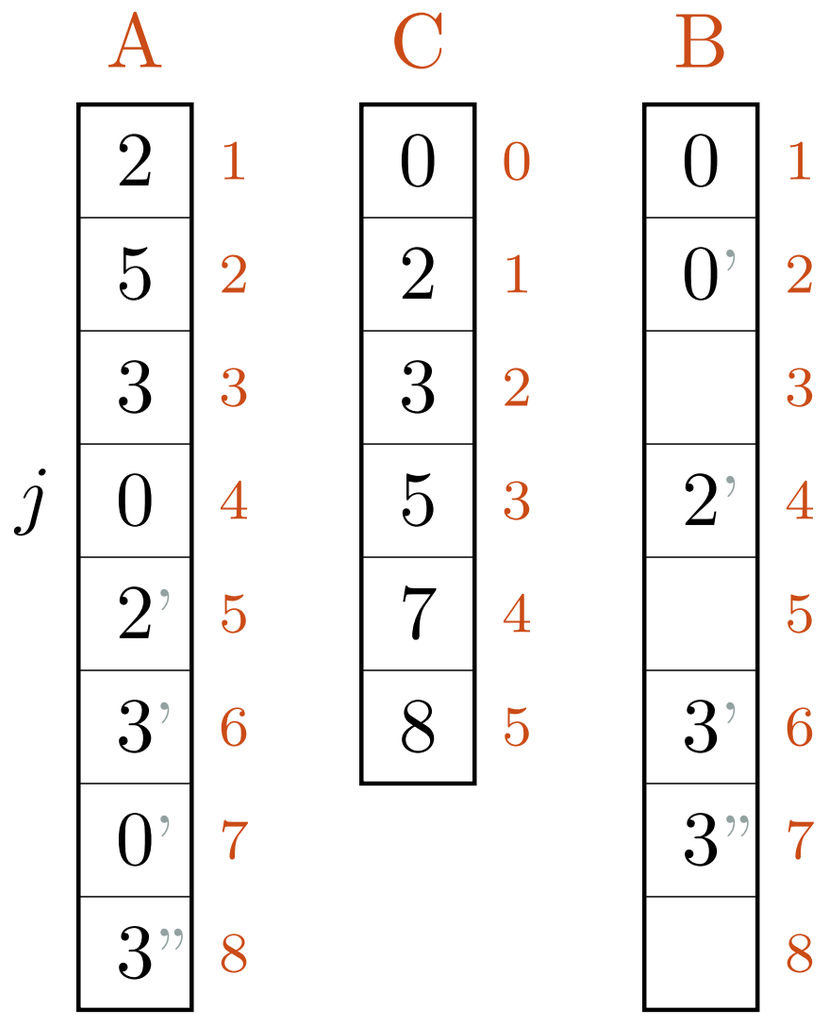
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



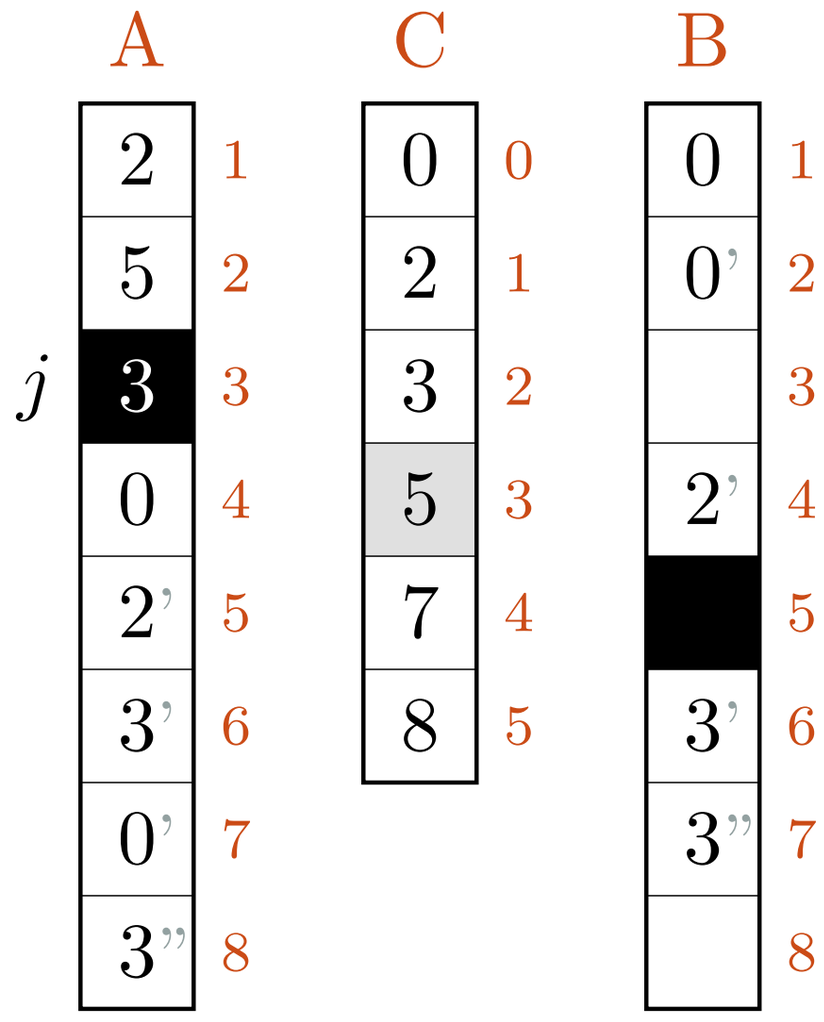
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



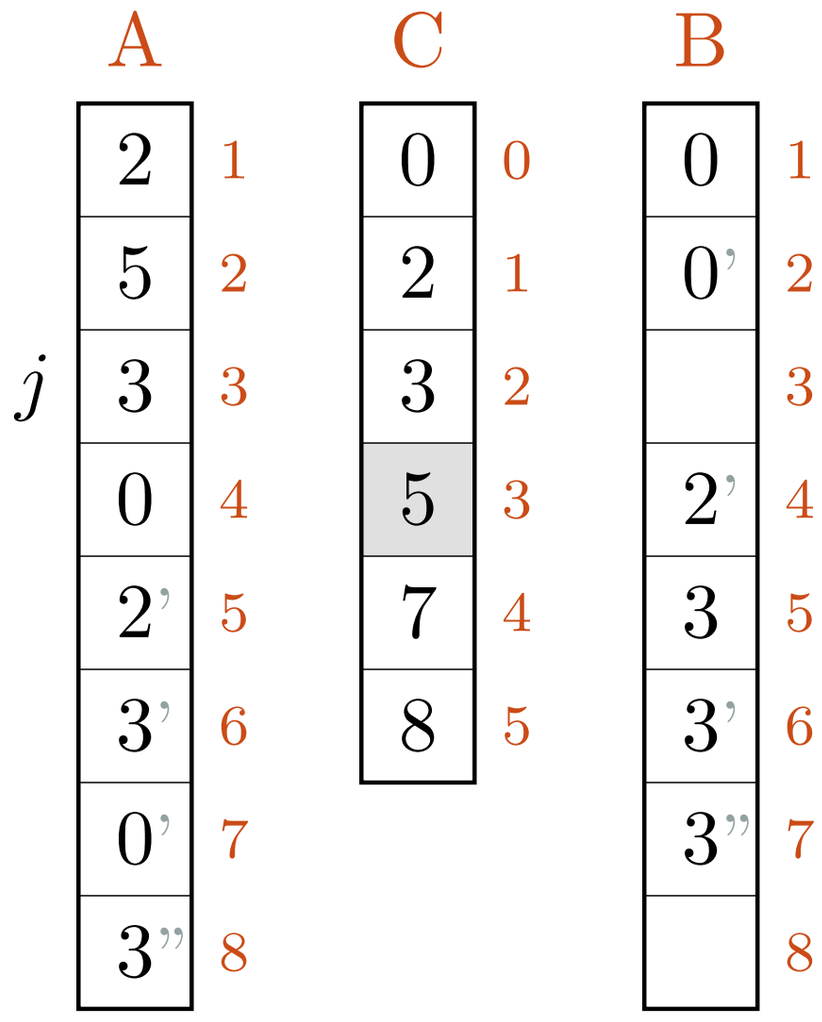
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



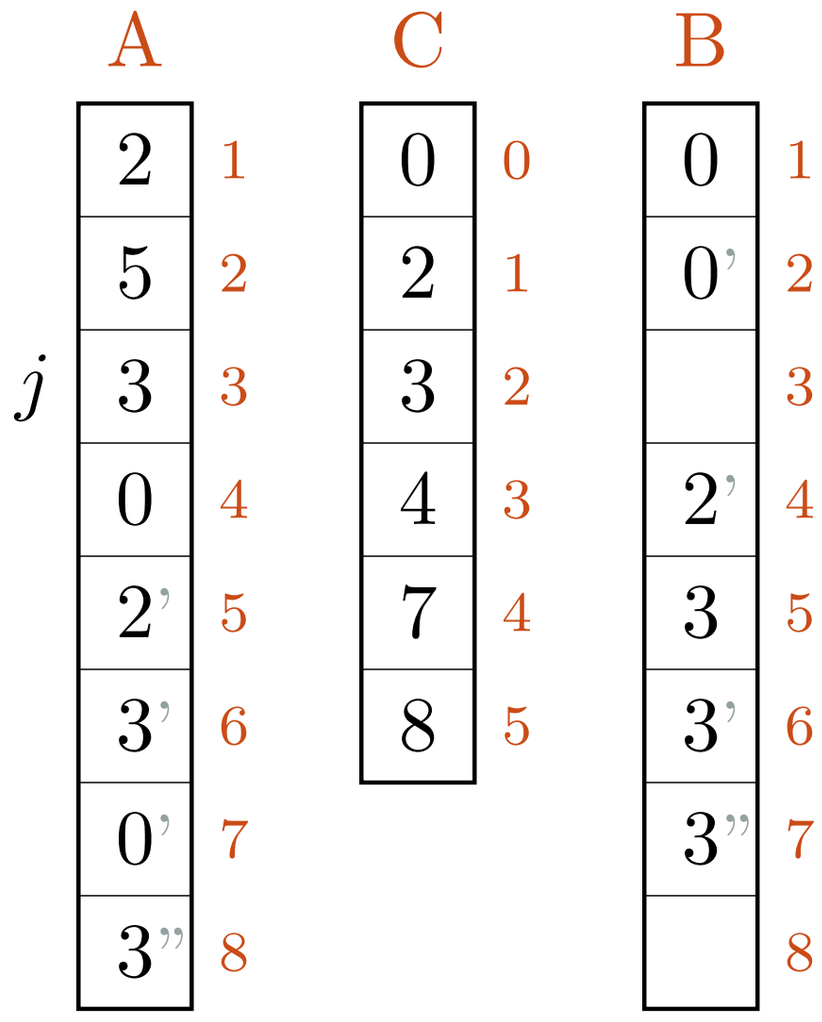
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



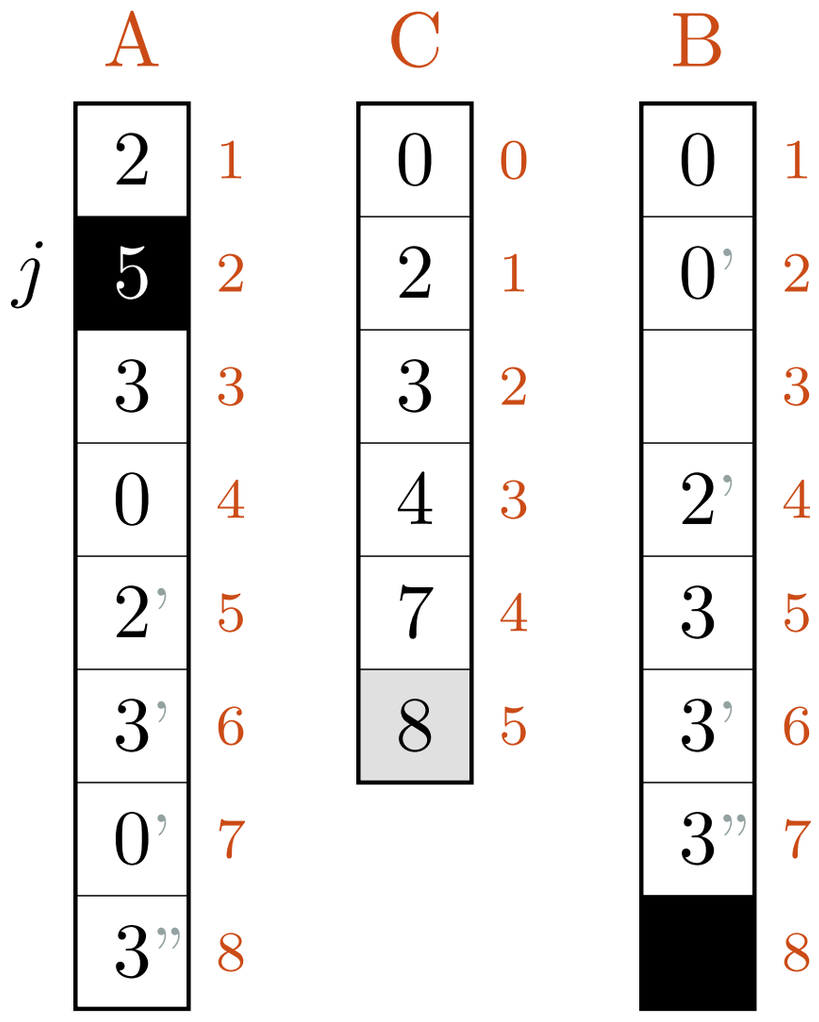
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



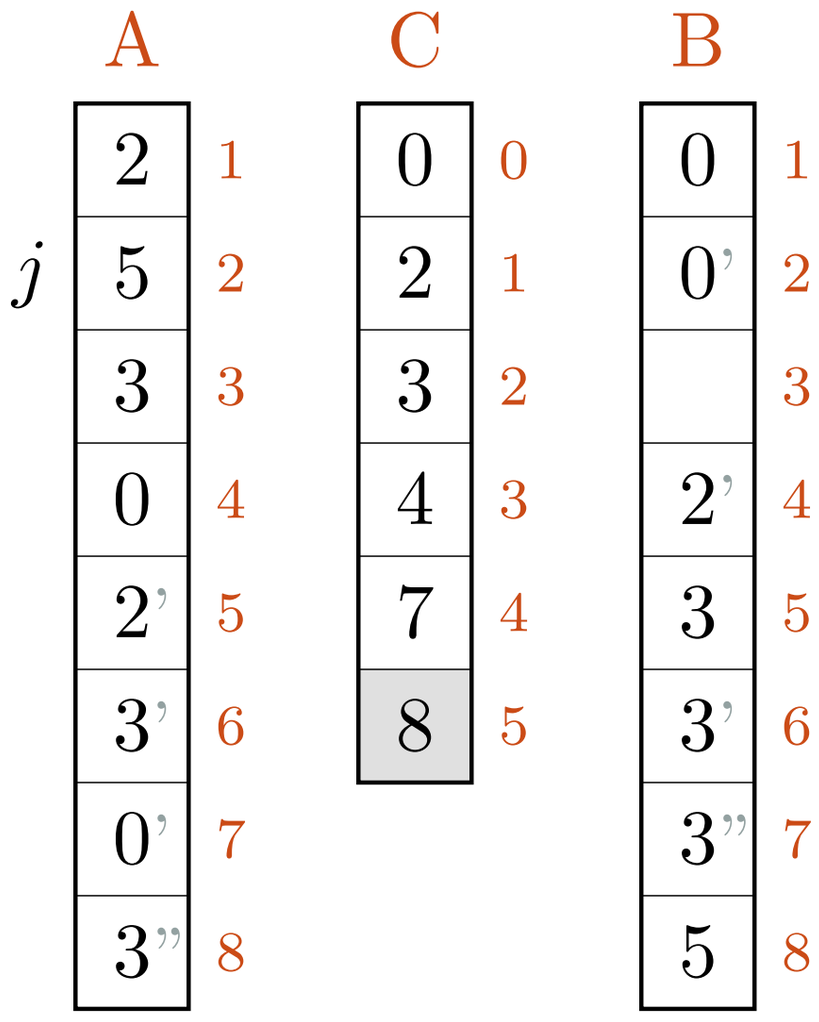
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



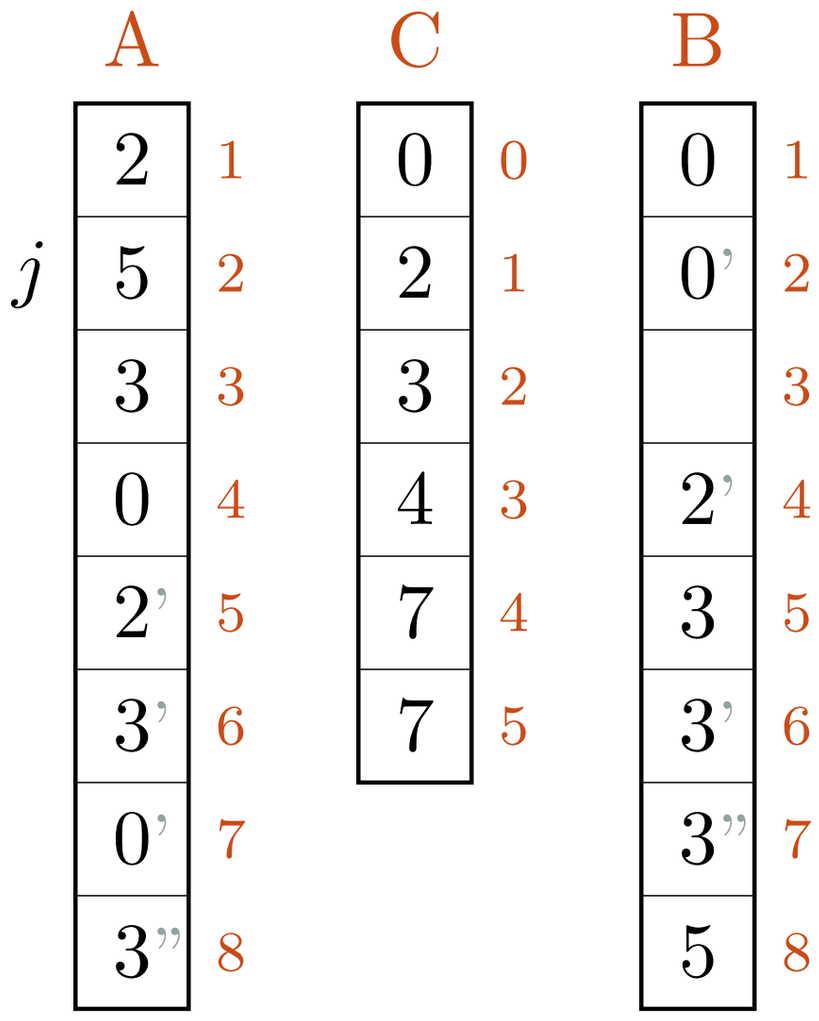
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



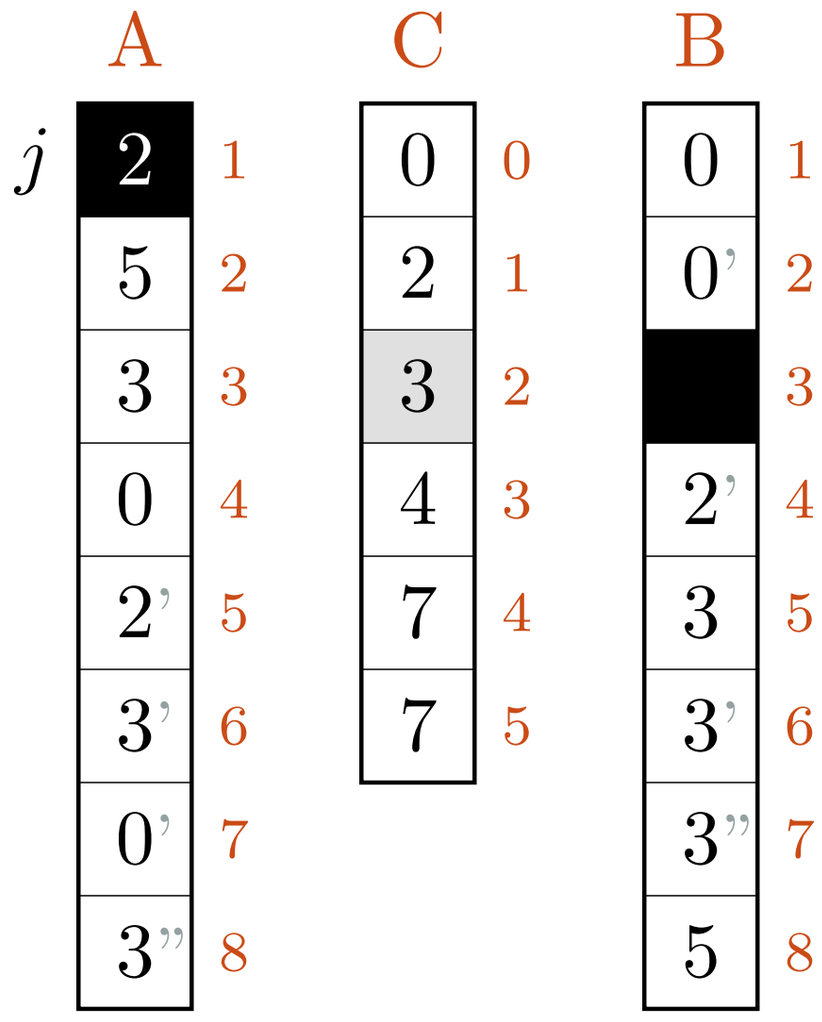
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



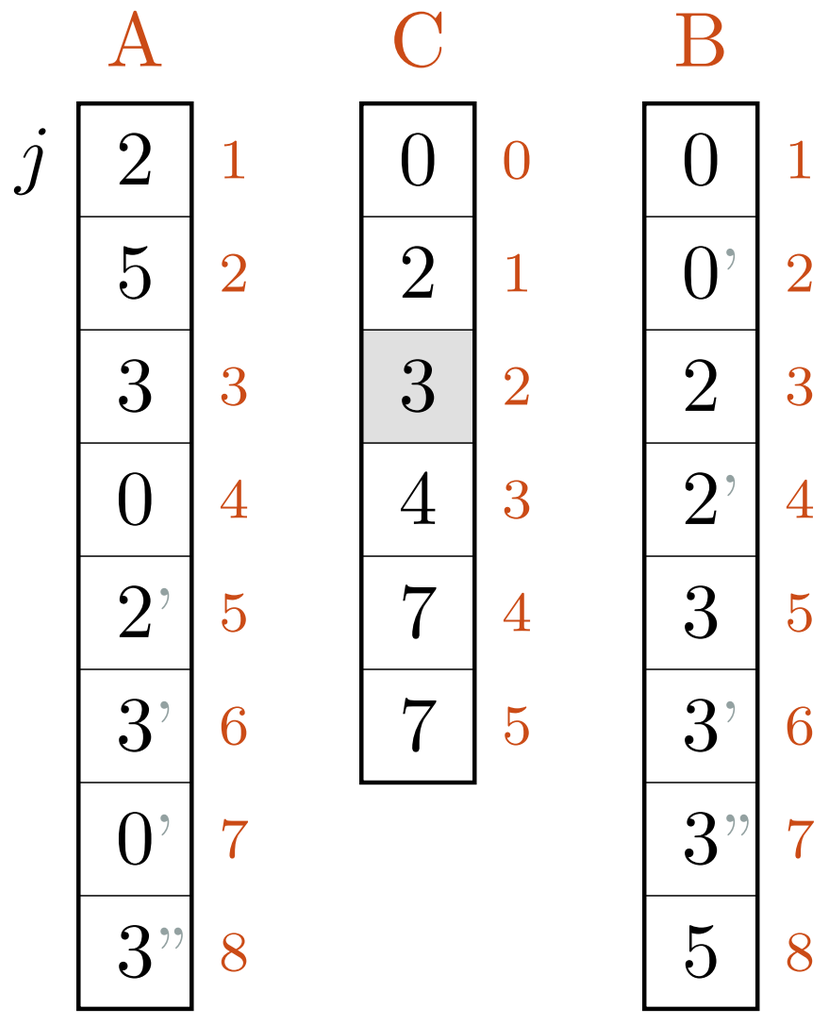
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```



```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  for i = 1 to k
 7      C[i] = C[i] + C[i − 1]
 8  for j = A.length downto 1
 9      B[C[A[j]]] = A[j]
10      C[A[j]] = C[A[j]] − 1
```

A		C		B	
2	1	0	0	0	1
5	2	2	1	0’	2
3	3	2	2	2	3
0	4	4	3	2’	4
2’	5	7	4	3	5
3’	6	7	5	3’	6
0’	7			3”	7
3”	8			5	8

$$\mathrm{T}(n) = \Theta(n + k)$$

$\Theta(1)$ iterasjoner over $\mathrm{A}[1\,..\,n]$, $\mathrm{B}[1\,..\,n]$ og $\mathrm{C}[0\,..\,k]$

Utvid verdiområdet

… uten lineær tidsøkning

3.6

Radikssortering

The Electrical Tabulating Machine.
By Dr. Herman Hollerith.

[Read before the Royal Statistical Society, 4th December, 1894.]

While engaged in work in the tenth census, that of 1880, my attention was called by Dr. Billings to the need of some mechanical device for facilitating the compilation of population and similar statistics. This led me to a consideration of the problems involved. I found, for example, that while we had collected the information regarding the conjugal condition of our 50,000,000 inhabitants, we were unable to compile this information even in its simplest form, so that, until the census of 1890, we never even knew the proportion of our population that was single, married, and widowed. Again, while we classed our population as native white, foreign white, and coloured, this was extremely unsatis-

RADIX-SORT(A, d)

```
RADIX-SORT(A, d)
1  for i = 1 to d
```

```
RADIX-SORT(A, d)
1  for i = 1 to d
2      sort* A by digit d
```

```
RADIX-SORT(A, d)
1  for i = 1 to d
2      sort* A by digit d
```

*Må være stabil

RADIX-SORT(A, d)
1 **for** $i = 1$ **to** d
2 sort* A by digit d

*Må være stabil†

```
RADIX-SORT(A, d)
1  for i = 1 to d
2      sort* A by digit d
```

*Må være stabil†

†Må ikke bytte om like verdier

```
RADIX-SORT(A, d)
1  for i = 1 to d
2      sort* A by digit d
```

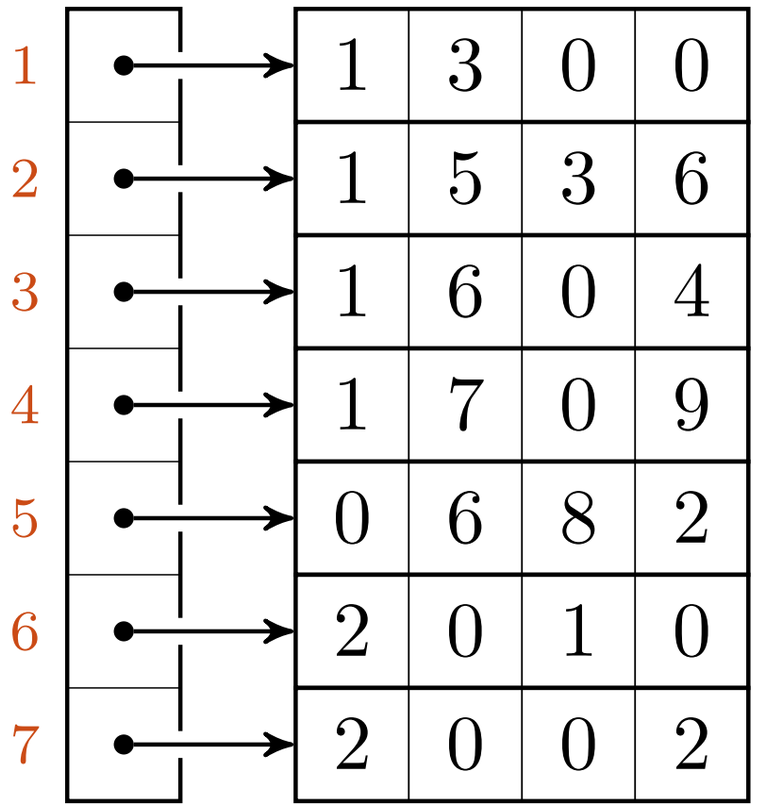
*Må være stabil†

†Må ikke bytte om like verdier

(Bruk f.eks. COUNTING-SORT)

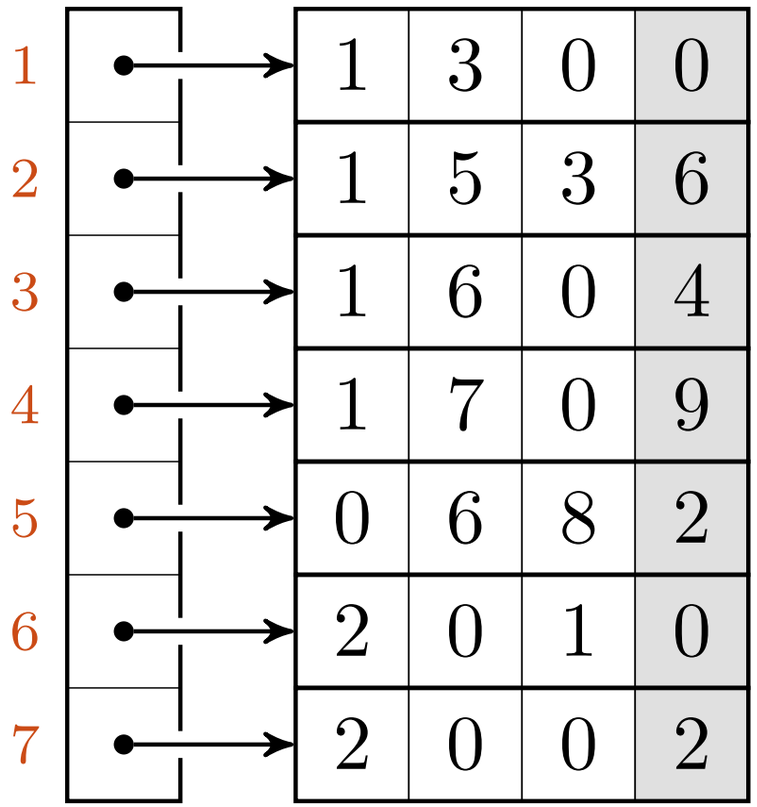
RADIX-SORT(A, d)

1 **for** $i = 1$ **to** d
2 sort A on digit d



i = –

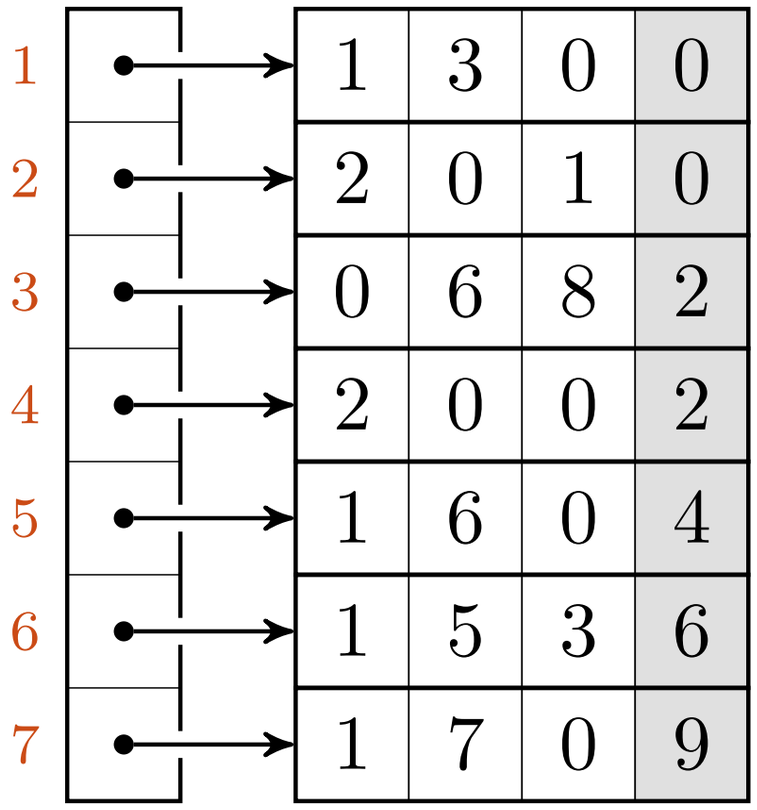
RADIX-SORT(A, d)
1 **for** $i = 1$ **to** d
2 sort A on digit d



$i = 4$

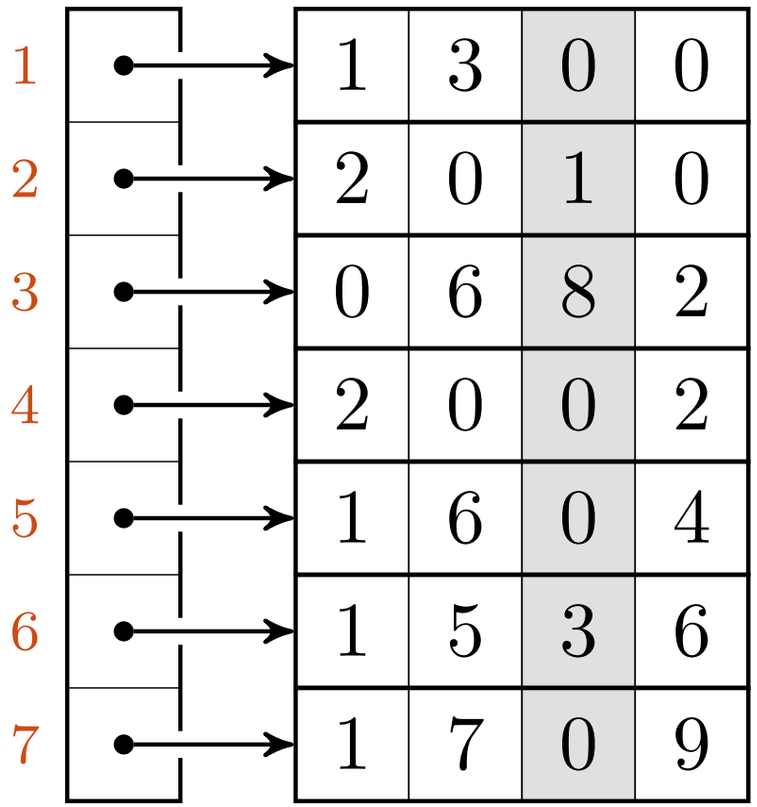
RADIX-SORT(A, d)

1 **for** $i = 1$ **to** d
2 sort A on digit d



$i = 4$

RADIX-SORT(A, d)
1 **for** $i = 1$ **to** d
2 sort A on digit d

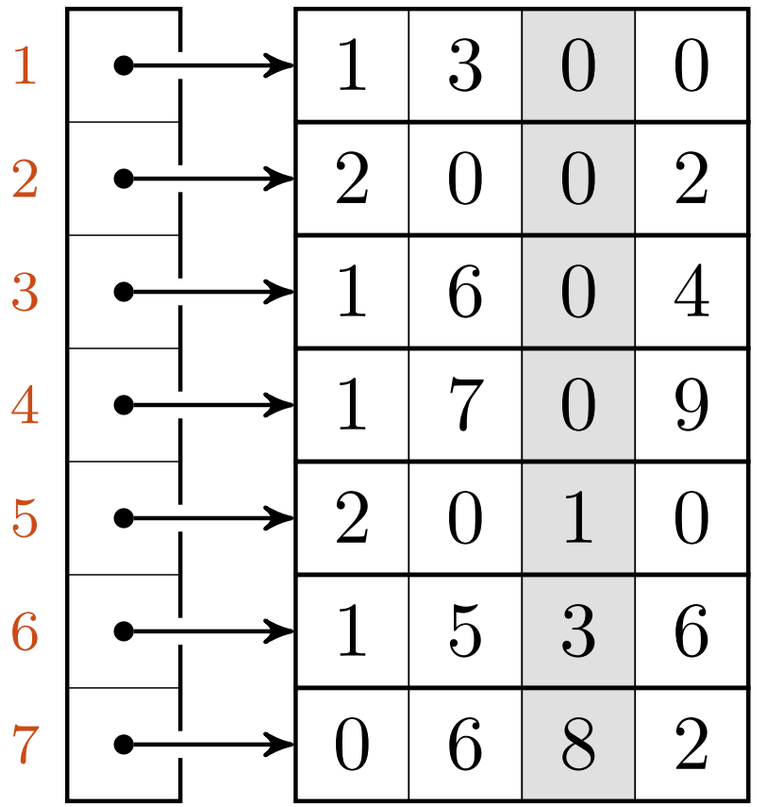


$i = 3$

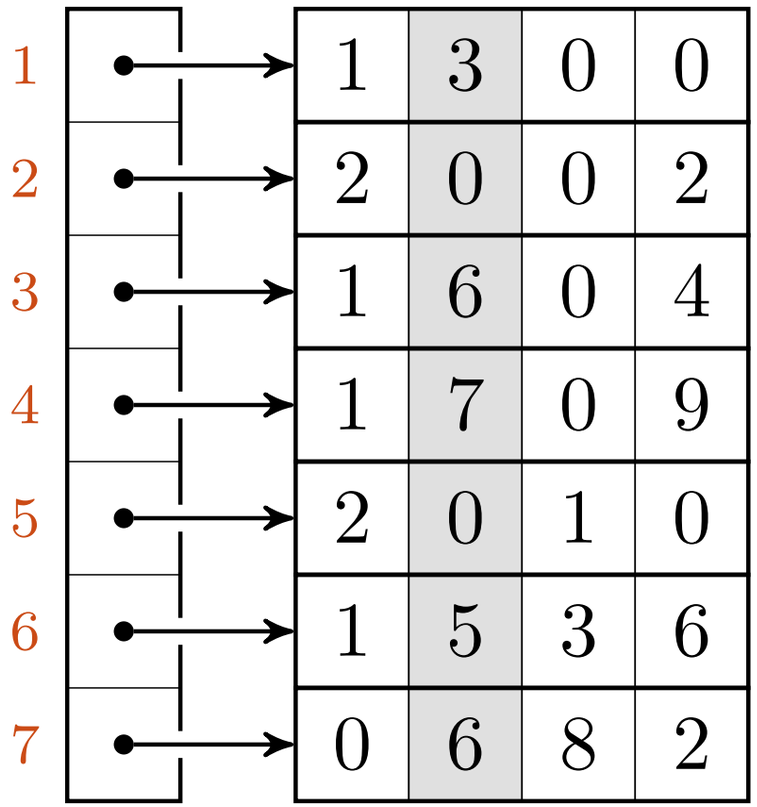
RADIX-SORT(A, d)

1 **for** $i = 1$ **to** d
2 sort A on digit d

$i = 3$



RADIX-SORT(A, d)
1 **for** $i = 1$ **to** d
2 sort A on digit d

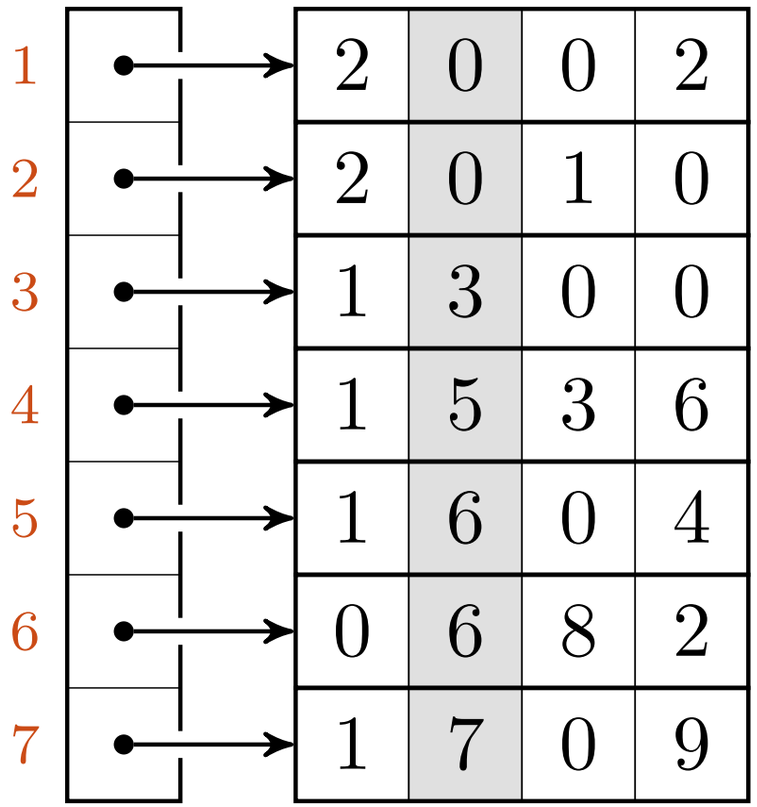


$i = 2$

RADIX-SORT(A, d)

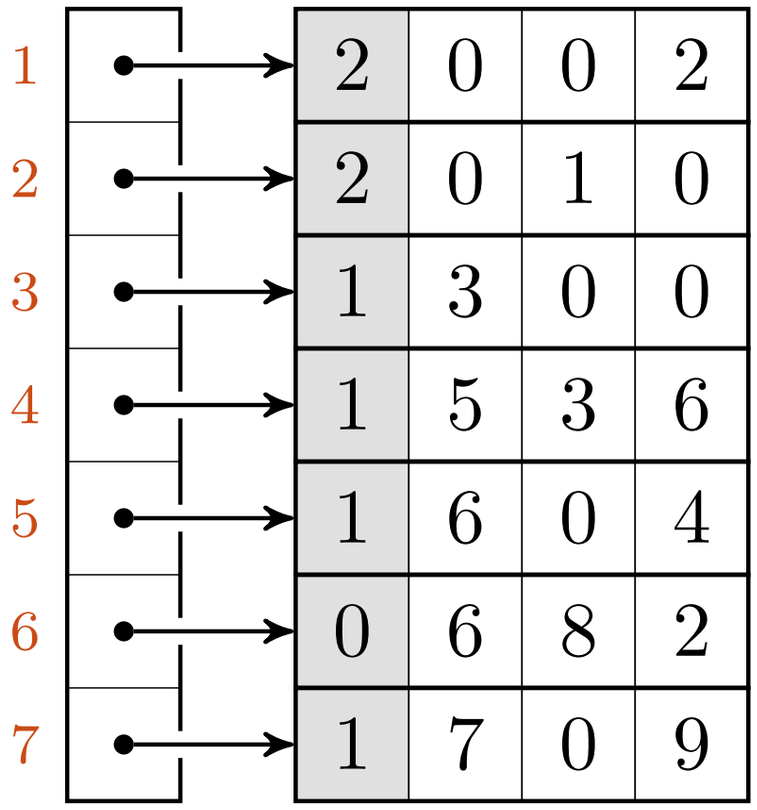
1 **for** $i = 1$ **to** d
2 sort A on digit d

$i = 2$

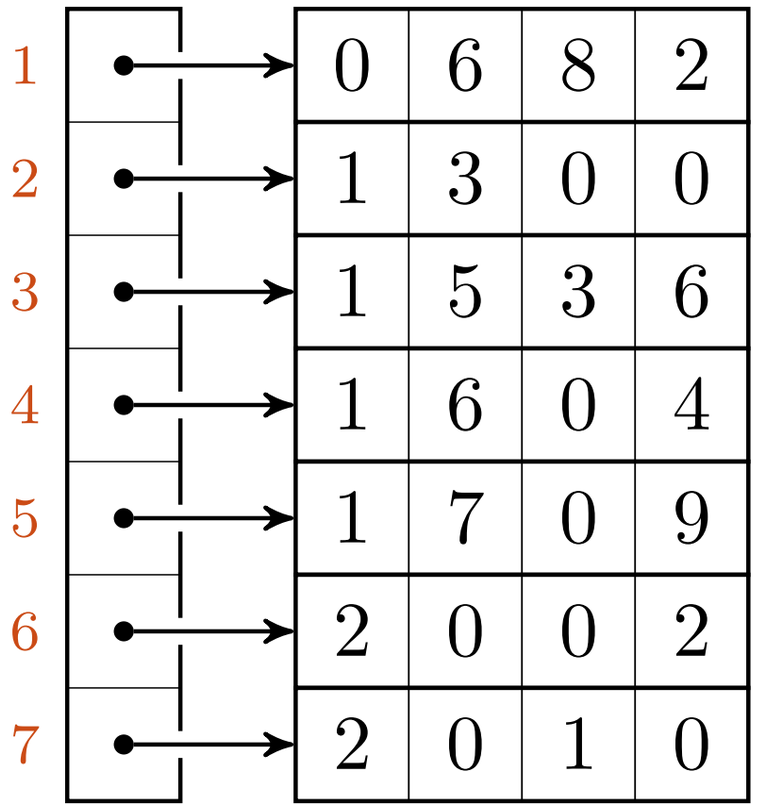


RADIX-SORT(A, d)
1 **for** $i = 1$ **to** d
2 sort A on digit d

$i = 1$

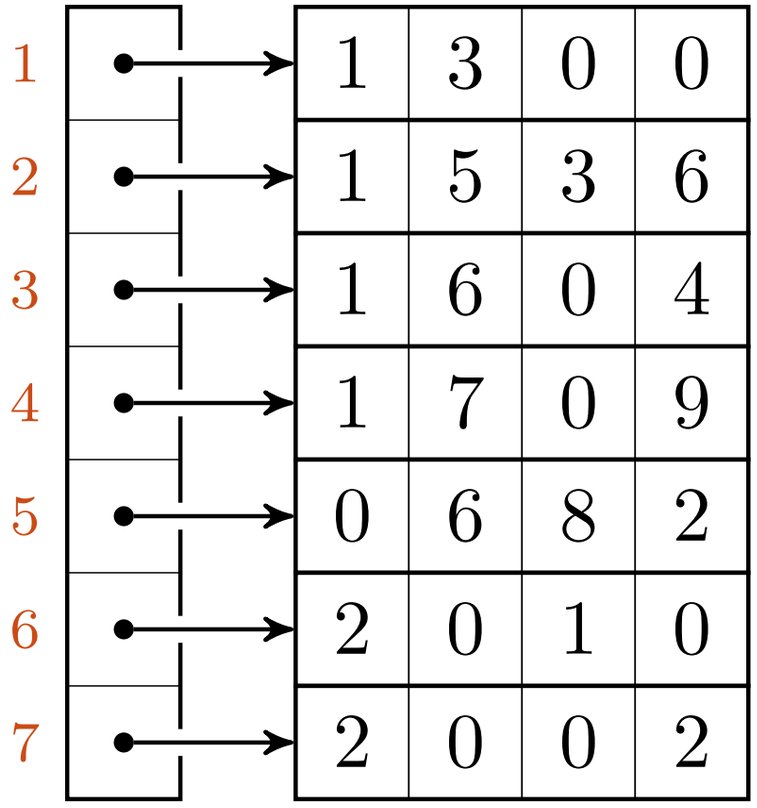


RADIX-SORT(A, d)
1 **for** $i = 1$ **to** d
2 sort A on digit d



i = –

Det *finnes* varianter som går denne veien også – men de gjør andre ting *i tillegg*. Så denne rett-frem-varianten er rett og slett feil.



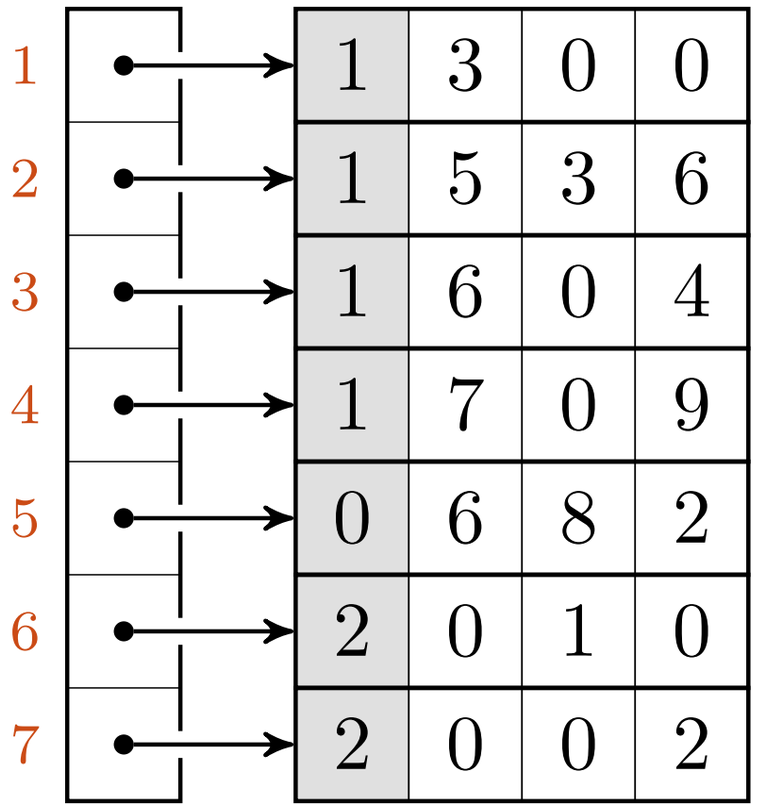
XIDAR-NON-SORT(A, d)

```
1  for i = d downto 1
2      sort A on digit d
```

i = –

XIDAR-NON-SORT(A, d)
1 **for** $i = d$ **downto** 1
2 sort A on digit d

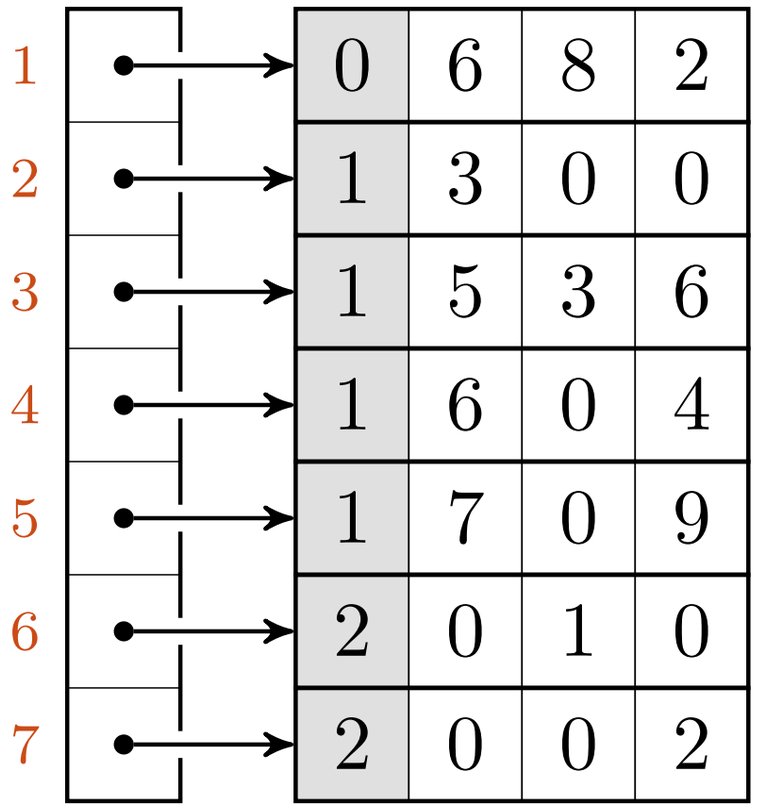
$i = 1$



XIDAR-NON-SORT(A, d)

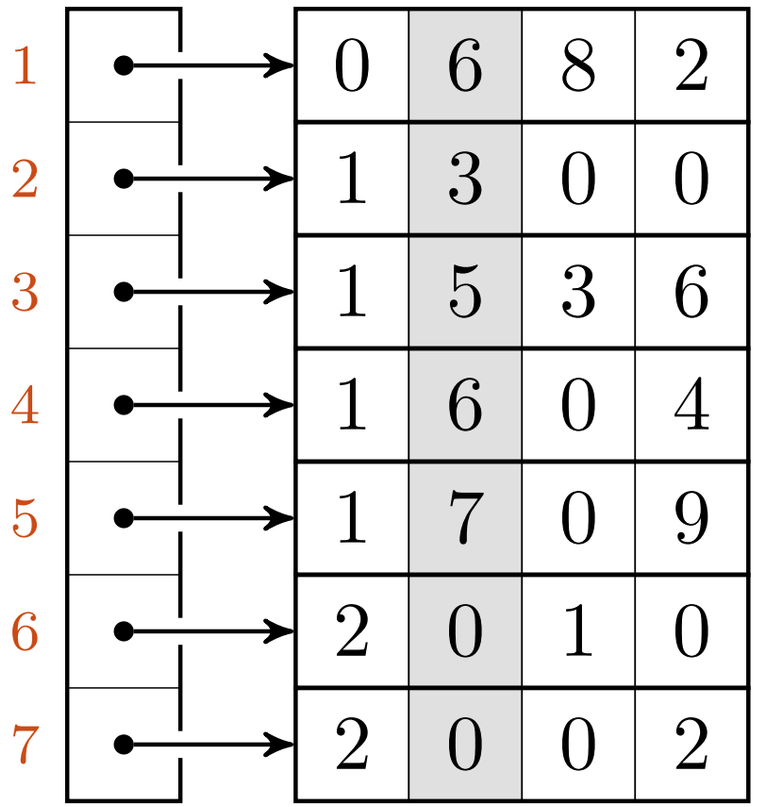
1 **for** $i = d$ **downto** 1
2 sort A on digit d

$i = 1$



XIDAR-NON-SORT(A, d)
1 **for** $i = d$ **downto** 1
2 sort A on digit d

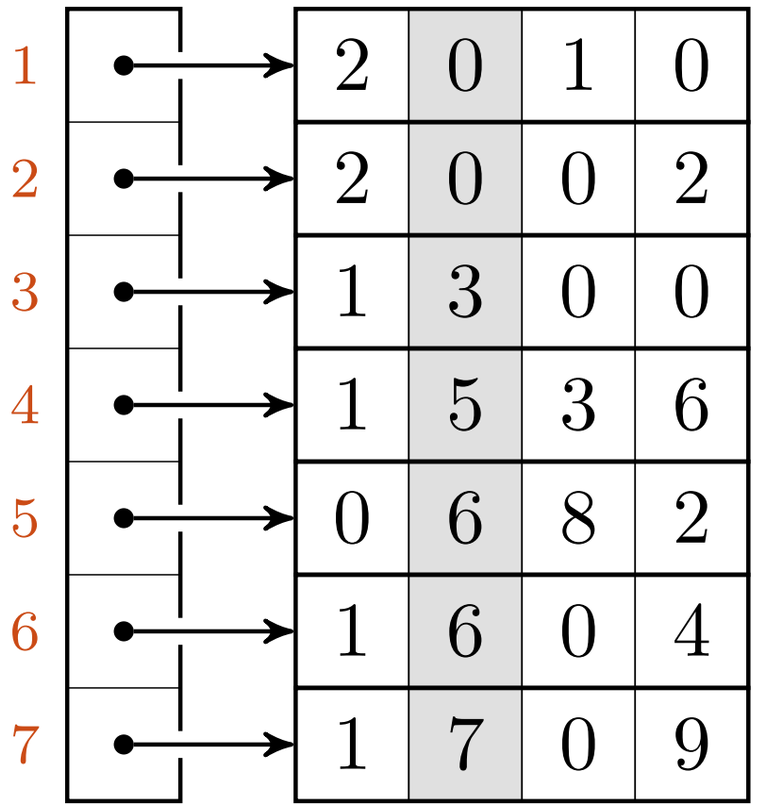
$i = 2$



XIDAR-NON-SORT(A, d)

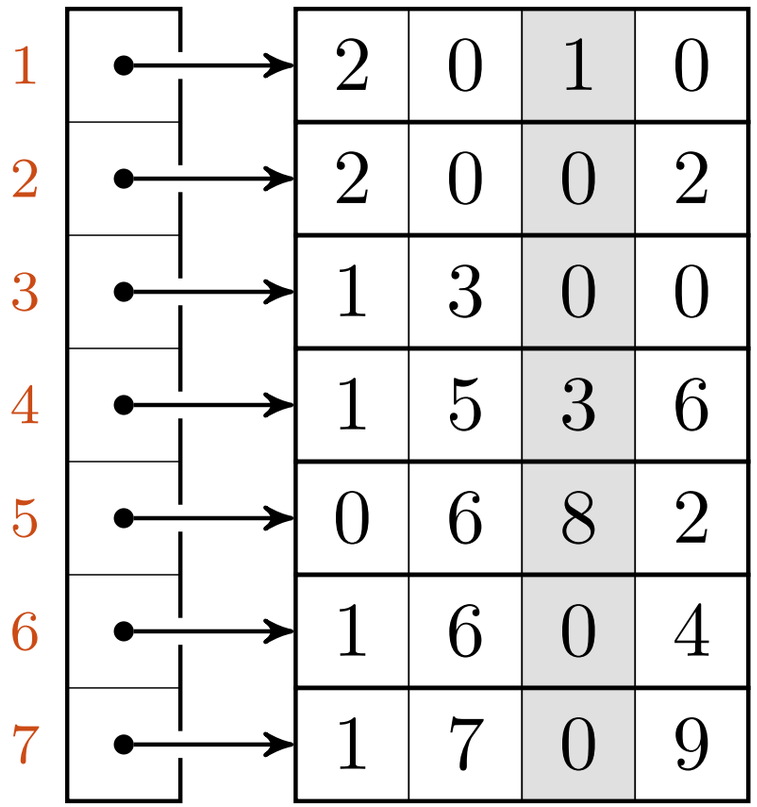
1 **for** $i = d$ **downto** 1
2 sort A on digit d

$i = 2$



XIDAR-NON-SORT(A, d)
1 **for** $i = d$ **downto** 1
2 sort A on digit d

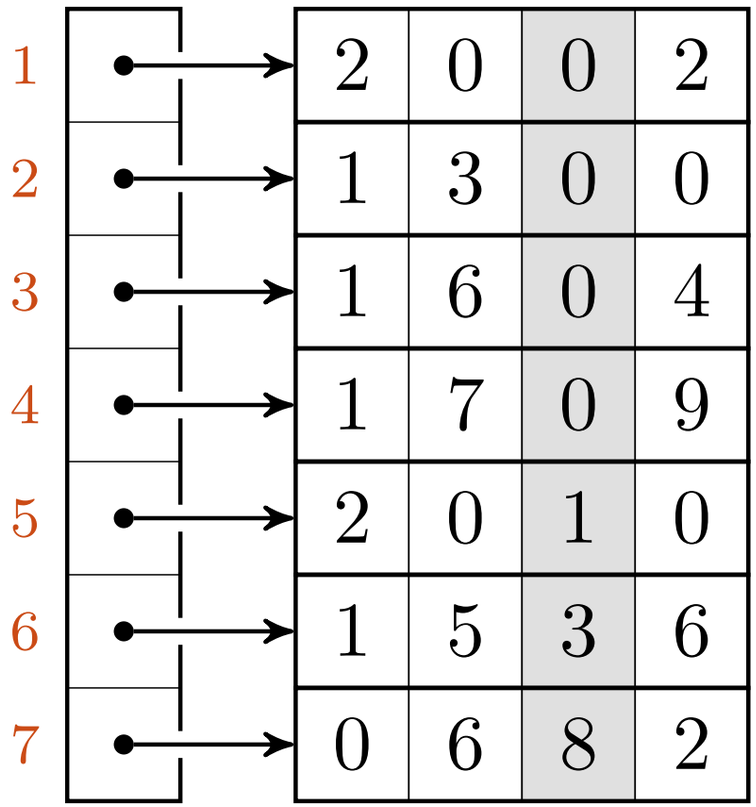
$i = 3$



XIDAR-NON-SORT(A, d)

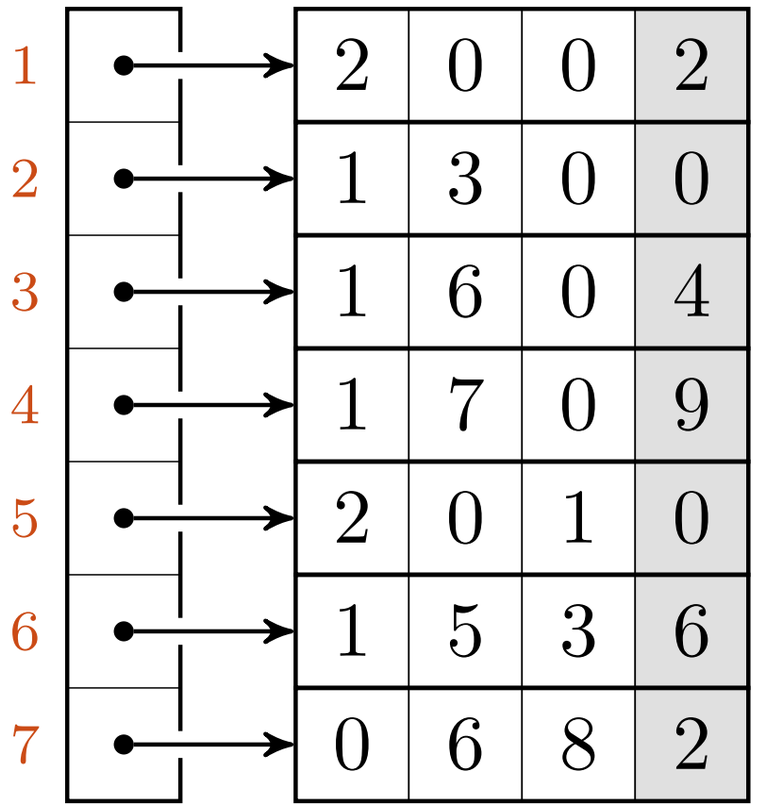
1 **for** $i = d$ **downto** 1
2 sort A on digit d

$i = 3$



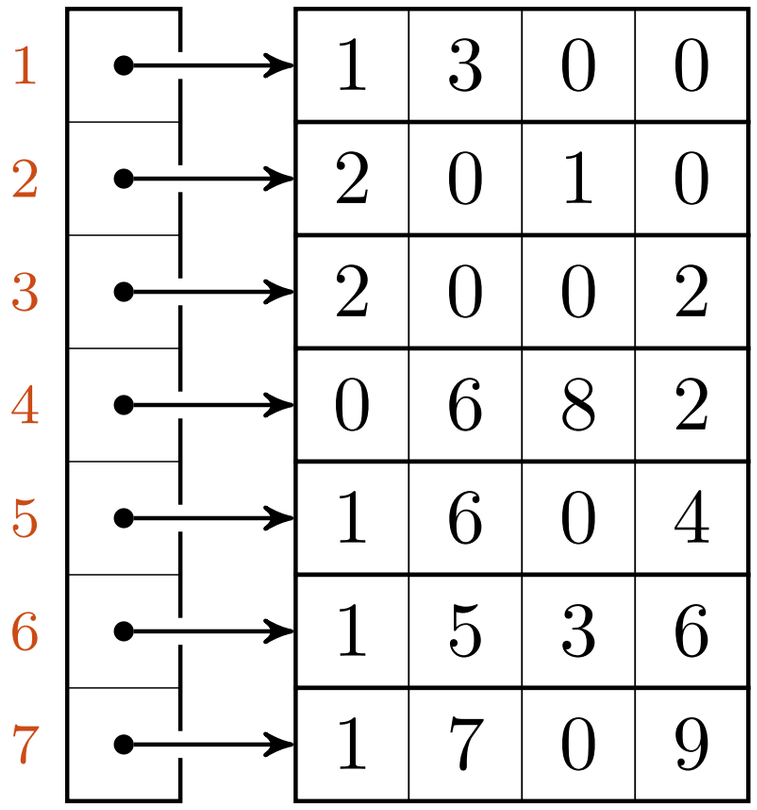
XIDAR-NON-SORT(A, d)
1 **for** $i = d$ **downto** 1
2 sort A on digit d

$i = 4$



XIDAR-NON-SORT(A, d)
1 **for** $i = d$ **downto** 1
2 sort A on digit d

i = –



$$\mathrm{T}(n) = \Theta(d \cdot (n + k))$$

Kjøretiden er altså avhengig av antall verdier (n), antall mulige verdier for hvert siffer (k) og antall siffer (d).

COUNTING-SORT (kjøretid $\Theta(n + k)$) for hvert av d siffer

Bryt grensen

… denne gang for AC

Ingen garantier, men antar at inputs er uniformt fordelte tilfeldige tall i intervallet [0,1).

4:6

Bøttesortering

Sorting by Address Calculation*

E. J. Isaac and R. C. Singleton

Stanford Research Institute, Menlo Park, California

General Description of Method

Sorting in a random access memory is essentially a process of associating the address of the location in which each item is to be placed with the identifying key of the item. The fewer times the items have to be moved from one location

BUCKET-SORT(A)

A input

Verdier i A: Uniformt fordelte, $[0, 1)$

```
BUCKET-SORT(A)
1  n = A.length
```

A input
n lengde

Antall bøtter; ett element per bøtte

```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
```

A input
n lengde
B bøtter

Hodepekere til bøttene (lister)

```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
```

A input
n lengde
B bøtter
i bøttenr

```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
```

A input
n lengde
B bøtter
i bøttenr

(Evt. bruk dynamiske tabeller)

```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
```

A input
n lengde
B bøtter
i bøttenr

```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
```

A input
n lengde
B bøtter
i bøttenr

«Ryddig hashing»; kan regne ut rett bøtte direkte

```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
```

A input
n lengde
B bøtter
i bøttenr

```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
```

A input
n lengde
B bøtter
i bøttenr

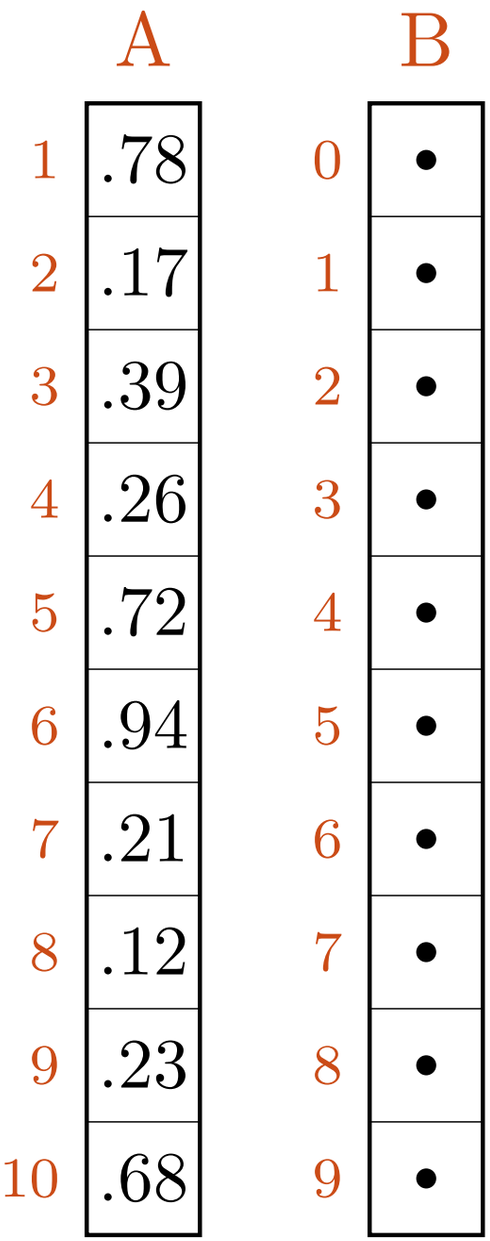
Bruk f.eks. INSERTION-SORT. Forventet: $\mathrm{B}[i].length = \mathrm{O}(1)$

```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```

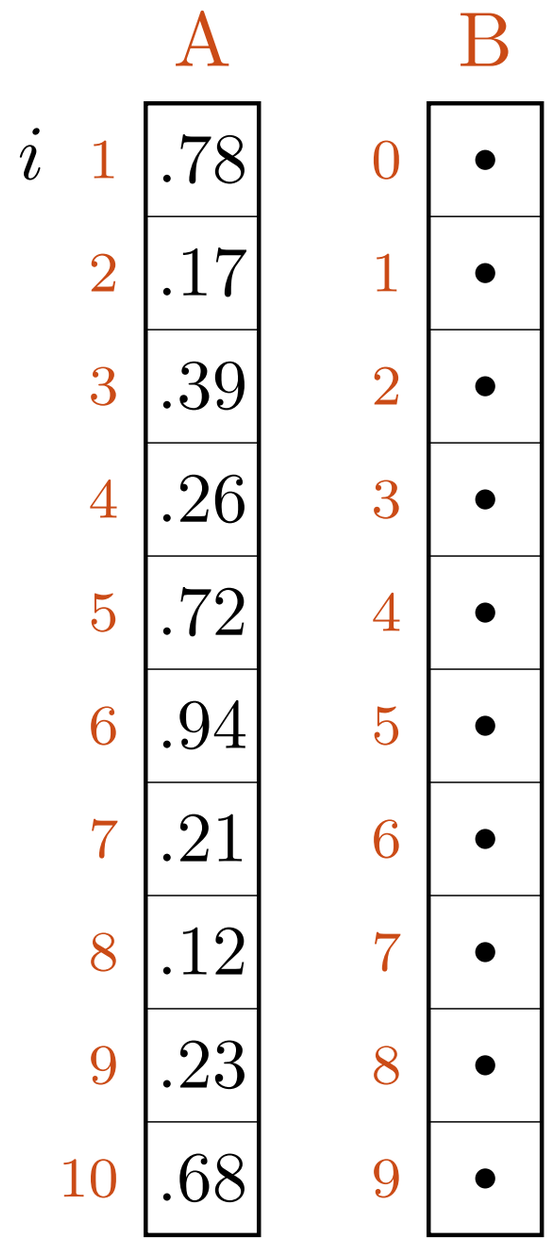
A input
n lengde
B bøtter
i bøttenr

Ferdig sorterte, i rett rekkefølge

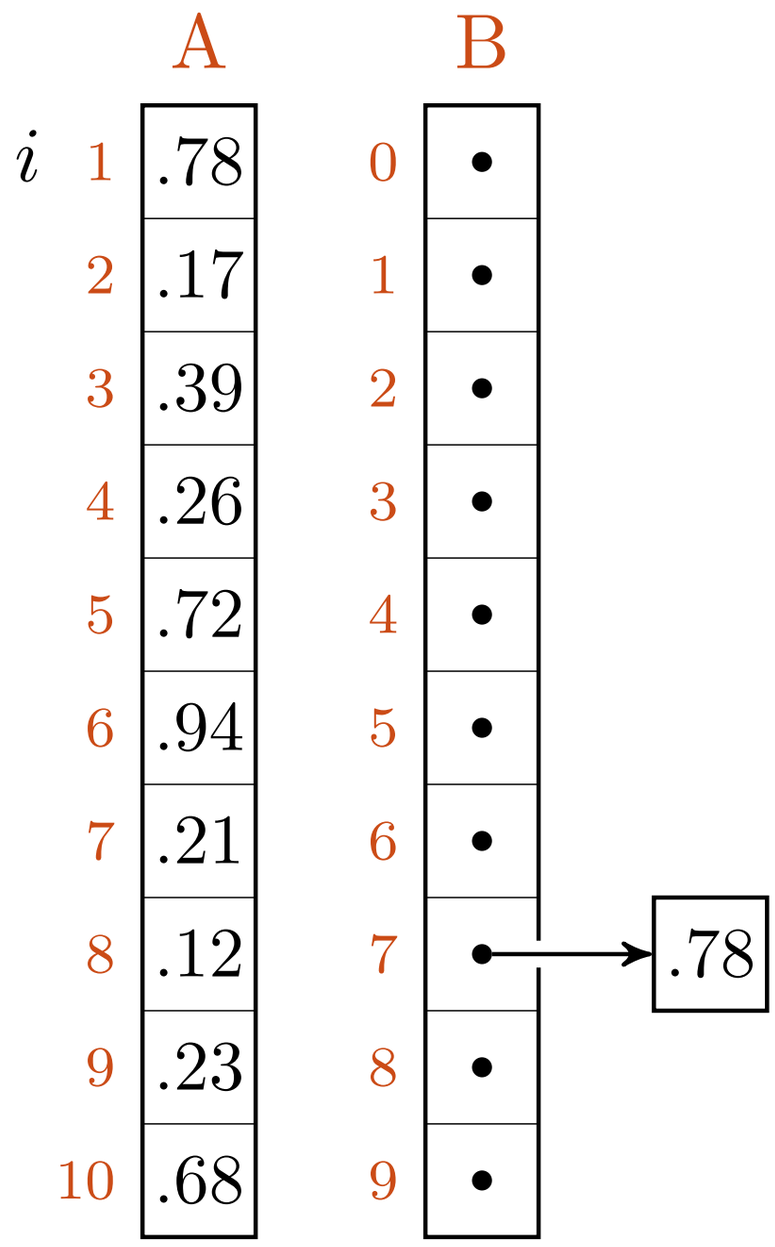
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



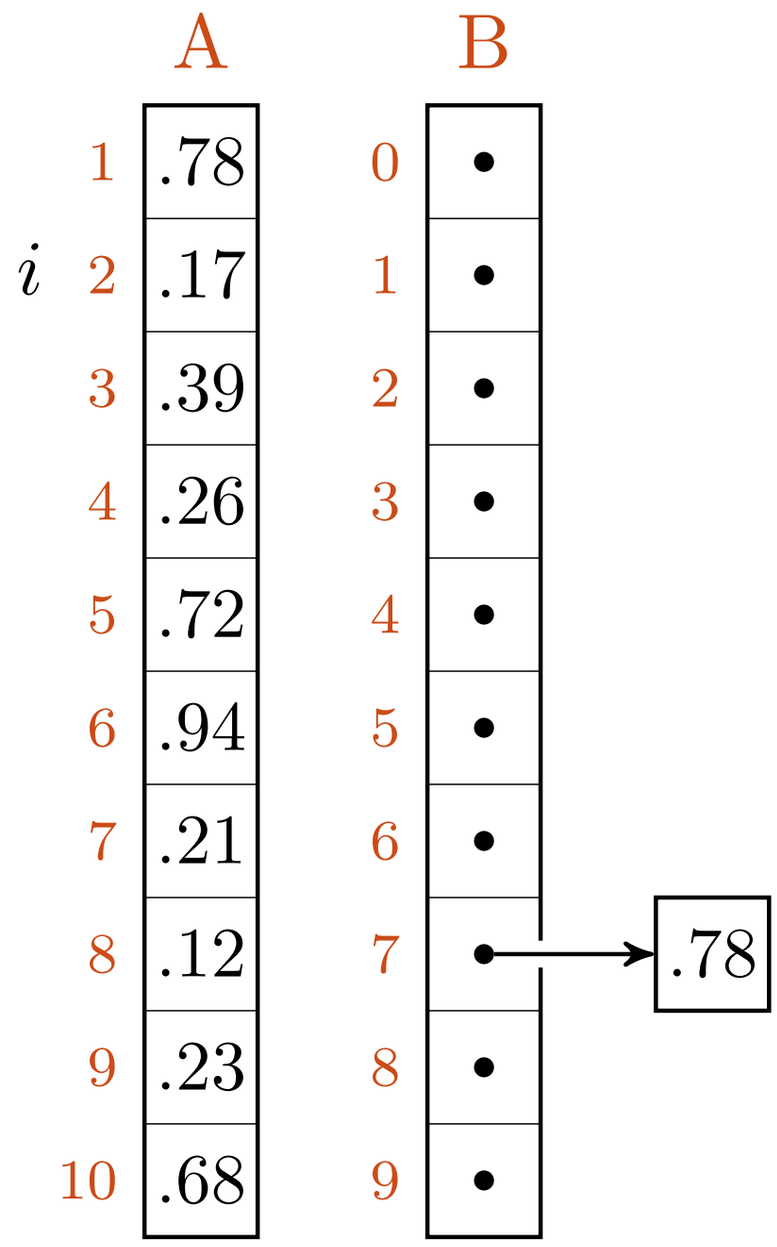
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



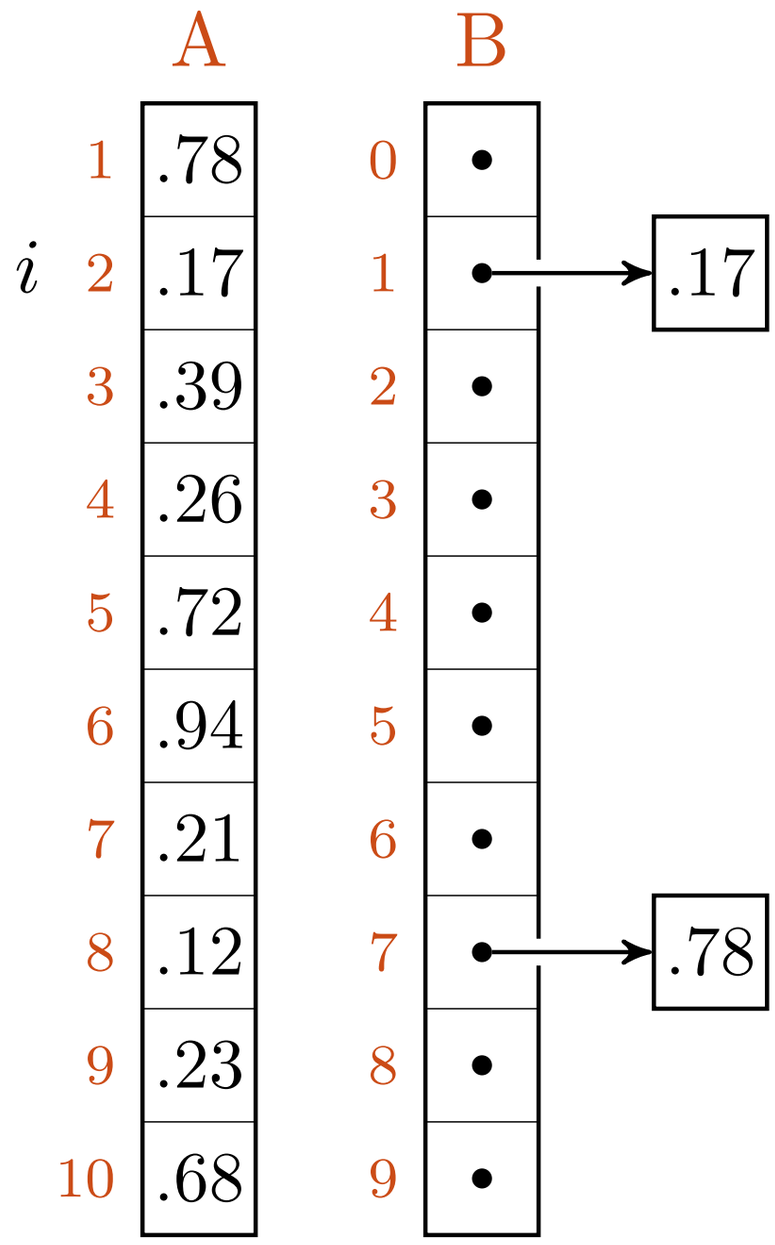
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



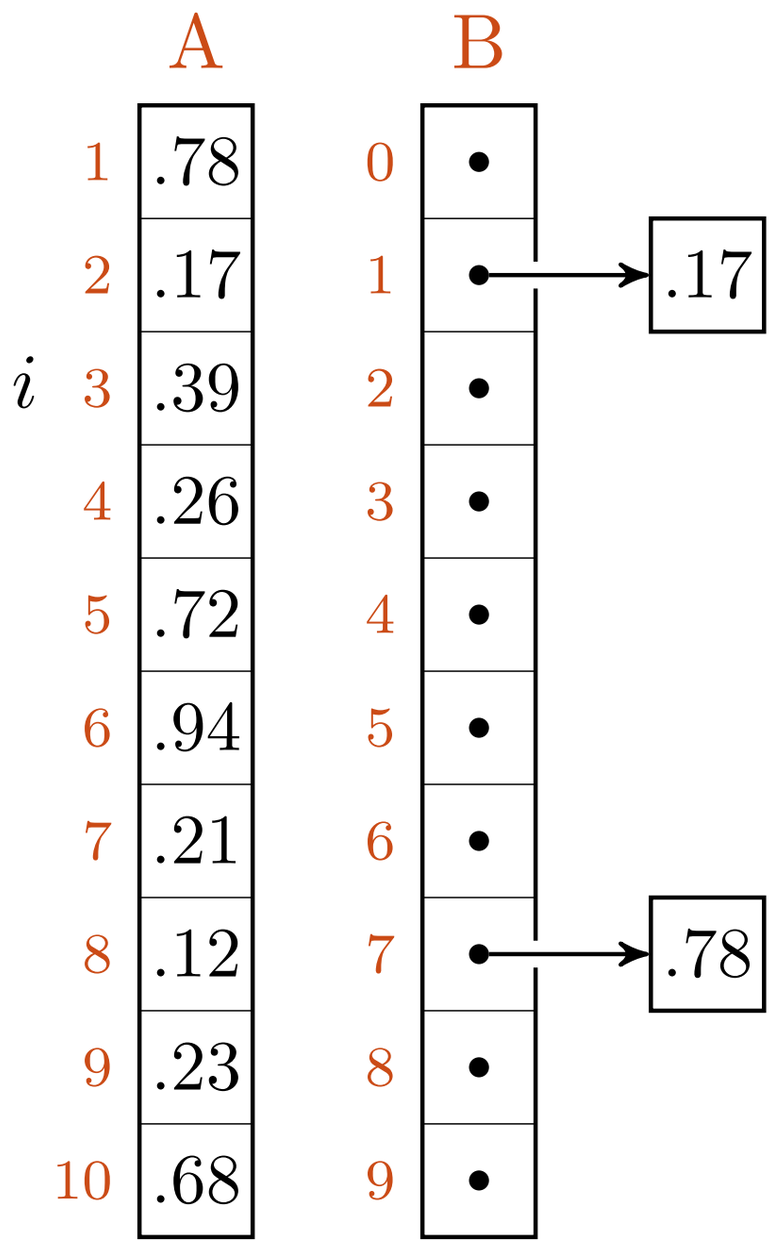
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



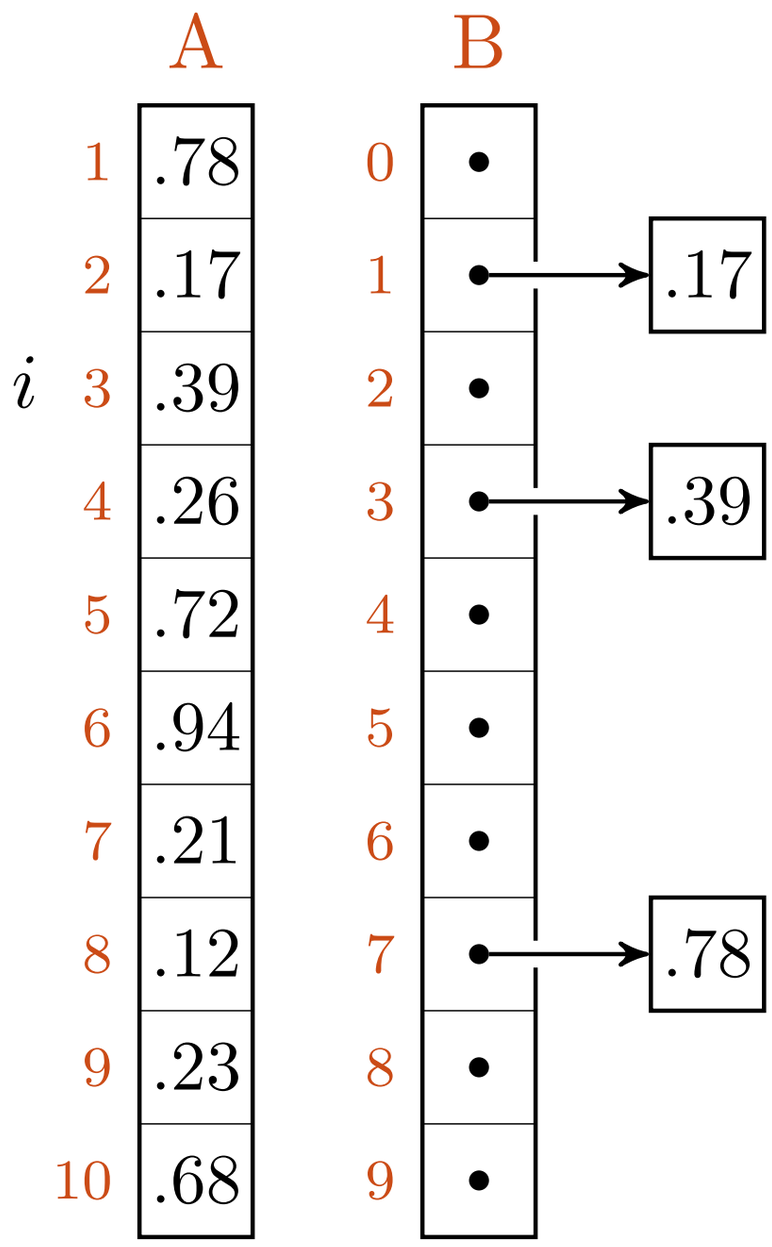
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



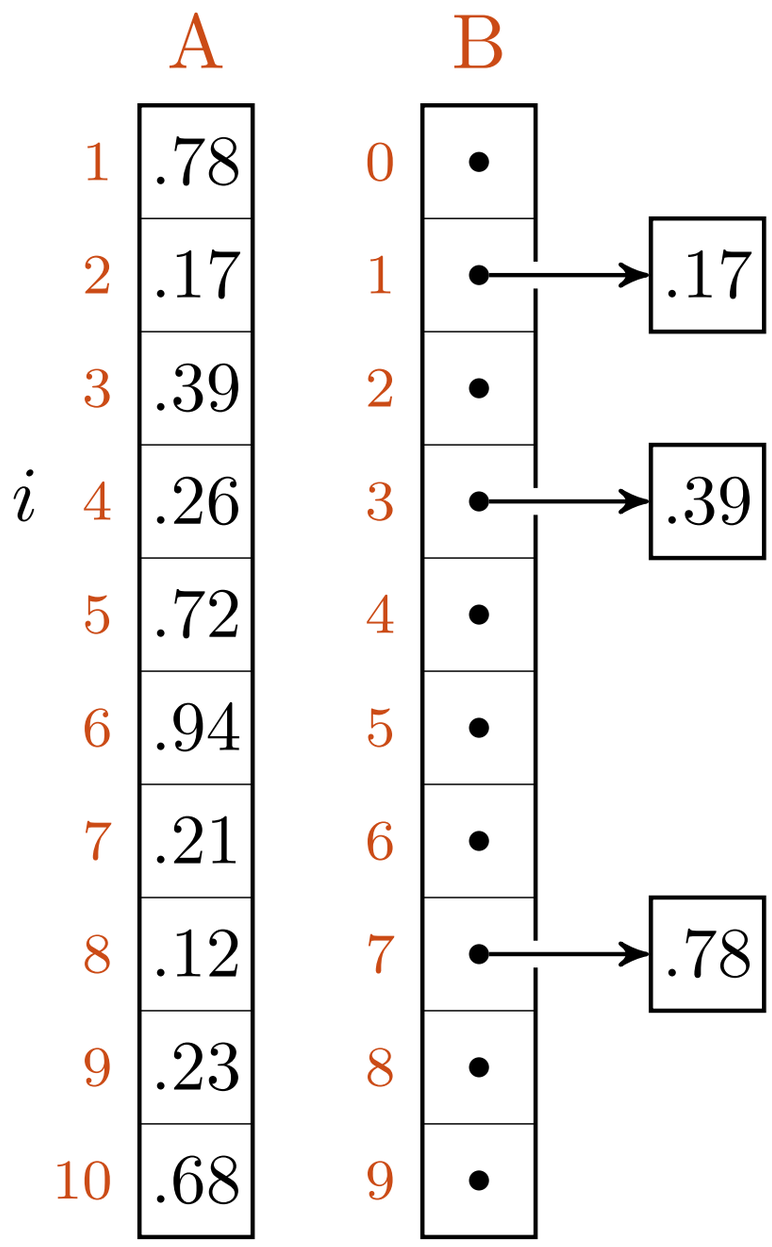
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



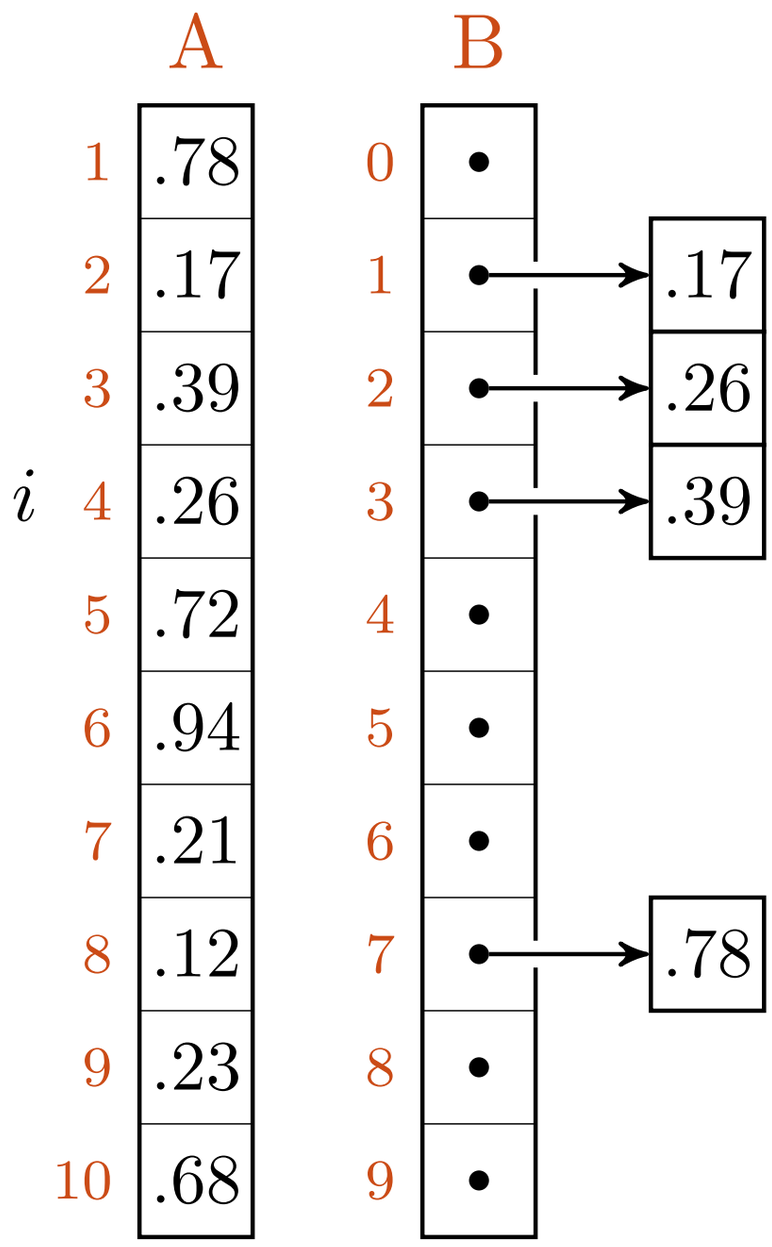
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



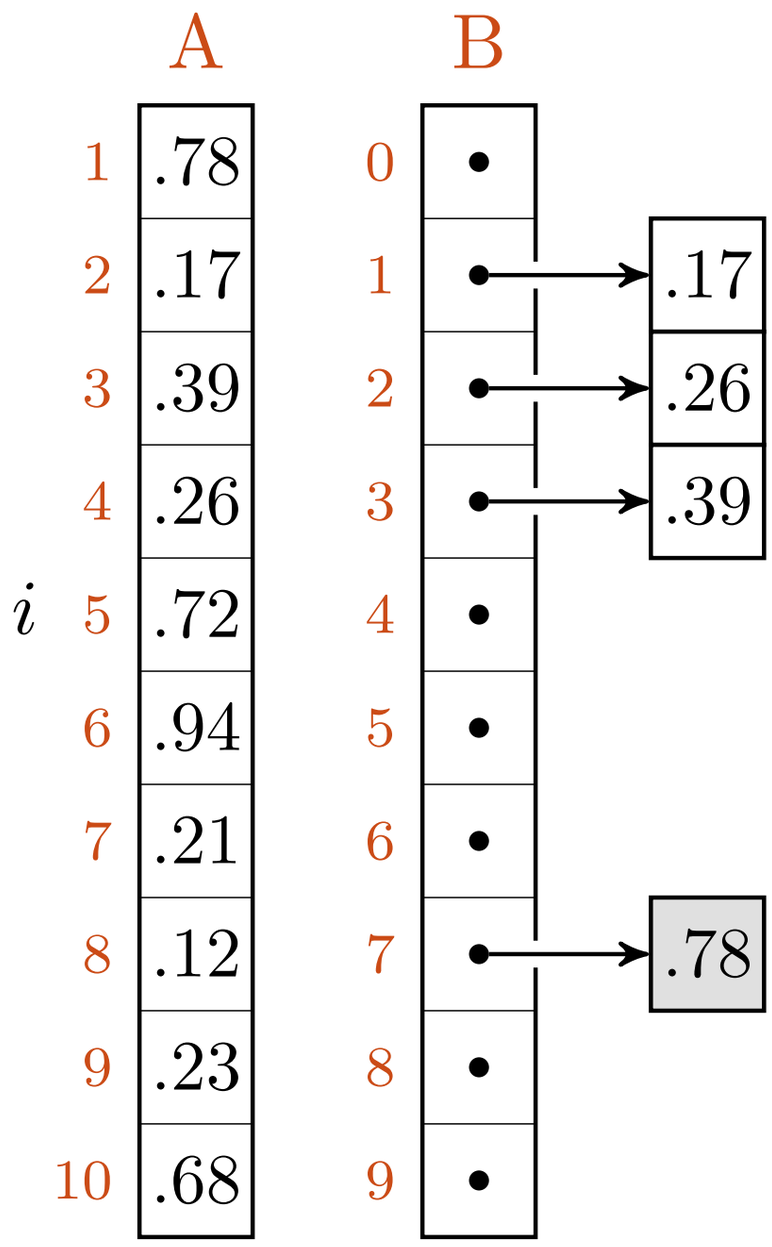
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



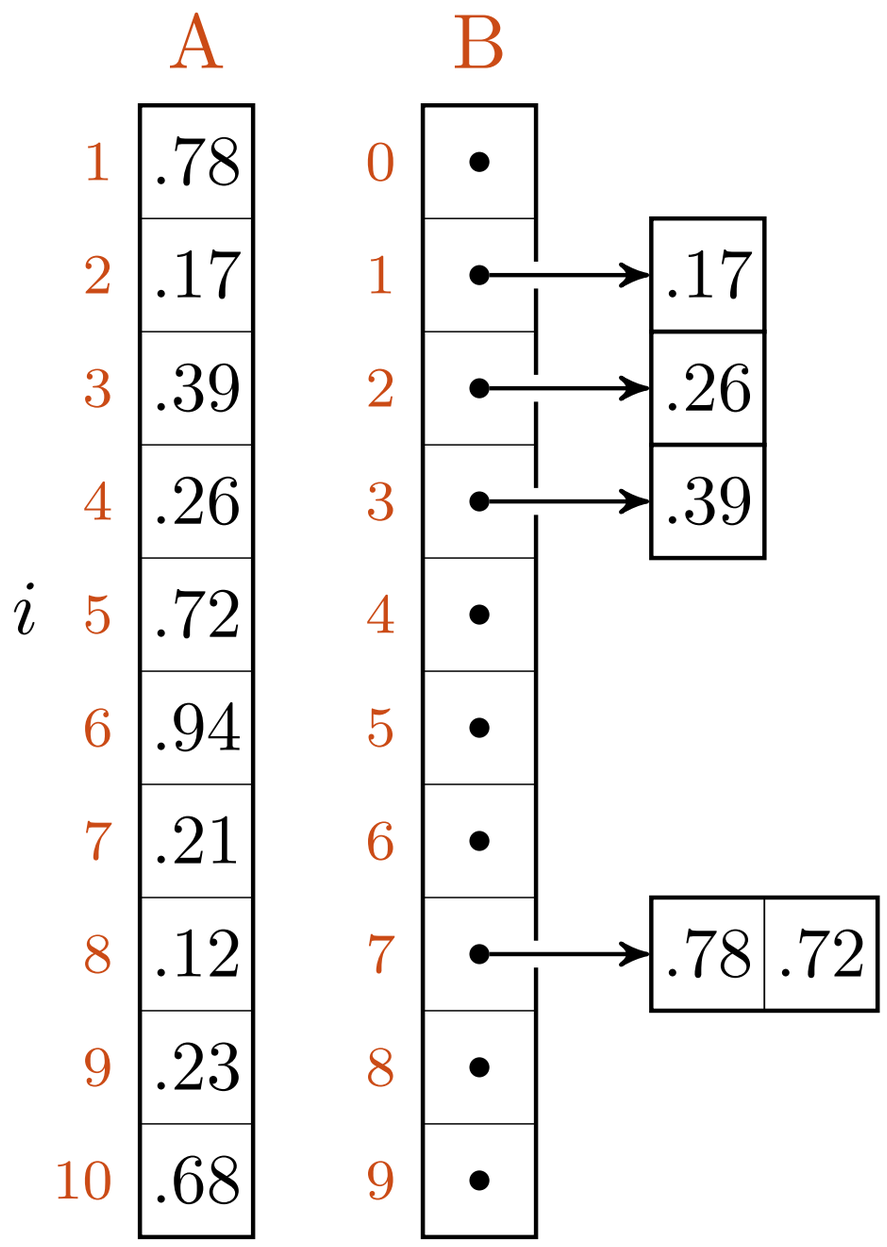
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



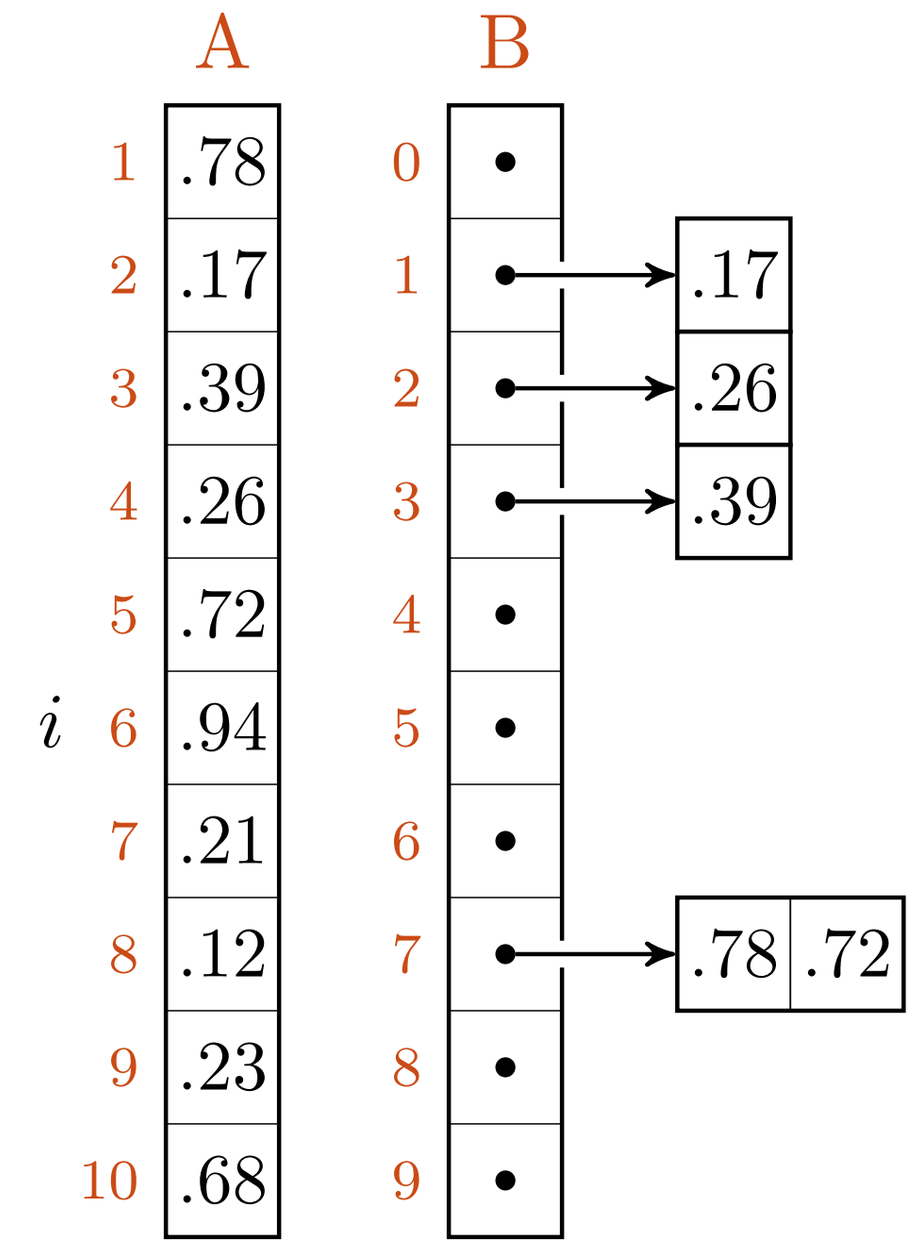
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



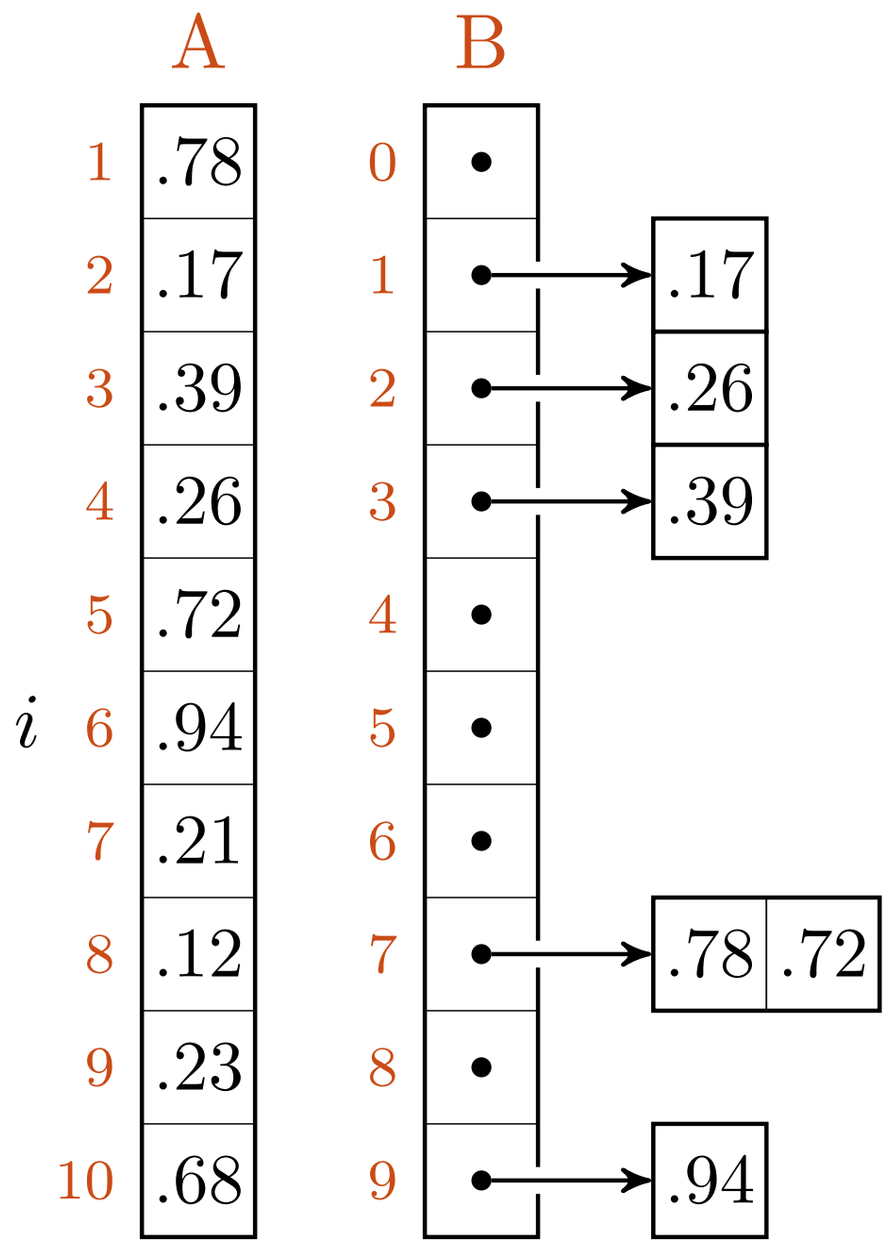
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



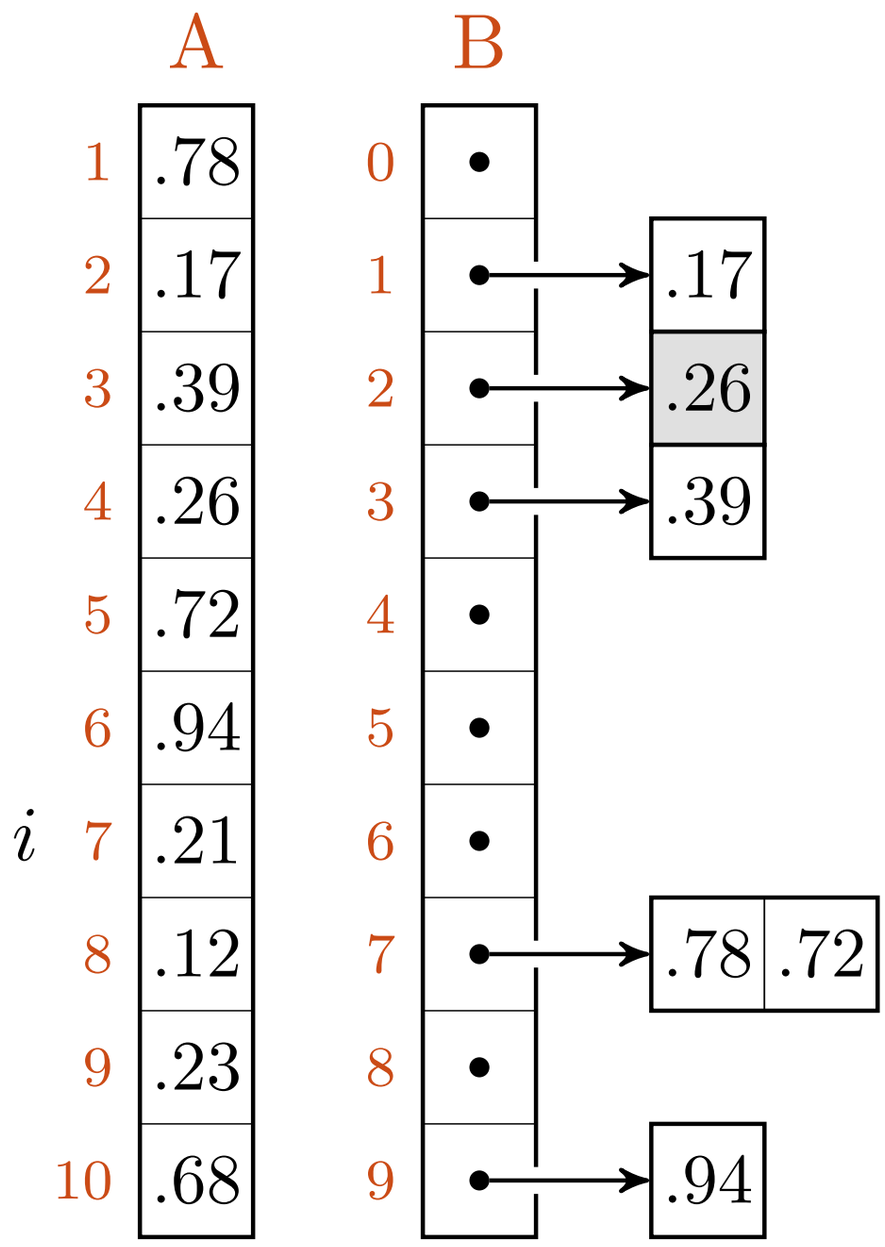
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



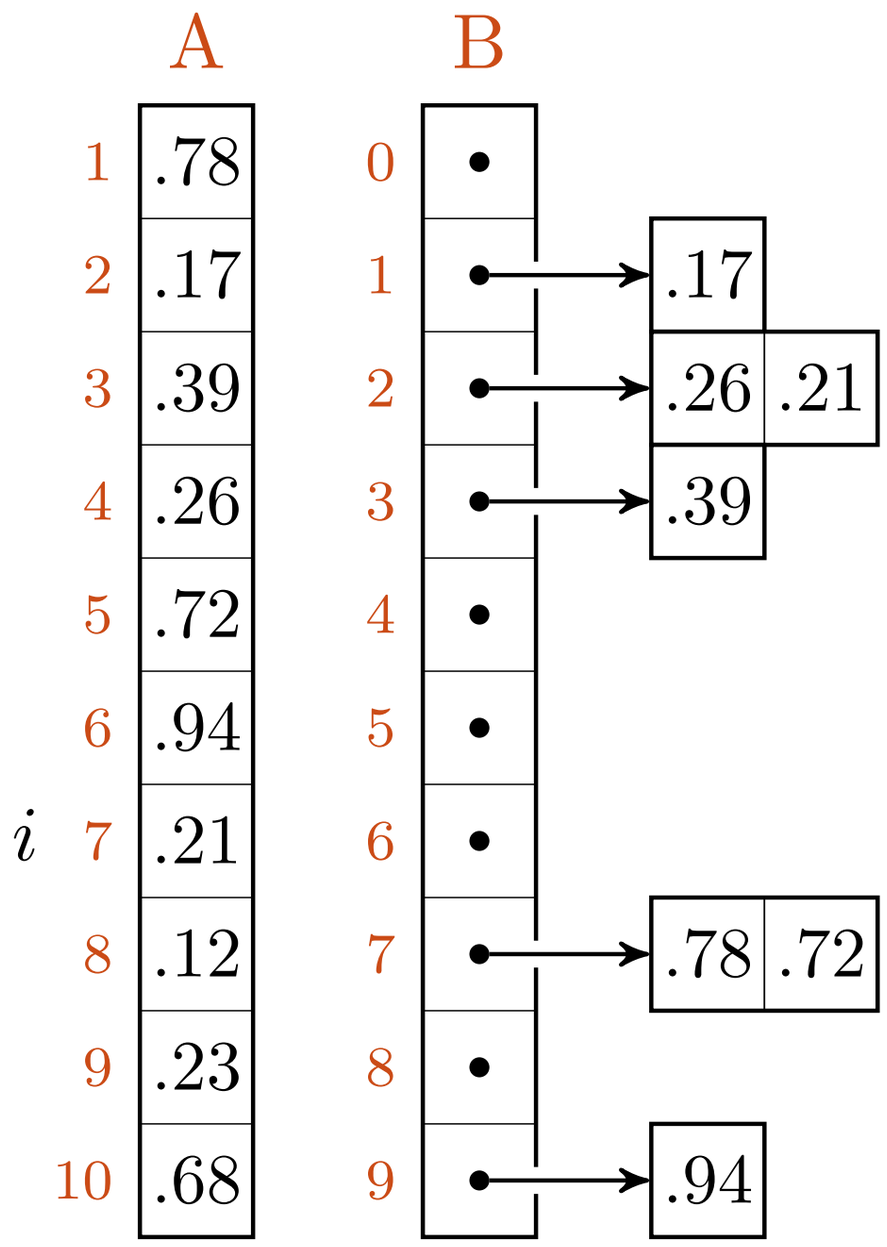
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



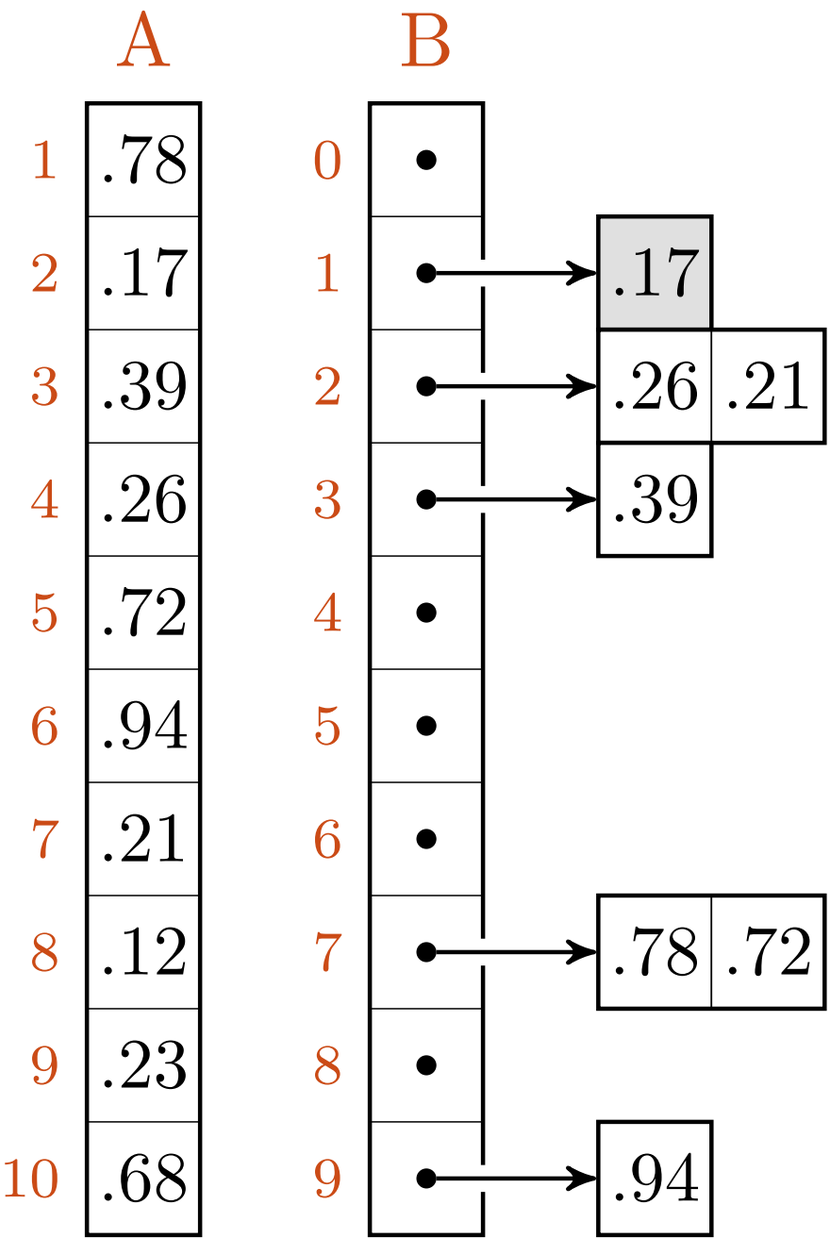
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



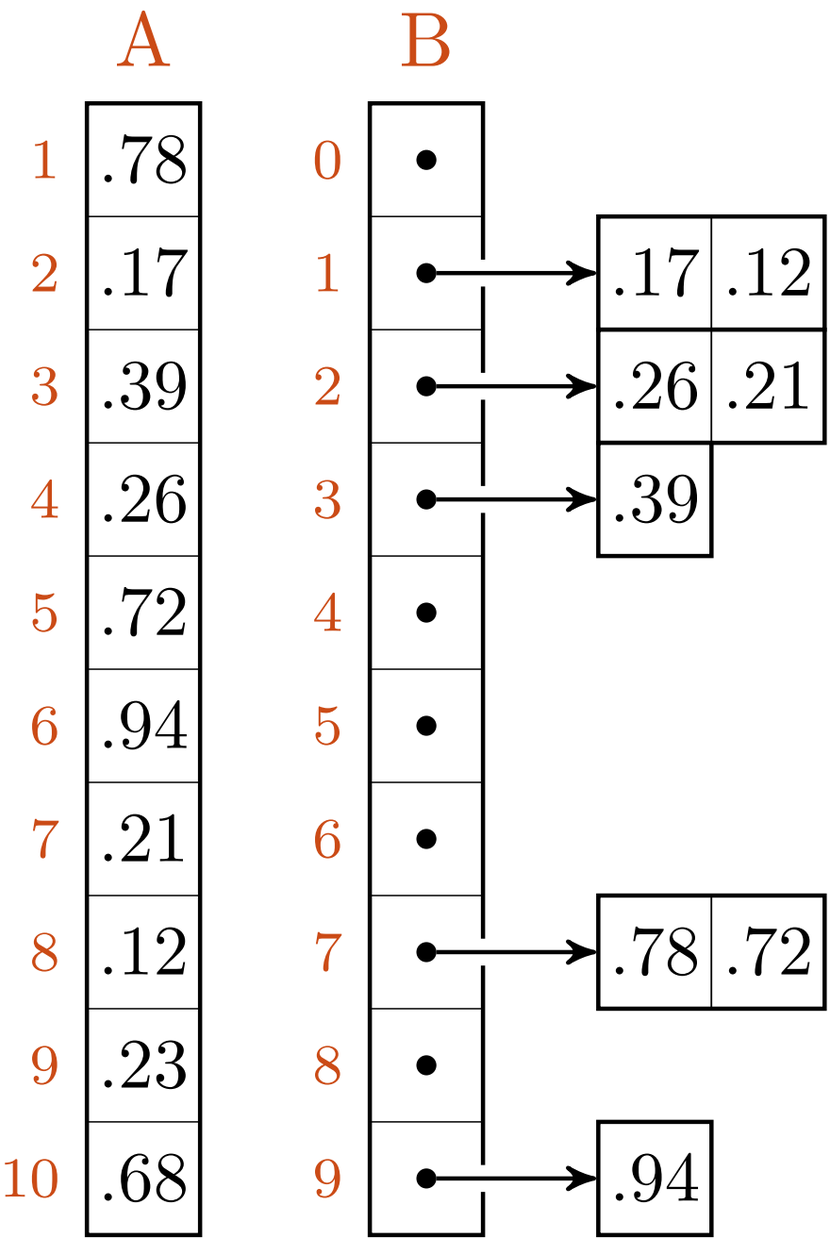
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



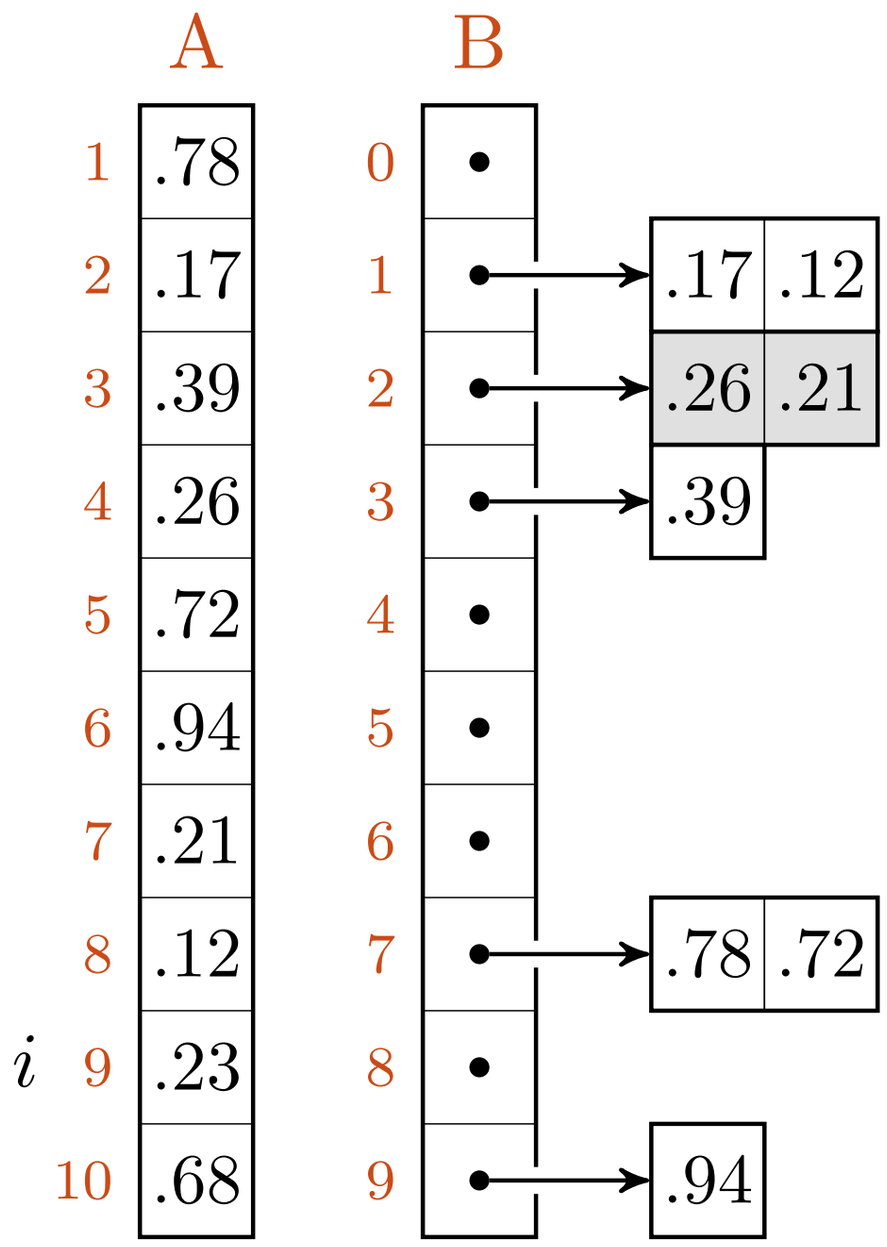
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



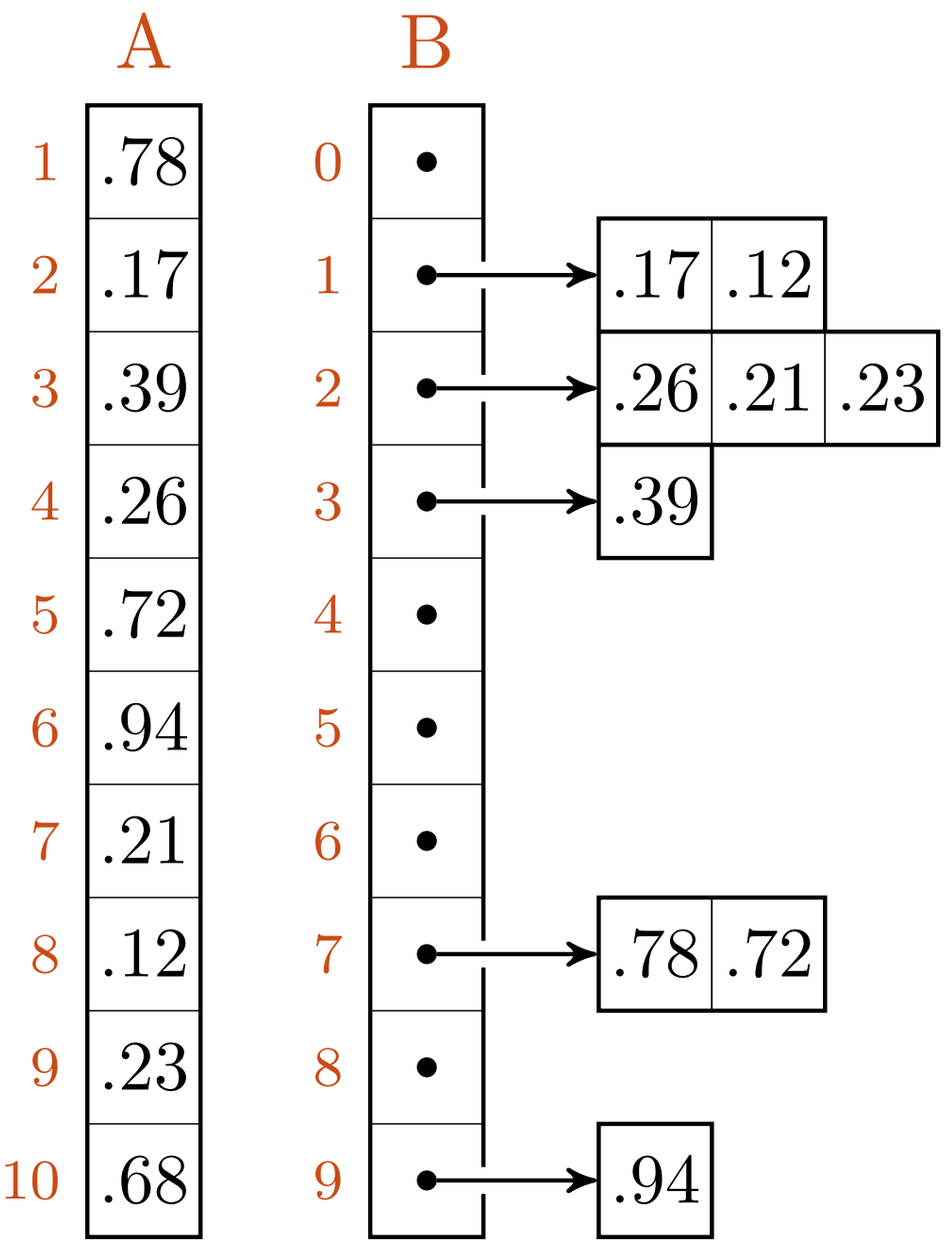
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



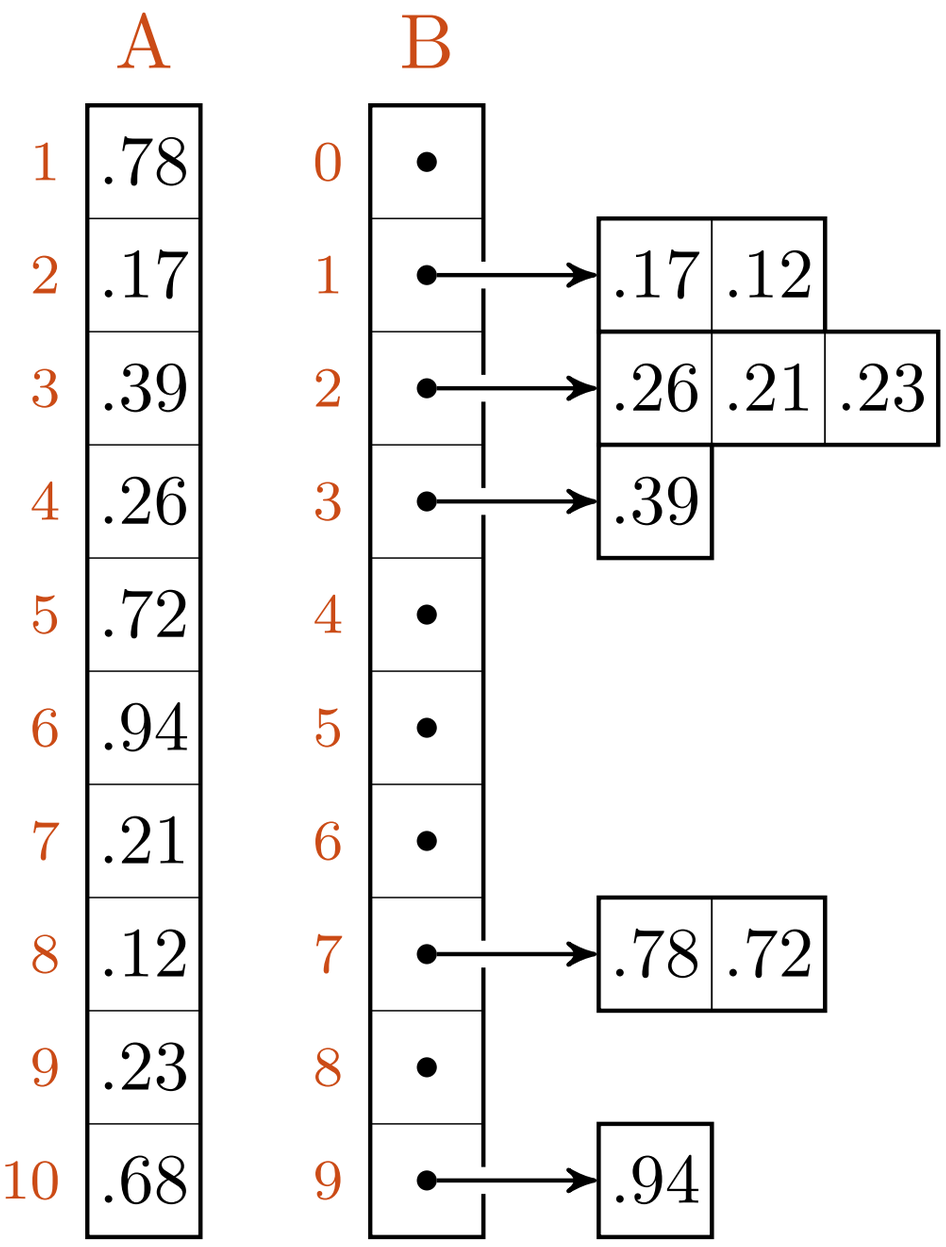
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



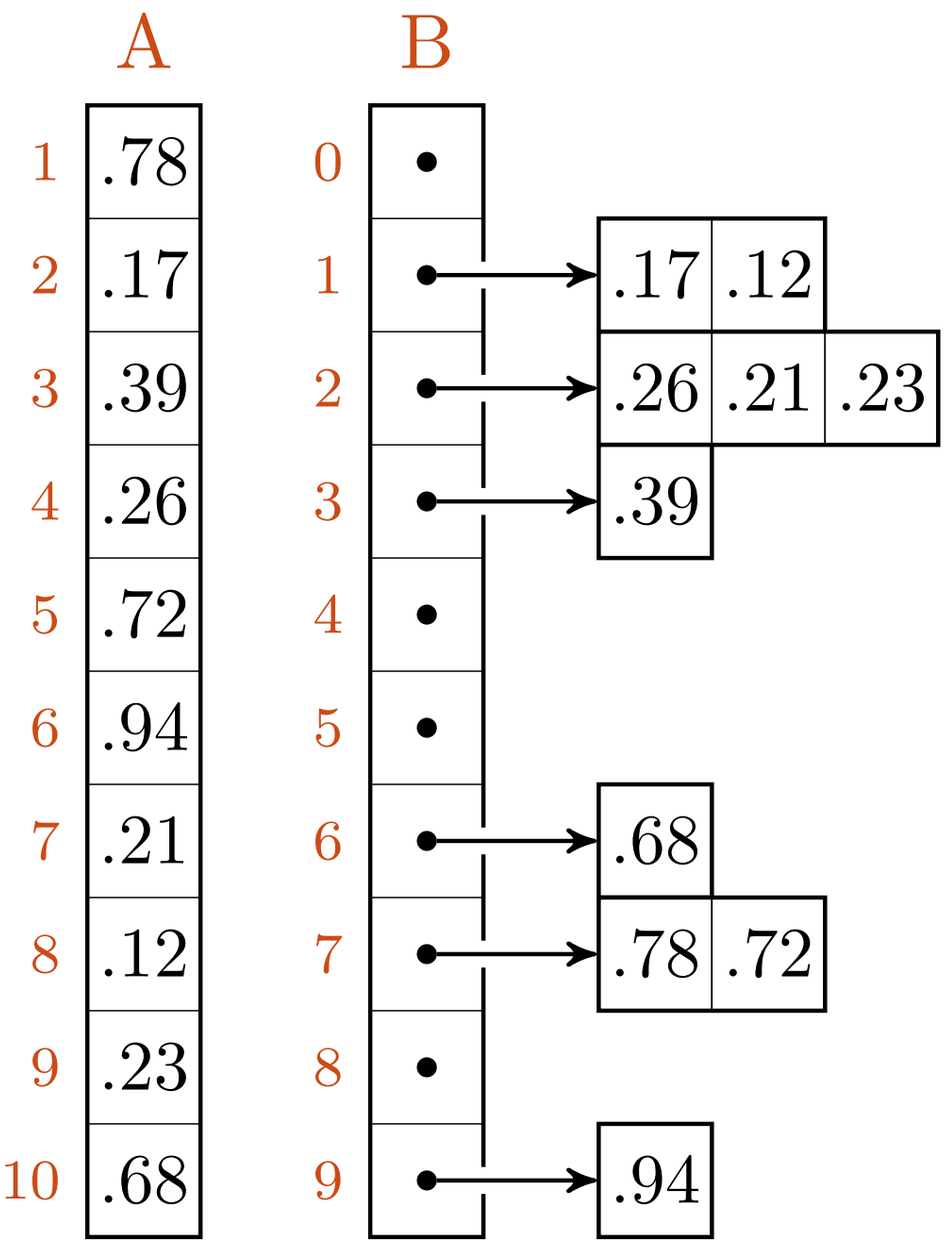
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



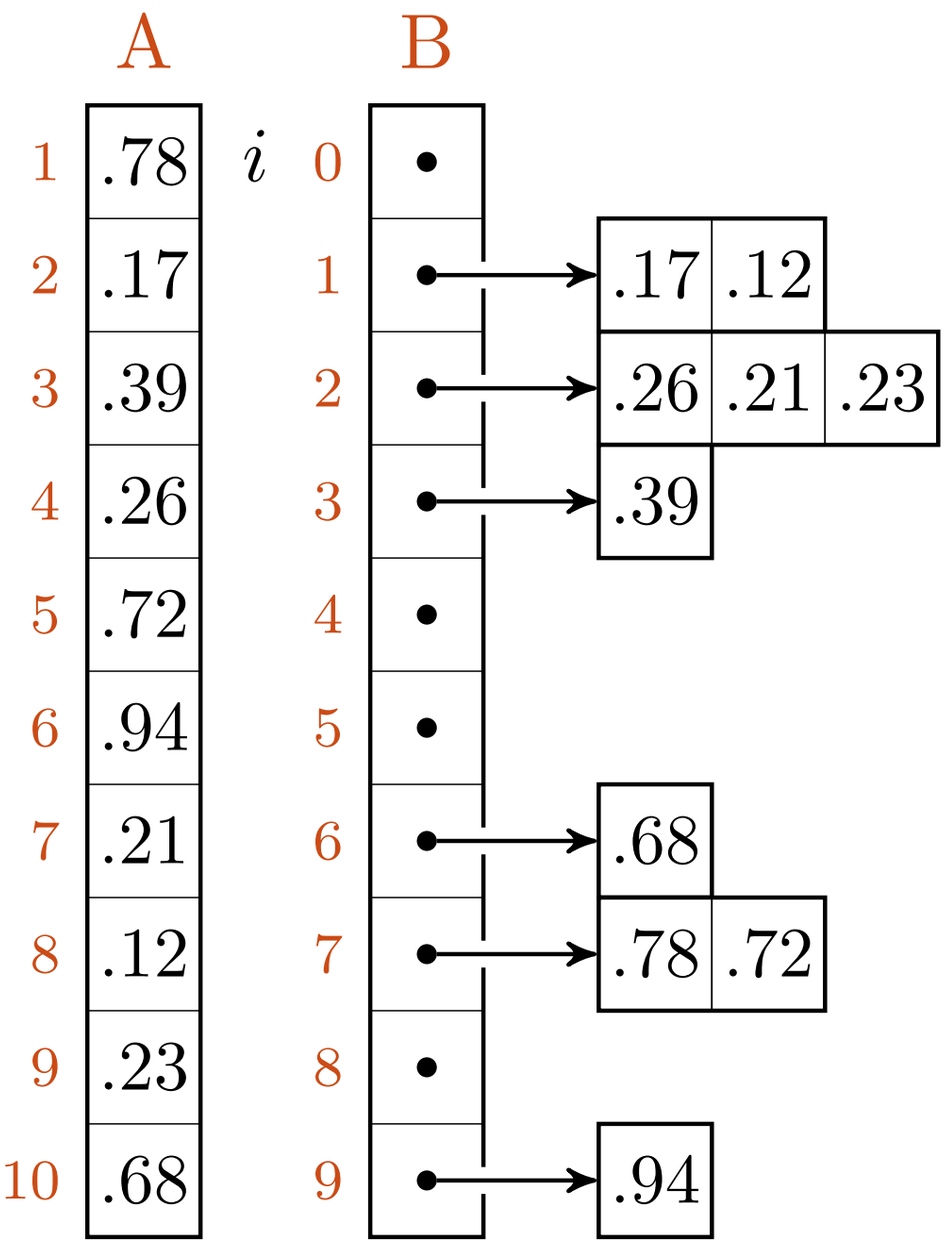
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



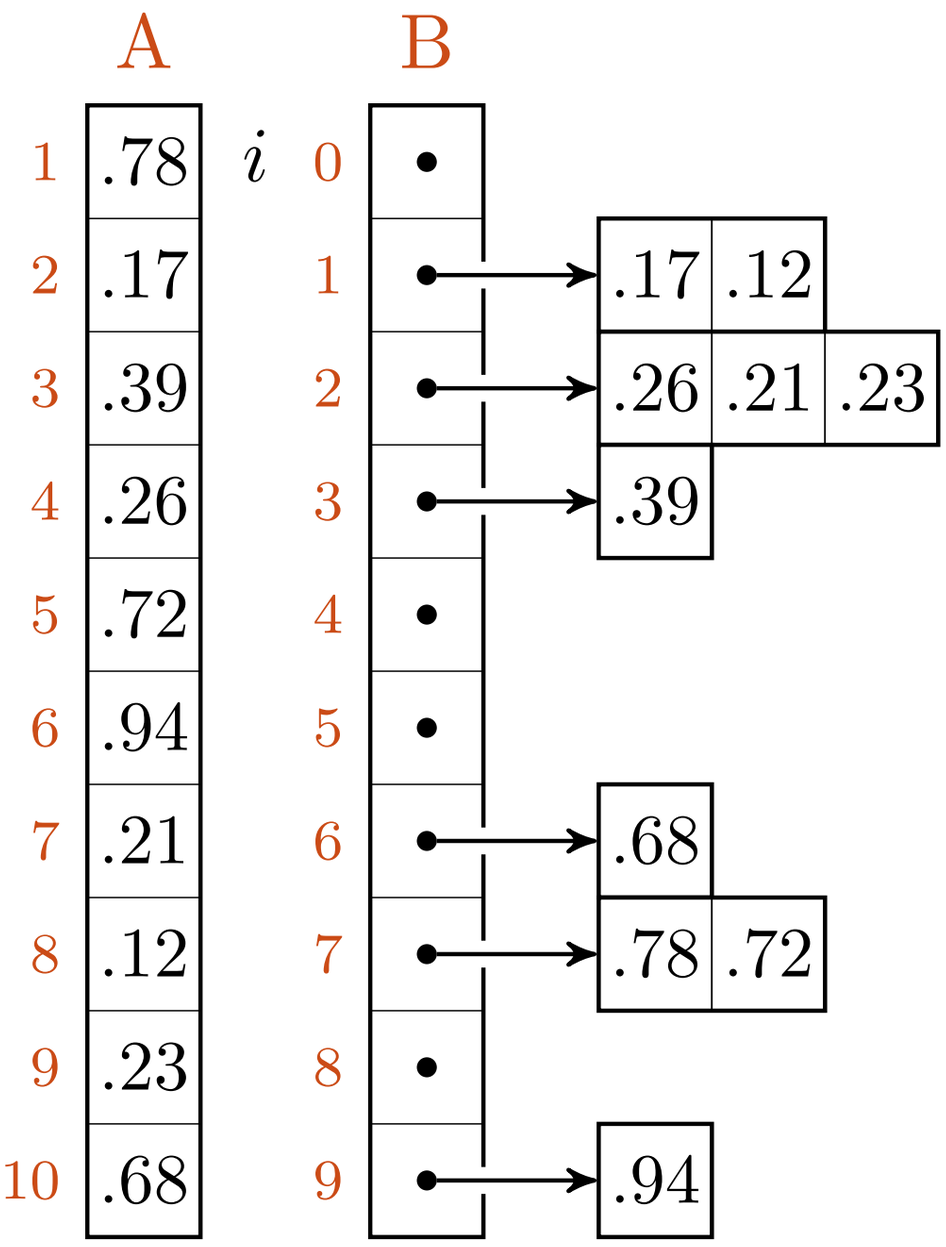
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



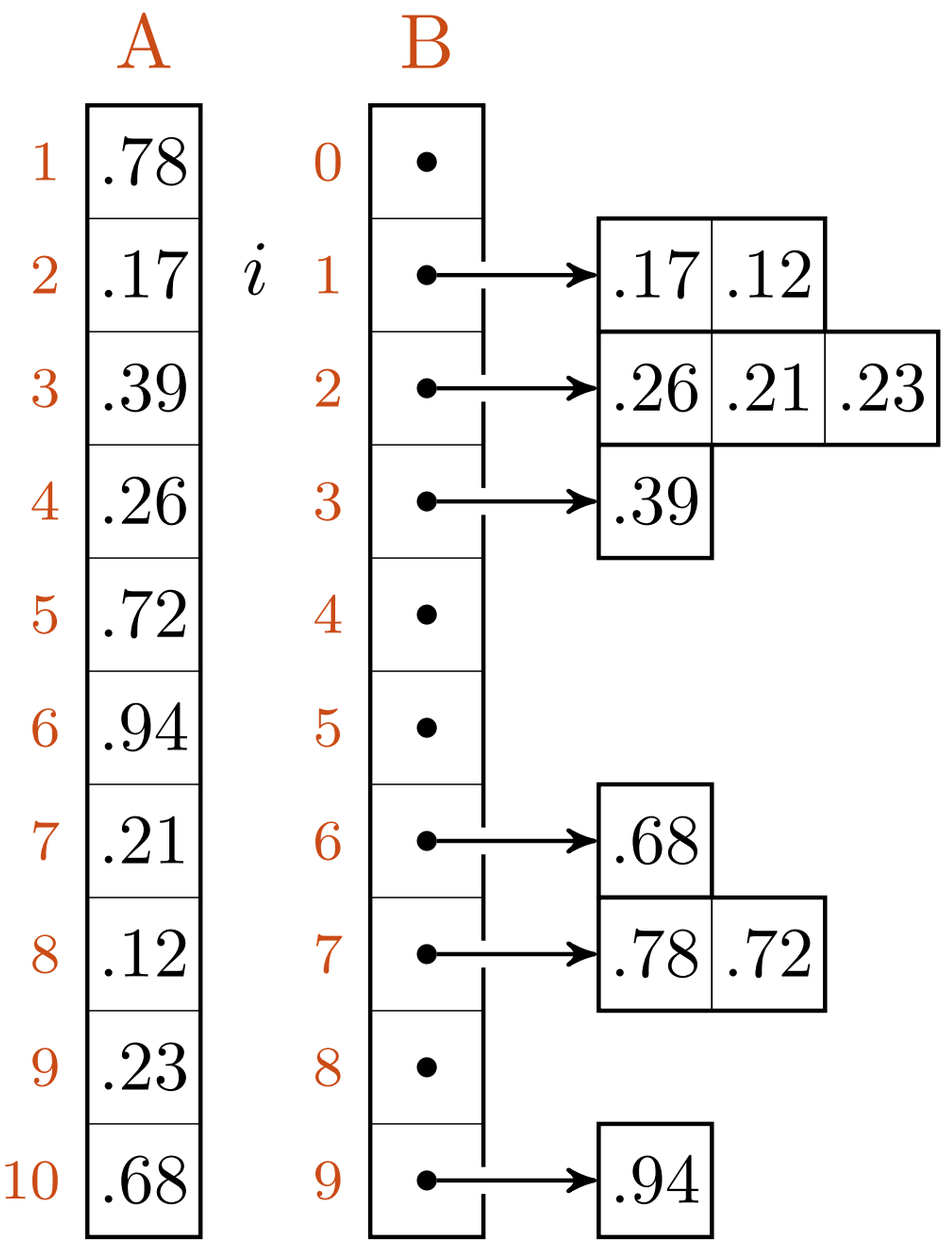
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



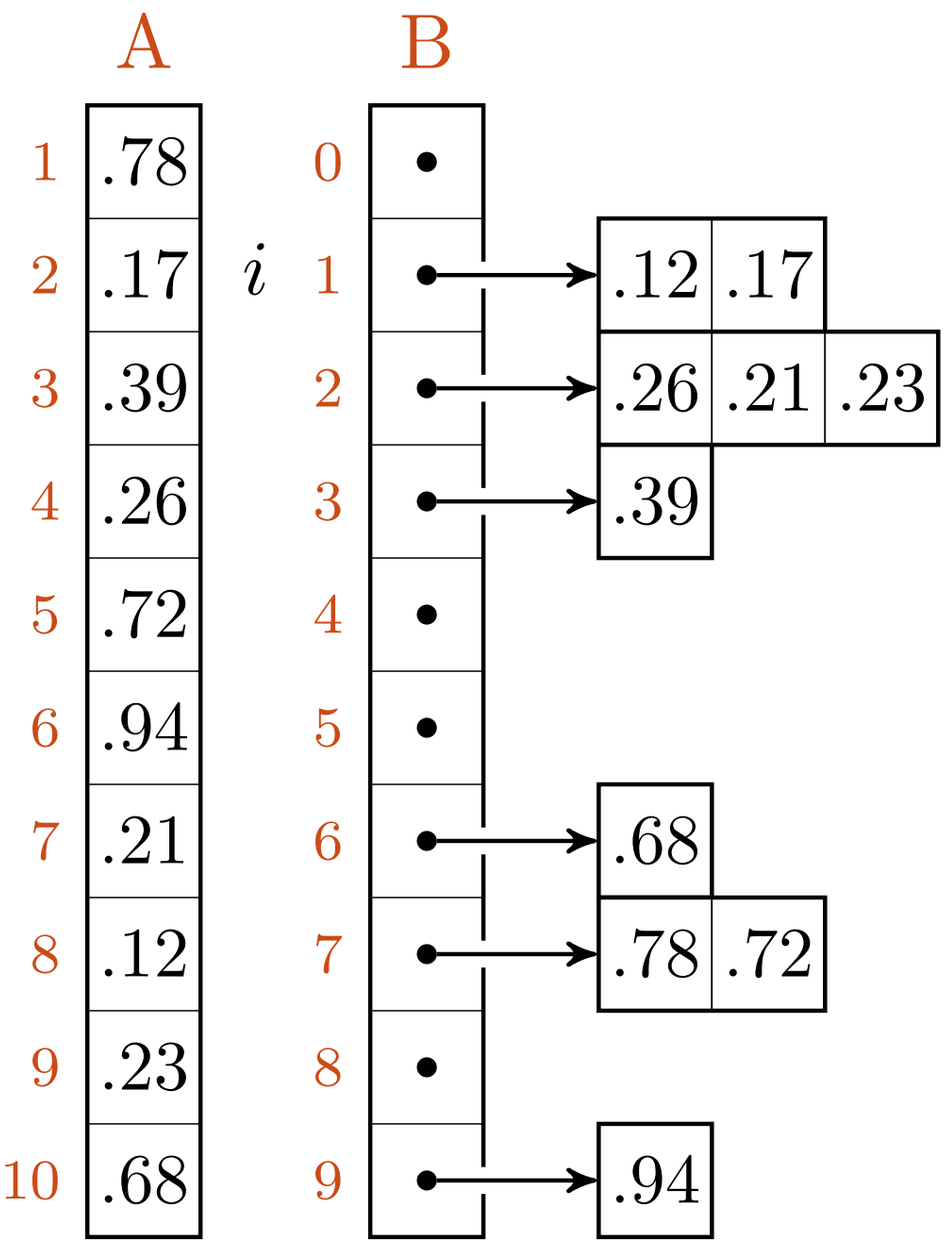
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



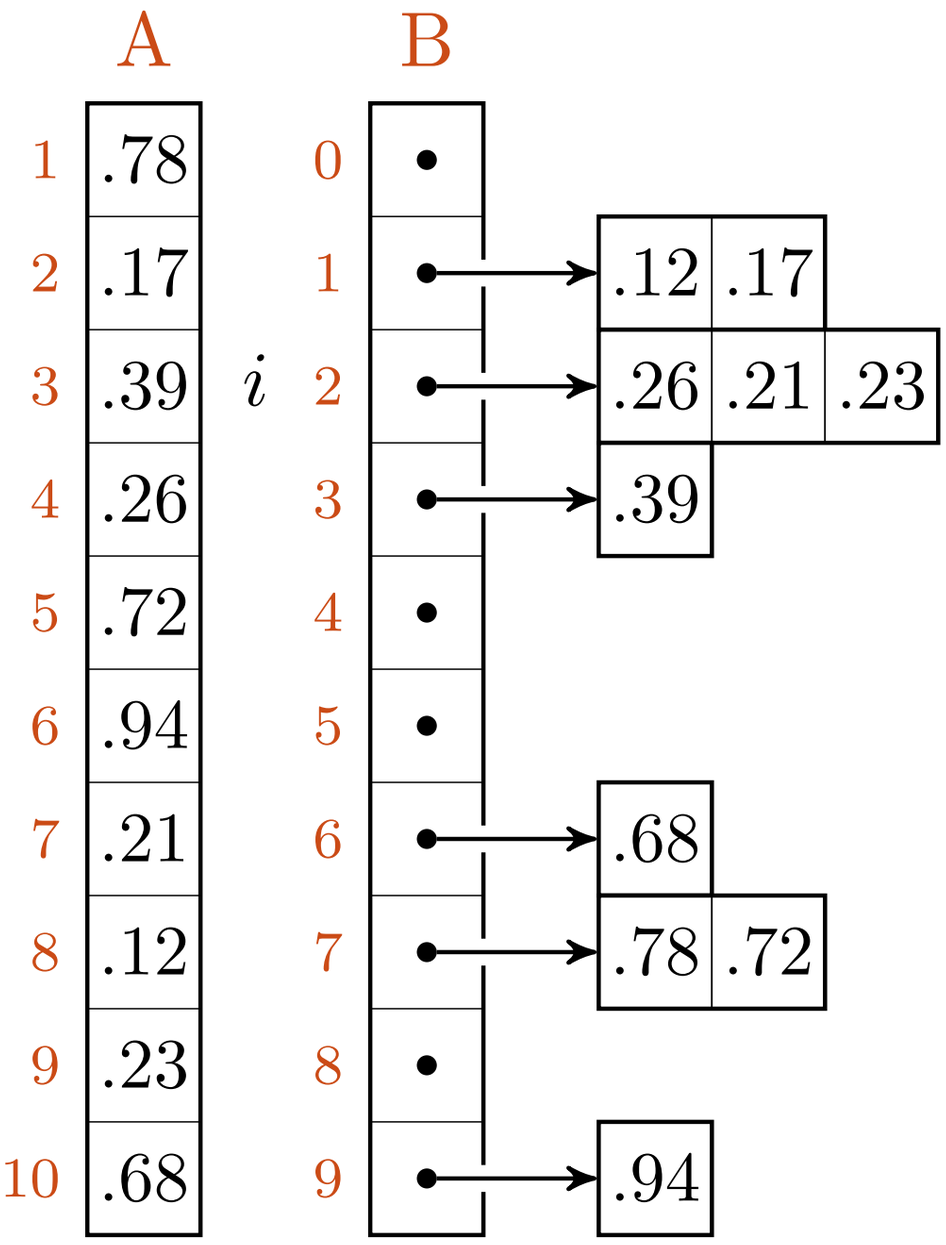
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



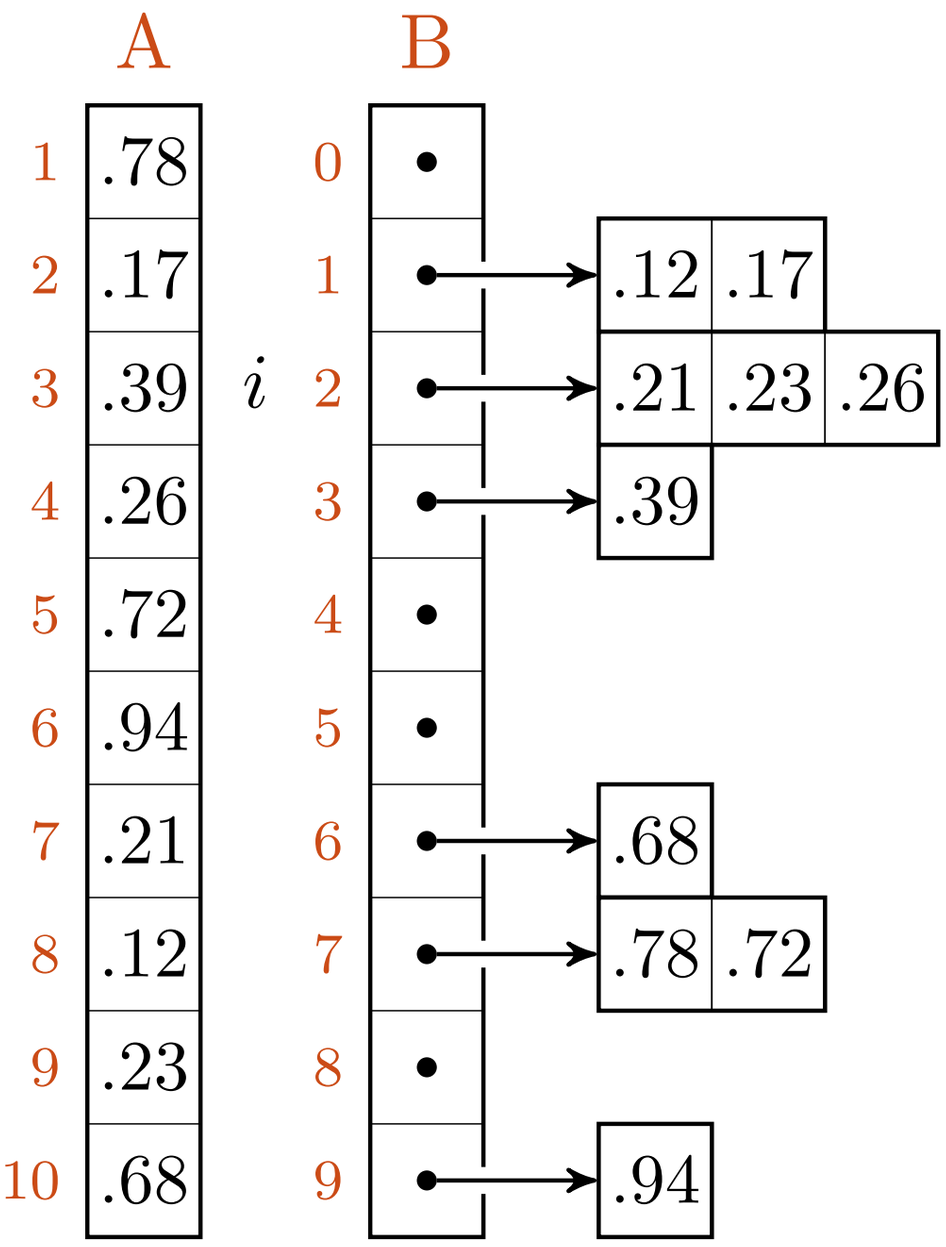
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



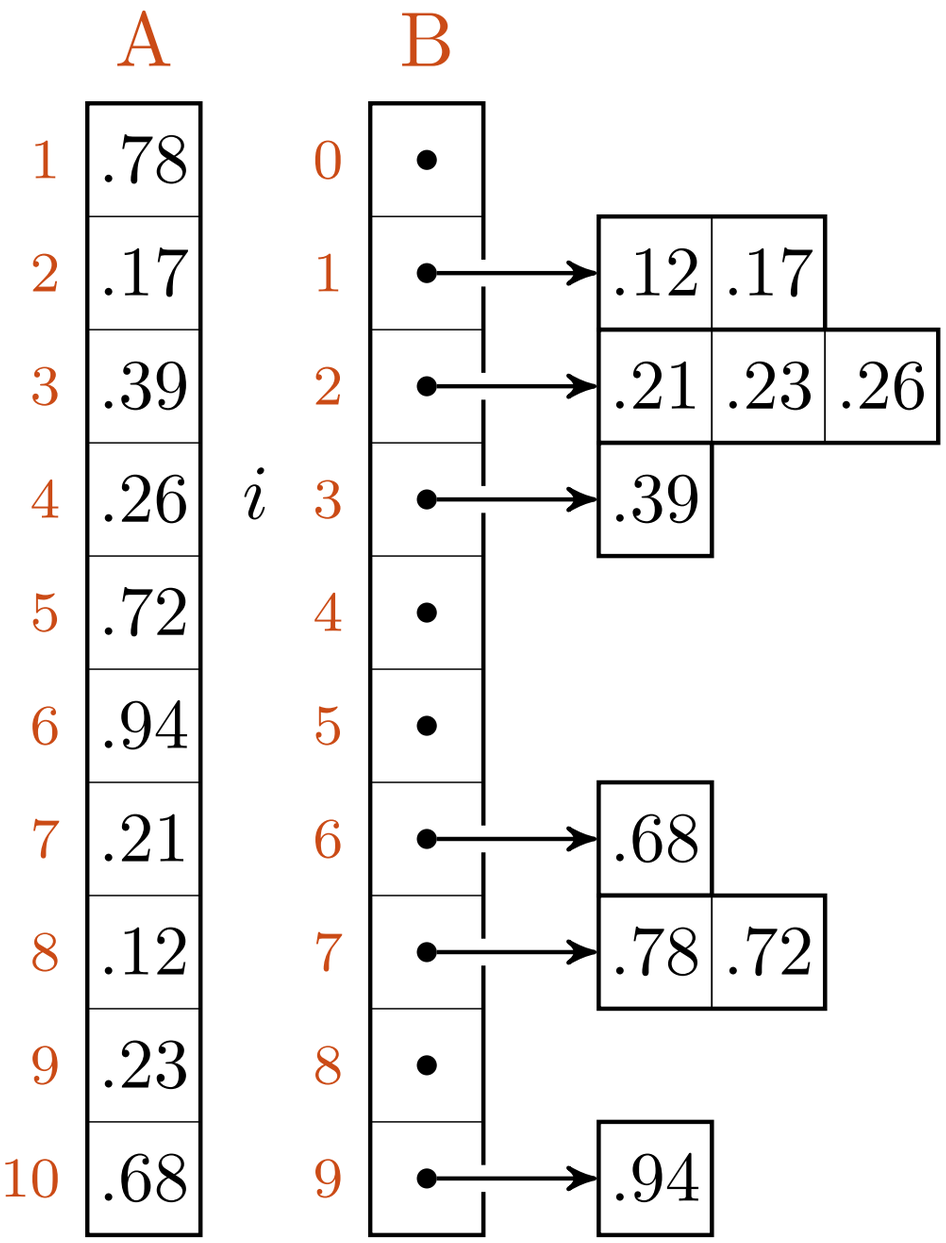
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



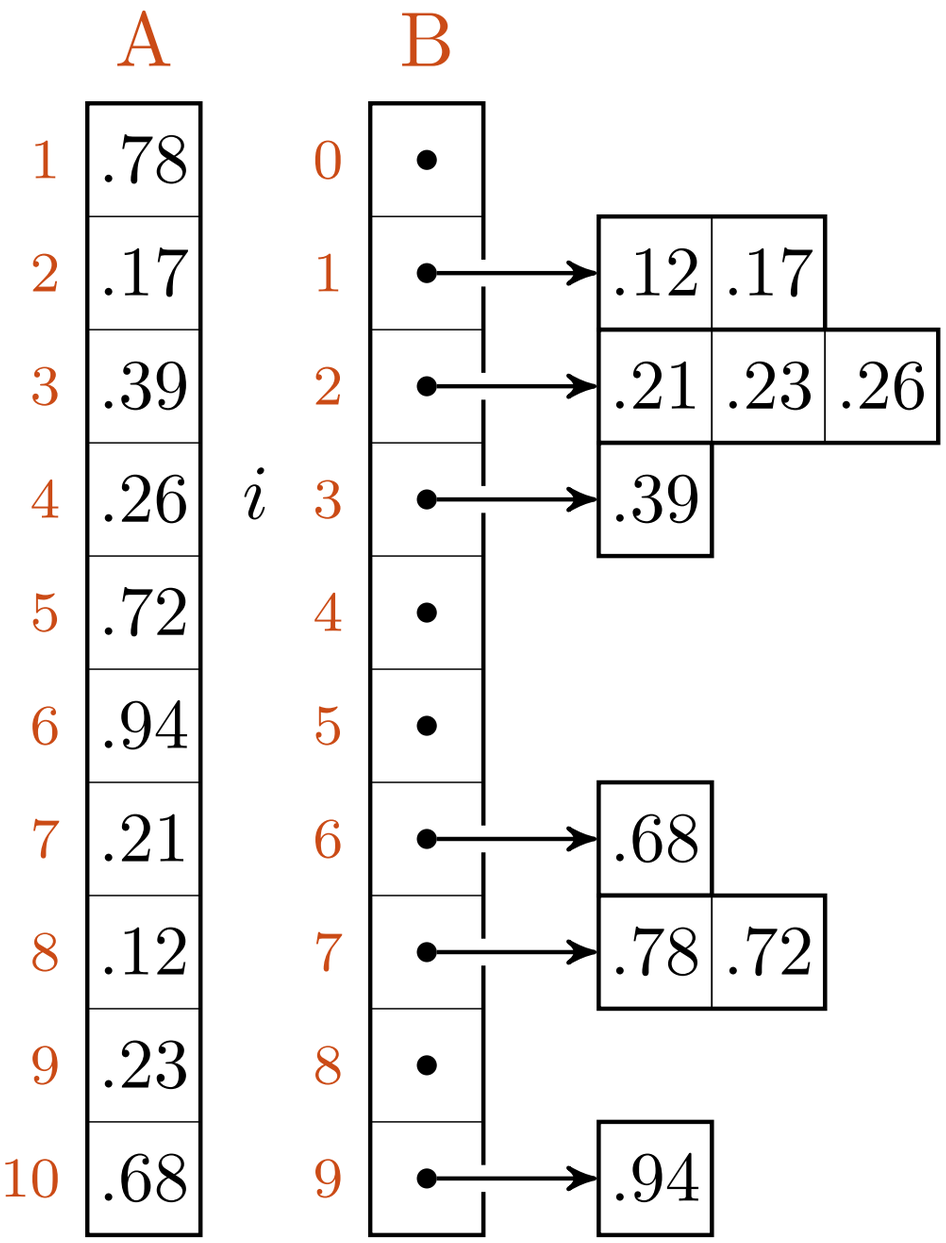
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



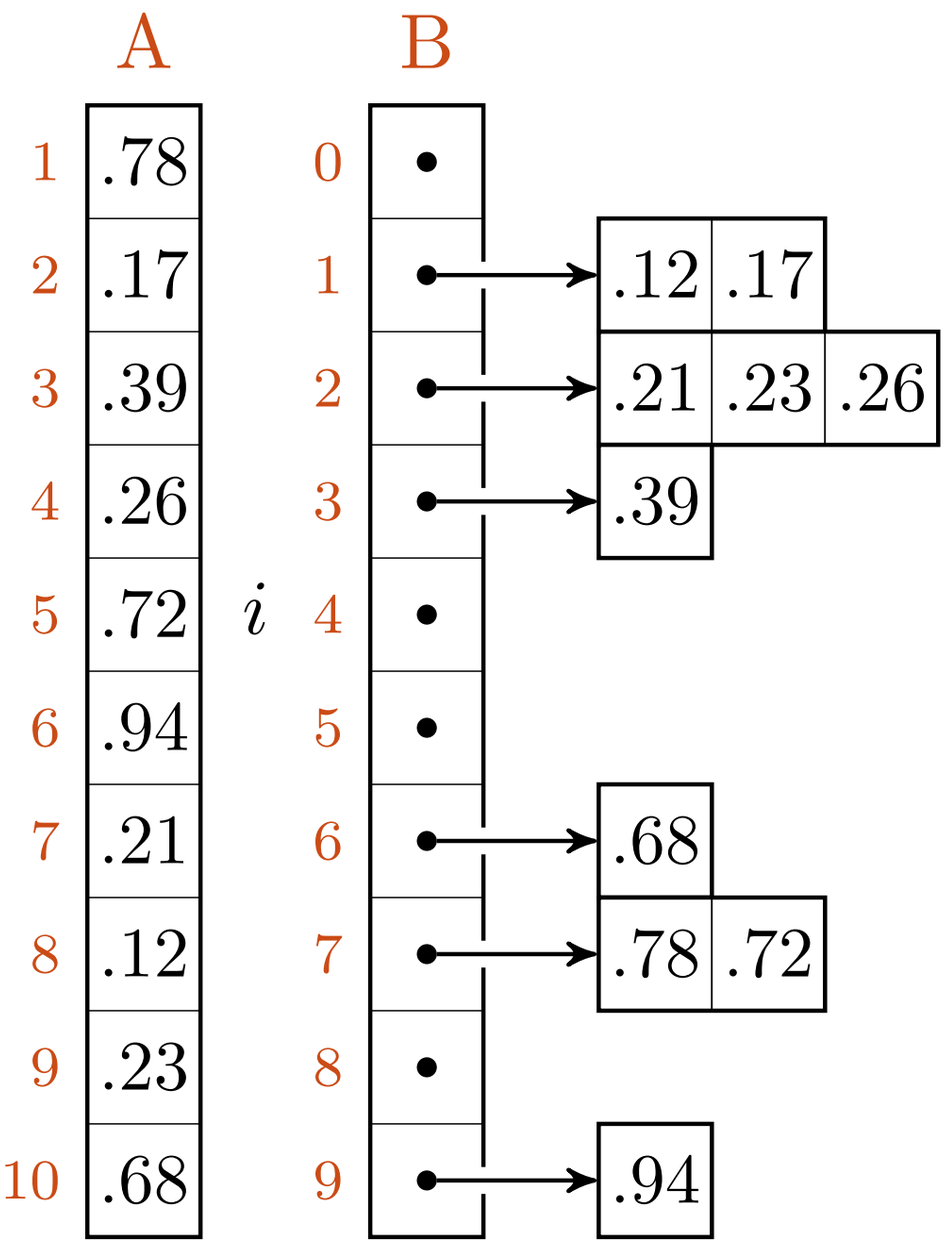
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



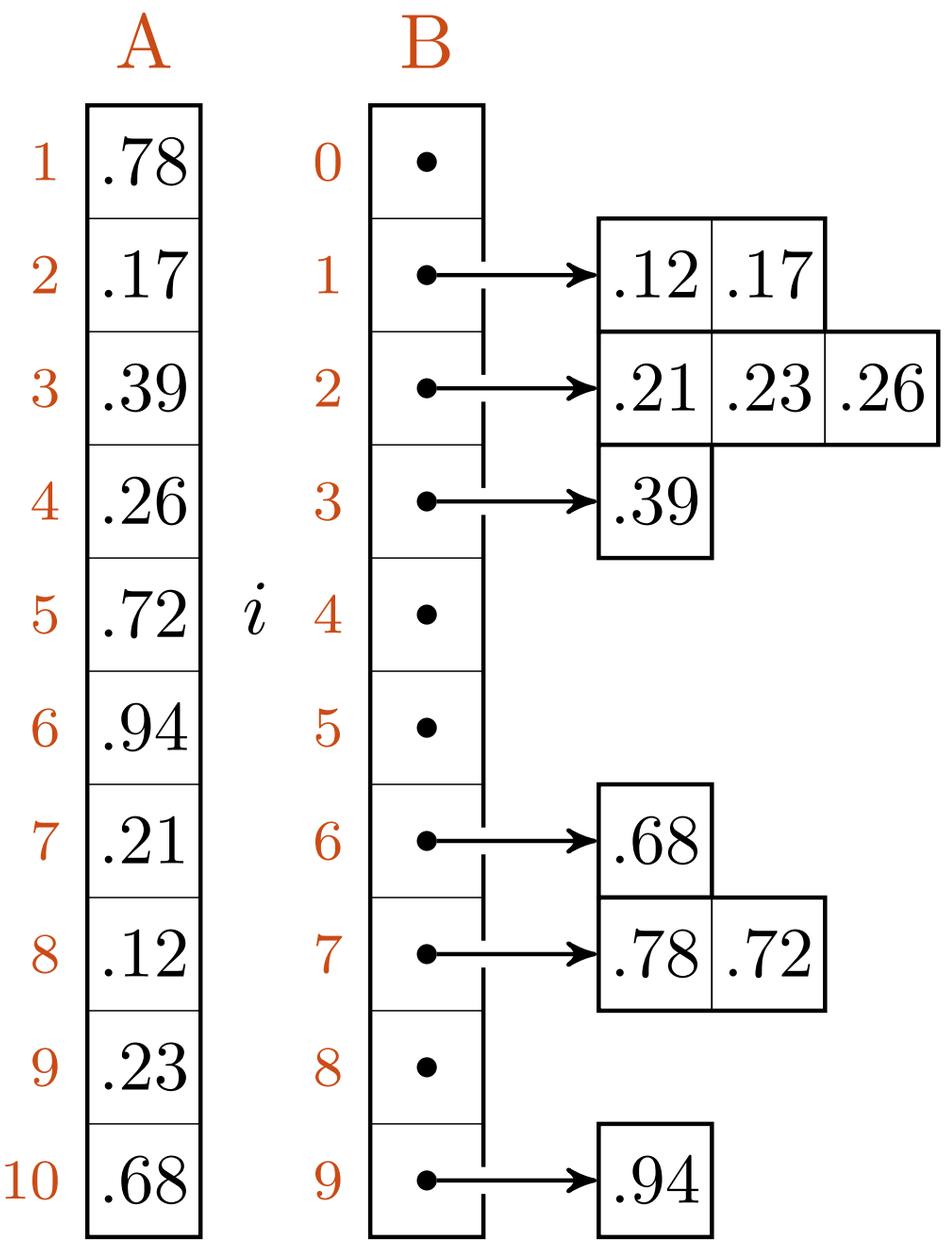
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



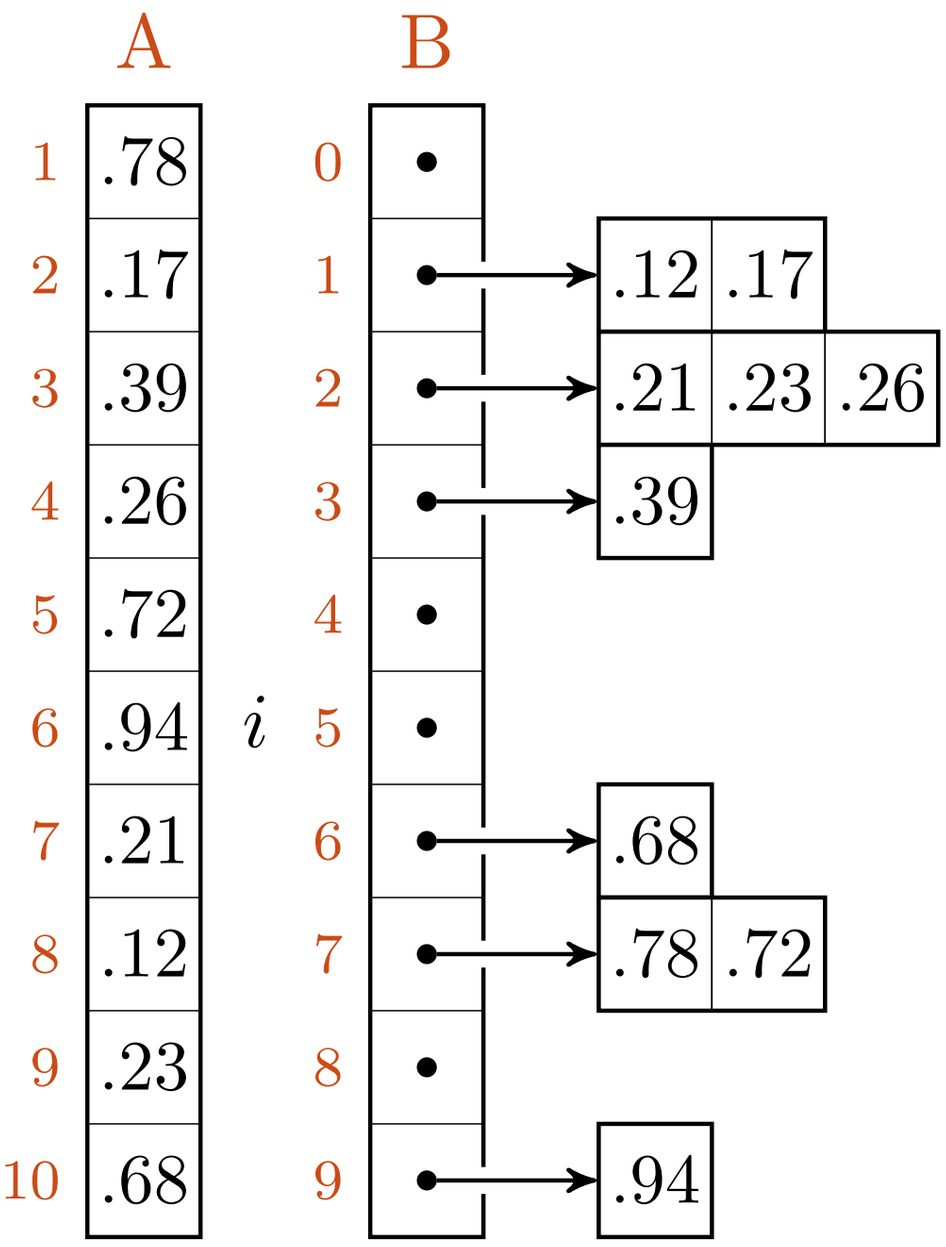
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



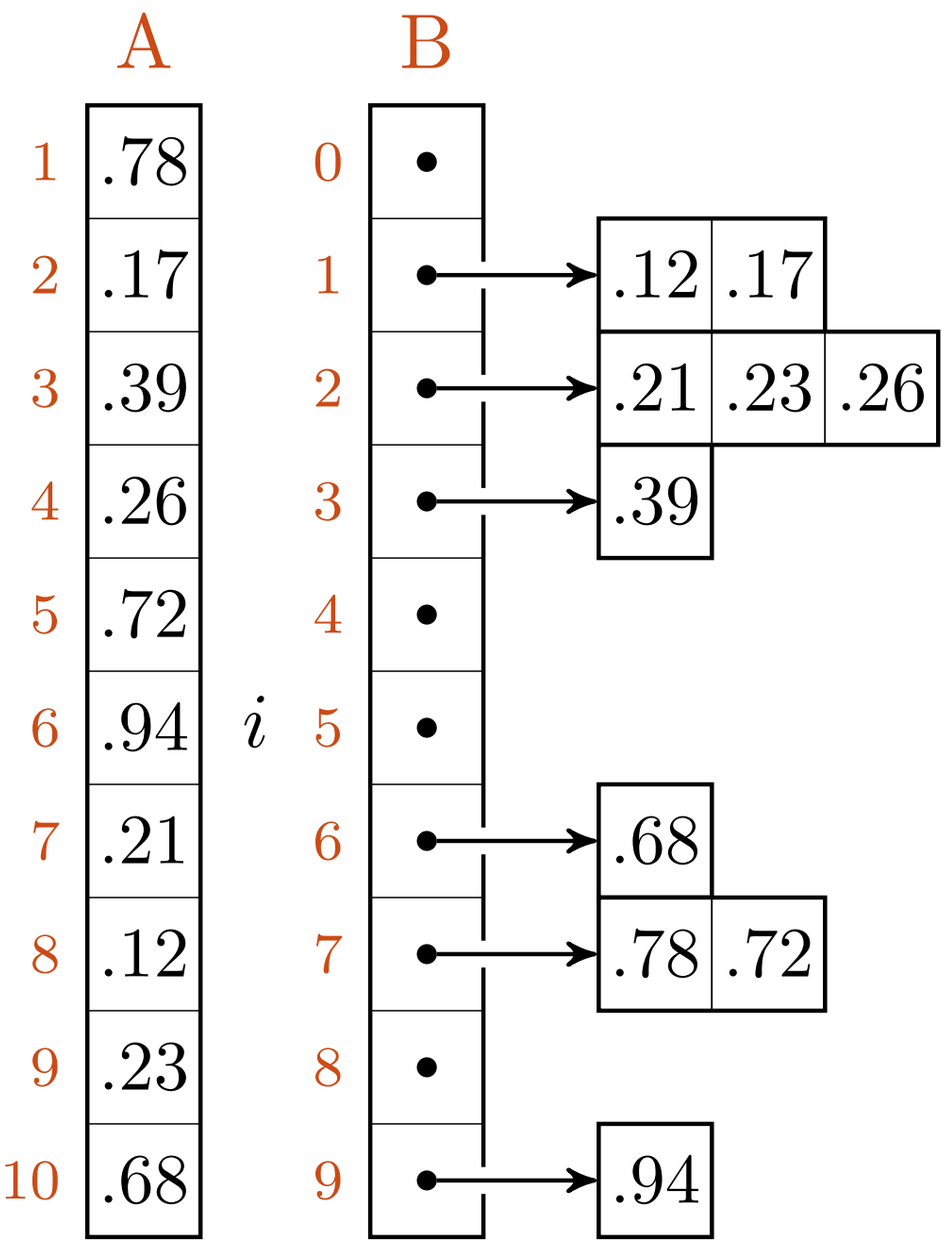
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



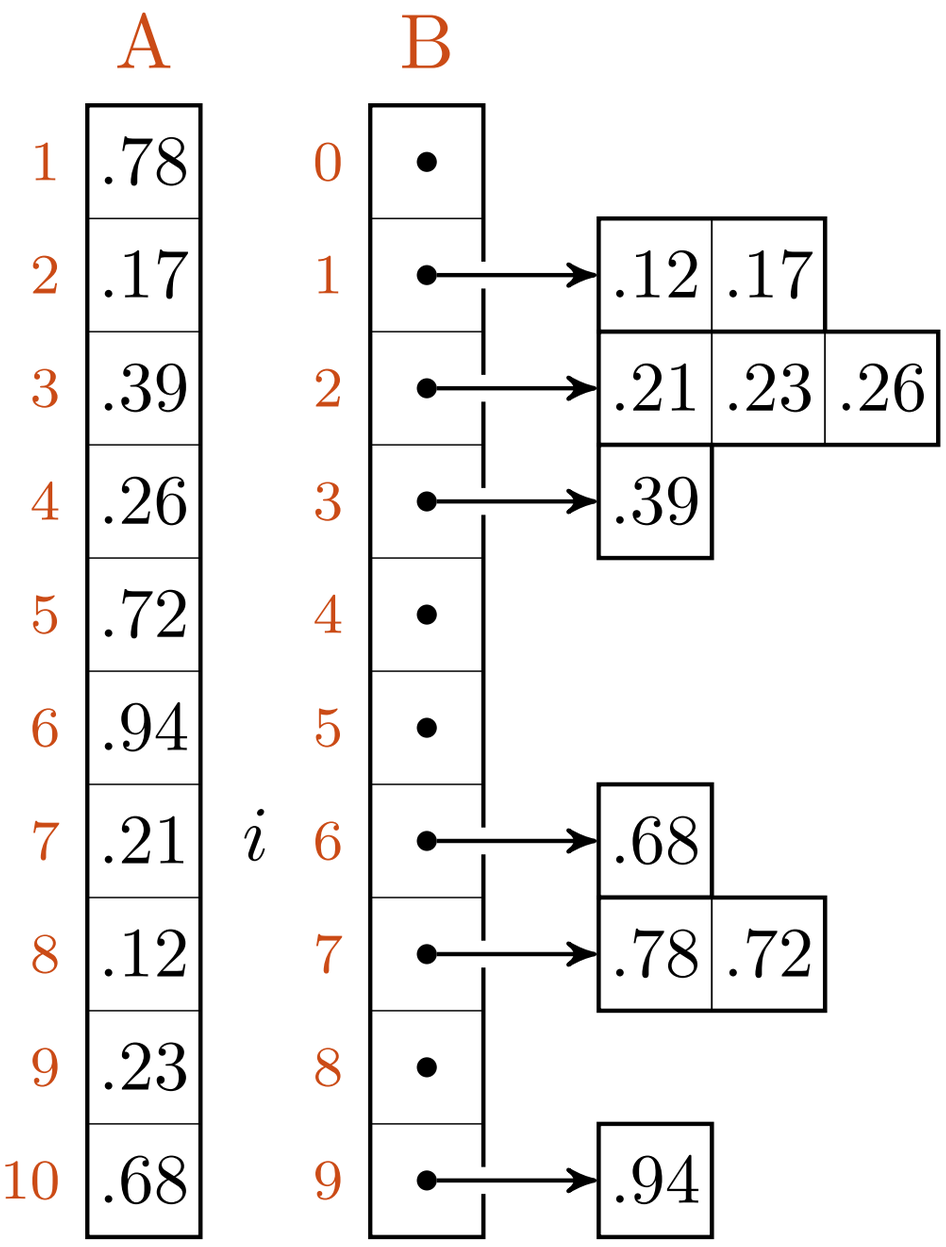
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



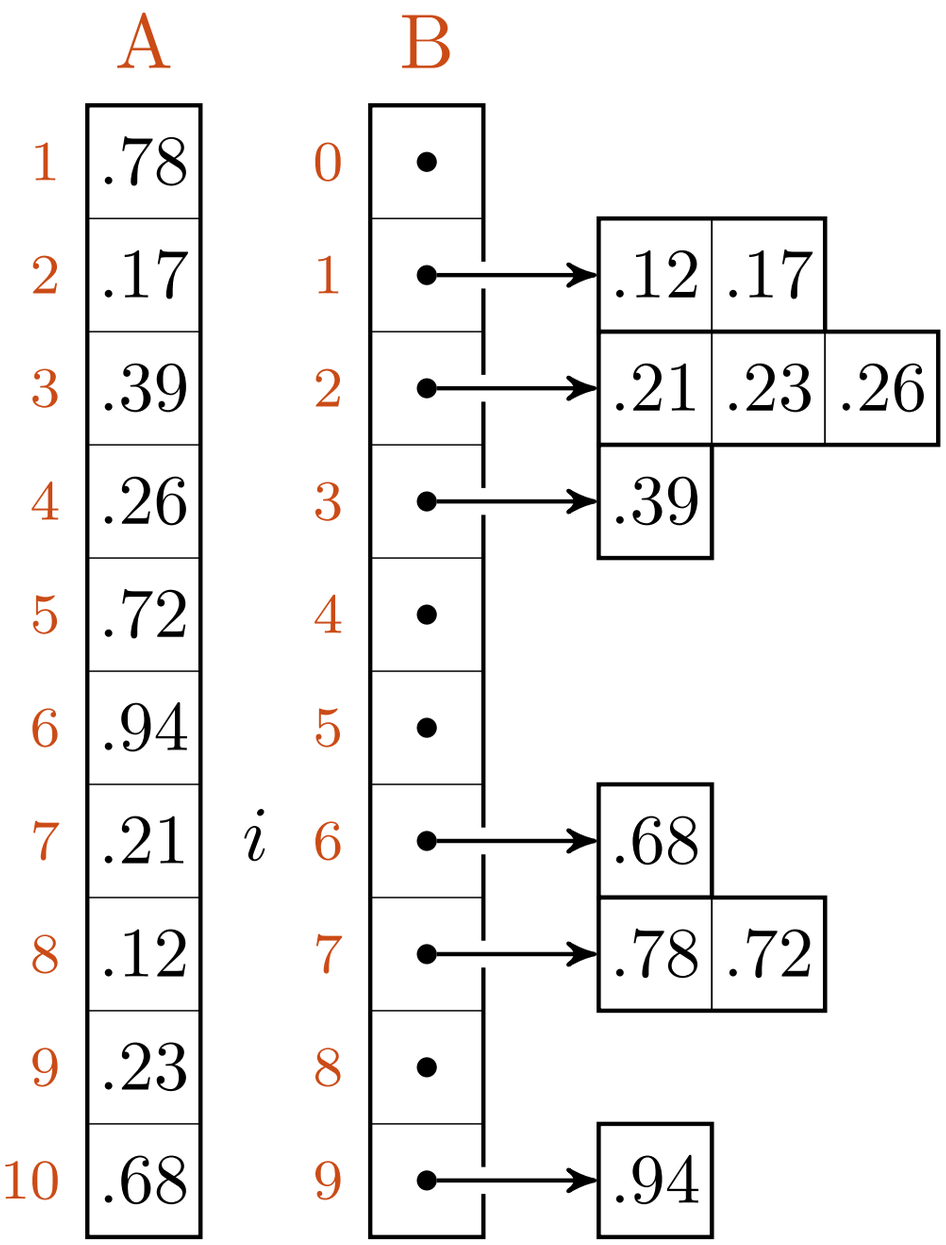
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



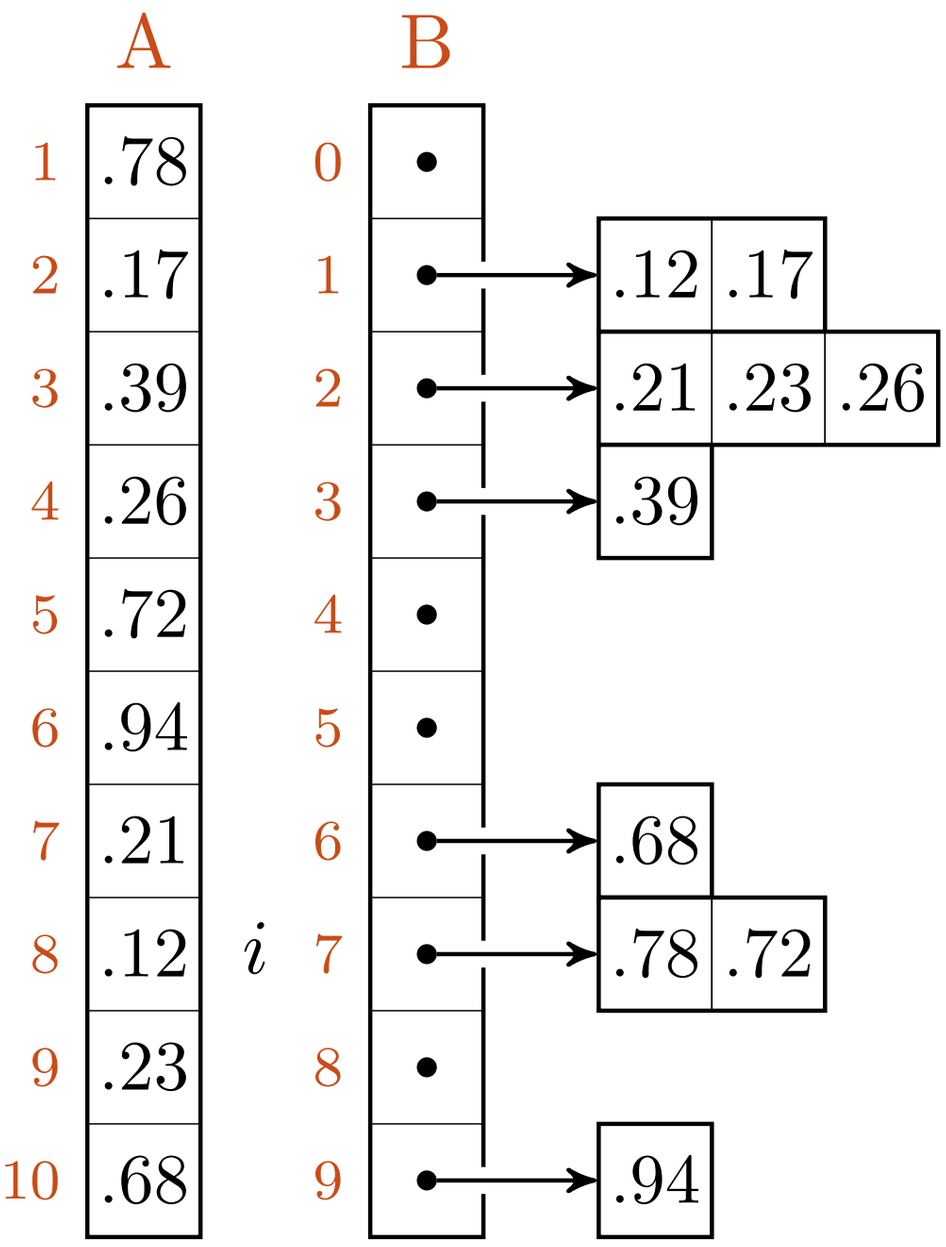
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



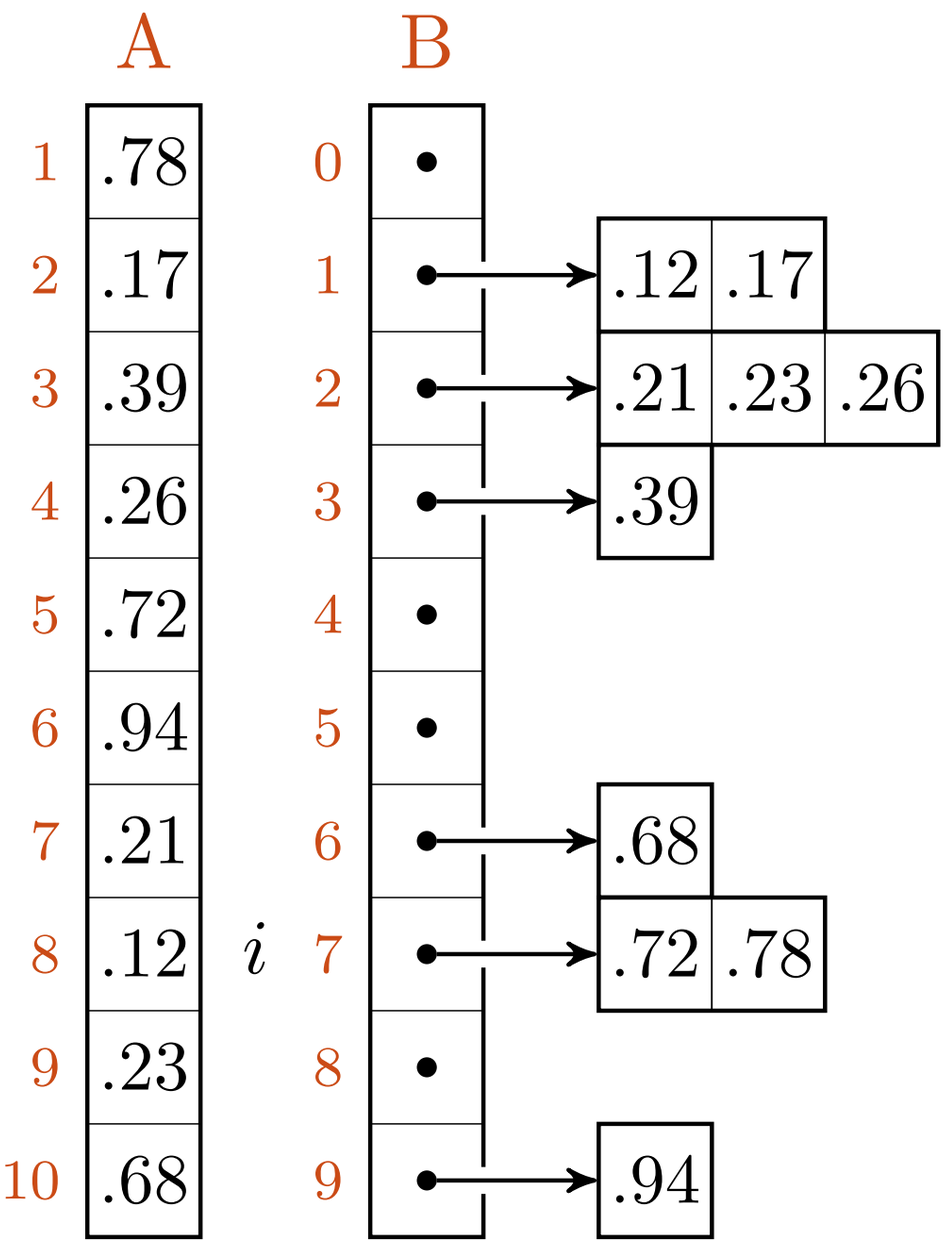
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



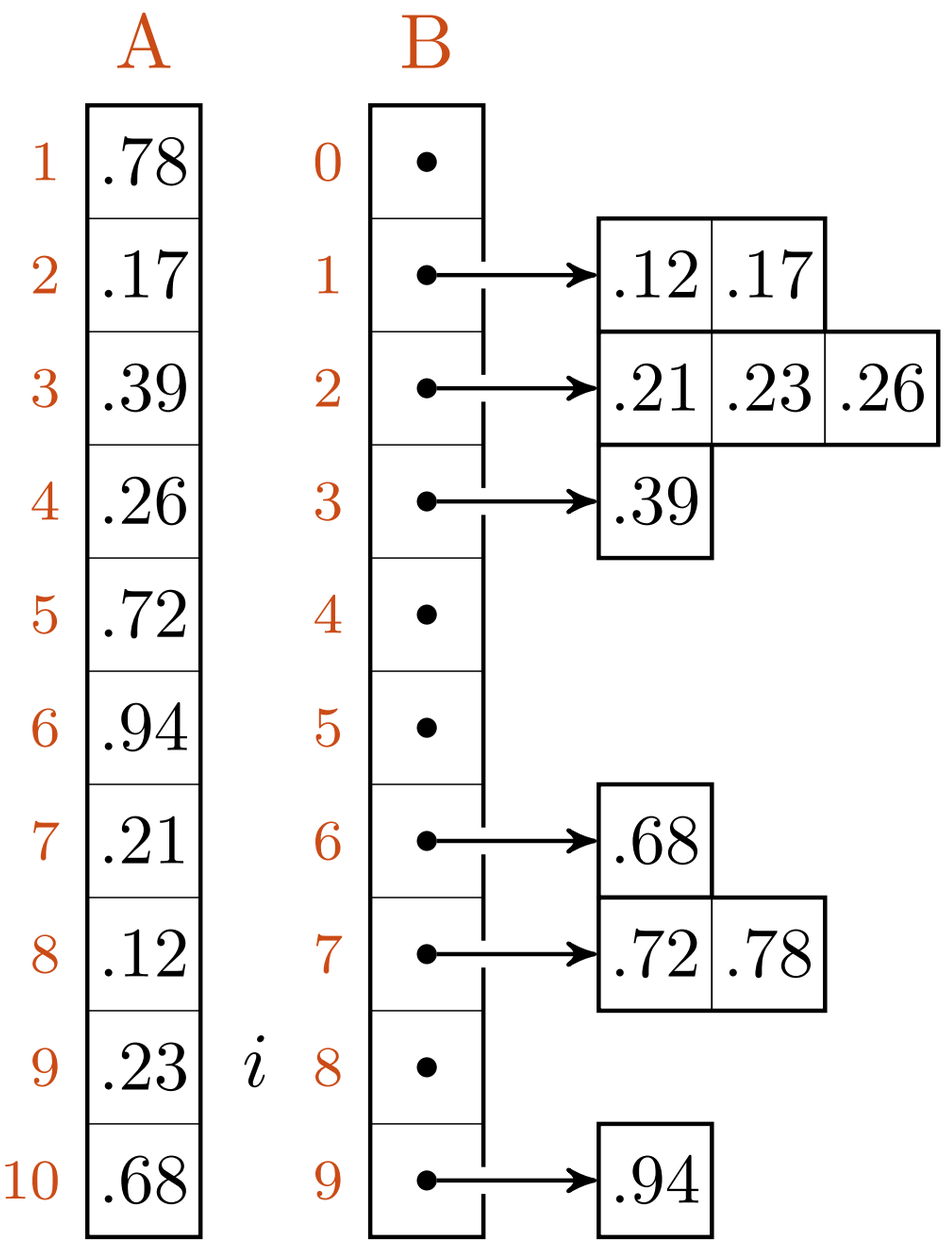
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



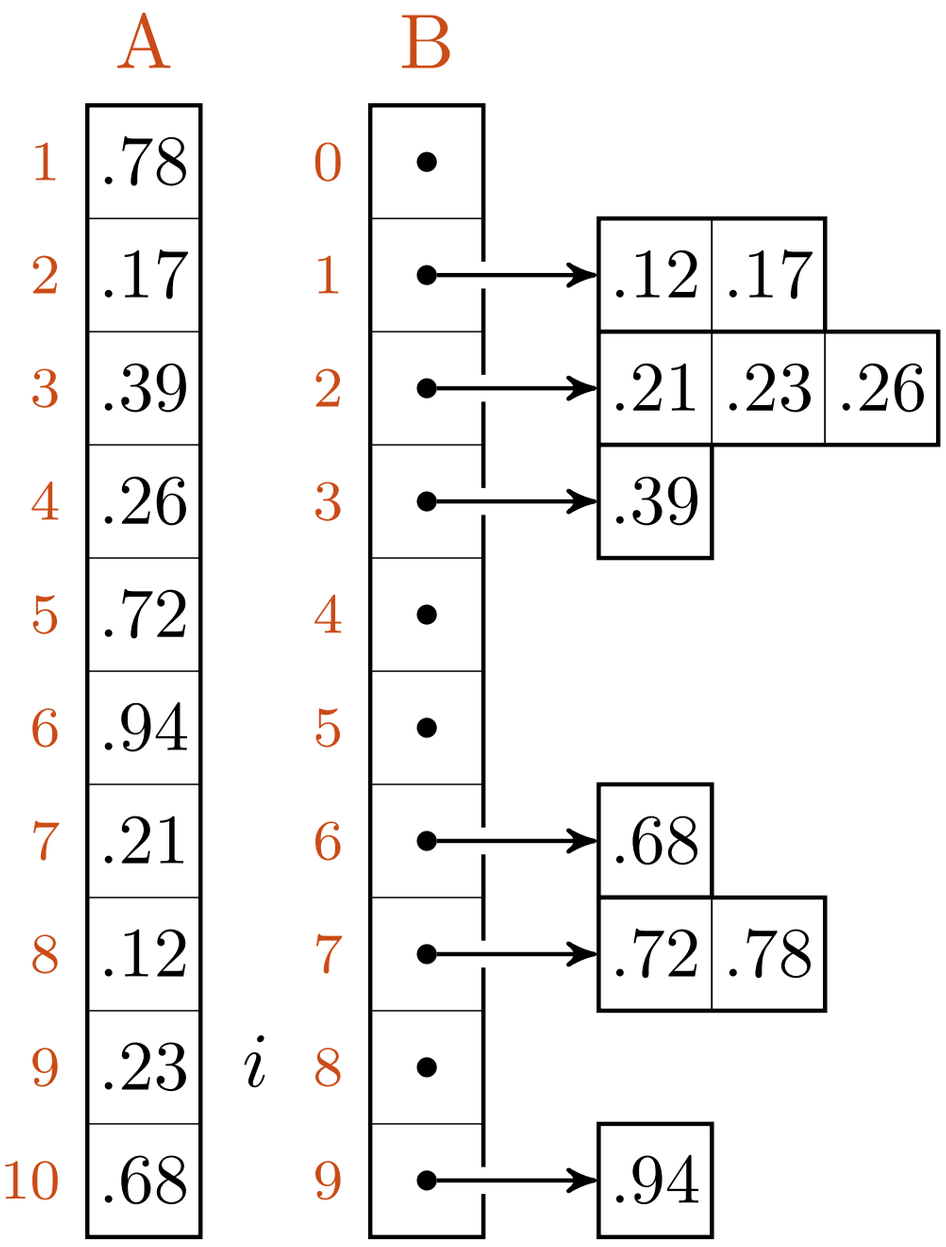
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



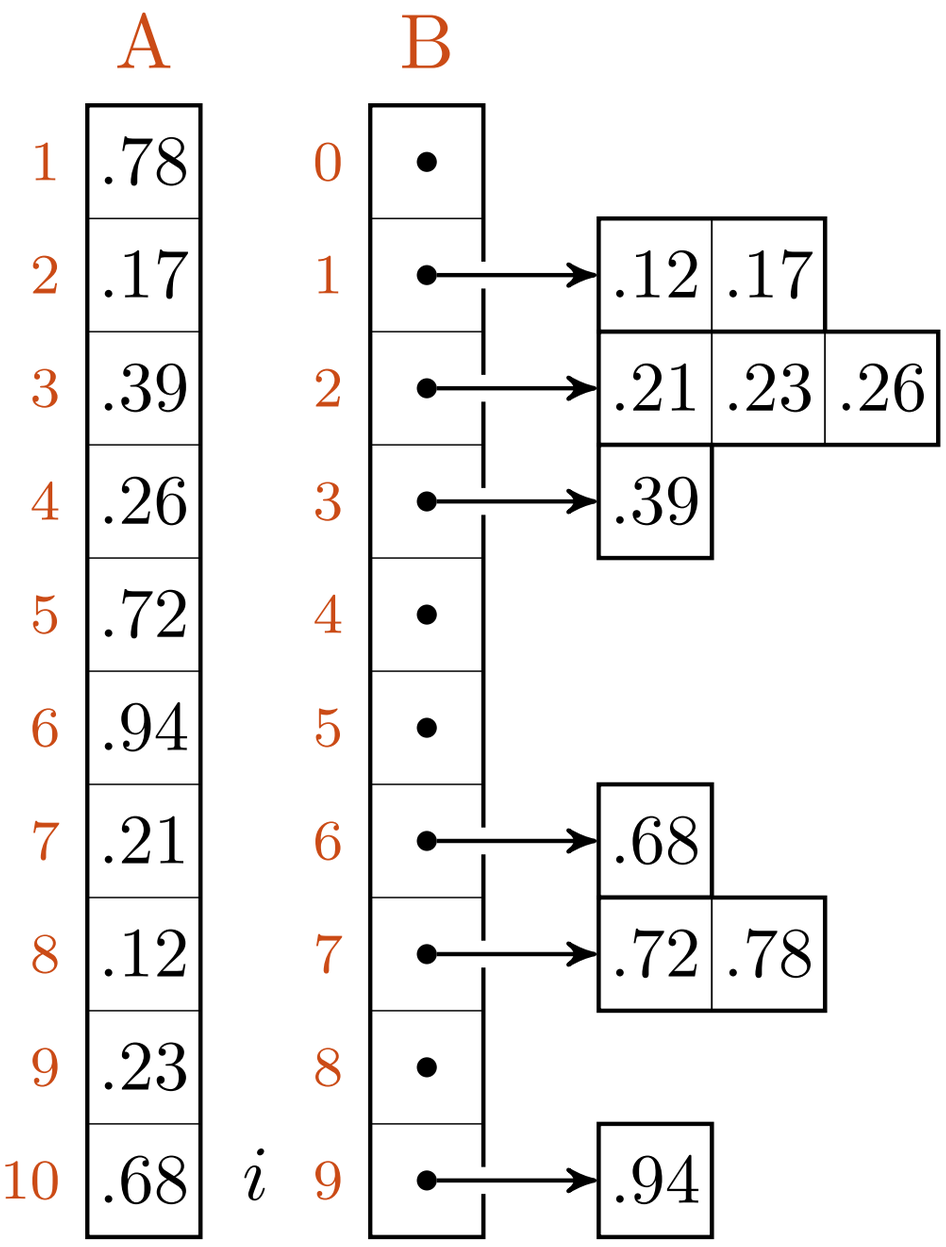
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



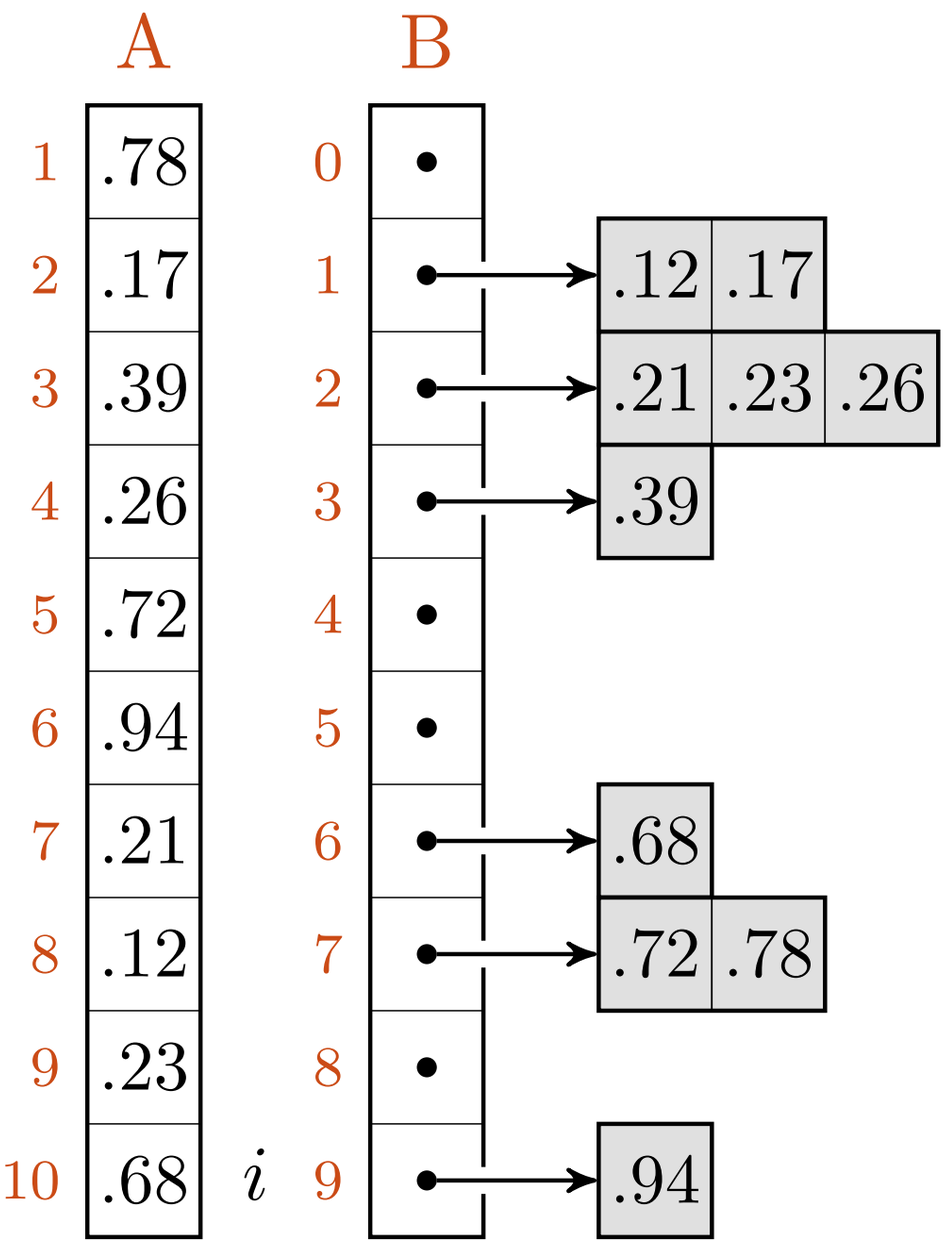
```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4        make B[i] an empty list
5  for i = 1 to n
6        add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8        sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



```
BUCKET-SORT(A)
1  n = A.length
2  create B[0..n − 1]
3  for i = 1 to n
4       make B[i] an empty list
5  for i = 1 to n
6       add A[i] to B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i]
9  concatenate B[0] ... B[n − 1]
```



$$\mathrm{T_W}(n) = \Theta(n^2)$$

$$\mathrm{T_A}(n) = \Theta(n)$$

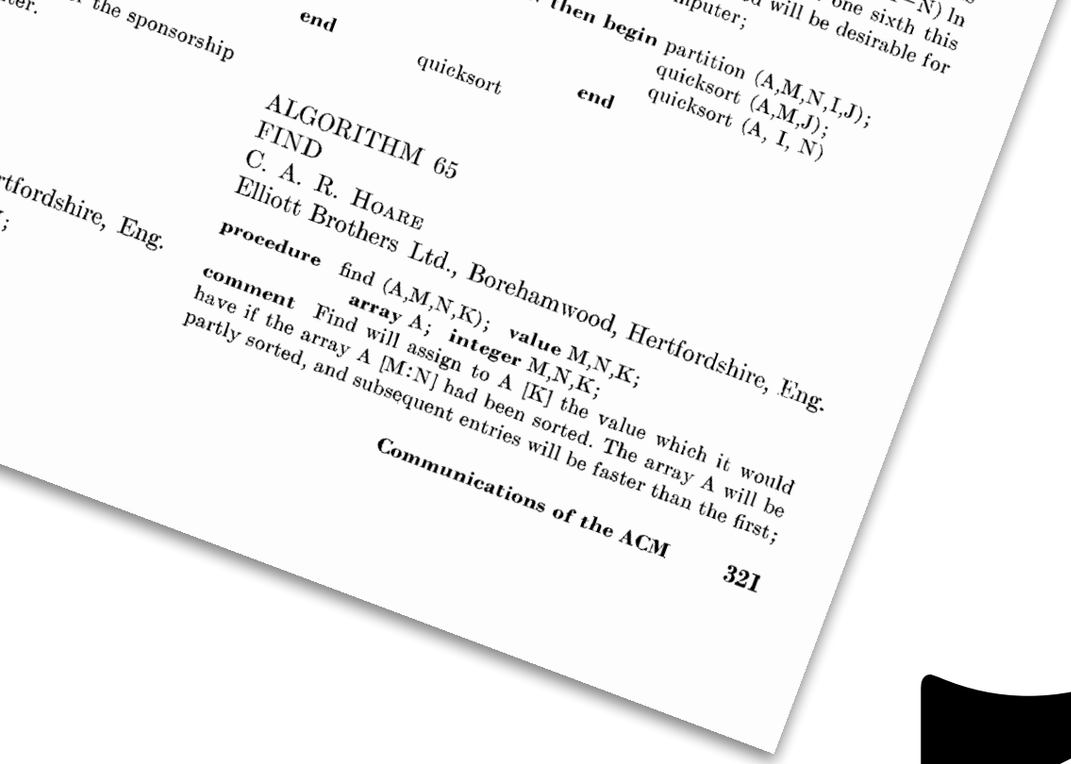
$$\mathrm{T_B}(n) = \Theta(n)$$

Insertion-Sort på n bøtter med forv. lengde $\Theta(1)$

Bryt grensen for AC

... ved å begrense problemet

Vi vil bare rangere *noen* elementer, heller enn *alle. Men vil antar her som for sammenligningsbasert sortering at vi bare kan rangere objekter parvis.



5.6

Randomized Select

Hvem er på F.eks. 10. plass?

Antar distinkte verdier!

Induksjon/rekursjon

Anta mindre instanser kan løses

Vi kan da løse disse rekursivt!

«Quicksort som binærsøk»

RAND-SEL(A, p, r, i)

A tabell
p venstre
r høyre
i rang

Finn det i-ende minste elementet i A$[p \,..\, r]$

```
RAND-SEL(A, p, r, i)
1  if p == r
```

A tabell
p venstre
r høyre
i rang

Bare ett element: Det må være det vi leter etter!

```
RAND-SEL(A, p, r, i)
1  if p == r
2        return A[p]
```

A tabell
p venstre
r høyre
i rang

Bare ett element: Det må være det vi leter etter!

```
RAND-SEL(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
```

A tabell
p venstre
r høyre
i rang
q splitt

Nå er A[q] på rett plass, akkurat som i QUICKSORT!

```
RAND-SEL(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
```

A tabell
p venstre
r høyre
i rang
q splitt
k rang, q

A$[q]$ er det k-ende minste elementet i A$[p\,..\,r]$

```
RAND-SEL(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
```

A tabell
p venstre
r høyre
i rang
q splitt
k rang, q

Er A$[q]$ det i-ende minste elementet i A$[p\,..\,r]$?

```
RAND-SEL(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
```

A tabell
p venstre
r høyre
i rang
q splitt
k rang, q

Da er det det vi leter etter!

```
RAND-SEL(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
```

A tabell
p venstre
r høyre
i rang
q splitt
k rang, q

Leter vi etter et mindre element?

```
RAND-SEL(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RAND-SEL(A, p, q − 1, i)
```

A tabell
p venstre
r høyre
i rang
q splitt
k rang, q

Let blant de små: Finn det k-ende minste i $A[p\,..\,q-1]$

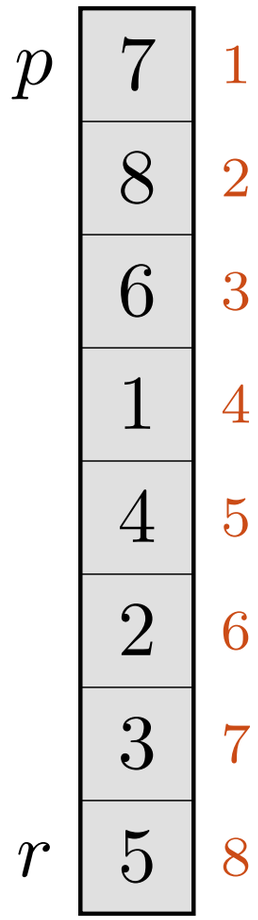
```
RAND-SEL(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RAND-SEL(A, p, q − 1, i)
9  else return RAND-SEL(A, q + 1, r, i − k)
```

A tabell
p venstre
r høyre
i rang
q splitt
k rang, q

Ellers: Finn det $(i-k)$-ende minste i $A[q+1\,..\,r]$

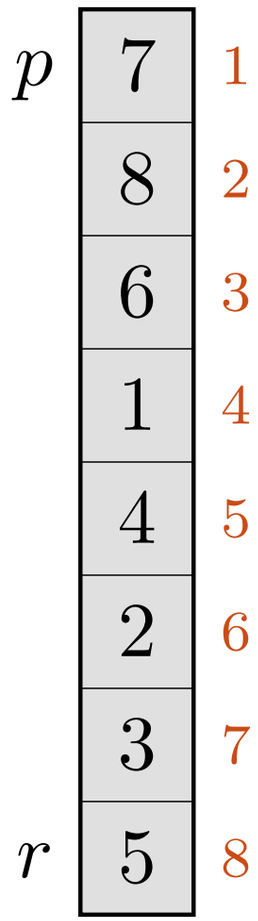
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i = -, 4$



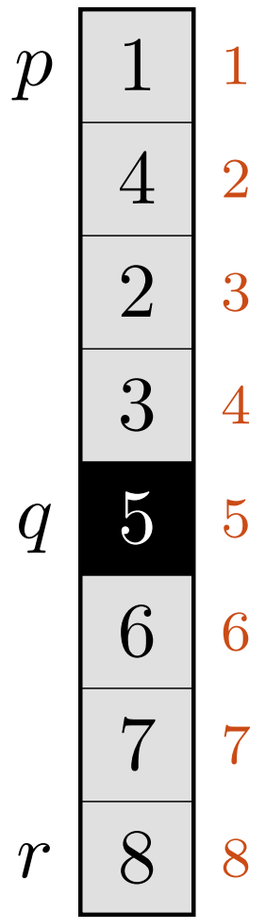
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i = -, 4$



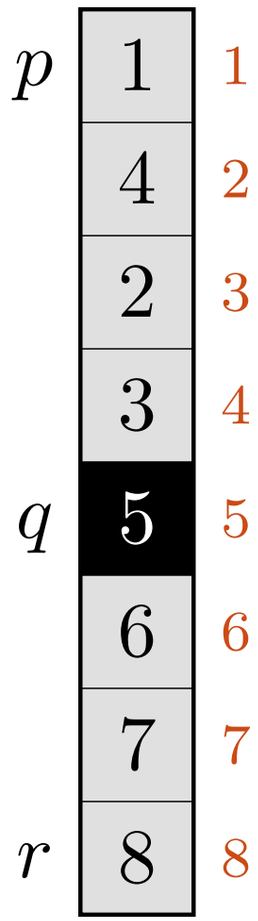
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i = -, 4$



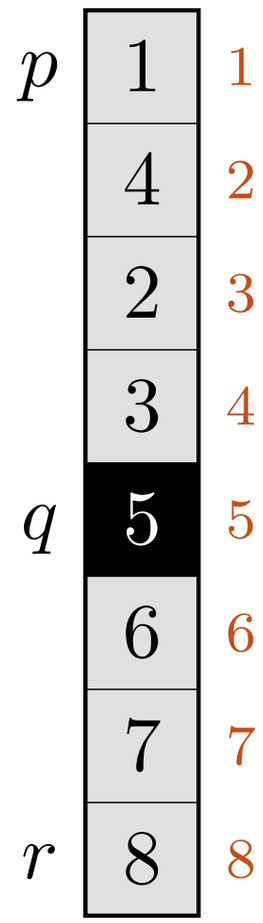
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i \;=\; 5, 4$



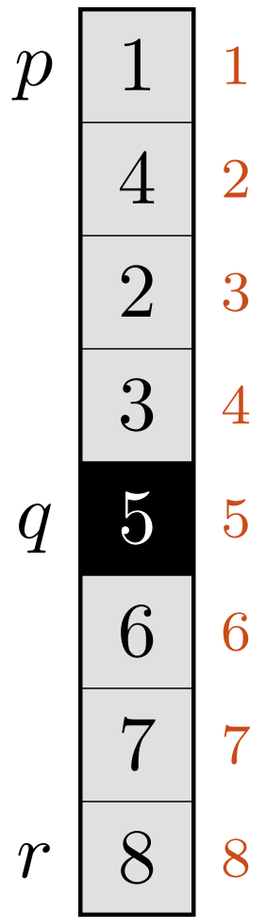
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i \;=\; 5, 4$



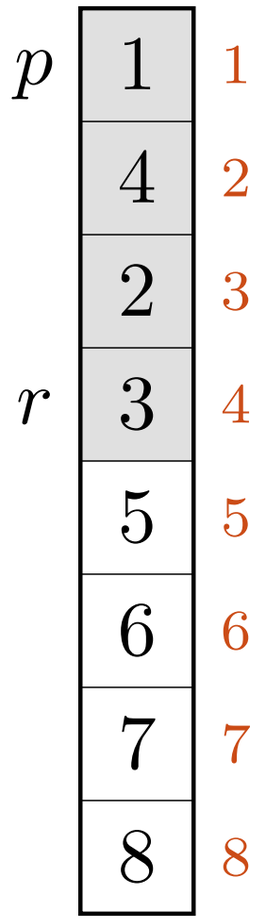
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i \;=\; 5, 4$



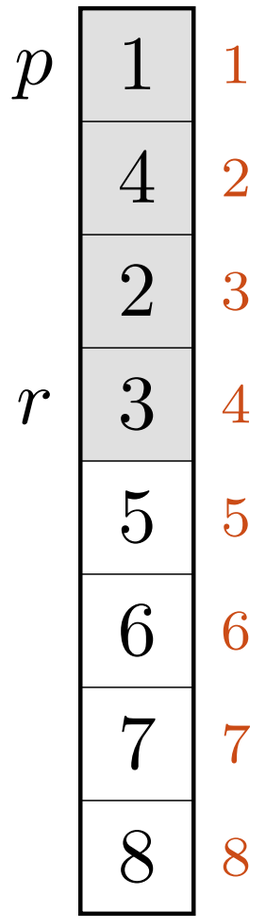
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i \;=\; 5, 4 \;\rangle\; -, 4$



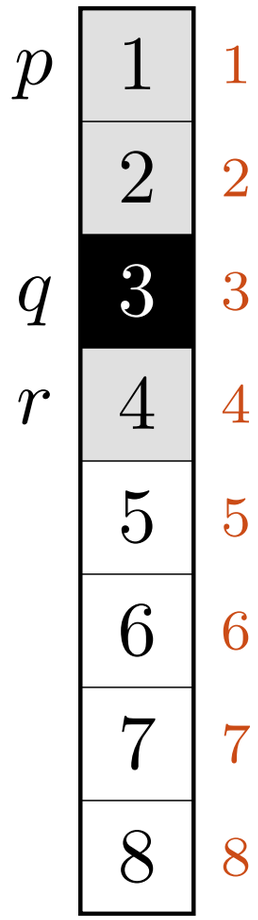
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i \;=\; 5, 4 \;\rightarrow\; -, 4$



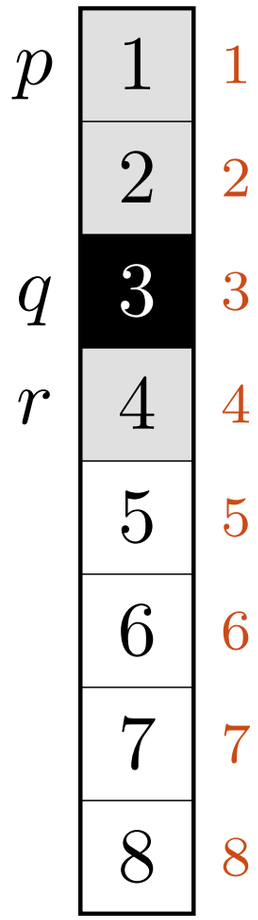
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i \;=\; 5, 4 \;\rightarrow\; -, 4$



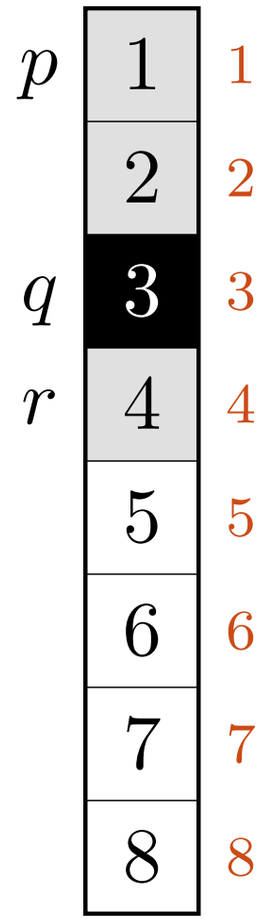
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i \;=\; 5, 4 \rightarrow 3, 4$

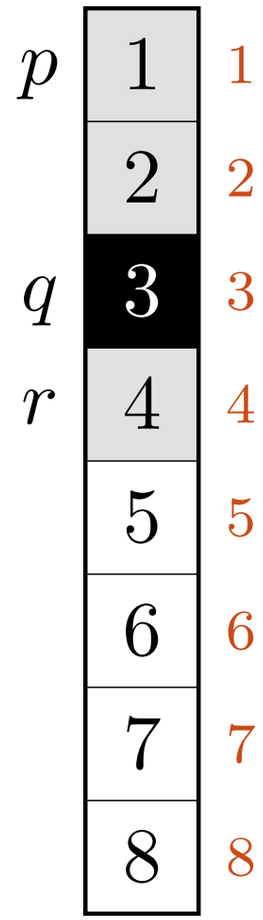


```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

$k, i \;=\; 5, 4 \rangle 3, 4$



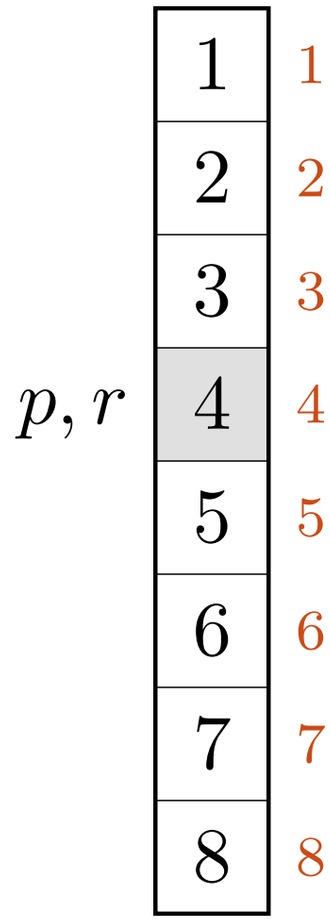
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```



$k, i \;=\; 5, 4 \rightarrow 3, 4$

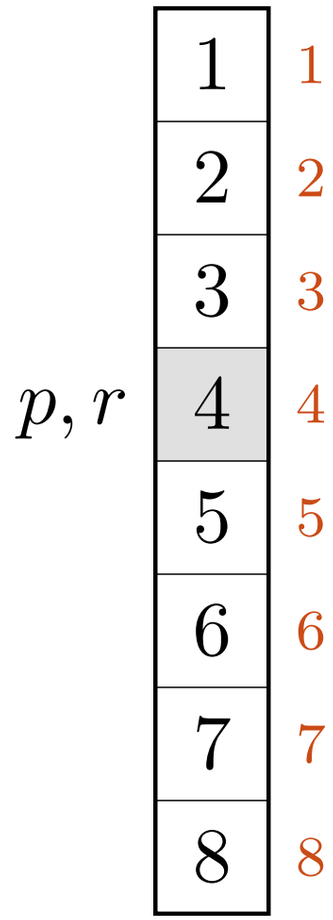
```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

k, i = 5, 4 › 3, 4 › –, 1



```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

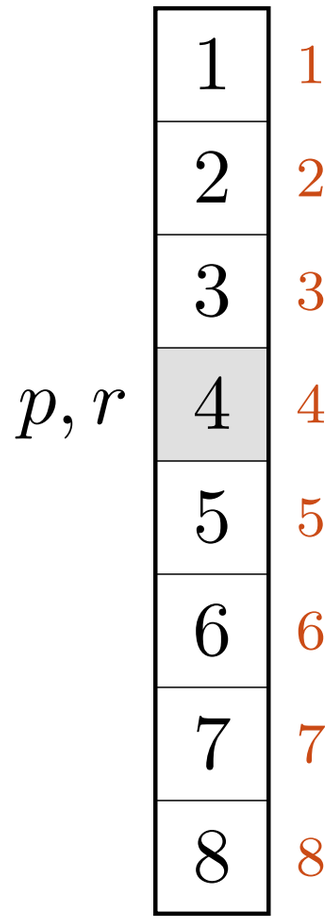
k, i = 5, 4 › 3, 4 › –, 1



```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

→ 4

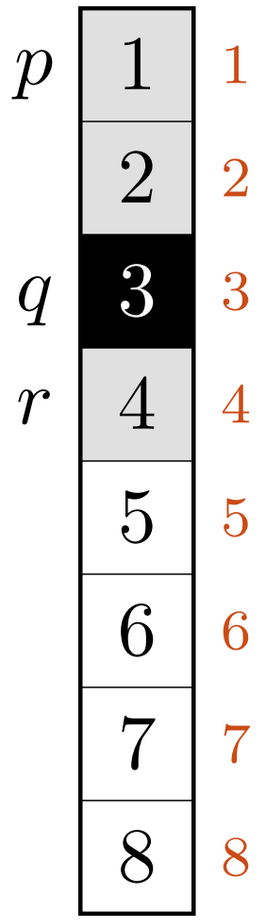
k, i = 5, 4 › 3, 4 › –, 1



```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

→ 4

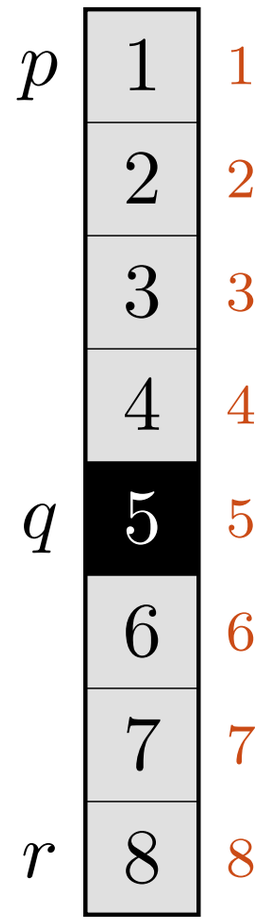
$k, i \;=\; 5, 4 \rangle 3, 4$



```
RS(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RS(A, p, q − 1, i)
9  else return RS(A, q + 1, r, i − k)
```

→ 4

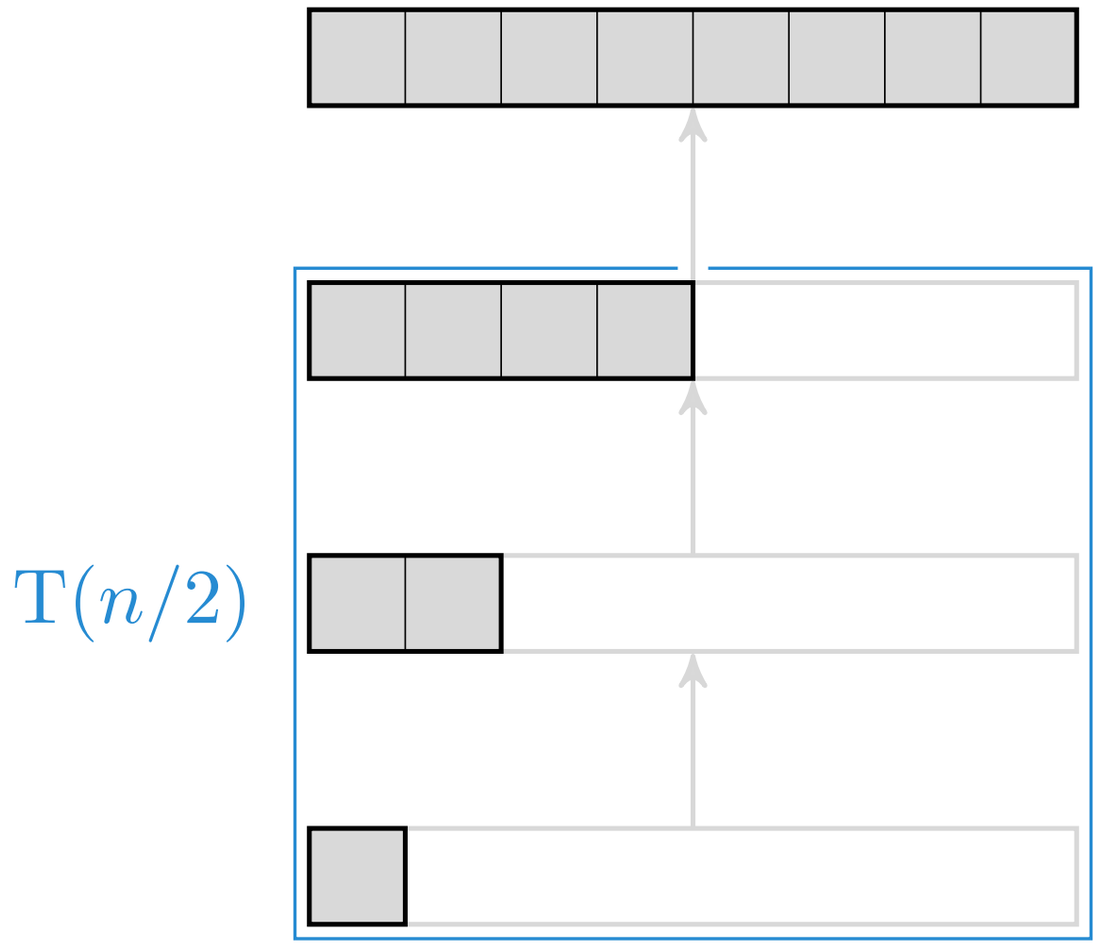
$k, i = 5, 4$



$$\mathrm{T}(n) = \mathrm{T}(n/2) + n$$

$$\mathrm{T}(n) = \mathrm{T}(n/2) + n$$

Ett rekursivt kall (størrelse $n/2$) pluss n operasjoner



$$\mathrm{T}(n) = n + \mathrm{T}(n/2) \tag{1}$$

$$\mathrm{T}(n) = n + \mathrm{T}(n/2) \tag{1}$$

$$\mathrm{T}(n) = n$$

$$+ n/2 \tag{1}$$

$$+ \mathrm{T}(n/2/2) \tag{2}$$

$$\begin{aligned}\mathrm{T}(n) &= n \\ &\quad + n/2 && (1) \\ &\quad + \mathrm{T}(n/4) && (2)\end{aligned}$$

$$\begin{aligned}
\mathrm{T}(n) &= n && \\
&\quad + n/2 && (1)\\
&\quad + n/4 && (2)\\
&\quad + \mathrm{T}(n/8) && (3)
\end{aligned}$$

$$
\begin{aligned}
\mathrm{T}(n) &= n && \\
&\quad + n/2 && (1) \\
&\quad + n/4 && (2) \\
&\quad + n/8 && (3) \\
&\quad \vdots && \vdots \\
&\quad + \mathrm{T}(n/2^{?}) && (?)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{T}(n) &= n && \\
&\quad + n/2 && (1) \\
&\quad + n/4 && (2) \\
&\quad + n/8 && (3) \\
&\quad \vdots && \vdots \\
&\quad + \mathrm{T}(n/2^{?}) && (\lg n)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{T}(n) &= n && \\
&\quad + n/2 && (1) \\
&\quad + n/4 && (2) \\
&\quad + n/8 && (3) \\
&\quad \vdots && \vdots \\
&\quad + \mathrm{T}(n/2^{\lg n}) && (\lg n)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{T}(n) &= n && \\
&\quad + n/2 && (1) \\
&\quad + n/4 && (2) \\
&\quad + n/8 && (3) \\
&\quad \vdots && \vdots \\
&\quad + \mathrm{T}(n/n) && (\lg n)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{T}(n) &= n && \\
&\quad + n/2 && (1) \\
&\quad + n/4 && (2) \\
&\quad + n/8 && (3) \\
&\quad \vdots && \vdots \\
&\quad + \mathrm{T}(1) && (\lg n)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{T}(n) &= n && \\
&\quad + n/2 && (1) \\
&\quad + n/4 && (2) \\
&\quad + n/8 && (3) \\
&\quad \vdots && \vdots \\
&\quad + 1 && (\lg n)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{T}(n) &= n && \\
&\quad + n/2 && (1) \\
&\quad + n/4 && (2) \\
&\quad + n/8 && (3) \\
&\quad \vdots && \vdots \\
&\quad + 1 && (\lg n)
\end{aligned}
$$

$$\mathrm{T}(n) = 2n - 1$$

Verifikasjon

Med substitusjon/induksjon

$$\mathrm{T}(n) = \mathrm{T}(n/2) + n$$

$$\mathrm{T}(n) = \mathrm{T}(n/2) + n$$

Gitt $\mathrm{T}(k) = 2k - 1$ for $k < n$, vis $\mathrm{T}(n) = 2n - 1$

$$\mathrm{T}(n) = \mathrm{T}(n/2) + n$$

Gitt $\mathrm{T}(k) = 2k - 1$ for $k < n$, vis $\mathrm{T}(n) = 2n - 1$

$$\begin{aligned}\mathrm{T}(n) &= \mathrm{T}(n/2) + n \\ &= \left(2 \cdot \frac{n}{2} - 1\right) + n\end{aligned}$$

Gitt $\mathrm{T}(k) = 2k - 1$ for $k < n$, vis $\mathrm{T}(n) = 2n - 1$

$$\begin{aligned}\mathrm{T}(n) &= \mathrm{T}(n/2) + n \\ &= \left(2 \cdot \frac{n}{2} - 1\right) + n \\ &= n - 1 + n\end{aligned}$$

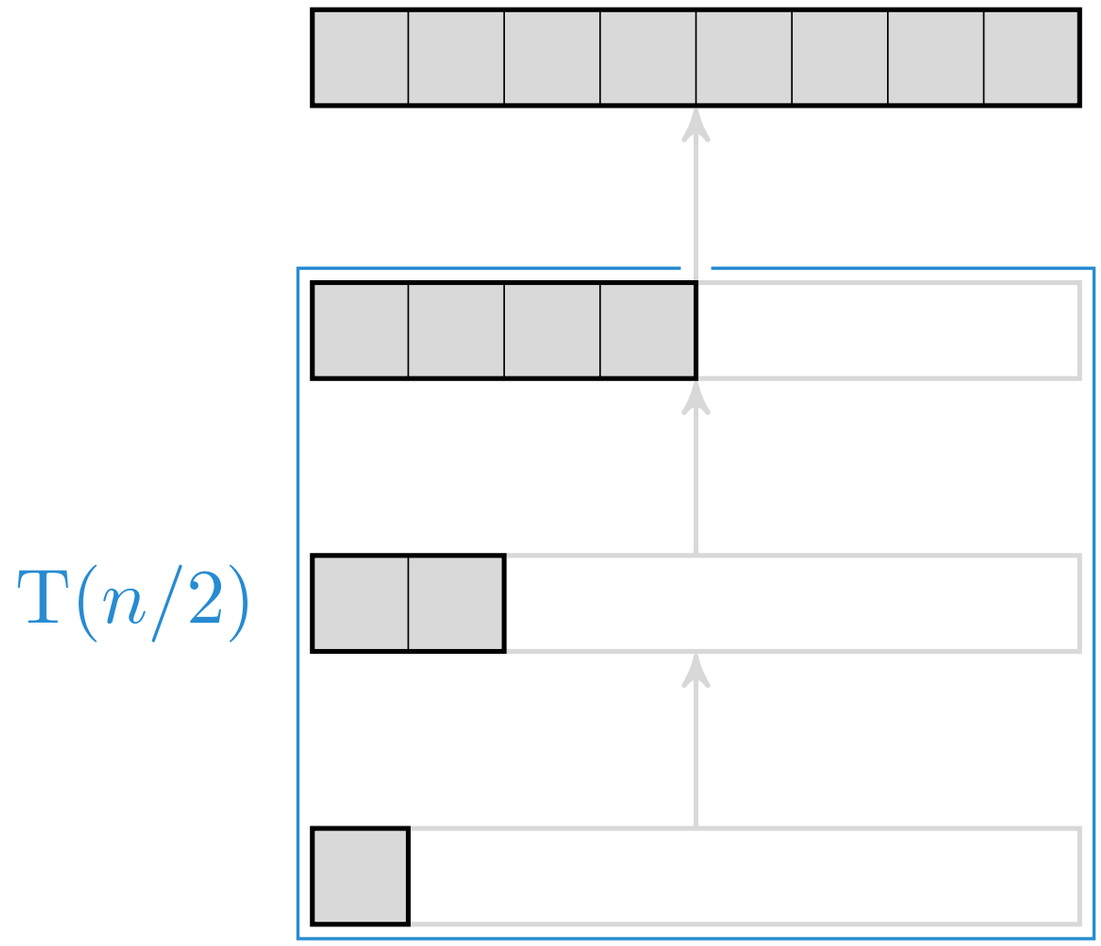
Gitt $\mathrm{T}(k) = 2k - 1$ for $k < n$, vis $\mathrm{T}(n) = 2n - 1$

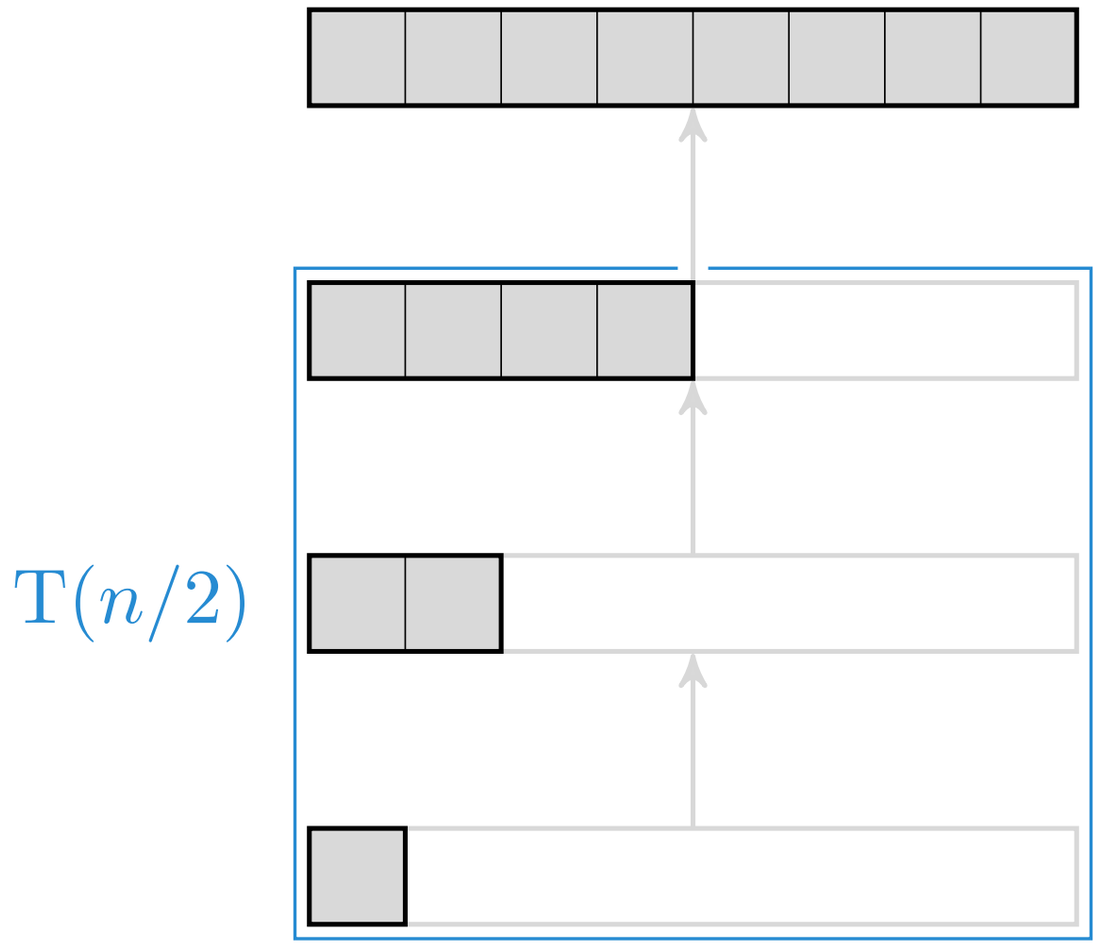
$$
\begin{aligned}
\mathrm{T}(n) &= \mathrm{T}(n/2) + n \\
&= \left(2 \cdot \frac{n}{2} - 1\right) + n \\
&= n - 1 + n \\
&= 2n - 1
\end{aligned}
$$

Gitt $\mathrm{T}(k) = 2k - 1$ for $k < n$, vis $\mathrm{T}(n) = 2n - 1$

$$\begin{aligned}
\mathrm{T}(n) &= \mathrm{T}(n/2) + n \\
&= \left(2 \cdot \frac{n}{2} - 1\right) + n \\
&= n - 1 + n \\
&= 2n - 1
\end{aligned}$$

Gitt $\mathrm{T}(k) = 2k - 1$ for $k < n$, vis $\mathrm{T}(n) = 2n - 1$





$\mathrm{T}(n/2)$

$$\mathrm{T}(n) = 2n - 1$$

$$\mathrm{T_W}(n) = \Theta(n^2)$$

$$\mathrm{T_A}(n) = \Theta(n)$$

$$\mathrm{T_B}(n) = \Theta(n)$$

I verste tilfelle: Pivot alene, akkurat som QUICKSORT

$$\mathrm{T_W}(n) = \Theta(n^2)$$

$$\mathrm{T_A}(n) = \Theta(n)$$

$$\mathrm{T_B}(n) = \Theta(n)$$

Forventet: $\Theta(n + n/2 + \cdots + 4 + 2 + 1)$ operasjoner

Gjenta suksessen!

... denne gang for WC

6:6

Select

JOURNAL OF COMPUTER AND SYSTEM SCIENCES 7, 448–461 (1973)

Time Bounds for Selection*

MANUEL BLUM, ROBERT W. FLOYD, VAUGHAN PRATT,
RONALD L. RIVEST, AND ROBERT E. TARJAN

Department of Computer Science, Stanford University, Stanford, California 94305

Received November 14, 1972

258

The number of comparisons required to select the i-th smallest of n numbers is shown

Trenger god pivot

Bruk … Select?

«Median av medianer»

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4        if A[j] ≤ x
5              i = i + 1
6              exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Bruker $A[r]$ som pivot

```
PARTITION-AROUND(A, p, r, x)
1  i = 1
2  while A[i] ≠ x
3        i = i + 1
4  exchange A[r] and A[i]
5  return PARTITION(A, p, r)
```

Får pivot oppgitt! Beskrevet uten kode i boka

```
RAND-SEL(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RAND-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return RAND-SEL(A, p, q − 1, i)
9  else return RAND-SEL(A, q + 1, r, i − k)
```

La oss bytte ut alle RANDOMIZED-tingene

```
SELECT(A, p, r, i)
1  if p == r
2        return A[p]
3  q = GOOD-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]
7  elseif i < k
8        return SELECT(A, p, q − 1, i)
9  else return SELECT(A, p + 1, r, i − k)
```

Partisjoneringen er kjernen: Finn en god pivot!

GOOD-PARTITION(A, p, r)

A tabell
p venstre
r høyre

Velger pivot nøye. Beskrevet uten kode i boka

GOOD-PARTITION(A, p, r)
1 $\quad n = r - p + 1$

A tabell
p venstre
r høyre
n antall

$n = \mathrm{A}[p\,..\,r].length$

GOOD-PARTITION(A, p, r)
1 $n = r - p + 1$
2 $m = \lceil n/5 \rceil$

A tabell
p venstre
r høyre
n antall
m grupper

Vi vil dele $\mathrm{A}[p\,..\,r]$ i grupper på fem

```
GOOD-PARTITION(A, p, r)
1  n = r − p + 1
2  m = ⌈n/5⌉
3  create B[1..m]
```

A	tabell
p	venstre
r	høyre
n	antall
m	grupper
B	medianer

Vil inneholde medianen for hver av femmergruppene

```
GOOD-PARTITION(A, p, r)
1  n = r − p + 1
2  m = ⌈n/5⌉
3  create B[1..m]
4  for i = 0 to m − 1
```

A	tabell
p	venstre
r	høyre
n	antall
m	grupper
B	medianer
i	gruppe − 1

For hver femmergruppe ...

```
GOOD-PARTITION(A, p, r)
1  n = r − p + 1
2  m = ⌈n/5⌉
3  create B[1..m]
4  for i = 0 to m − 1
5      q = p + 5i
```

A	tabell
p	venstre
r	høyre
n	antall
m	grupper
B	medianer
i	gruppe − 1
q	v., gruppe

Gruppen starter med $A[q]$

GOOD-PARTITION(A, p, r)

```
1  n = r − p + 1
2  m = ⌈n/5⌉
3  create B[1..m]
4  for i = 0 to m − 1
5      q = p + 5i
6      sort A[q..q + 4]
```

A	tabell
p	venstre
r	høyre
n	antall
m	grupper
B	medianer
i	gruppe − 1
q	v., gruppe

For å finne medianen i gruppen. Bruk f.eks. INSERTION-SORT

```
GOOD-PARTITION(A, p, r)
1  n = r − p + 1
2  m = ⌈n/5⌉
3  create B[1..m]
4  for i = 0 to m − 1
5      q = p + 5i
6      sort A[q..q + 4]
7      B[i] = A[q + 3]
```

A	tabell
p	venstre
r	høyre
n	antall
m	grupper
B	medianer
i	gruppe − 1
q	v., gruppe

Sett medianen inn i B

GOOD-PARTITION(A, p, r)
1 $n = r - p + 1$
2 $m = \lceil n/5 \rceil$
3 create $\mathrm{B}[1\,..\,m]$
4 **for** $i = 0$ **to** $m - 1$
5 $\quad q = p + 5i$
6 $\quad$ sort $\mathrm{A}[q\,..\,q + 4]$
7 $\quad \mathrm{B}[i] = \mathrm{A}[q + 3]$
8 $x =$ SELECT$(\mathrm{B}, 1, m, \lfloor m/2 \rfloor)$

A	tabell
p	venstre
r	høyre
n	antall
m	grupper
B	medianer
i	gruppe – 1
q	v., gruppe
x	splitt

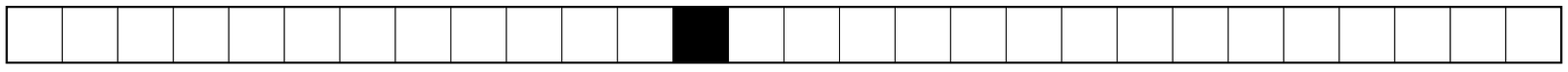
Finn medianen av medianene . . . med SELECT!

```
GOOD-PARTITION(A, p, r)
1  n = r − p + 1
2  m = ⌈n/5⌉
3  create B[1..m]
4  for i = 0 to m − 1
5        q = p + 5i
6        sort A[q..q + 4]
7        B[i] = A[q + 3]
8  x = SELECT(B, 1, m, ⌊m/2⌋)
9  return PARTITION-AROUND(A, p, r, x)
```

A	tabell
p	venstre
r	høyre
n	antall
m	grupper
B	medianer
i	gruppe − 1
q	v., gruppe
x	splitt

Bruk medianen av medianene som pivot

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5}\right\rceil\right\rceil - 2\right) \geqslant \frac{3n}{10} - 6$$



Hvor mange har vi på hver side av pivot?

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5}\right\rceil \right\rceil - 2\right) \geqslant \frac{3n}{10} - 6$$

Vi har delt inn i $\lceil n/5 \rceil$ grupper

$$3\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil\right\rceil-2\right)\geqslant\frac{3n}{10}-6$$

Minst halvparten bidrar med 3 verdier mindre enn pivot

$$3\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil\right\rceil - 2\right) \geqslant \frac{3n}{10} - 6$$

Unntatt én, om $\lceil n/5 \rceil > n/5$ …

$$3\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil\right\rceil - 2\right) \geqslant \frac{3n}{10} - 6$$

. . . og unntatt gruppen med pivot

$$3\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil\right\rceil-2\right)\geqslant\frac{3n}{10}-6$$

Vi har altså minst så mange elementer mindre enn pivot …

$$3\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil\right\rceil-2\right)\geqslant\frac{3n}{10}-6$$

. . . og så mange som er større

mlh@ntnu.no

$$\mathrm{T}(n) = \Theta(n)$$

Garantert en viss prosent på hver side av pivot

1. Sorteringsgrensen
2. Tellesortering
3. Radikssortering
4. Bøttesortering
5. Randomized Select
6. Select

Bonusmateriale

$$\mathrm{T}(n) \leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + \mathrm{O}(n)$$

$$\mathrm{T}(n) \leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + \mathrm{O}(n)$$

Rekursiv bruk av SELECT for å finne median av $\lceil n/5 \rceil$ gruppemedianer

$$\mathrm{T}(n) \leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + \mathrm{O}(n)$$

Rekursivt kall i største «halvdel», med $n - (3n/10 - 6)$ elementer

$$\mathrm{T}(n) \leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + \mathrm{O}(n)$$

PARTITION, gruppering, sortering av grupper, etc.

$$\mathrm{T}(n) \leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + an$$

Her er a konstanten fra O-notasjonen ($n_0 = 1$)

$$\mathrm{T}(n) \leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + an$$

Vil vise $\mathrm{T}(n) \leq cn$ med substitusjon

$$\begin{aligned}\mathrm{T}(n) &\leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + an \\ &\leqslant c\lceil n/5 \rceil + c(7n/10 + 6) + an\end{aligned}$$

Induksjonshypotese: $\mathrm{T}(k) \leq ck$ for alle $k < n$

$$
\begin{aligned}
\mathrm{T}(n) &\leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + an \\
&\leqslant c\lceil n/5 \rceil + c(7n/10 + 6) + an \\
&\leqslant cn/5 + c + 7cn/10 + 6c + an
\end{aligned}
$$

Vanlig algebra/«ligningsløsning»

$$
\begin{aligned}
\mathrm{T}(n) &\leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + an \\
&\leqslant c\lceil n/5 \rceil + c(7n/10 + 6) + an \\
&\leqslant cn/5 + c + 7cn/10 + 6c + an \\
&= 9cn/10 + 7c + an
\end{aligned}
$$

Vanlig algebra/«ligningsløsning»

$$
\begin{aligned}
\mathrm{T}(n) &\leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + an \\
&\leqslant c\lceil n/5 \rceil + c(7n/10 + 6) + an \\
&\leqslant cn/5 + c + 7cn/10 + 6c + an \\
&= 9cn/10 + 7c + an \\
&= cn + (an - (cn/10 + 7c))
\end{aligned}
$$

Vanlig algebra/«ligningsløsning»

$$
\begin{aligned}
\mathrm{T}(n) &\leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + an \\
&\leqslant c\lceil n/5 \rceil + c(7n/10 + 6) + an \\
&\leqslant cn/5 + c + 7cn/10 + 6c + an \\
&= 9cn/10 + 7c + an \\
&= cn + (an - (cn/10 + 7c)) \\
&\leqslant cn
\end{aligned}
$$

. . . hvis c er stor nok, så $an \leqslant cn/10 - 7c$

$$
\begin{aligned}
\mathrm{T}(n) &\leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + an \\
&\leqslant c\lceil n/5 \rceil + c(7n/10 + 6) + an \\
&\leqslant cn/5 + c + 7cn/10 + 6c + an \\
&= 9cn/10 + 7c + an \\
&= cn + (an - (cn/10 + 7c)) \\
&\leqslant cn
\end{aligned}
$$

F.eks. $c \geqslant 20a$, hvis $n \geqslant 140$. (La $n < 140$ være grunntilfellet!)

$$
\begin{aligned}
\mathrm{T}(n) &\leqslant \mathrm{T}(\lceil n/5 \rceil) + \mathrm{T}(7n/10 + 6) + an \\
&\leqslant c\lceil n/5 \rceil + c(7n/10 + 6) + an \\
&\leqslant cn/5 + c + 7cn/10 + 6c + an \\
&= 9cn/10 + 7c + an \\
&= cn + (an - (cn/10 + 7c)) \\
&\leqslant cn
\end{aligned}
$$

Og dermed er induksjonstrinnet komplett!