מיונים
Sorts

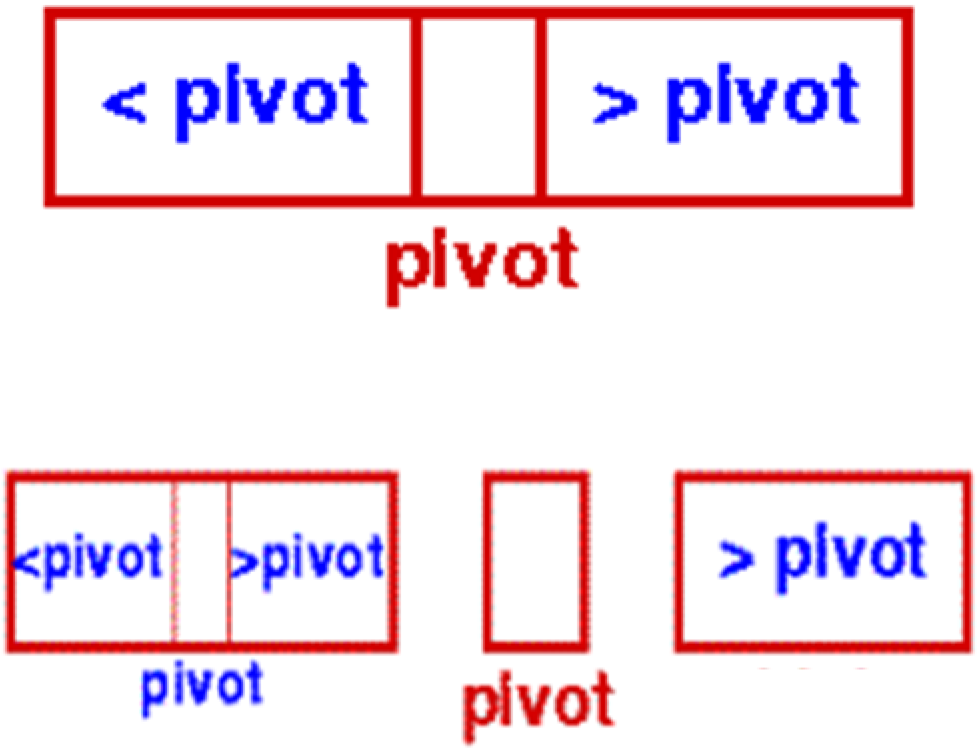
מה נלמד?

- מיון מהיר Quicksort
- חסם תחתון על זמן ריצה של מיון מבוסס השוואות
- מיונים בזמן ליניארי
  - Counting sort
  - Radix Sort

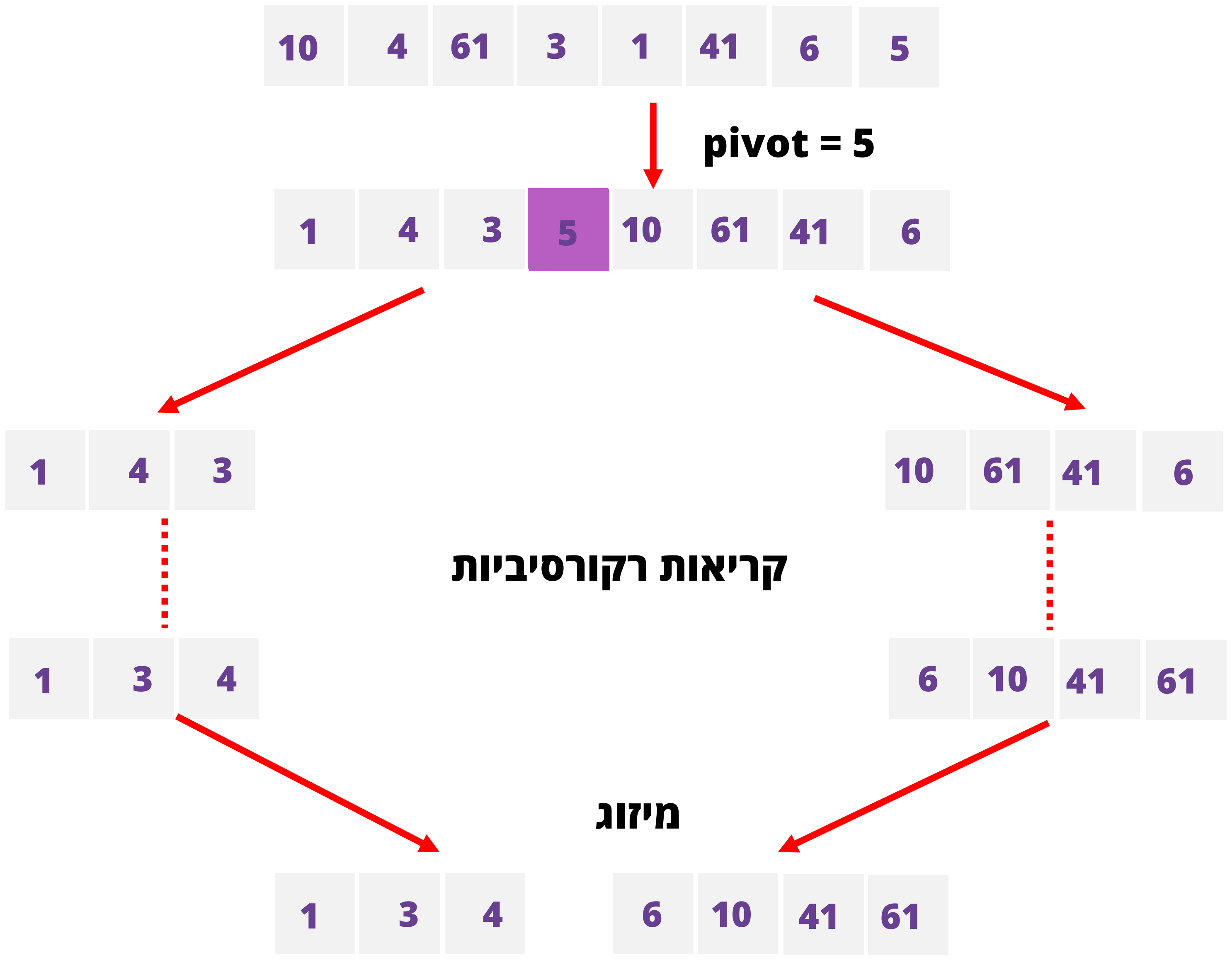
מיון מהיר
Quicksort

- פותח ב-1960 על ידי C.A.R.Hoare
- פעול לפי אסטרטגיית הפרד ומשול
- זמן הריצה במקרה הגרוע ביותר: $O(n^2)$
- זמן ריצה צפוי: $O(n\log n)$
- קבועים המסתתרים ב- $O(n\log n)$ קטנים
- ממיין במקום (in-place)

Quicksort
הרעיון

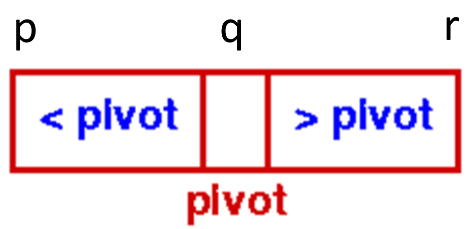


QuickSort


10 4 61 3 1 41 6 5
pivot = 5
1 4 3 5 10 61 41 6
1 4 3
10 61 41 6
קריאות רקורסיביות
1 3 4
6 10 41 61
מיזוג
1 3 4
6 10 41 61

Quicksort

- To sort the sub-array A[p . . r]:
  - **Divide (חלק)**:
    - Partition A[p . . r] into A[p . . q − 1] and A[q + 1 . . r], such that
      - each element in A[p . . q − 1] is ≤ A[q] and
      - A[q] is ≤ each element in A[q + 1 . . r].
  - **Conquer (משול)**:
    - Sort the two sub-arrays by recursive calls to Quicksort.
  - **Combine (צרף)**:
    - No work is needed to combine the sub-arrays, because they are sorted in place.



Quicksort

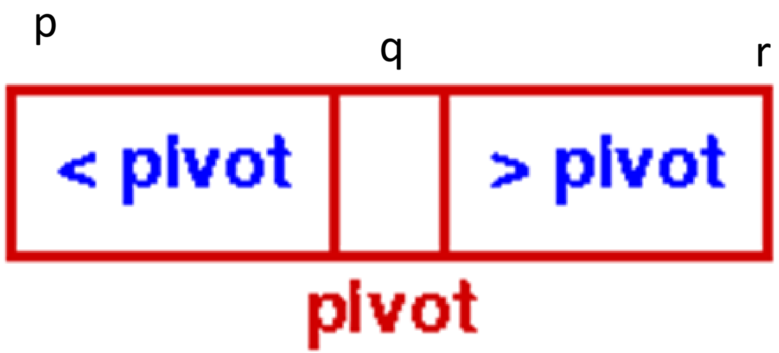
Quicksort (A, p, r)

if $(p < r)$ **then**

q = **partition** (A, p, r)

Quicksort $(A, p, q - 1)$

Quicksort $(A, q + 1, r)$



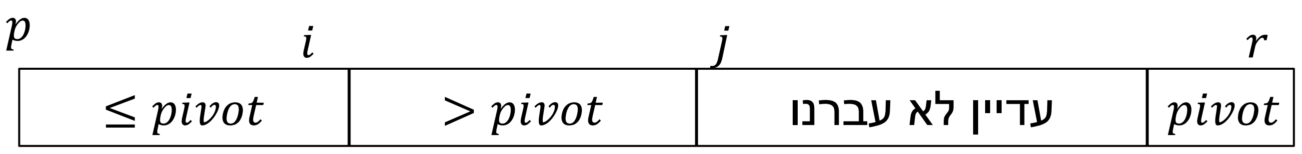
- **Initial call** is Quicksort $(A, 1, n)$.

Partition

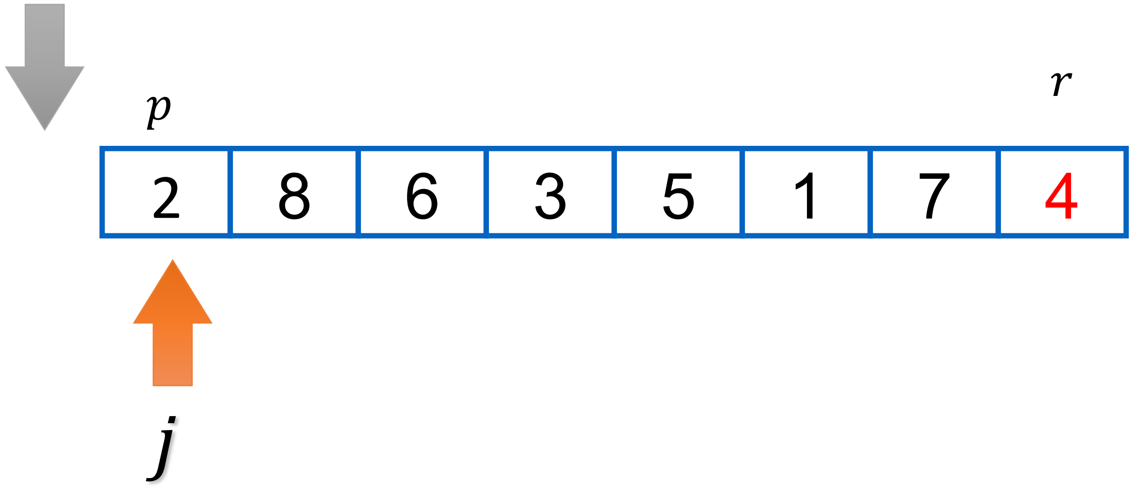
p	i	j	r
$\leq pivot$	$> pivot$	עדיין לא עברנו	$pivot$

- $A[r]$ = pivot
- All entries in $A[p\,..\,i\,]$ are ≤ pivot
- All entries in $A[i+1\,..\,j-1]$ are > pivot
- All entries in $A[j\,..\,r-1]$ are not yet examined.

Partition

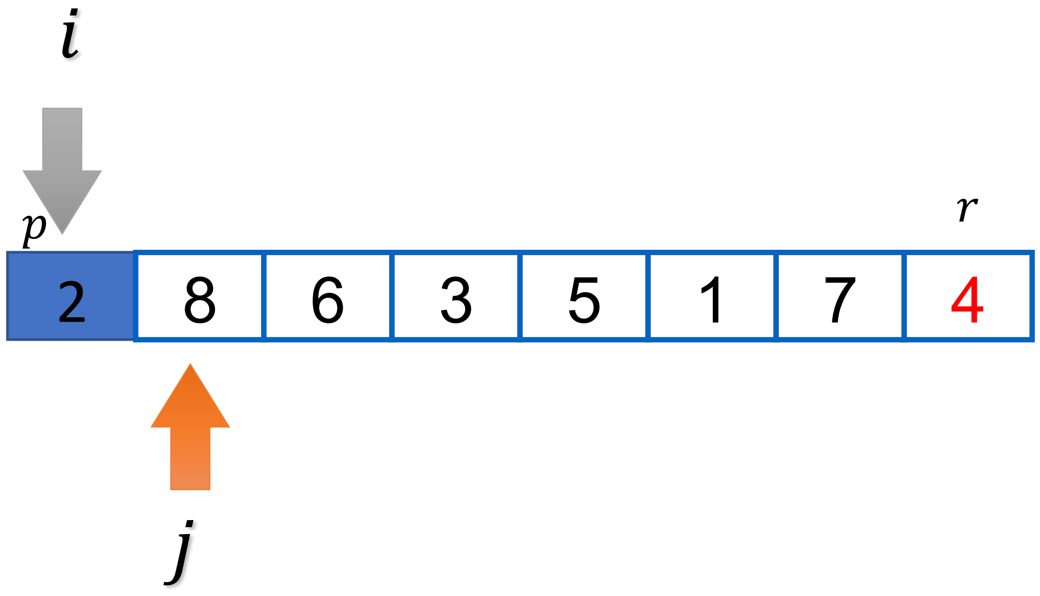


$$i = p - 1$$



Partition

p i	j		r
$\leq pivot$	$> pivot$	עדיין לא עברנו	$pivot$



Partition

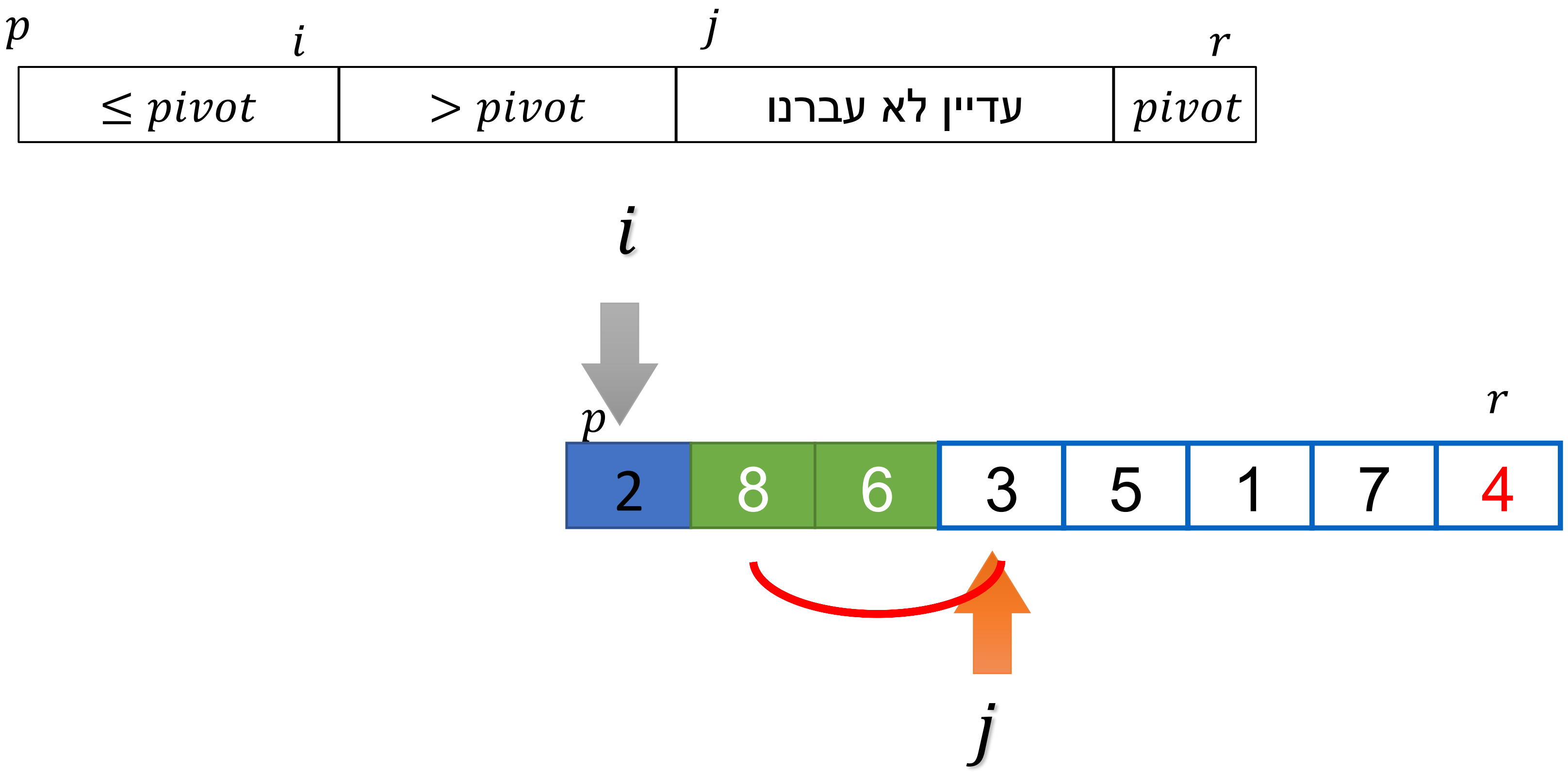
p $\quad\quad\quad$ i	j		r
$\leq pivot$	$> pivot$	עדיין לא עברנו	$pivot$

i

p							r
2	8	6	3	5	1	7	4

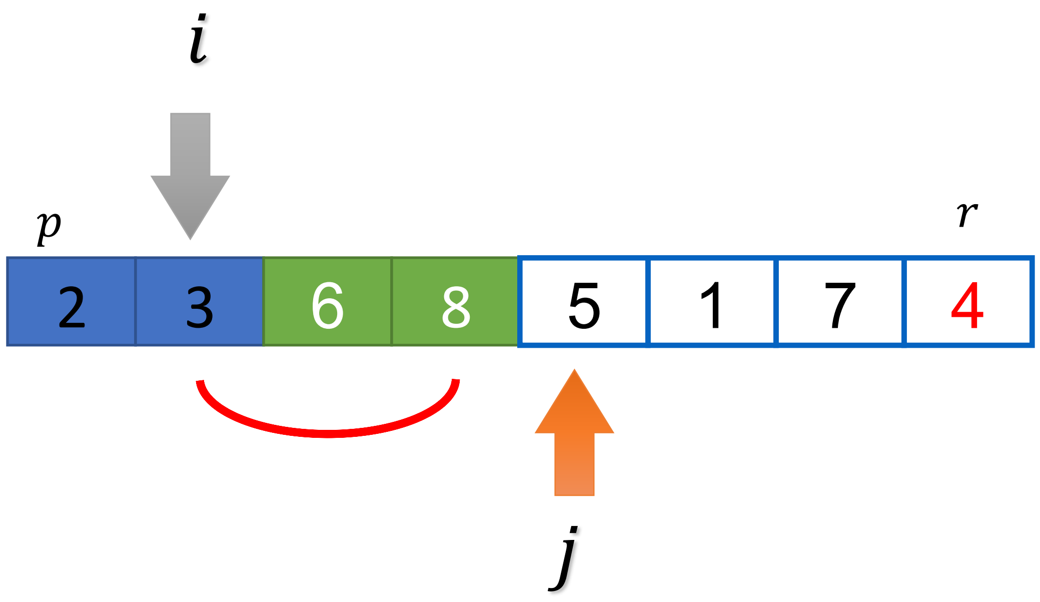
j

Partition

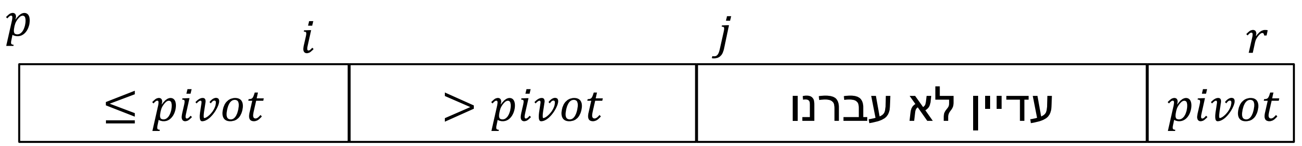


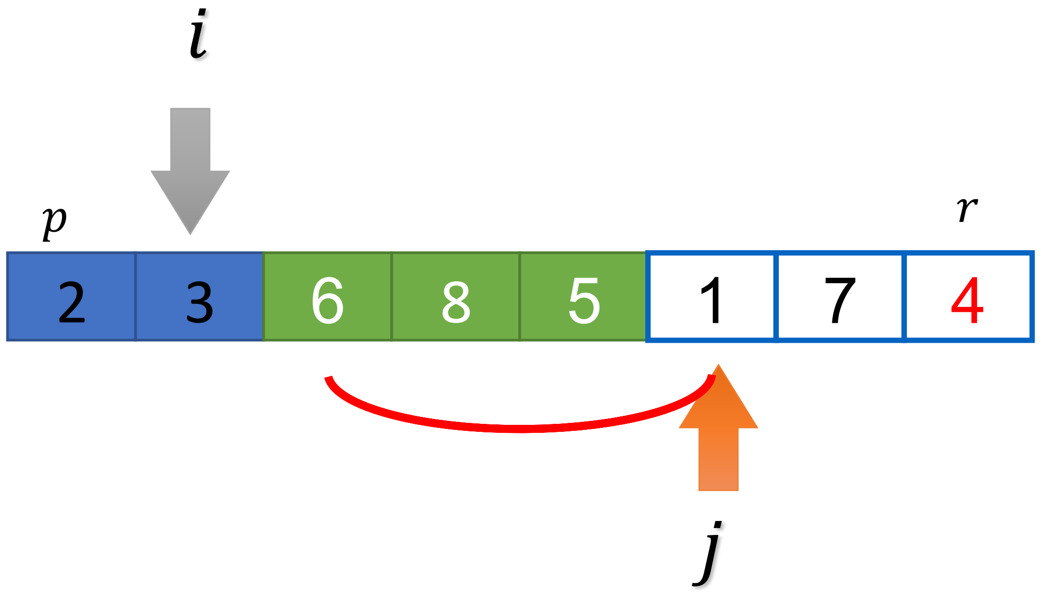
Partition

p	i	j	r
$\leq pivot$	$> pivot$	עדיין לא עברנו	$pivot$



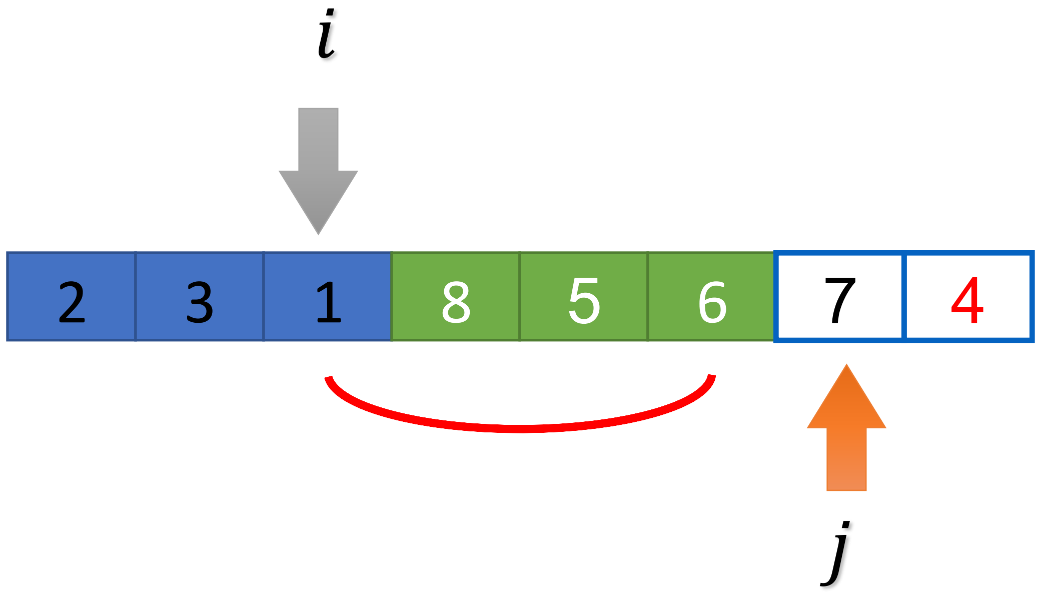
Partition



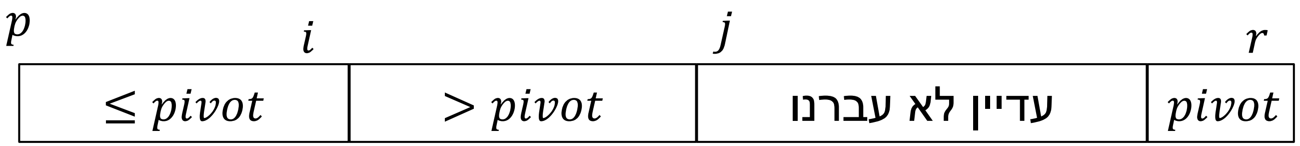


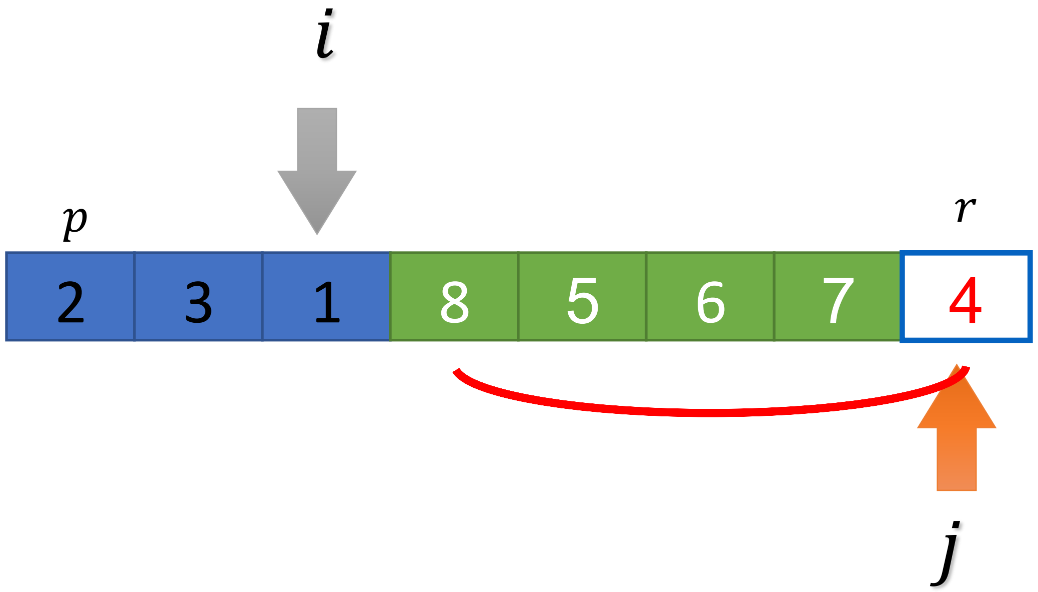
Partition

p i		j	r
$\leq pivot$	$> pivot$	עדיין לא עברנו	$pivot$

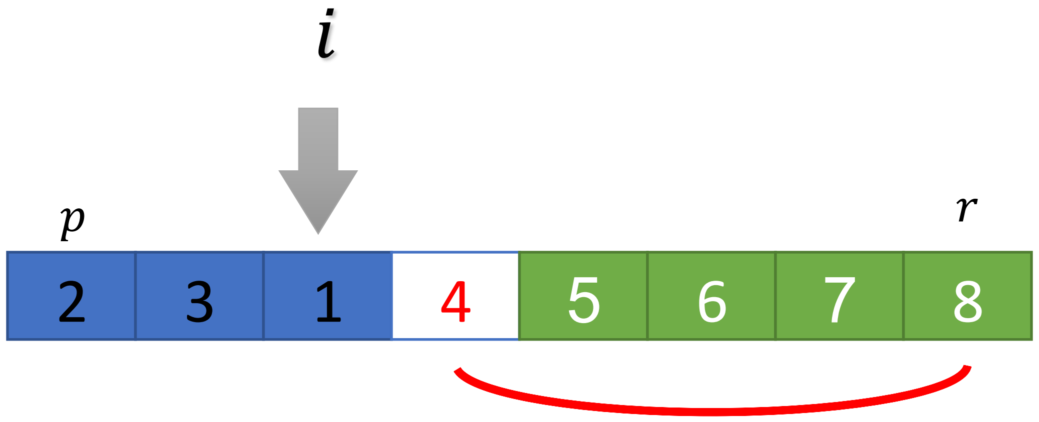


Partition





Partition



Partition

partition (A, p, r)

$i = p - 1$

for $(j = p$ **to** $r - 1)$

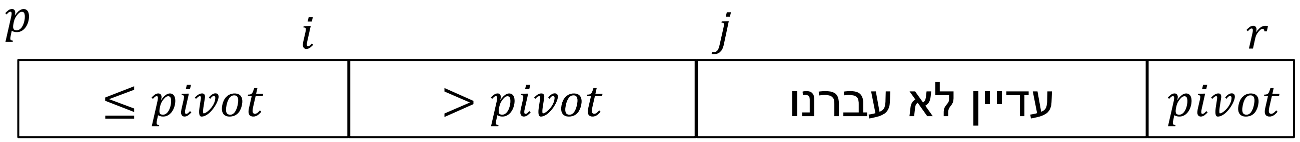
if $(A[j] \leq A[r])$

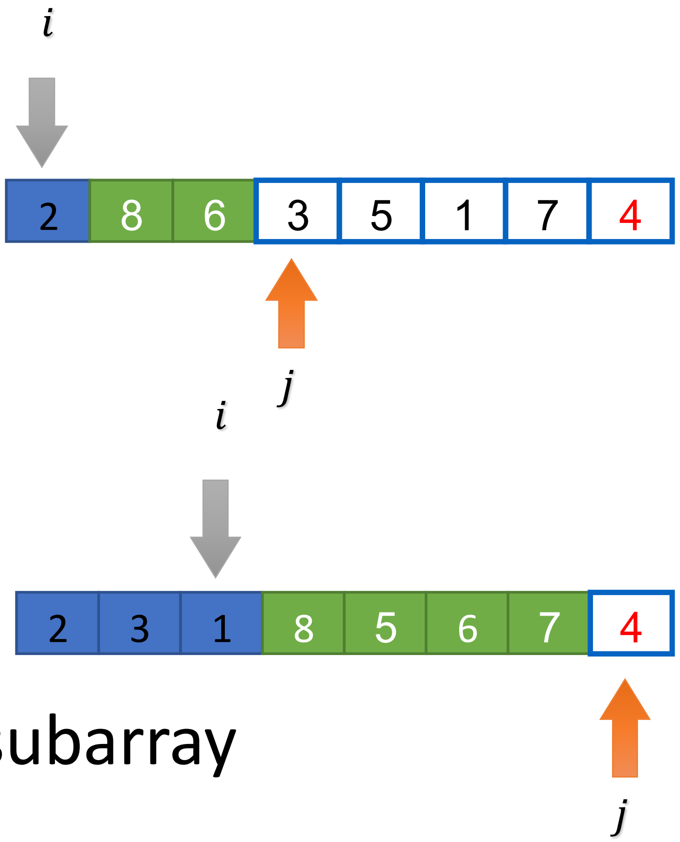
$i = i + 1$

exchange $A[i]$ **with** $A[j]$

exchange $A[i + 1]$ **with** $A[r]$

return $i + 1$





- **Complexity**: $\Theta(n)$ to partition an n-element subarray

זמן ריצה של מיון מהיר
Performance

- The running time of Quicksort depends on the partitioning of the sub-arrays:
  - If the sub-arrays are balanced, then Quicksort can run as fast as Mergesort.
  - If they are unbalanced, then Quicksort can run as slowly as Insertion sort.

מקרה הגרוע ביותר
Worst Case

- Occurs when the sub-arrays are **completely unbalanced** every time.
  - When Quicksort takes a sorted array as input.
- Have 0 elements in one sub-array and $n - 1$ elements in the other sub-array.
- Get the recurrence

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) . \end{aligned}$$

- Same running time as insertion sort.

מקרה הטוב ביותר
Best Case

- Occurs when the sub-arrays are **completely balanced** every time.
- Each sub-array has $\approx\ n/2$ elements.
- Get the recurrence

$$
\begin{aligned}
T(n) &= 2T(n/2) + \Theta(n) \\
&= \Theta(n \lg n)\,.
\end{aligned}
$$

זמן ריצה צפוי
expected running time

- If the pivot is the k'th element, the runtime is $T(n)=T(k)+T(n-1-k)+cn$
- The probability that the pivot is the k'th element is $1/n$
- Therefore, the expected runtime can be expressed as:

$$T(n)=cn+\frac{1}{n}\sum_{k=0}^{n-1}\left(T(k)+T(n-1-k)\right)$$

זמן ריצה צפוי
expected running time

$$T(n) = cn + \frac{1}{n}\sum_{k=0}^{n-1}\left(T(k) + T(n-1-k)\right)$$

$$nT(n) = cn^2 + 2\sum_{k=0}^{n-1} T(k)(*)$$

$$(n-1)T(n-1) = c(n-1)^2 + 2\sum_{k=0}^{n-2} T(k)(**)$$

$$nT(n) - (n-1)T(n-1) = c(2n-1) + 2T(n-1)$$

$$nT(n) = c(2n-1) + (n+1)T(n-1)$$

$$\frac{T(n)}{n+1} = \frac{c(2n-1)}{n(n+1)} + \frac{T(n-1)}{n} \le \frac{2nc}{n(n+1)} + \frac{T(n-1)}{n} = \frac{2c}{n+1} + \frac{T(n-1)}{n}$$

$$\frac{T(n)}{n+1} \le \frac{2c}{n+1} + \frac{T(n-1)}{n} \le \frac{2c}{n+1} + \frac{2c}{n} + \frac{T(n-2)}{n-1} \le \ldots \le 2c\sum_{i=3}^{n+1}\frac{1}{i} + \frac{T(1)}{2}$$

$$\boxed{T(n) = O(n\log n)}$$

זמן ריצה של מיון מהיר
סיכום

- זמן הריצה במקרה הגרוע ביותר: $O(n^2)$
- זמן ריצה צפוי: $O(n \log n)$
- קבועים המסתתרים ב- $O(n \log n)$ קטנים
- ממיין במקום (in-place)

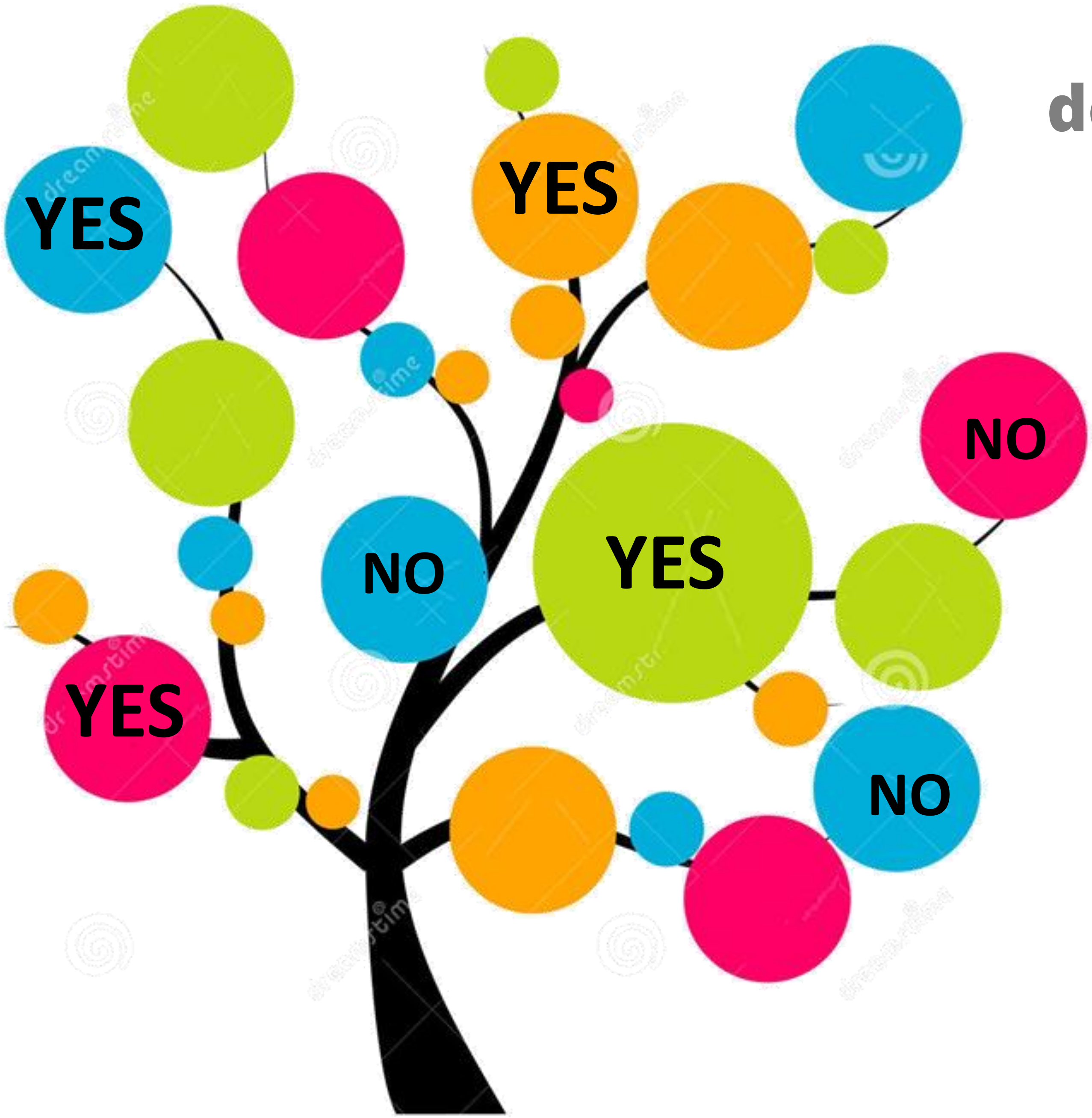
MergeSort	$O(n\log n)$
HeapSort	$O(n\log n)$
Insertion Sort	$O(n^2)$
QuickSort	$O(n\log n)$ בהסתברות גבוהה

?

האם קיים אלגוריתם מיון מבוסס השוואות שזמן ריצתו במקרה הגרוע קטן מ- $O(n\log n)$**?**

מיונים מבוססים השוואה
Comparisons based sorts

- מיון מבוסס השוואות: הוא מיון שבו מידע על סדר האיברים מתקבל אך ורק ע"י השוואת שני איברים.
- כל המיונים שראינו עד עכשיו הם מבוססי השוואות.
- סיבוכיות הזמן של כל מיון מבוסס השוואות היא $\Omega(n\log n)$.



עץ החלטה – decision tree

- נייצג אלגוריתם מבוסס השוואות בעזרת עץ החלטה - decision tree

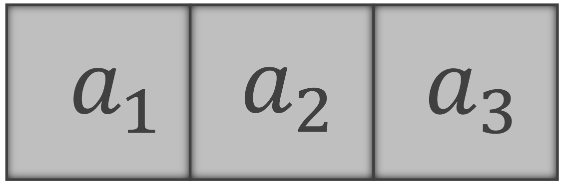
$$>, <, \geq, \leq$$

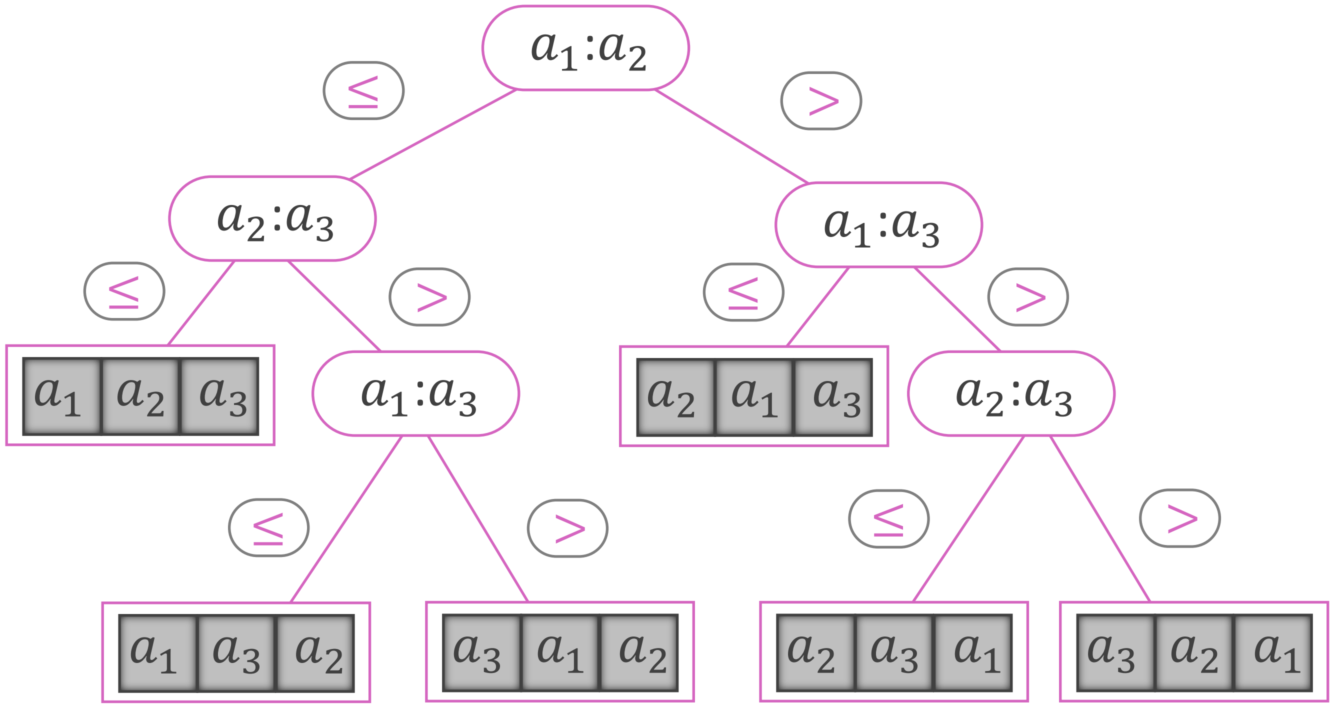
- בלי הגבלת הכלליות, נניח שכל המספרים שונים זה מזה, ונגביל את עצמינו להשוואה "$\leq$"

עץ החלטה – decision tree
מיון הכנסה של מערך בגודל n=3

a_1	a_2	a_3

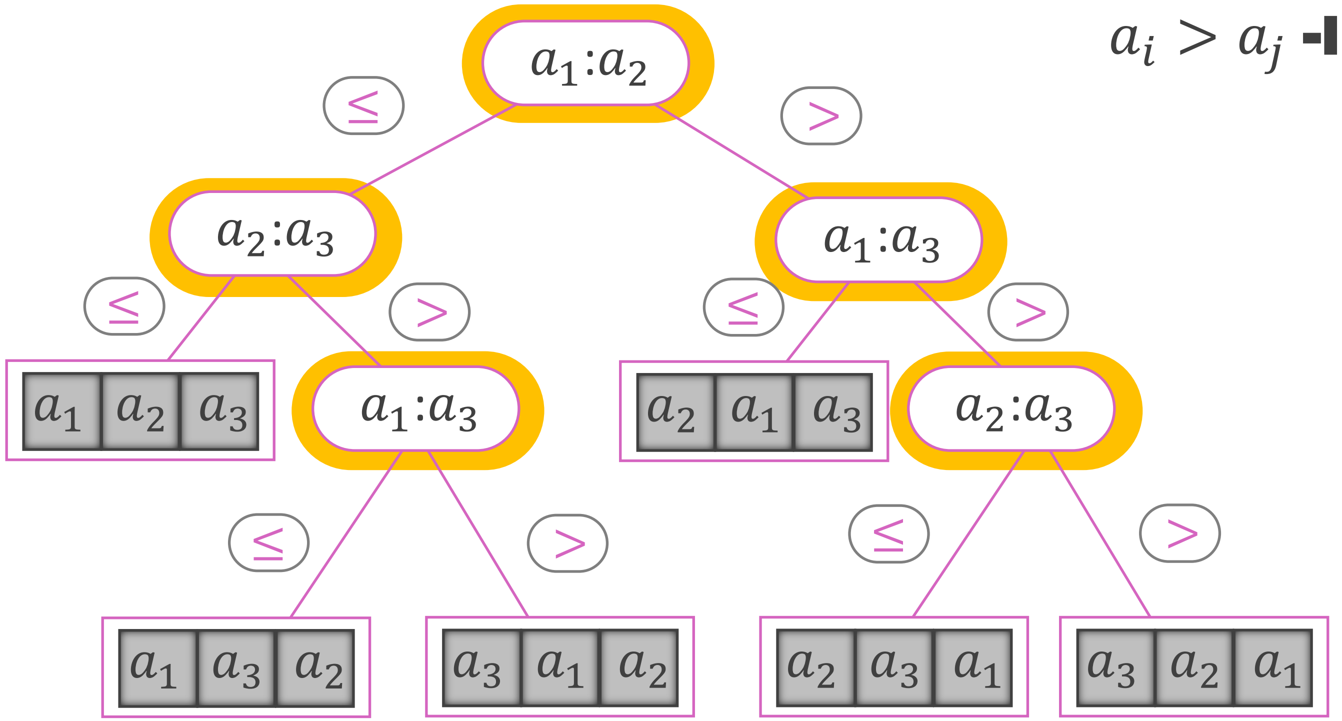
עץ החלטה – decision tree
מיון הכנסה של מערך בגודל n=3





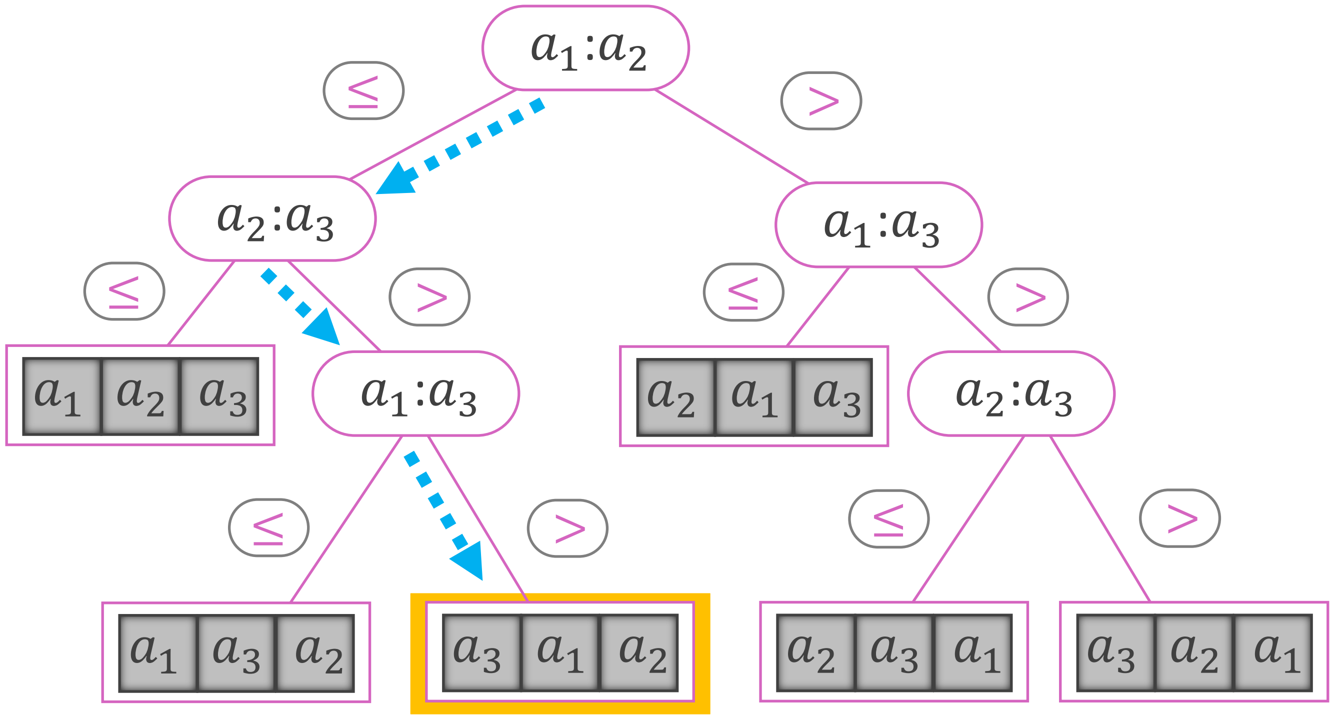
עץ החלטה – decision tree

- כל צומת פנימית $a_i : a_j$ מסמנת השוואה בין a_i לבין a_j
- תת-העץ השמאלי מתאר החלטות במידה ו- $a_i \leq a_j$
- תת-העץ הימני מתאר החלטות במידה ו- $a_i > a_j$
- כל עלה הוא תמורה של איברי המערך



עץ החלטה – decision tree

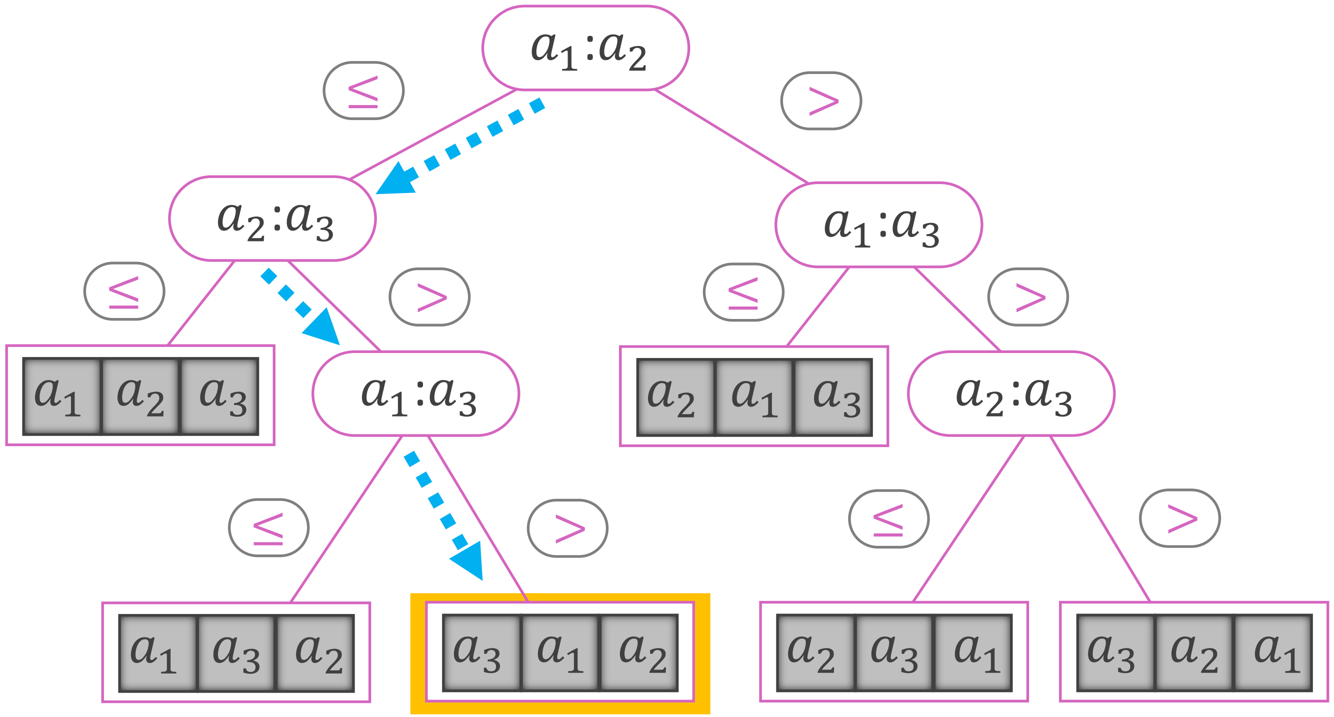
- כל מסלול משורש לעלה מתאר סידרת השוואות אפשרית שמבצע אלגוריתם על מערך הקלט בגודל n
- המסלול הכי ארוך מתאר מספר השוואות המרבי שאלגוריתם מבצע



עץ החלטה – decision tree

- המסלול הכי ארוך מתאר מספר השוואות המרבי שאלגוריתם מבצע

- גובה של עץ ההחלטה קובע חסם תחתון על זמן ריצה במקרה הגרוע של אלגוריתם מיון

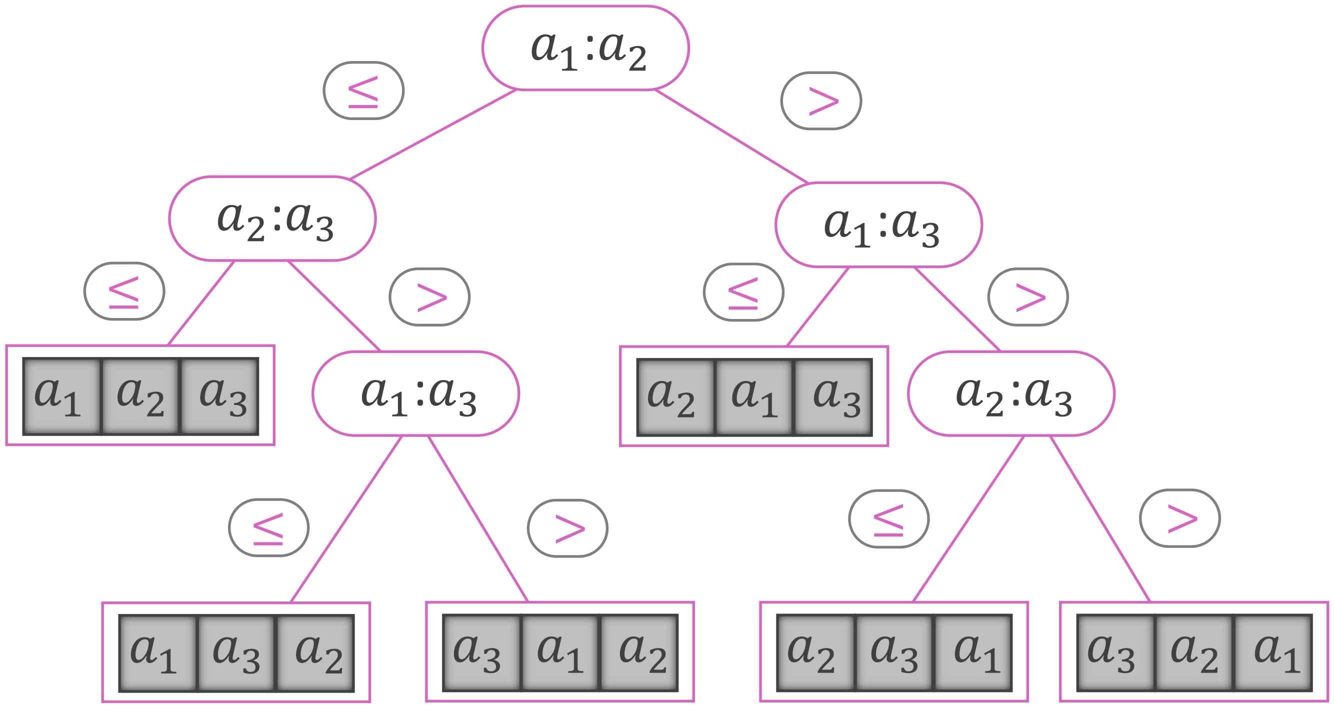


משפט: החסם התחתון של מיון מבוסס השוואות

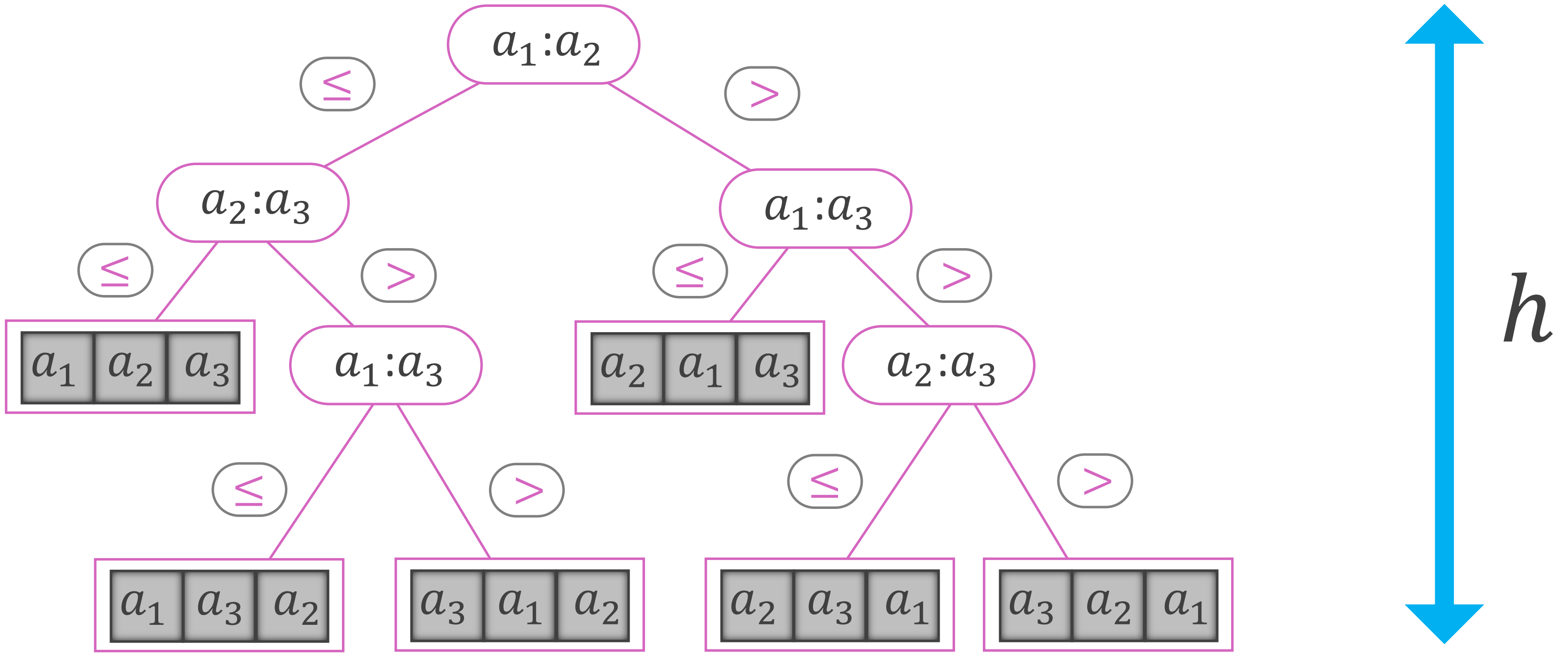
כל מיון מבוסס השוואות על מערך בגודל n דורש $\Omega(n\log n)$ השוואות במקרה הגרוע.

הוכחה

מהדיון הקודם יש למצוא חסם תחתון על גובה של עץ ההחלטה.



מהו חסם תחתון על גובה של עץ ההחלטה ?



.1 $\Omega(n \log n)$

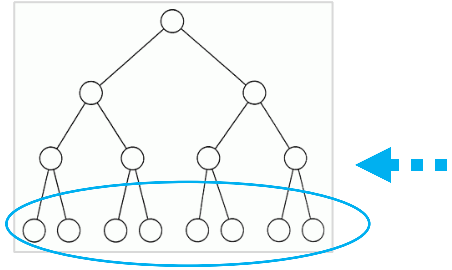
.2 $\Omega(n^2)$

.3 $\Omega(n \log^2 n)$

.4 $\Omega(n^{1.5} \log n)$

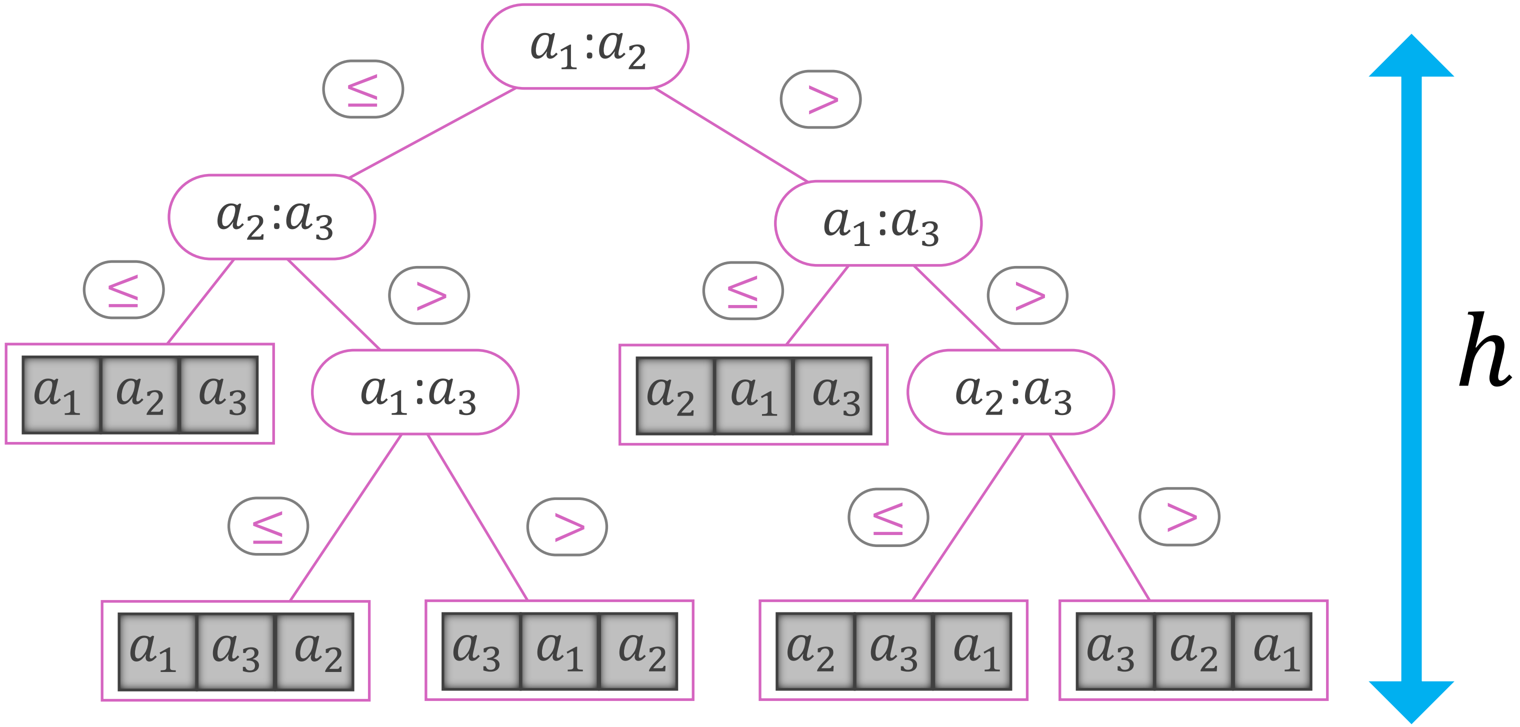
משפט החסם התחתון של מיון מבוסס השוואות – הוכחה

יהי T עץ החלטה בגובה h המתאר מיון מבוסס השוואות.



כל אחת מ $n!$ תמורות של איברי המערך חייבת להופיע באחד העלים.

מצד שני, מספר המירבי של עלים שיכול להיות בעץ בינארי בגובה h הוא 2^h.



מכאן, $n! \le 2^h$.

$$h \ge \log n!$$

מכאן, $h = \Omega(n \log n)$.

מ.ש.ל.

מסקנה: זמן ריצה במקרה הגרוע של כל אלגוריתם מבוסס השוואות על קלט בגודל n חסום מלמטה ע"י $\Omega(n \log n)$.

?

האם קיים איזשהו אלגוריתם למיון שזמן ריצתו במקרה הגרוע קטן יותר מ $O(n\log n)$

!

לא – אם האלגוריתם משתמש בהשוואות בלבד

כן – אם אלגוריתם משתמש במידע נוסף על מערך הקלט

מיונים בזמן לינארי

- ישנם מיונים לא מבוססי השוואות שסיבוכיות הזמן שלהם היא לינארית כלומר $O(n)$.
- מיונים אלה מניחים הנחות מסוימות על הקלט.
  - מיון מנייה counting sort
  - מיון בסיס Radix Sort

מיון מניה

Counting Sort

- הנחה: איברי הקלט הם מספרים שלמים בתחום מ- 0 עד k, עבור k שלם חיובי
- רעיון: לכל איבר x בקלט סופרים את כמות האיברים שקטנים או שווים לו (כולל x עצמו).

A	2	5	3	0	2	3	0	3

A לאחר מיון

מיון מנייה

שלבי האלגוריתם

1. שלב המנייה
2. שלב הצבירה
3. בניית מערך הפלט

מיון מניה דוגמת הרצה

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

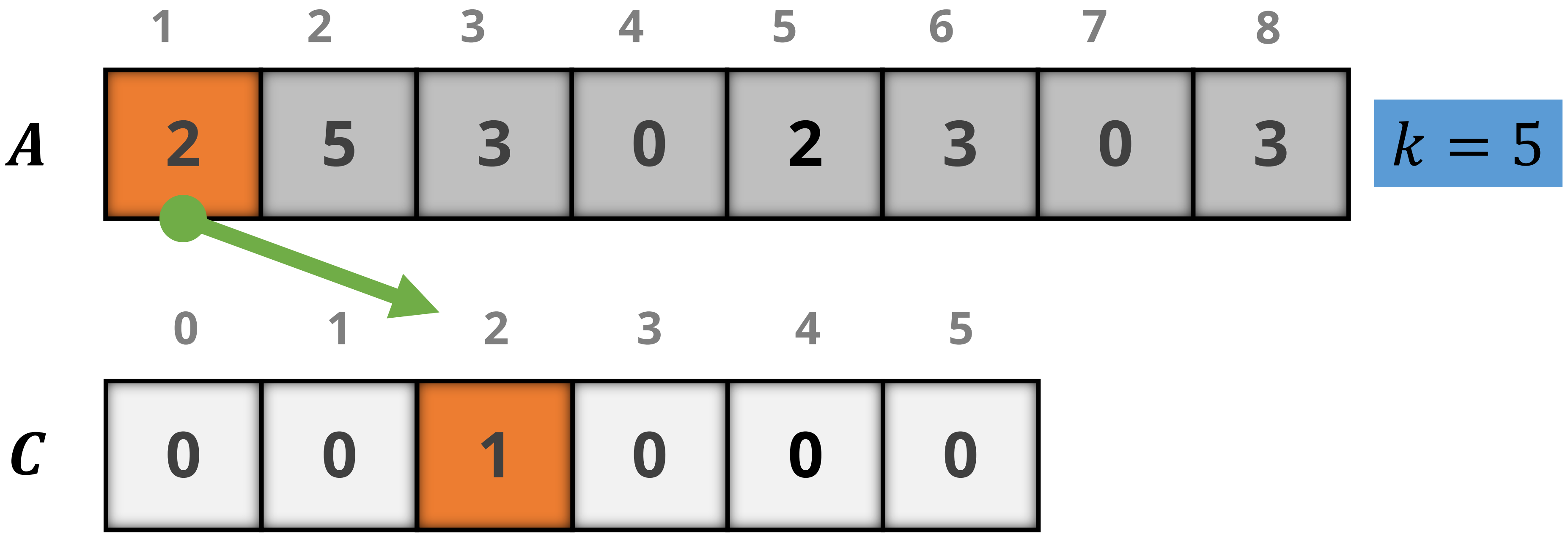
$k = 5$

	0	1	2	3	4	5
C	0	0	0	0	0	0

שלב I - שלב מנייה

- עבור כל איבר i, נספור כמה איברים שווים ל i יש ב A
- $C[i]$ – כמות האיברים ששווים ל i

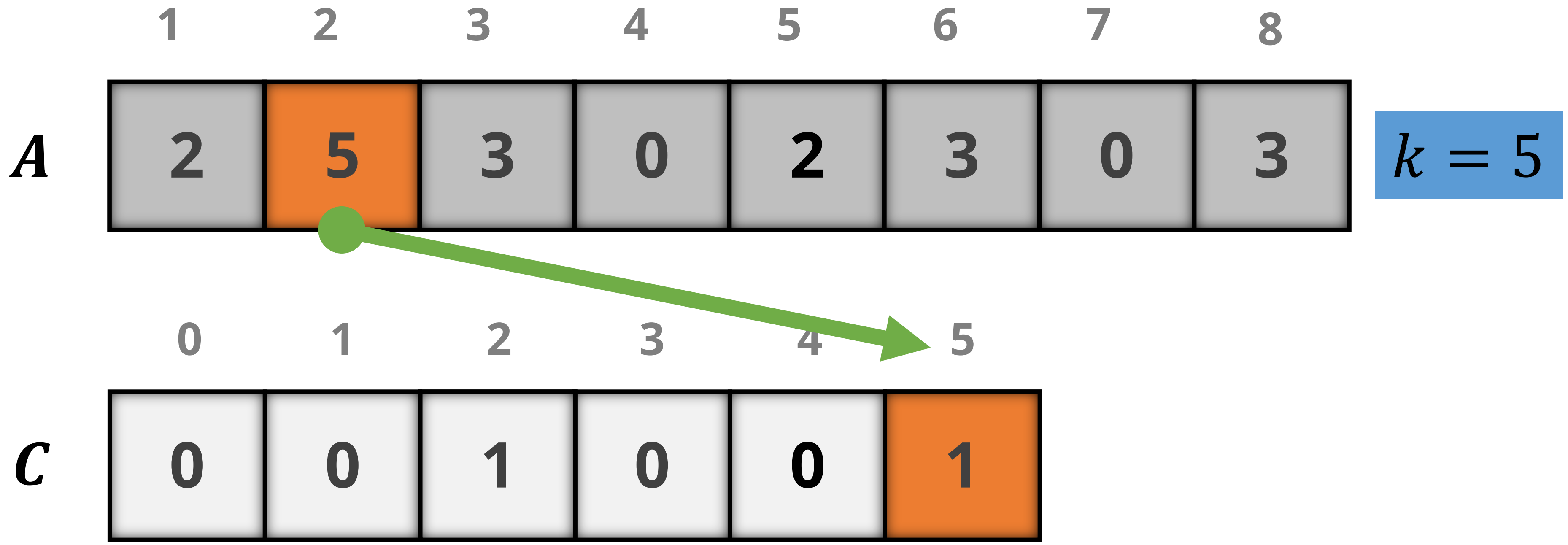
מיון מניה דוגמת הרצה



שלב I - שלב מנייה

- עבור כל איבר i, נספור כמה איברים שווים ל i יש ב A
- $C[i]$ – כמות האיברים ששווים ל i

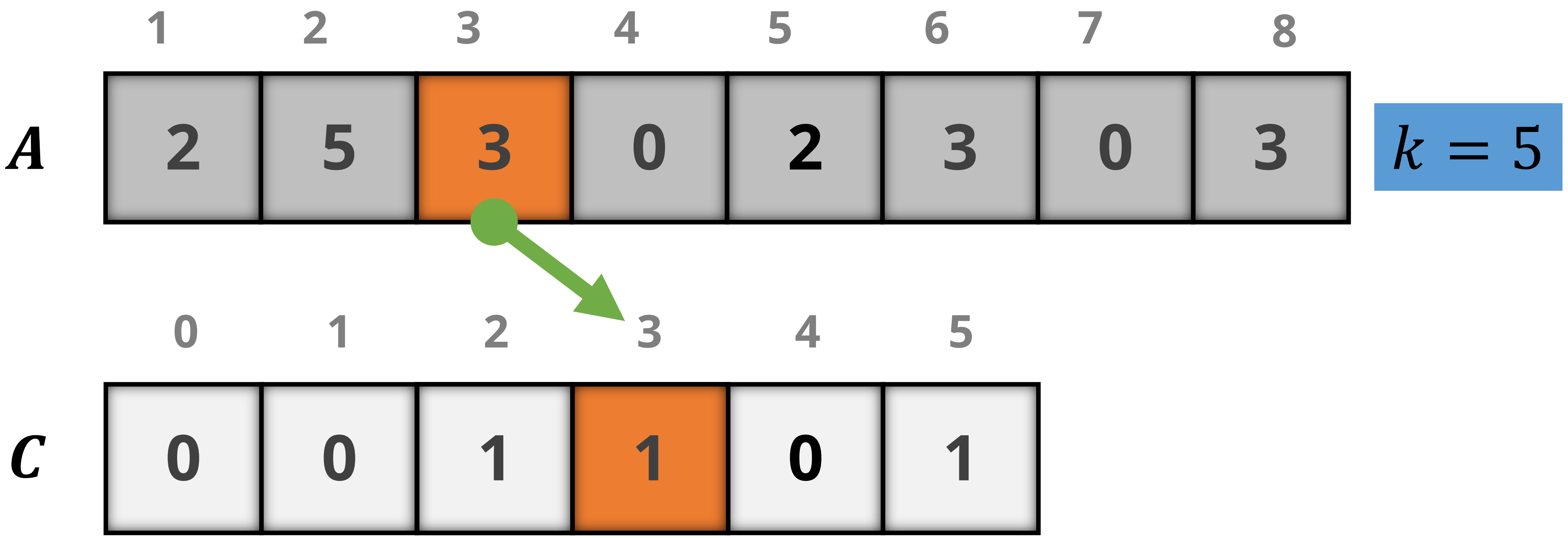
מיון מניה דוגמת הרצה



שלב I - שלב מנייה

- עבור כל איבר i, נספור כמה איברים שווים ל i יש ב A
- $C[i]$ – כמות האיברים ששווים ל i

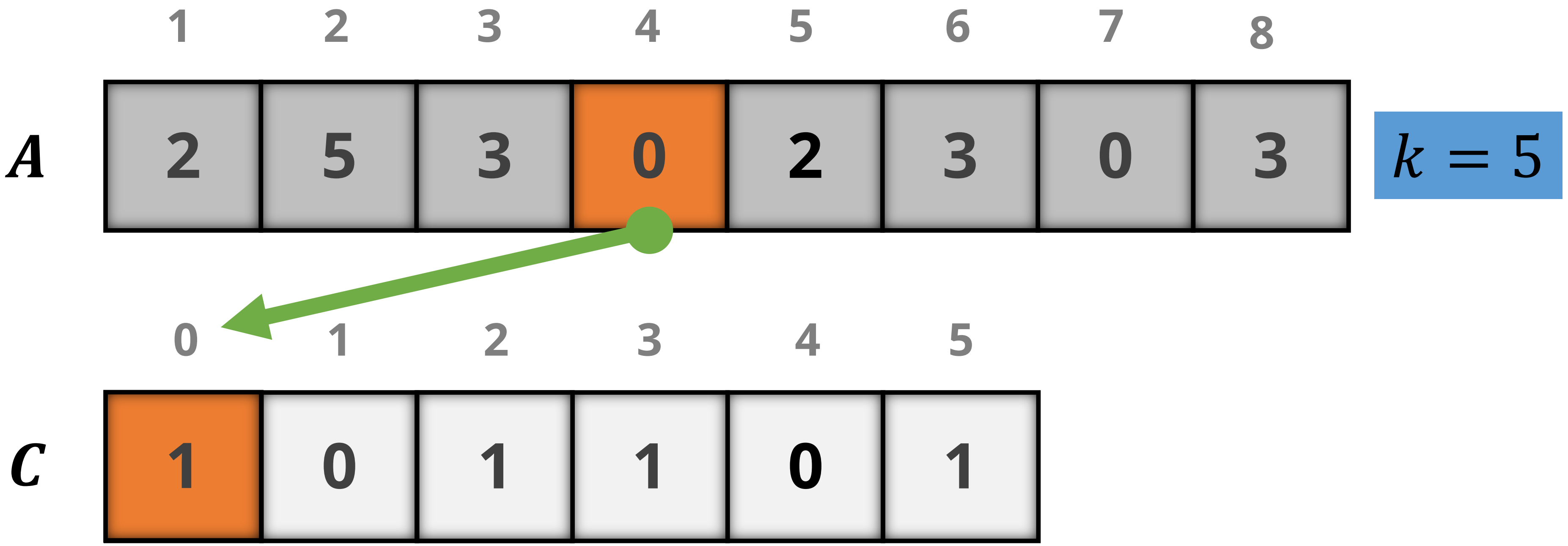
מיון מניה דוגמת הרצה



שלב I - שלב מנייה

- עבור כל איבר i, נספור כמה איברים שווים ל i יש ב A
- $C[i]$ – כמות האיברים ששווים ל i

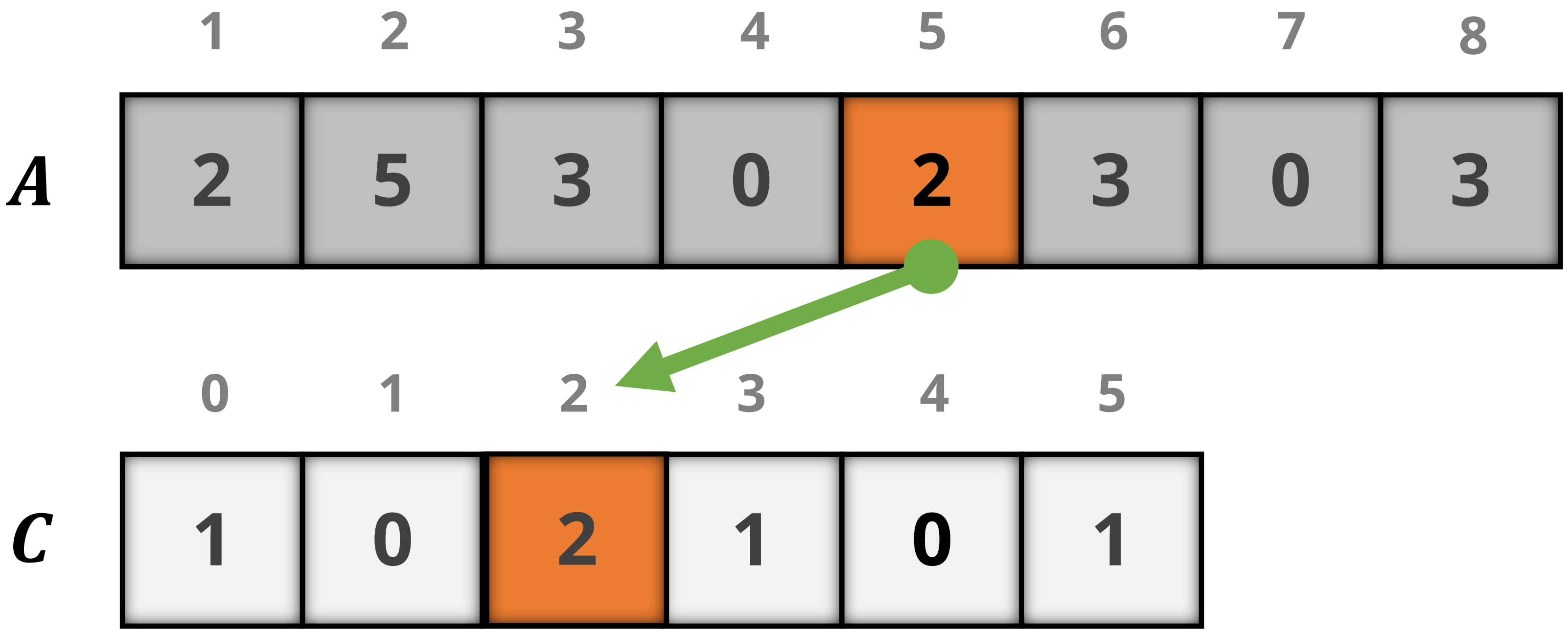
מיון מניה דוגמת הרצה



שלב I - שלב מנייה

- עבור כל איבר i, נספור כמה איברים שווים ל i יש ב A
- $C[i]$ – כמות האיברים ששווים ל i

מיון מניה דוגמת הרצה



$k = 5$

שלב I - שלב מנייה

- עבור כל איבר i, נספור כמה איברים שווים ל i יש ב A
- $C[i]$ – כמות האיברים ששווים ל i

מיון מניה דוגמת הרצה

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

$k = 5$

	0	1	2	3	4	5
C	2	0	2	3	0	1

שלב I - שלב מנייה

- עבור כל איבר i, נספור כמה איברים שווים ל i יש ב A
- $C[i]$ – כמות האיברים ששווים ל i

מיון מניה דוגמת הרצה

A

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

$k = 5$

C

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

≤ 0

שלב *II* - שלב צבירה

- נעבור על C משמאל לימין, ונסכום $C[i] \leftarrow C[i] + C[i-1]$
- $C[i]$ – כמות האיברים שקטנים או שווים ל i

מיון מניה דוגמת הרצה

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

$k = 5$

	0	1	2	3	4	5
C	2	2	2	3	0	1

≤ 1

שלב II - שלב צבירה

- נעבור על C משמאל לימין, ונסכום $C[i] \leftarrow C[i] + C[i-1]$
- $C[i]$ – כמות האיברים שקטנים או שווים ל i

מיון מניה דוגמת הרצה

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

$k = 5$

	0	1	2	3	4	5
C	2	2	4	3	0	1

≤ 2

שלב *II* - שלב צבירה

- נעבור על C משמאל לימין, ונסכום $C[i] \leftarrow C[i] + C[i-1]$
- $C[i]$ – כמות האיברים שקטנים או שווים ל i

מיון מניה דוגמת הרצה

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

$k = 5$

	0	1	2	3	4	5
C	2	2	4	7	0	1

≤ 3

שלב II - שלב צבירה

- נעבור על C משמאל לימין, ונסכום $C[i] \leftarrow C[i] + C[i-1]$
- $C[i]$ – כמות האיברים שקטנים או שווים ל i

מיון מניה דוגמת הרצה

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

$k = 5$

	0	1	2	3	4	5
C	2	2	4	7	7	1

≤ 4

שלב II - שלב צבירה

- נעבור על C משמאל לימין, ונסכום $C[i] \leftarrow C[i] + C[i-1]$
- $C[i]$ – כמות האיברים שקטנים או שווים ל i

מיון מניה דוגמת הרצה

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

$k = 5$

	0	1	2	3	4	5
C	2	2	4	7	7	8

שלב II - שלב צבירה

- נעבור על C משמאל לימין, ונסכום $C[i] \leftarrow C[i] + C[i-1]$
- $C[i]$ – כמות האיברים שקטנים או שווים ל i

מיון מניה דוגמת הרצה

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

$k = 5$

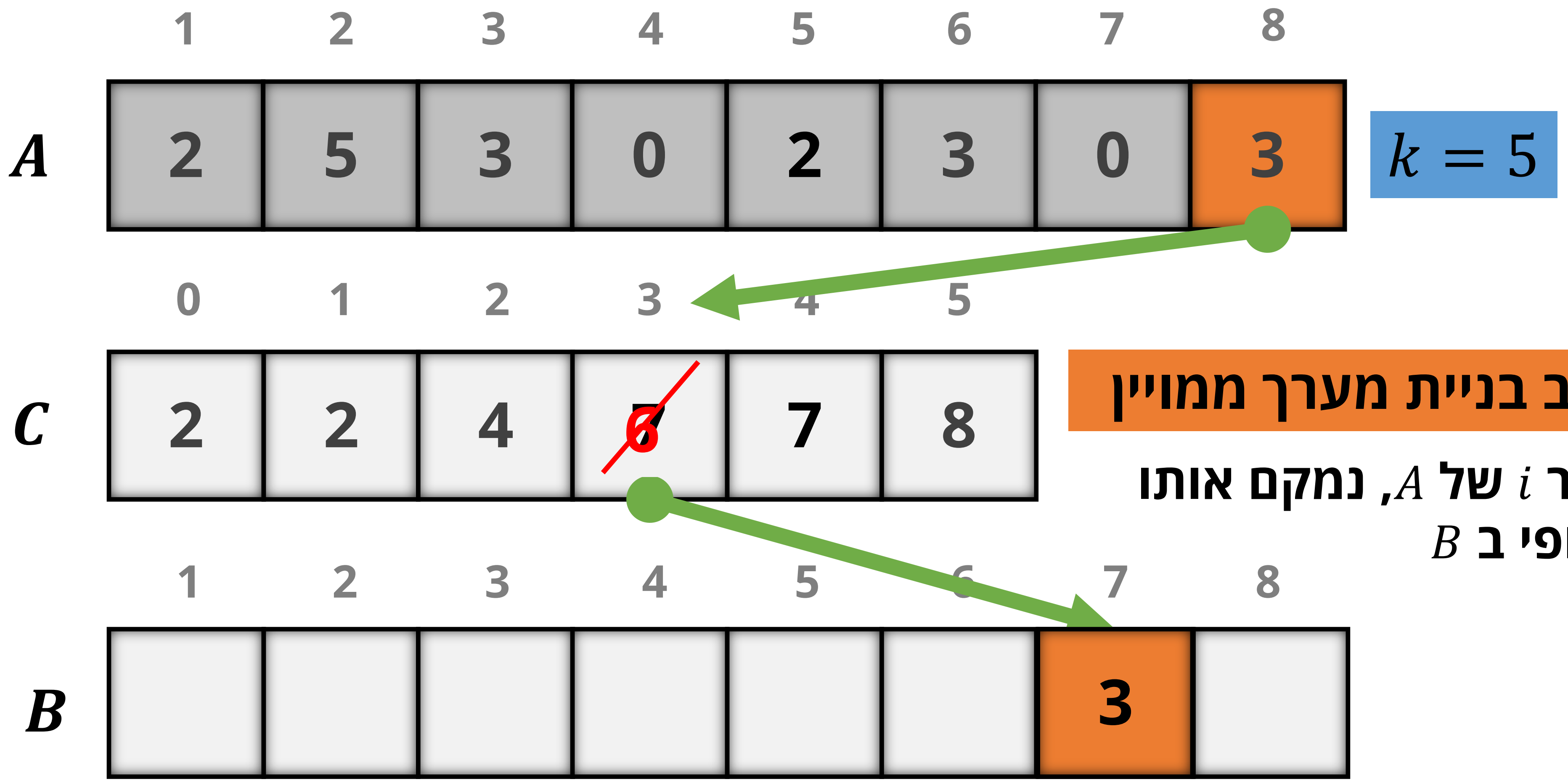
	0	1	2	3	4	5
C	2	2	4	7	7	8

שלב III - שלב בניית מערך ממוין

- עבור כל איבר i של A, נמקם אותו במיקומו הסופי ב B

	1	2	3	4	5	6	7	8
B								

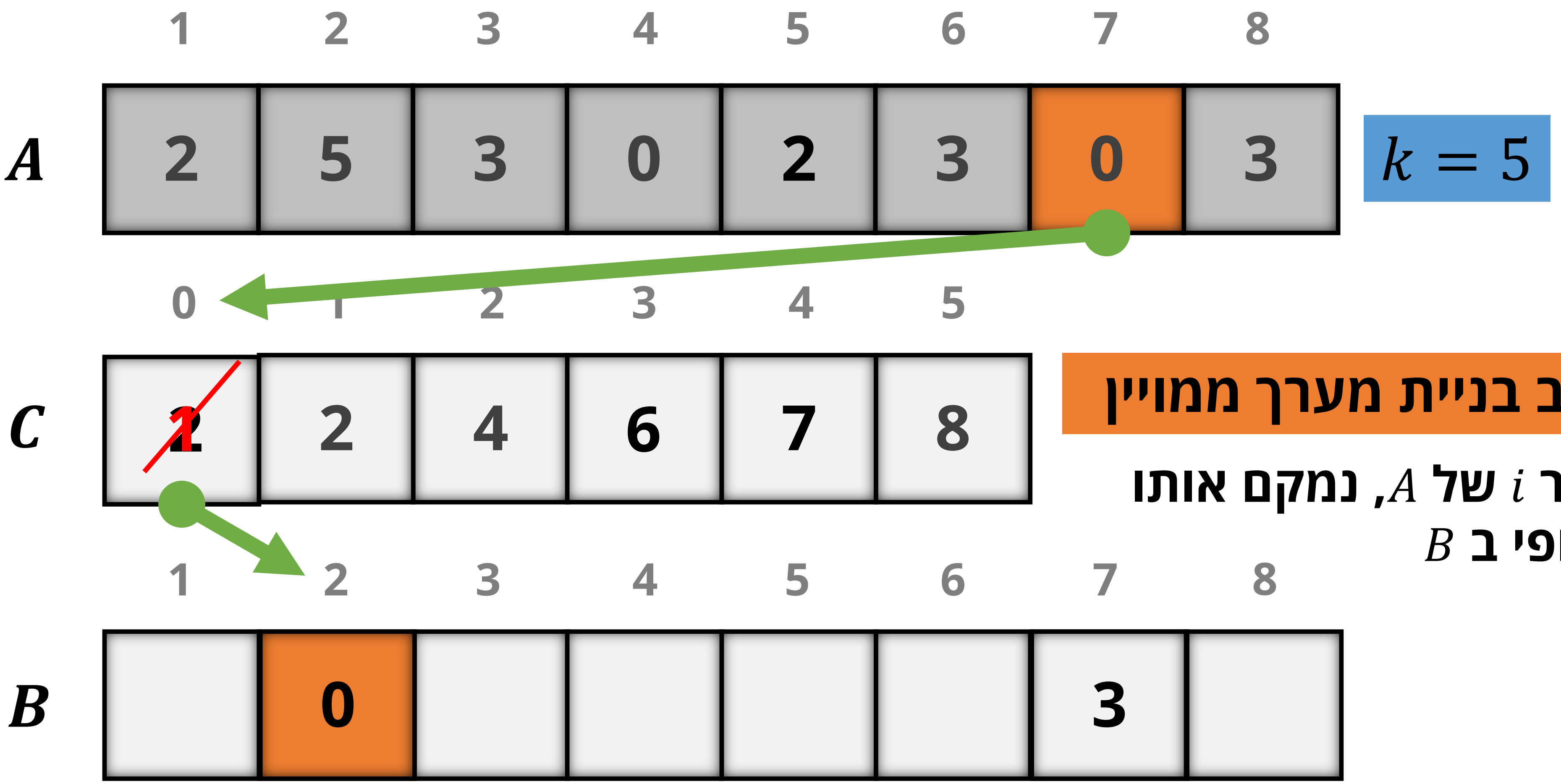
מיון מניה דוגמת הרצה



שלב *III* - שלב בניית מערך ממויין

- עבור כל איבר i של A, נמקם אותו במיקומו הסופי ב B

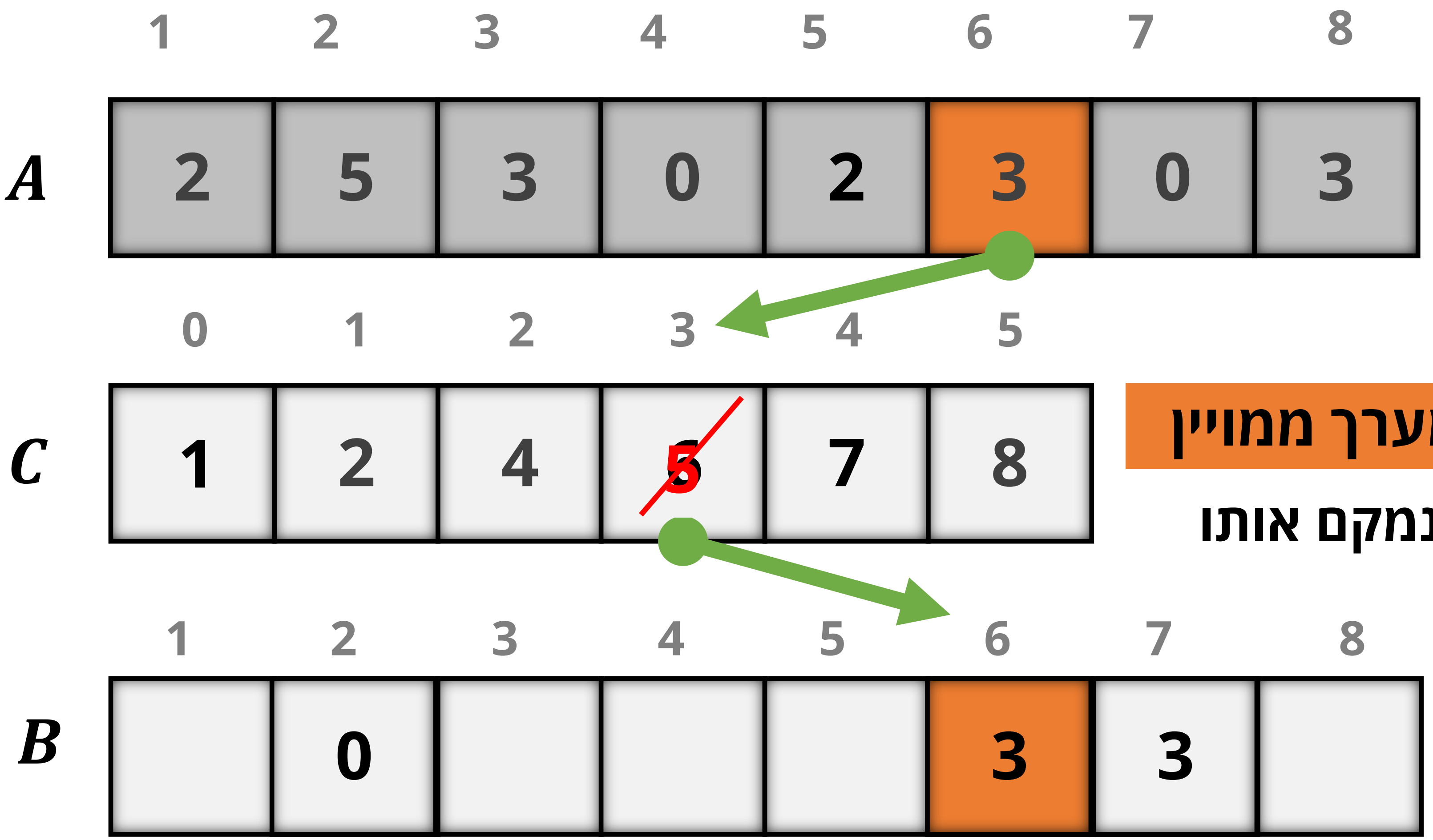
מיון מניה דוגמת הרצה



שלב *III* - שלב בניית מערך ממויין

- עבור כל איבר i של A, נמקם אותו במיקומו הסופי ב B

מיון מניה דוגמת הרצה

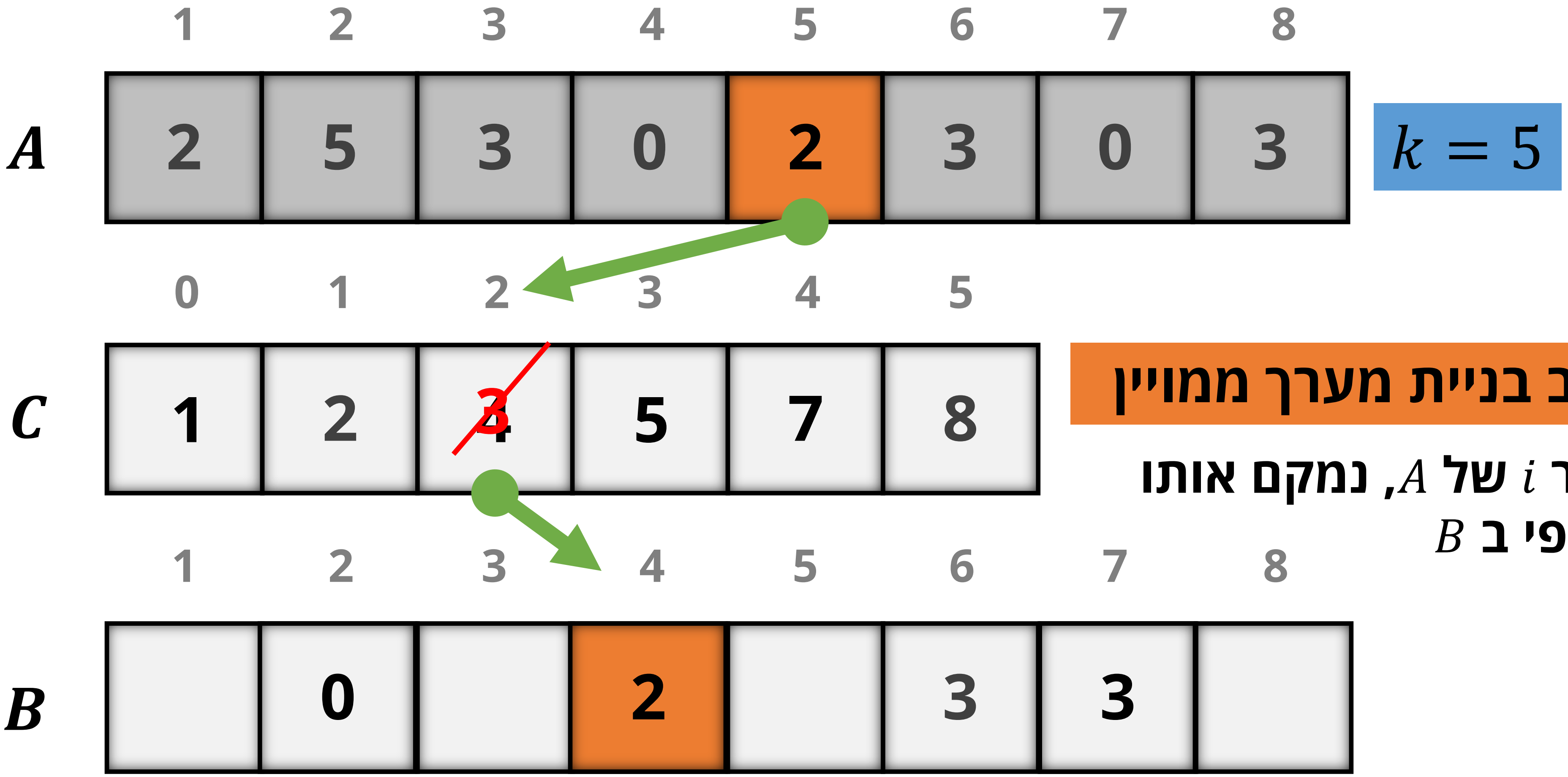


$k = 5$

שלב III - שלב בניית מערך ממויין

- עבור כל איבר i של A, נמקם אותו במיקומו הסופי ב B

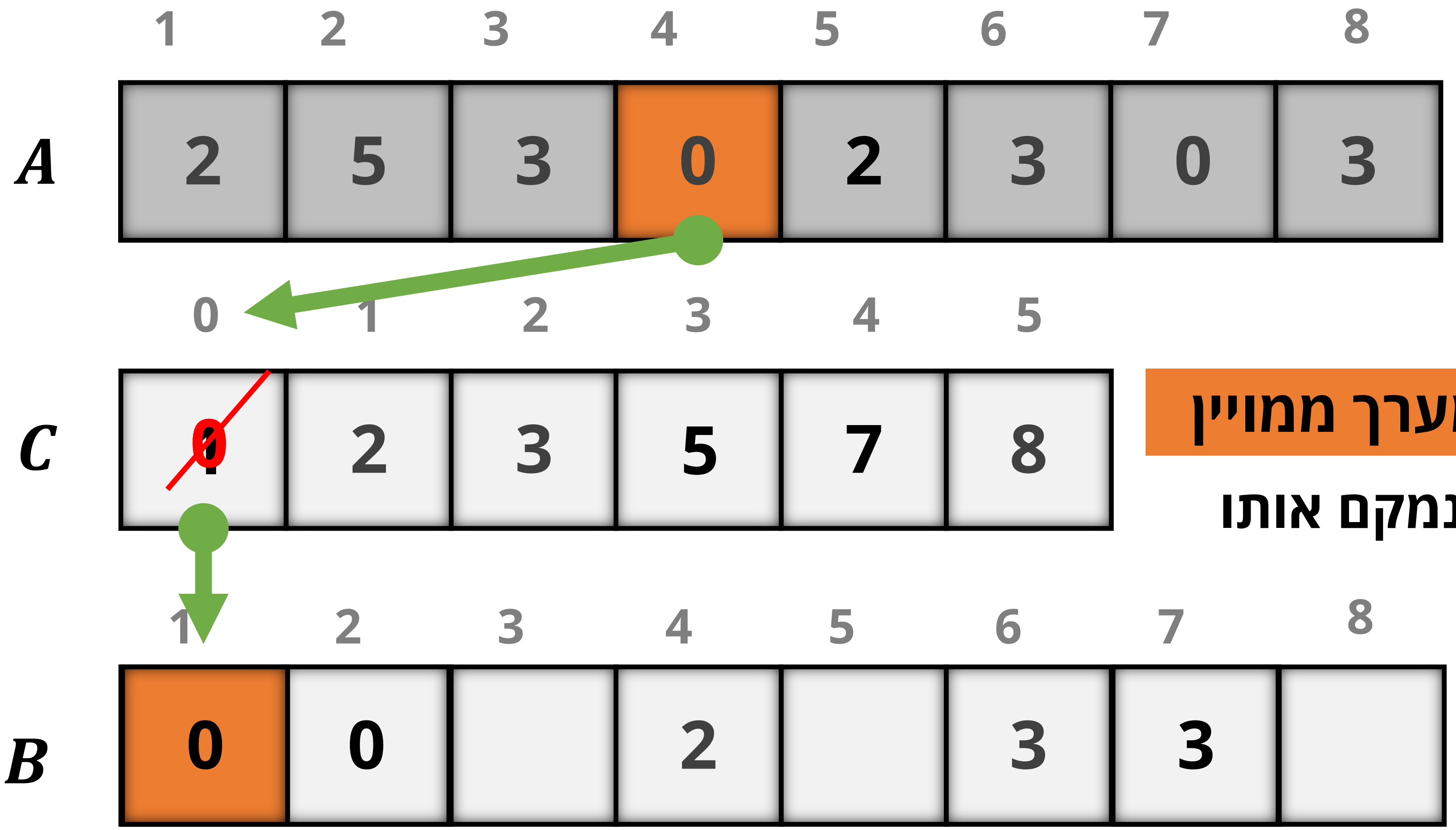
מיון מניה דוגמת הרצה



שלב III - שלב בניית מערך ממויין

- עבור כל איבר i של A, נמקם אותו במיקומו הסופי ב B

מיון מניה דוגמת הרצה

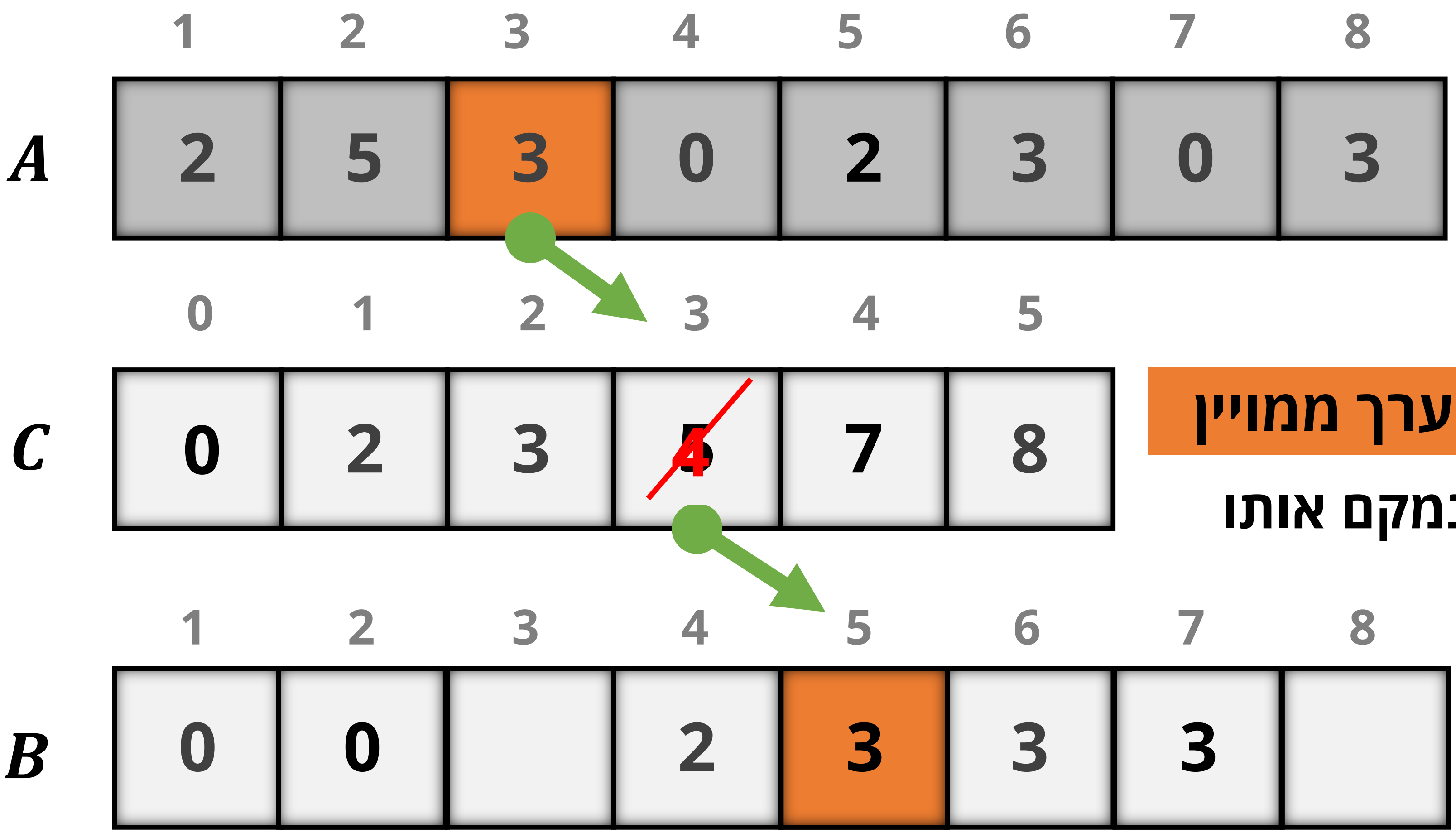


$k = 5$

שלב III - שלב בניית מערך ממויין

- עבור כל איבר i של A, נמקם אותו במיקומו הסופי ב B

מיון מניה דוגמת הרצה



$k = 5$

שלב III - שלב בניית מערך ממויין

- עבור כל איבר i של A, נמקם אותו במיקומו הסופי ב B

מיון מניה דוגמת הרצה

A

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

$k = 5$

C

0	1	2	3	4	5
0	2	3	4	7	~~8~~ 7

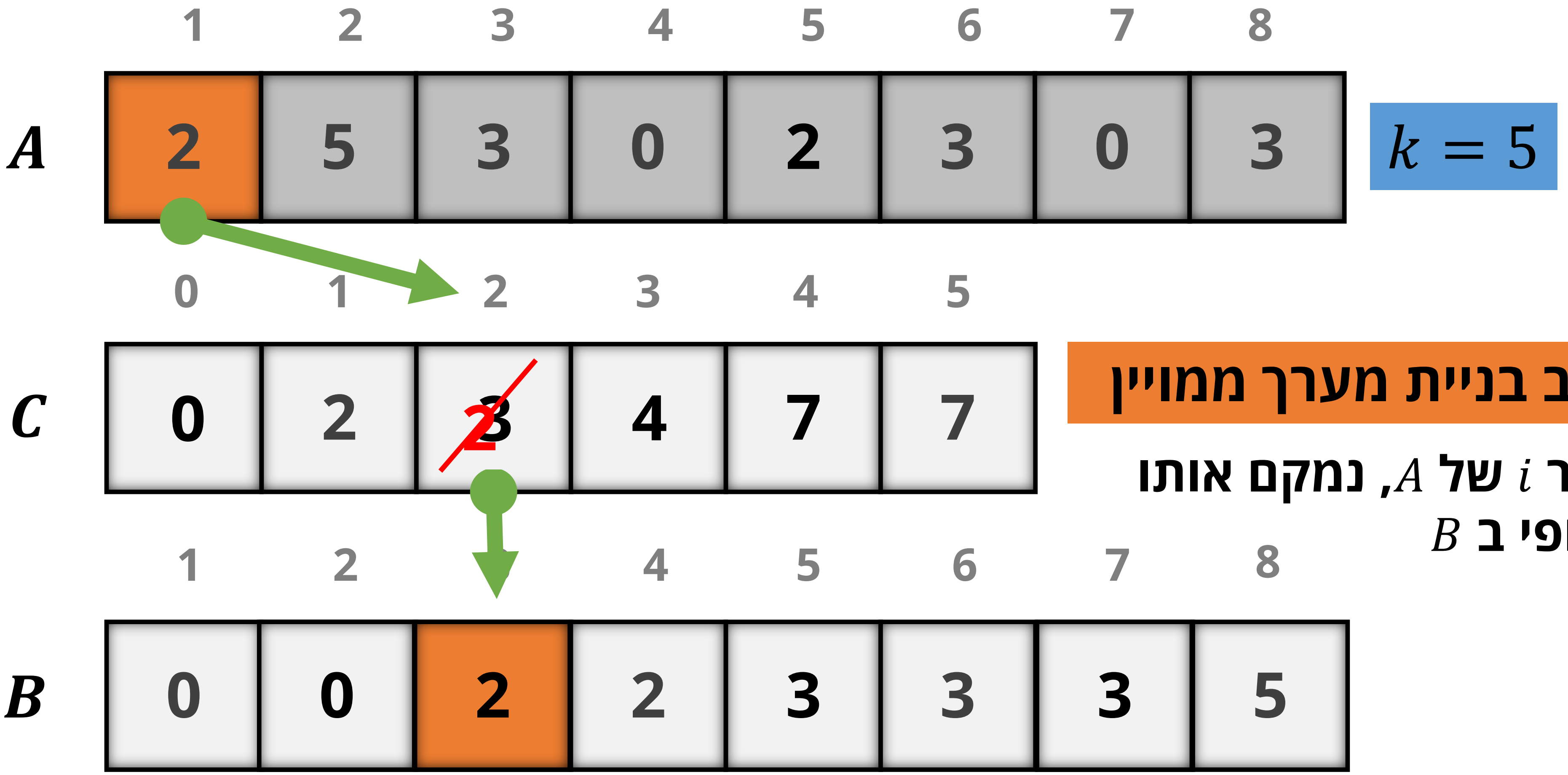
B

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

שלב III - שלב בניית מערך ממויין

- עבור כל איבר i של A, נמקם אותו במיקומו הסופי ב B

מיון מניה דוגמת הרצה



שלב III - שלב בניית מערך ממויין

- עבור כל איבר i של A, נמקם אותו במיקומו הסופי ב B

מיון מניה דוגמת הרצה

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

$k = 5$

	0	1	2	3	4	5
C	0	2	2	4	7	7

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

שלב III - שלב בניית מערך ממויין

- עבור כל איבר i של A, נמקם אותו במיקומו הסופי ב B

מיון מניה

זמן ריצה

$Counting\ Sort(A, B, k)$

1 $\quad for\ (i \leftarrow 0\ to\ k)$

2 $\quad\quad C[i] \leftarrow 0$

A

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

C

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

מיון מניה

זמן ריצה

$Counting\ Sort(A, B, k)$

1 $for\ (i \leftarrow 0\ to\ k)$

2 $\quad C[i] \leftarrow 0$

3 $for\ (i \leftarrow 1\ to\ A.length)$

4 $\quad C[A[i]] \leftarrow C[A[i]] + 1$

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

	0	1	2	3	4	5
C	2	0	2	3	0	1

מיון מניה

זמן ריצה

$Counting\ Sort(A, B, k)$

1 $for\ (i \leftarrow 0\ to\ k)$

2 $\quad C[i] \leftarrow 0$

3 $for\ (i \leftarrow 1\ to\ A.length)$

4 $\quad C[A[i]] \leftarrow C[A[i]] + 1$

5 $for\ (i \leftarrow 1\ to\ k)$

6 $\quad C[i] \leftarrow C[i] + C[i-1]$

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

	0	1	2	3	4	5
C	2	2	4	7	7	8

מיון מניה

זמן ריצה

$Counting\ Sort(A, B, k)$

```
for (i ← 0 to k)
    C[i] ← 0
for (i ← 1 to A.length)
    C[A[i]] ← C[A[i]] + 1
for (i ← 1 to k)
    C[i] ← C[i] + C[i − 1]
for (j ← A.length downto 1)
    B[C[A[j]]] ← A[j]
    C[A[j]] − −
```

A

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

C

0	1	2	3	4	5
2	2	4	7	7	8

B

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

מיון מניה

זמן ריצה

$Counting\ Sort(A, B, k)$

```
for (i ← 0 to k)
    C[i] ← 0
for (i ← 1 to A.length)
    C[A[i]] ← C[A[i]] + 1
for (i ← 1 to k)
    C[i] ← C[i] + C[i − 1]
for (j ← A.length downto 1)
    B[C[A[j]]] ← A[j]
    C[A[j]] − −
return B
```

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

מיון מניה

זמן ריצה

$\boldsymbol{Counting\ Sort(A, B, k)}$

$\boldsymbol{for}\ (i \leftarrow 0\ \boldsymbol{to}\ k)$

$\quad C[i] \leftarrow 0$

$O(k)$

$\boldsymbol{for}\ (i \leftarrow 1\ \boldsymbol{to}\ A.length)$

$\quad C[A[i]] \leftarrow C[A[i]] + 1$

$O(n)$

$\boldsymbol{for}\ (i \leftarrow 1\ \boldsymbol{to}\ k)$

$\quad C[i] \leftarrow C[i] + C[i-1]$

$O(k)$

$\boldsymbol{for}\ (j \leftarrow A.length\ downto\ 1)$

$\quad B[C[A[j]]] \leftarrow A[j]$

$\quad C[A[j]] - -$

$O(n)$

$return\ B$

זמן ריצה כולל הינו $\boldsymbol{O(n+k)}$

אם בבניית מערך הפלט היינו עוברים על A מהתחלה לסוף (ולא מסוף להתחלה), האם המיון היה נכון ?

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

	0	1	2	3	4	5
C	2	2	4	7	7	8

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

1. כן, אין חשיבות לכיוון המעבר
2. לא, רק מעבר מסוף להתחלה היה נותן מיון תקין
3. אפשר לעבור מהתחלה לסוף, אך היינו מאבדים תכונת מיון חשובה

אם בבניית מערך הפלט היינו עוברים על A מהתחלה לסוף (ולא מסוף להתחלה), האם המיון היה נכון ?

	1	2	3	4	5	6	7	8
A	2_1	5	3_1	0_1	2_2	3_2	0_2	3_3

	1	2	3	4	5	6	7	8
B	0_2	0_1	2_2	2_1	3_3	3_2	3_1	5

	1	2	3	4	5	6	7	8
B	0_1	0_2	2_1	2_2	3_1	3_2	3_3	5

מיון יציב הגדרה

שמירה של הסדר המקורי בין ערכים זהים נקראת **תכונת היציבות** –
stability, **ומיון שיש לו את התכונה הזו נקרא מיון יציב** – stable sort.

	1	2	3	4	5	6	7	8
A	2_1	5	3_1	0_1	2_2	3_2	0_2	3_3

	1	2	3	4	5	6	7	8
B	0_1	0_2	2_1	2_2	3_1	3_2	3_3	5

מיון בסיס
Radix Sort

- הנחה: מספר הקלט הם שלמים בני d ספרות
- דוגמה:

?

למה חשוב להשתמש במיון יציב למיון ספרות?
מה עלול לקרות אם היינו משתמשים במיון לא יציב?

.1
אפשר גם מיון לא יציב, זה לא משנה

.2
מיון לא יציב היה עלול להחזיר תוצאת מיון שגויה

.3
מיון לא יציב תמיד היה מחזיר תוצאת מיון שגויה

.4
מיון יציב חשוב לנו רק כדי לקבל זמן ריצה טוב

מיון בסיס
Radix Sort

RadixSort (A, d)

for $(i = 1 \text{ to } d)$ **do**

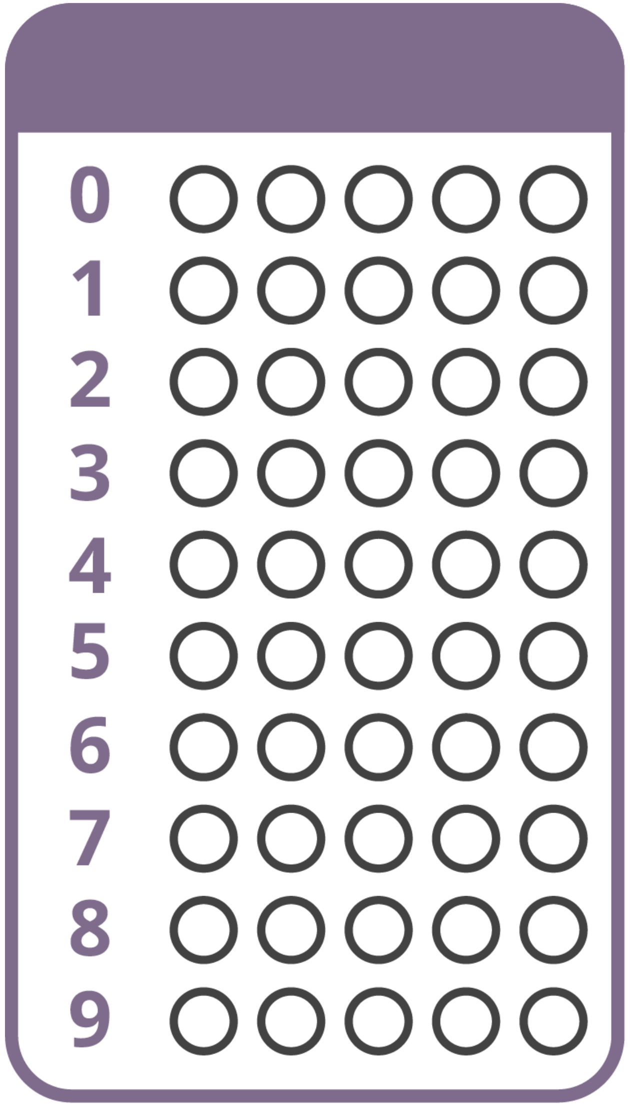
use a **stable sort** to sort array A on digit i

Radix Sort
זמן ריצה

```
RadixSort (A, d)
        for (i = 1 to d) do
                use a stable sort to sort array A on digit i
```

- Assume that we use counting sort as the intermediate sort.
- $\Theta(n + k)$ per pass (digits in range $0, \ldots, k$).
- d passes.
- $\Theta(d(n + k))$ total.
- If $k = O(n)$ and $d = O(1)$, then $T(n) = \Theta(n)$.

מיון בסיסי
Radix Sort



סיכום