

# מיון - Sorting

**משפט:** כל אלגוריתם מיון מבוסס השוואות הממין  $n$  מפתחות כלשהם מבצע לכל הפחות  $\Omega(n \log n)$  פעולות השוואה במקרה הגרוע ובמקרה הממוצע.

לפעמים, אם ידוע מידע נוסף על הקלט, ניתן למיין בסבוכיות זמן טובה יותר.

## Counting Sort

אם  $n$  המספרים לקוחים מהטווח  $[1, k]$ , ניתן למיין אותם בזמן  $O(n + k)$ .

- עבור  $k = O(n)$  נקבל מיון בזמן לינארי.

Counting Sort הינו אלגוריתם מיון **יציב**:

**המדד:** אלגוריתם מיון הוא יציב אם הוא שומר על הסדר היחסי בין איברים בעלי ערכים זהים.

3 1 3 0 5 3 → 0 1 3 3 3 5

מברך נתונים 1 - טכניון ©

1

## Counting Sort

בהינתן מערך  $A$  בגודל  $n$  שאיבריו לקוחים מהטווח  $[1, k]$ :

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

1. נקצה מערך  $C$  בגודל  $k$  שיוספור כמה פעמים מופיע כל איבר  $i \in [1, k]$ :

```
for (int i=1; i <= k; i++) C[i] ← 0;
for (int i=1; i <= n; i++) C[A[i]]++;
```

	1	2	3	4	5	6
$C$	2	3	0	0	1	2

2. נמצא לכל איבר  $i \in [1, k]$  את האינדקס האחרון בסדרה הממויינת שבו מופיע  $i$ .

```
for (int i=2; i <= k; i++)
    C[i] ← C[i-1] + C[i];
```

	1	2	3	4	5	6
$C$	2	5	5	5	6	8

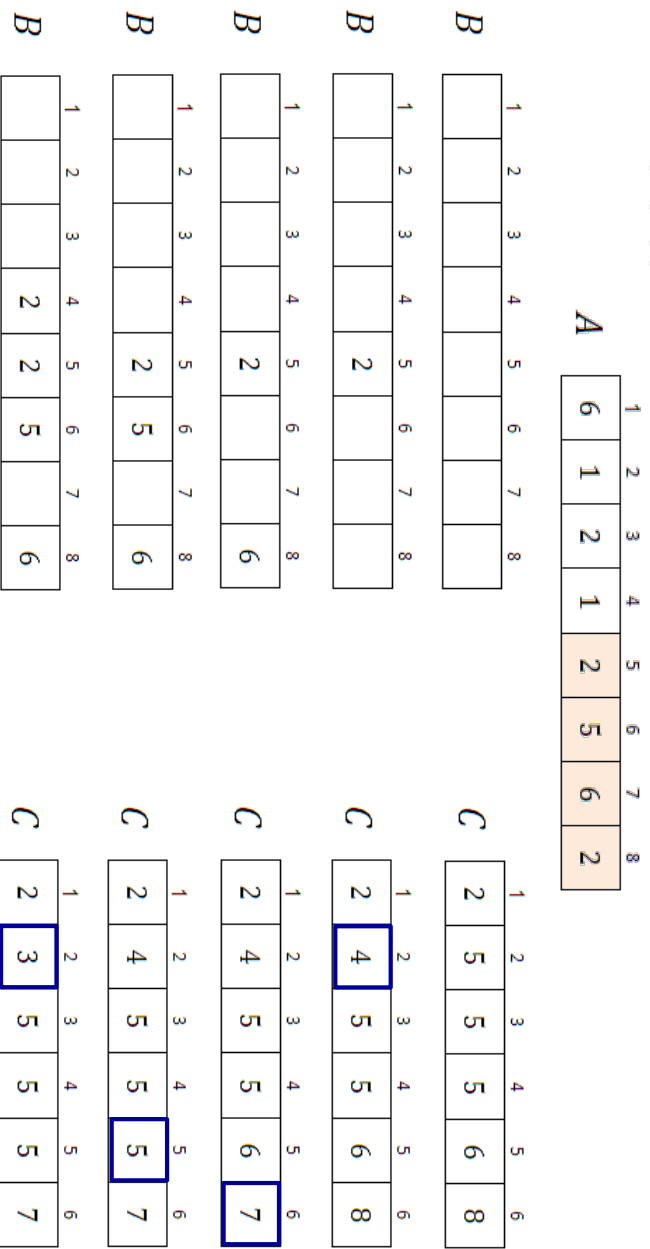
מברך נתונים 1 - טכניון ©

2

## Counting Sort

3. נעבור על האיברים מהסוף להתחלה ונמקם אותם במערך הפלט  $B$ .

```
for (int i=n; i >= 1; i--)
    B[C[A[i]]] ← A[i];
    C[A[i]]--;
```



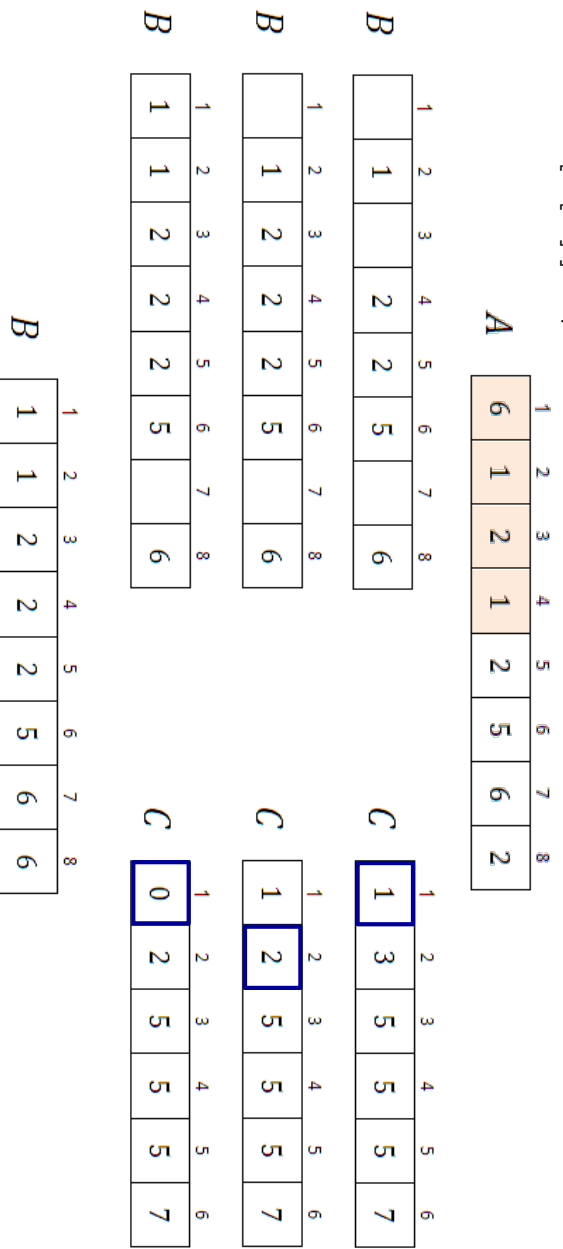
מברך נתונים 1 - טכניון ©

3

## Counting Sort

3. נעבור על האיברים מהסוף להתחלה ונמקם אותם במערך הפלט  $B$ .

```
for (int i=n; i >= 1; i--)
    B[C[A[i]]] ← A[i];
    C[A[i]]--;
```



מברך נתונים 1 - טכניון ©

4

## מיון ב- $O(n)$ ?

מיון Counting Sort לוקח  $O(n + k)$  זמן, כאשר המספרים לקוחים מהטווח  $[1, k]$ .  
נסתכל על האלגוריתם הבא:

- נעבור על המערך בזמן  $O(n)$  ונמצא את האיבר המקסימלי בו. נסמן איבר זה ב- $k$ .
- נשתמש ב-Counting Sort על מנת למיין את המערך.
- היות ו- $k$  הוא קבוע, נקבל מיון בזמן  $O(n)$ .

איפה הטעות?

האיבר המקסימלי במערך יכול להיות  $2^n$ .

Counting Sort במצב זה "יקח"  $O(2^n) = O(n + 2^n)$  זמן.