מבני נתונים ומבוא לאלגוריתמים
מפגש הנחיה מס' 5

מדעי המחשב, קורס מס' 20407

סמסטר 2016ב

מנחה: ג'ון מרברג

מה ראינו בפעם הקודמת?

- ערמה
  - תכונות הערימה
  - בניית ערמה $O(n)$
  - תיקון ערמה $O(\log n)$
  - מיון ערמה $O(n\log n)$
- שמוש בערמה כמבנה נתונים
  - תור קדימויות
  - פעולות: הכנסה, מציאת והוצאת מקסימום, שינוי קדימות

מפגש חמישי

- נושאי השיעור
  - פרק 7 בספר – מיון-מהיר
    - מיון-מהיר: הגרסה שבספר, הגרסה של Hoare
    - ניתוח זמן הריצה
    - מימוש אקראי
    - תרגילים

מבוסס על מצגת של ברוך חייקין ואיציק בייז

תרגיל חזרה: ערימה

נתונה ערימת מקסימום A בגודל n, המכילה מספרים ממשיים.
כמו כן נתון מספר ממשי z, ומספר שלם m המקיים $1 \le m \le n$.

יהי $\{x_1, x_2, x_3, \ldots, x_n\}$ הסדר <u>הממוין</u> (מגדול לקטן) של המספרים בערימה.
למשל, x_1 הוא המספר הגדול ביותר בערימה, $x_{\lceil n/2 \rceil}$ הוא החציון התחתון, וכד'.

כתבו אלגוריתם בשם HeapCheck(A, z, m) שמחזיר תוצאה כדלקמן:
אם $x_m < z$ יוחזר -1, אם $x_m > z$ יוחזר 1, ואחרת יוחזר 0.

על האלגוריתם לרוץ בסיבוכיות זמן $O(m)$.

לתשומת לב:
- אין צורך לזהות את המספר x_m
- כמובן שלא ניתן לבצע מיון של הערימה בסיבוכיות $O(m)$.
- <u>רמז</u>: נצלו את תכונת הערימה כדי להחליט האם יש טעם לסרוק תת עץ נתון

האוניברסיטה הפתוחה

תרגיל חזרה (המשך): ערימה

נתחזק שני משתנים <u>גלובלים</u> שמשמשים בתור מונים

- *count_ge* מונה את האיברים בערימה שגדולים או שווים ל-*z*
- *count_g* מונה את האיברים בערימה שגדולים ממש מ-*z*

```
HeapCount_GE(A, i, z, m)
if i > heapsize[A] or A[i] < z or count_ge = m
    then return
count_ge ← count_ge + 1
HeapCount_GE(A, left[i], z, m)
HeapCount_GE(A, right[i], z, m)
```

```
HeapCheck(A, z, m)
count_ge ← 0
count_g ← 0
HeapCount_GE(A, 1, z, m)
if count_ge < m
    then return –1
HeapCount_G(A, 1, z, m)
if count_g < m
    then return 0
return 1
```

```
HeapCount_G(A, i, z, m)
if i > heapsize[A] or A[i] ≤ z or count_g = m
    then return
count_g ← count_g + 1
HeapCount_G(A, left[i], z, m)
HeapCount_G(A, right[i], z, m)
```

תרגיל חזרה (סוף): ערימה

- **נכונות**

האלגוריתם HeapCount_GE סורק רקורסיבית את אברי הערימה, החל בשורש הערימה.
כל איבר נבדק פעם אחת לכל היותר.

המונה הגלובלי count_ge מוגדל ב-1 בכל פעם שנמצא עוד איבר שערכו גדול או שווה z.
לפי תנאי העצירה בשורה 1, הרקורסיה תמשיך כל עוד לא נמצאו m איברים גדולים או שווים
ל-z, וכל עוד יש טעם להמשיך ולסרוק את תת העץ של האיבר הנוכחי לפי תכונת הערימה.

לפיכך, אחרי החזרה ל-HeapCheck, אם count_ge<m אז חייב להתקיים $x_m<z$.
אחרת, מופעל האלגוריתם HeapCount_G, ובטיעון דומה לעיל, אם $count\_g<m$ אז חייב
להתקיים $x_m=z$. אחרת, כמובן $x_m>z$.

- **סבוכיות**

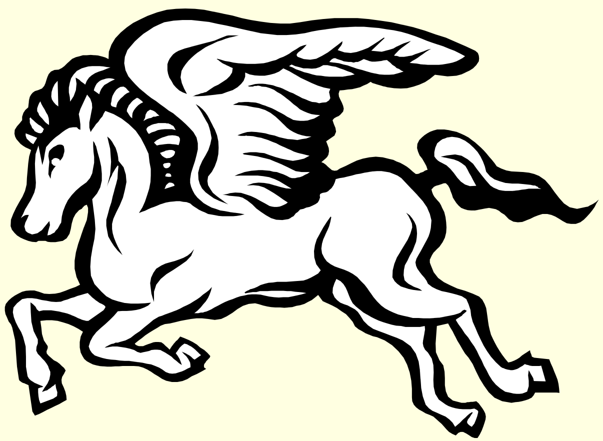
סבוכיות האלגורתם נקבעת על ידי מספר הקריאות הרקורסיביות. כל שאר הפעולות הן $O(1)$.

נשים לב כי הקריאות הרקורסיביות נעשות בזוגות (שורות 4-5 בכל אחד מאלגוריתמי המניה).
לפני כל זוג קריאות כזה, המונה הגלובלי הרלוונטי מוגדל ב-1.
תנאי העצירה בשורה 1 מונע מהמונה לגדול מעבר לערך m.

לפיכך, מספר הקריאות הרקורסיביות של כל אחד משני האלגוריתמים הוא לכל היותר $2m$,
כלומר זמן הריצה הכולל הוא $O(m)$.

מיון מהיר Quicksort

- הוצע ע"י C.A.R. Hoare ב-1962
- אלגוריתם רקורסיבי, בשיטת הפרד ומשול (בדומה למיון מיזוג)
- קיים מימוש כאלגוריתם אקראי
- זמן ריצה:
  - במקרה הגרוע $\Theta(n^2)$
  - במקרה הממוצע $\Theta(n\lg n)$
- האלגוריתם ממיין במקום (בניגוד למיון מיזוג)
- אלגוריתם מהיר מאוד באופן מעשי



האוניברסיטה הפתוחה

האלגוריתם

- האלגוריתם (עמ' 122 בספר):

```
QuickSort (A, p, r)
if p < r
      then q ← Partition (A, p, r)
            QuickSort (A, p, q–1)
            QuickSort (A, q+1, r)
```

- הקריאה הראשונה: QuickSort $(A, 1, length[A])$
- *מקרה בסיס:* מערך בגודל 1 הוא ממוין
- *הפרד:* מבצעים חלוקה של המערך לשני תת-מערכים, $L = A[p..q–1]$ ו- $R = A[q+1..r]$, כך שכל איבר באזור השמאלי קטן או שווה ל- $A[q]$ וכל איבר באזור הימני גדול או שווה ל- $A[q]$.
האיבר $A[q]$ נמצא לאחר החלוקה במקומו הסופי.
- *משול:* ממיינים כל אזור באופן רקורסיבי

האוניברסיטה הפתוחה

שגרת החלוקה של Lomuto

- שגרת החלוקה (עמ' 122 בספר) הומצאה ע"י Lomuto

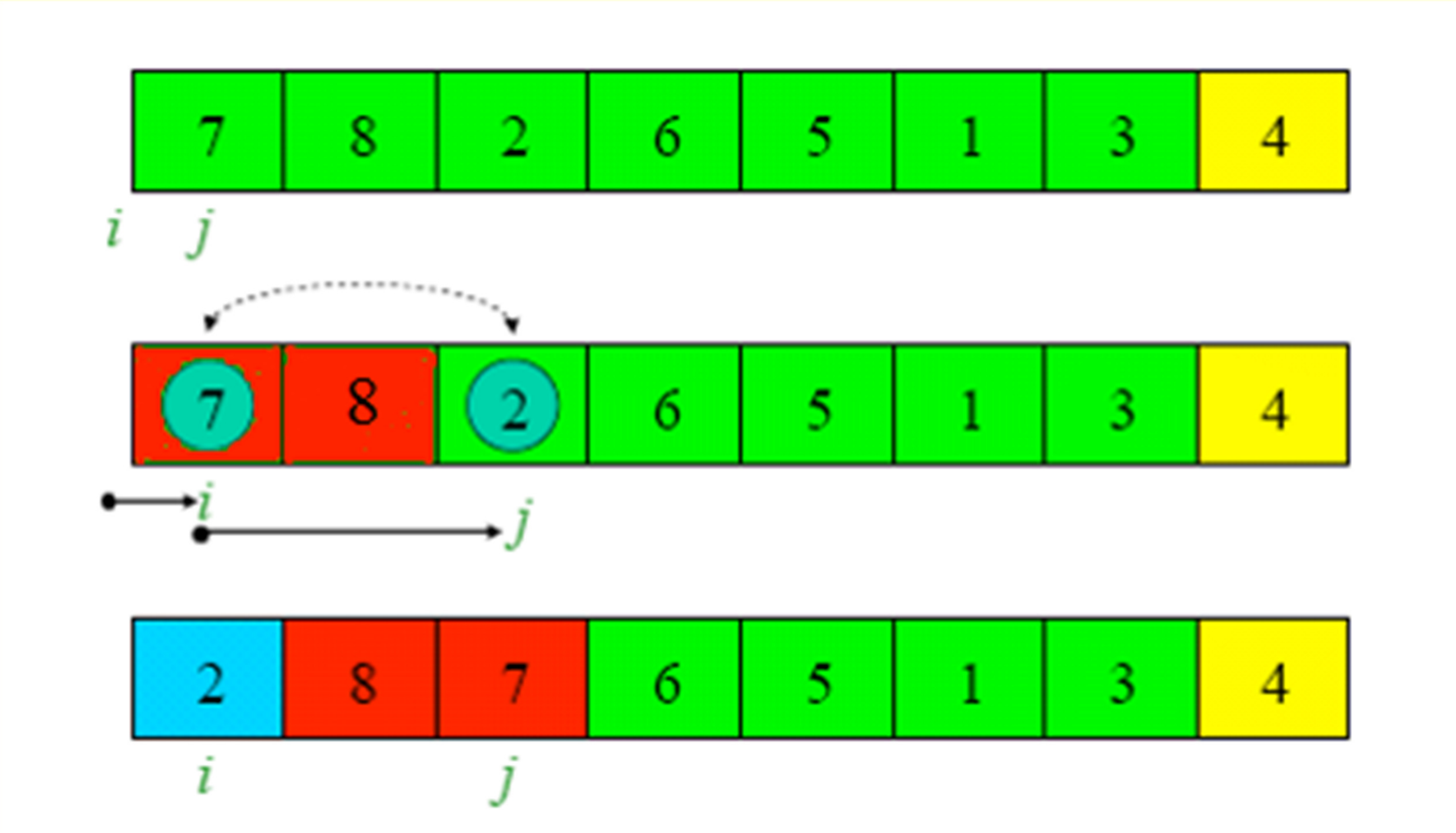
Partition (*A*, *p*, *r*)

$x \leftarrow A[r]$	// בחירת איבר ציר
$i \leftarrow p - 1$	// מצביע לגבול החלק השמאלי
for $j \leftarrow p$ **to** $r - 1$	// מצביע לגבול החלק הימני
do if $A[j] \le x$	// האיבר הנוכחי קטן מאיבר הציר
then $i \leftarrow i + 1$	// הגדלת הגבול של החלק השמאלי
exchange $A[i] \leftrightarrow A[j]$	// האיבר הנוכחי נכנס לקצה החלק השמאלי
	// והאיבר שהיה שם נכנס לקצה החלק הימני
exchange $A[i+1] \leftrightarrow A[r]$	// איבר הציר נכנס למקומו הסופי
return $i + 1$	// מוחזר האינדקס של איבר הציר

Running time is O(*n*)

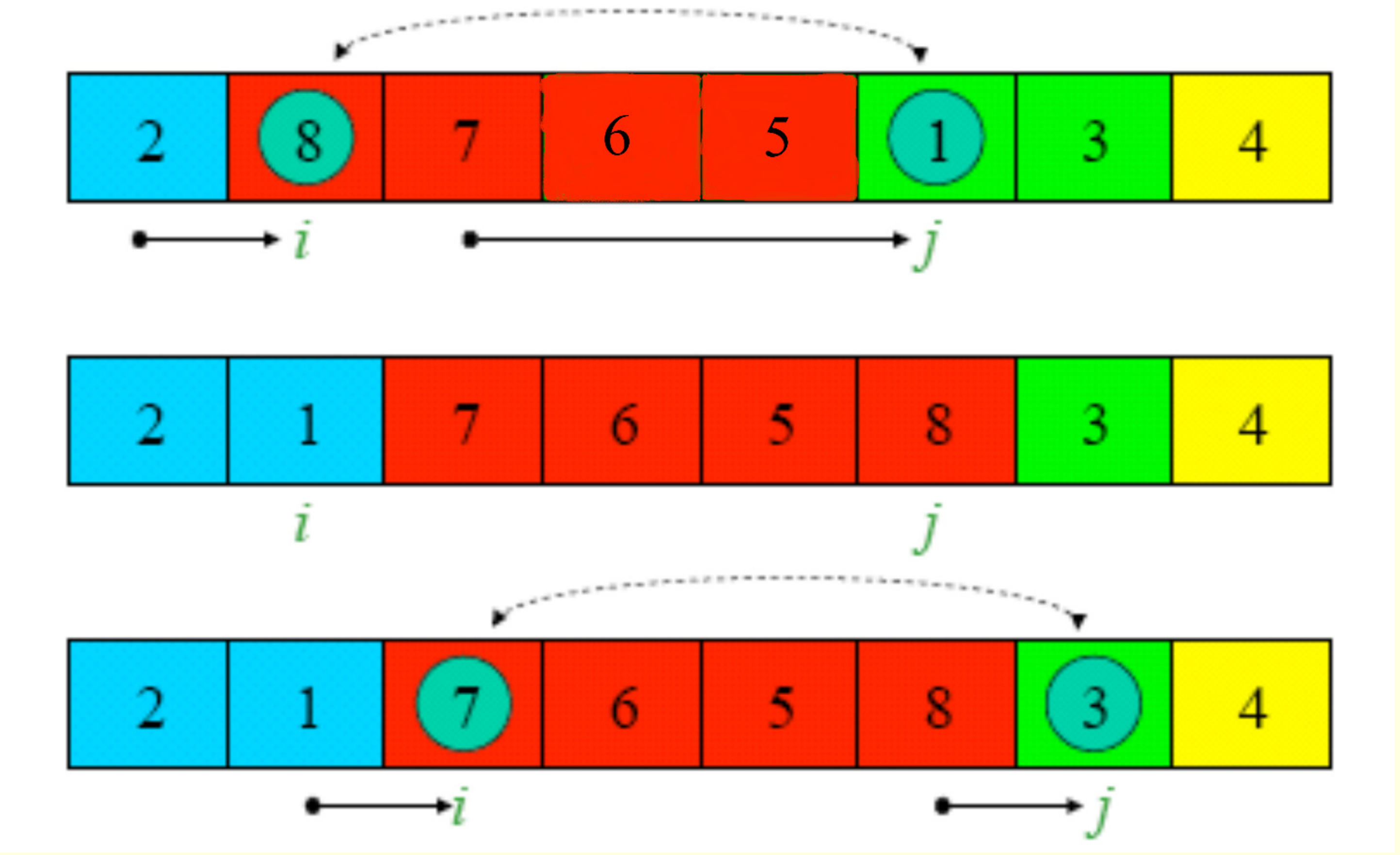
שגרת החלוקה – דוגמא

איבר הציר הוא 4



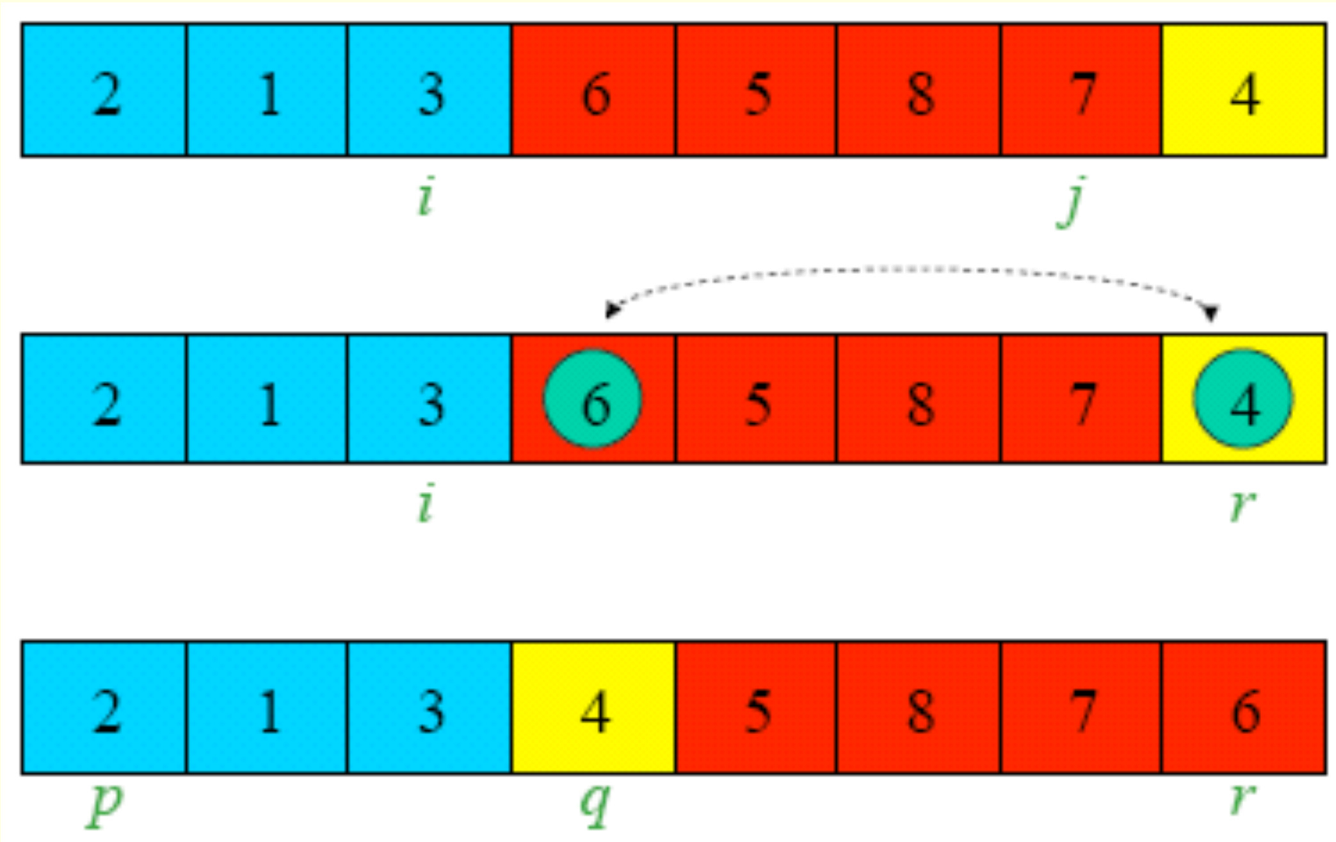
שגרת החלוקה – דוגמא (המשך)

איבר הציר הוא 4



שגרת החלוקה – דוגמא (המשך)

איבר הציר הוא 4



מיון מהיר - הגרסה של Hoare (בעיה 7-1)

- האלגוריתם:

Hoare-QuickSort (A, p, r)

1. **if** $p < r$
2. **then** $q \leftarrow$ Hoare-Partition (A, p, r)
3. Hoare-QuickSort (A, p, q)
4. Hoare-QuickSort $(A, q+1, r)$

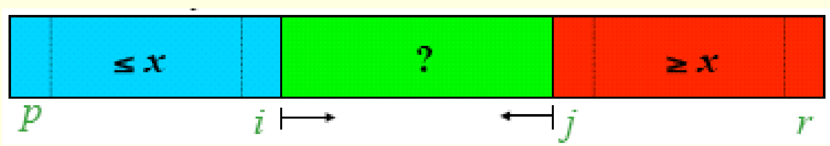
- ההבדל לעומת הגרסה שבספר:
  - מבצעים חלוקה של המערך לשני תת-מערכים, $L = A[p..q]$ ו- $R = A[q+1..r]$, כך שכל איבר באזור השמאלי קטן או שווה לכל איבר באזור הימני.
  - איבר הציר עצמו יכול להימצא בכל אחד משני האזורים!

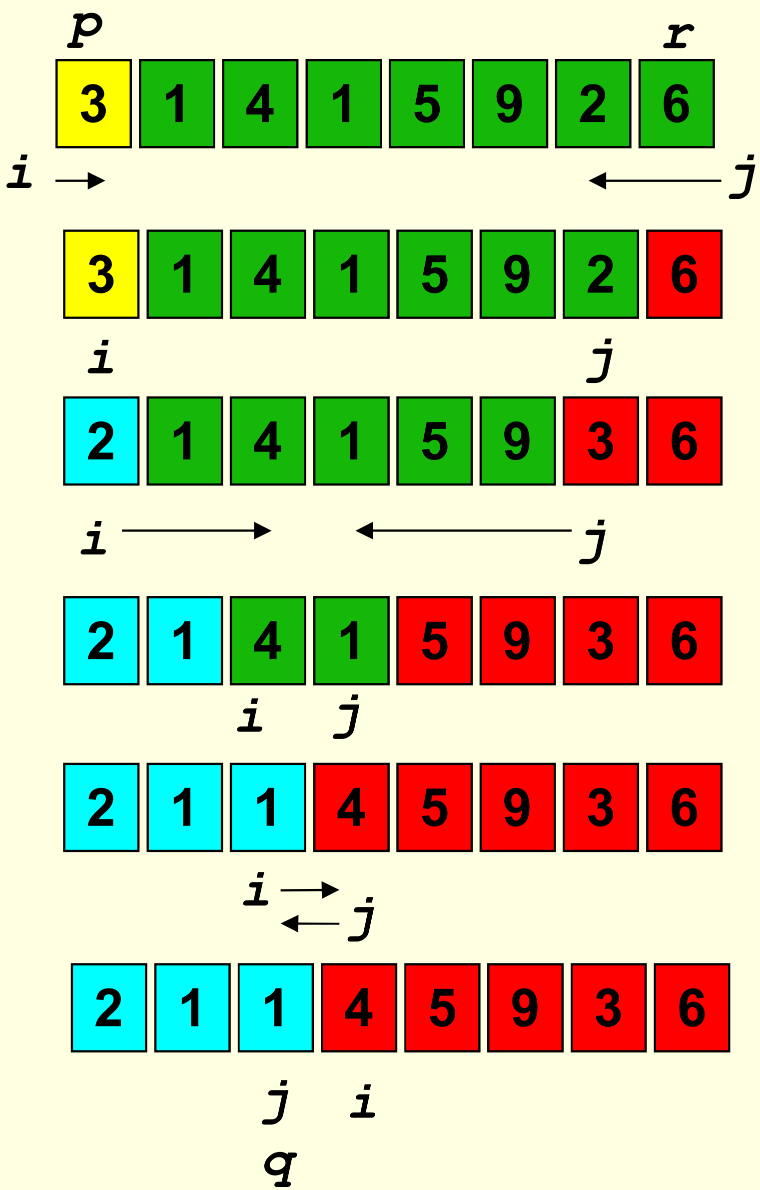
שגרת החלוקה של Hoare (בעיה 7-1)

Hoare-Partition(A, p, r)

```
x ← A[p]
i ← p – 1
j ← r + 1
while true do
    repeat j ← j – 1 until A[j] ≤ x
    repeat i ← i + 1 until A[i] ≥ x
    if i < j
        then exchange A[i] ↔ A[j]
        else return j
```

זמן הריצה: $O(n)$



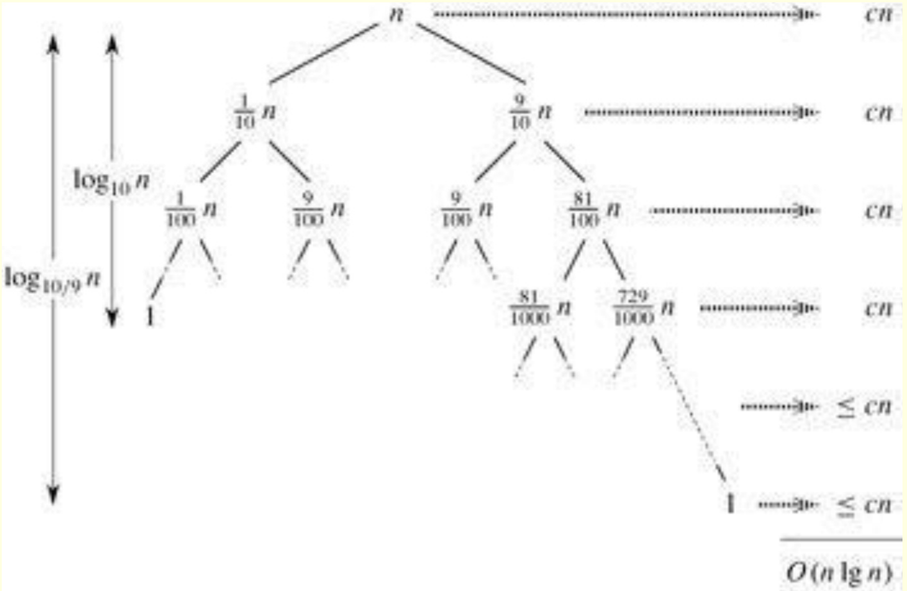


המקרה הטוב ביותר

- המקרה הטוב ביותר – כאשר החלוקה (כמעט) מאוזנת
  - $|R| = |L| \pm 1$
  - זה קורה, למשל, כאשר איבר הציר הוא *חציון* של איברי המערך
  - אם זה קורה (כמעט) לאורך כל אלגוריתם המיון, נוסחת הנסיגה היא:

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n\lg n)$$

- נוצר עץ קריאות בינארי שגובהו $\Theta(\lg n)$ וסך הפעולות בכל רמה הוא $\Theta(n)$
  - למעשה, כל חלוקה ביחס קבוע 1:k יוצרת עץ כזה! (ראו דיון בספר)



המקרה הגרוע ביותר

- המקרה הגרוע – כאשר החלוקה לא מאוזנת
  - זה קורה, למשל, כאשר איבר הציר הוא הקטן ביותר או הגדול ביותר במערך
    $|L| = 0$ ו- $|R| = n–1$, או להיפך
  - יריב מרושע המכיר את האלגוריתם יכול ללא קושי ליצור קלט כזה (למשל, מערך ממוין)
  - אם זה קורה (כמעט) לאורך כל אלגוריתם המיון, נוסחת הנסיגה היא:
    $$T(n) = T(n–1) + \Theta(n) = \Theta(n^2)$$
- ברצוננו להבטיח שהסיכוי להתרחשות המקרה הגרוע יהיה קטן מאוד
- בחירה אקראית של איבר הציר בכל קריאה רקורסיבית תביא להתנהגות המבוקשת.

אלגוריתם אקראי

- **אלגוריתם אקראי** – אלגוריתם המבצע במהלך ריצתו בחירות אקראיות
- קיימים שני סוגים של אלגוריתמים אקראיים:
  - **לאס וגאס**: האלגוריתם תמיד מחזיר תוצאה נכונה, אבל הבחירות האקראיות משפיעות על זמן הריצה והוא עלול להיות גדול
  - **מונטה קארלו**: זמן הריצה לא מושפע מהבחירות האקראיות, אבל האלגוריתם עלול להחזיר תוצאה שגויה (בהסתברות נמוכה)
- באלגוריתם אקראי, ריצות שונות על אותם נתונים יכולות להתנהג אחרת זו מזו (בצעדי הפעולות ו/או בסיבוכיות)
  - במונטה קרלו התוצאות גם כן יכולות להיות שונות זו מזו

- האלגוריתם RandomizedQuickSort הוא אלגוריתם לאס וגאס.

מימוש אקראי של מיון מהיר (מבוסס על Lomuto)

- כדי להקטין את הסיכוי שניתקל במקרה הגרוע, נשתמש באלגוריתם אקראי
- ההבדל היחידי הוא בבחירה אקראית של איבר הציר

RandomizedPartition (*A*, *p*, *r*)

1. $i \leftarrow$ Random (p, r)
2. exchange $A[r] \leftrightarrow A[i]$
3. **return** Partition (A, p, r)

RandomizedQuickSort (*A*, *p*, *r*)

1. **if** $p < r$
2. **then** $q \leftarrow$ RandomizedPartition(A, p, r)
3. RandomizedQuickSort (A, p, $q-1$)
4. RandomizedQuickSort (A, $q+1$, r)

מימוש אקראי – זמן ריצה

- אינטואיטיבית: נקבל זמן ריצה טוב כאשר החלוקות יהיו טובות.
  - נזכור כי כל חלוקה של 1:k היא טובה כל עוד k הוא קבוע
- מה הסיכוי לחלוקה רעה? קטן מאוד. שכן רוב איברי הציר האפשריים יתנו חלוקה סבירה.
- לפיכך רוב ההרצות יניבו עץ הרקורסיה פחות או יותר מאוזן
  - רק מעט הרצות יחרגו (במקרים הנדירים בהם בחרנו אקראית איבר ציר "לא טוב" כמעט בכל שלב).
- ברוב המוחלט של ההרצות של האלג', זמן הריצה יהיה $\Theta(n\lg n)$
  - הספר מוכיח את תוחלת זמן הריצה באופן פורמאלי

תוחלת מספר ההשוואות בשגרת החלוקה של Lomuto

- תוחלת – ממוצע צפוי (expected): ממוצע משוקלל של הערכים הצפויים. דוגמא – תוחלת תוצאת זריקת קוביה.
- נחשב את תוחלת מספר ההשוואות שיתבצעו במהלך המיון.
- נשים לב שבין כל זוג איברים תיערך לכל היותר השוואה אחת, ורק כאשר אחד מן האיברים הוא איבר הציר.
- מה הסיכוי שהאיבר שמיקומו הסופי הוא i יושווה עם איבר שמיקומו הסופי הוא j ?
- לדוגמה: נניח והקלט הוא המספרים 1-10
  - לכן מיקומו הסופי של 1 הוא 1, של 2 הוא 2 וכו'.
- מה הסיכוי ש- 2 ו- 7 יושוו?
  - רק אם 2 או 7 ייבחרו לאיבר ציר לפני כל אחד מהאיברים שביניהם
  - באופן כללי הסיכוי הוא $2/(j-i+1)$, כאשר $i<j$ (ראה עמ' 132 בספר)

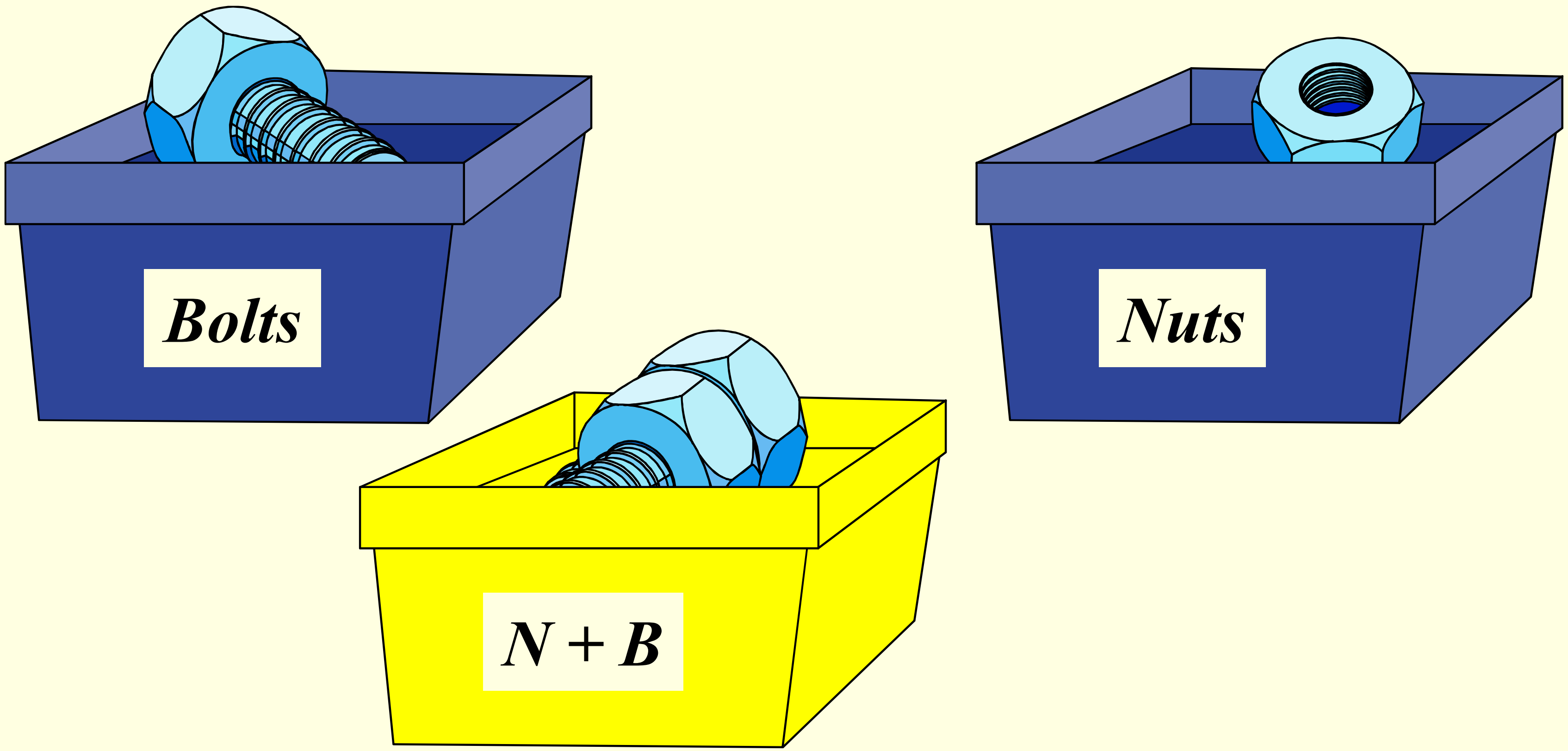
תוחלת מספר ההשוואות
בשגרת החלוקה של Lomuto

- נחשב את תוחלת מספר ההשוואות שיתבצעו במהלך המיון
  - נסמן ב-X_{ij} משתנה מקרי המקבל 1 אם האיבר שמיקומו הסופי i מושווה עם האיבר שמיקומו הסופי j, אחרת 0
  - מספר ההשוואות: $X = \Sigma_i \Sigma_{j>i} X_{ij}$
  - מלינאריות התוחלת: $E[X] = E[\Sigma_i \Sigma_{j>i} X_{ij}] = \Sigma_i \Sigma_{j>i} E[X_{ij}]$
  - X_{ij} אינדיקטור, לכן $E[X_{ij}]$ שווה לסיכוי ששני האיברים הנ"ל יושוו
  - סיכוי זה שווה ל- $2/(j-i+1)$
  - מכאן:

$$
\begin{aligned}
E[X] &= \Sigma_{i=1}^{n} \Sigma_{j>i}^{n}\, 2/(j-i+1) \\
&= \Sigma_{i=1}^{n} \Sigma_{k=2}^{n-i+1}\, 2/k \\
&\le 2 \cdot \Sigma_{i=1}^{n} \Sigma_{k=1}^{n}\, 1/k \\
&= 2nH_n \qquad \blacktriangleright\ H_n = O(\lg n) \text{ is the } n\text{th harmonic number} \\
&= O(n \lg n)
\end{aligned}
$$

מיון-מהיר – תרגול

- נתונים *n* ברגים בגדלים שונים ו-*n* אומים המתאימים להם.
הציעו אלגוריתם יעיל להתאמת הברגים והאומים.
מותר לאלגוריתם לבצע השוואה רק בין בורג לאום.



תרגול

בעיה 7-3 בספר: StoogeSort

StoogeSort(*A*, *i*, *j*)

1. **if** $A[i] > A[j]$
2. **then** exchange $A[i] \leftrightarrow A[j]$
3. **if** $i+1 \geq j$
4. **then return**
5. $k \leftarrow \lfloor (j - i + 1)/3 \rfloor$
6. StoogeSort(A, i, $j - k$)
7. StoogeSort(A, $i + k$, j)
8. StoogeSort(A, i, $j - k$)

תרגול

בעיה 7-3 – המשך

א. הראו ש- StoogeSort (A, 1, length[A]) ממיינת נכון את מערך הקלט $A[1..n]$ כאשר $n = length[A]$

ב. כתבו נוסחת נסיגה עבור זמן הריצה של StoogeSort במקרה הגרוע, ומצאו חסם הדוק אסימפטוטית על זמן הריצה במקרה הגרוע.

ג. השוו את זמן הריצה של StoogeSort במקרה הגרוע לזמני הריצה של מיון-הכנסה, מיון-ערמה ומיון-מהיר.