הוכן ע"י אמיר רובינשטיין

מבני נתונים
ומבוא לאלגוריתמים

נושא 4

מיון מהיר, הוכחת נכונות של אלגוריתמים
Quick-Sort, proof of correctness

בתוכנית

פרק 7 בספר הלימוד

- נלמד על האלגוריתם מיון מהיר (Quick-Sort),
  ועל אלגוריתם החלוקה (Partition).

- נראה שתי גרסאות של מיון מהיר:

1. בסיסי 2. אקראי (בהמשך הקורס נראה גרסה נוספת)

- נכיר את המושג "אלגוריתמים אקראיים"
- נלמד מהי יציבות (stability) של מיון

- נלמד מהי הוכחת נכונות (correctness) של אלגוריתם

מיון מהיר – Quick-Sort

- פותח ע"י C.A.R. Hoare ב-1962, בהיותו בן 26

- בדומה למיון מיזוג - אלגוריתם רקורסיבי,
  עובד בשיטת הפרד-משול-צרף

- בשונה ממיון מיזוג –
  - ממיין במקום (in-place), כלומר $\Theta(1)$ איברים מאוחסנים בכל רגע מחוץ למערך הקלט.
  - עושה את כל העבודה לפני הקריאות הרקורסיביות (שלב ה- Divide): מחלק את המערך לשני חלקים (לאו דווקא חצאים!).

- נלמד שלוש גרסאות:

גרסה	מקרה גרוע	ממוצע	מקרה טוב
בסיסית	$\Theta(n^2)$	$\Theta(n\log n)$	$\Theta(n\log n)$
אקראית	$\Theta(n^2)$		
דטרמיניסטית	$\Theta(n\log n)$		

בהמשך הקורס

- אלגוריתם מהיר מאוד באופן מעשי

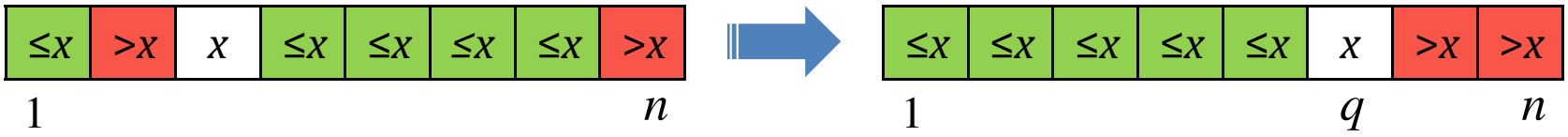
הרעיון

הקלט: מערך A בגודל n.

1. " הפרד "

- בחר איבר ציר (x (pivot מתוך המערך*

- חלק את A לשני אזורים: $\leq x$ ו- $> x$ כאשר x ביניהם



2. " משול "

- מיין רקורסיבית את $A[1\,..\,q\text{-}1]$
- מיין רקורסיבית את $A[q\text{+}1\,..\,n]$

3. " צרף "

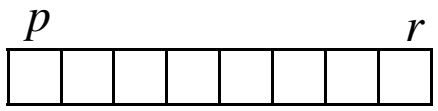
- כלום, המערך ממוין !

גרסה	הציר (בכל שלב) נבחר מתוך תת המערך עליו עובדים:
בסיסית	כאיבר הימני ביותר
אקראית	כאיבר אקראי
דטרמיניסטית	כחציון

* בגרסאות אחרות מאלו שנראה, x יכול להיות איבר שכלל אינו ב- A

האלגוריתם

הקלט: מערך A,
גבול שמאלי p
וגבול ימני r.



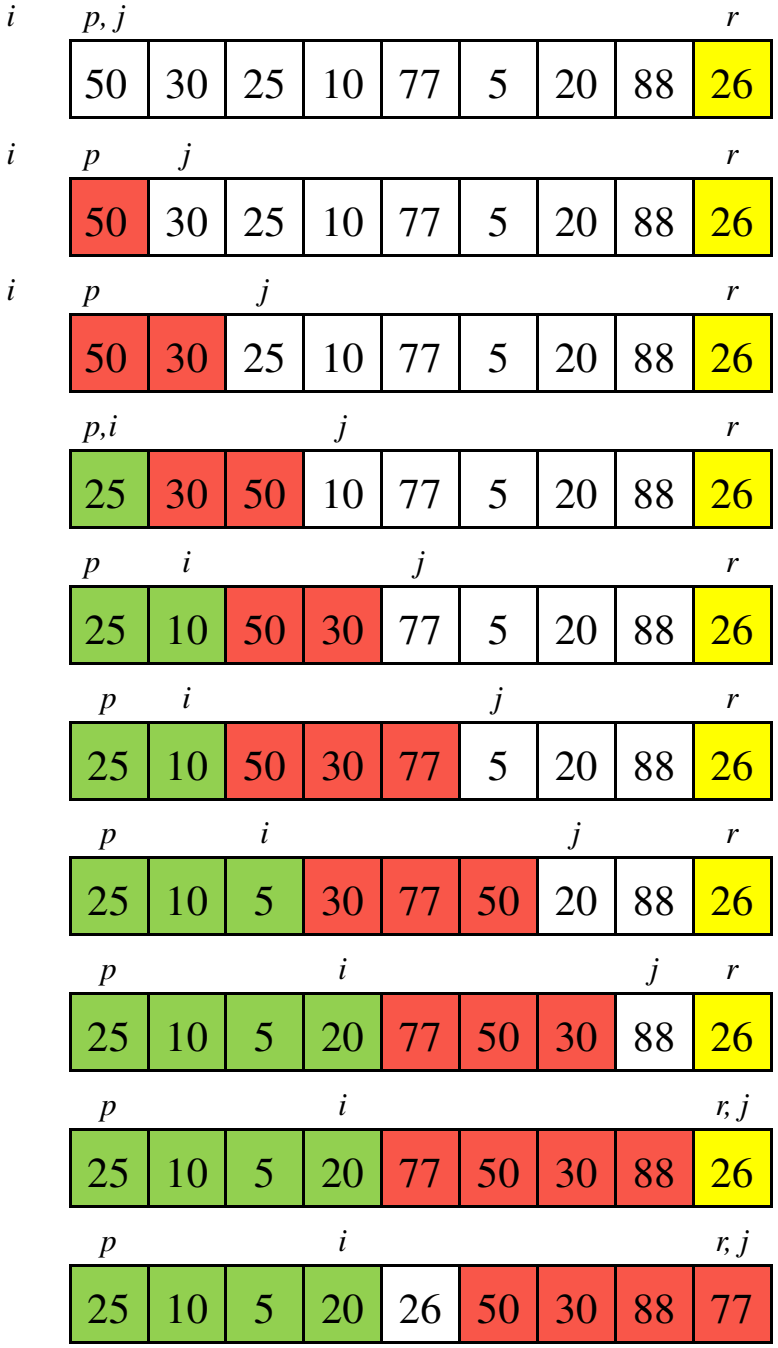

```
Quick-Sort (A, p, r)
1.    if  p < r
2.          q ← Partition(A, p, r)
3.          Quick-Sort (A, p, q−1)
4.          Quick-Sort (A, q+1, r)
```

הקריאה הראשית (n הוא גודל המערך):

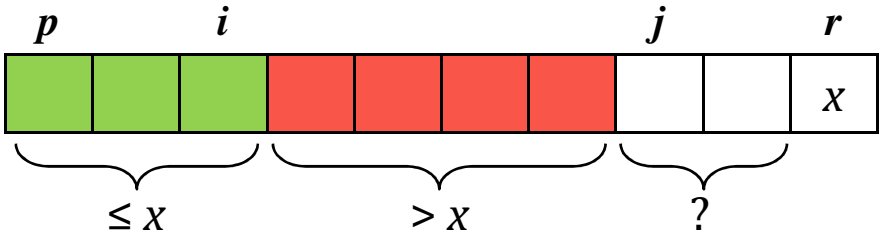
```
Quick-Sort (A, 1, n)
```

מהו תנאי העצירה?

Partition - החלוקה

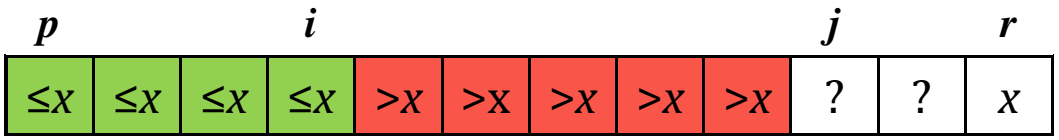



```
Partition(A, p, r)
1.   x ← A[r]
2.   i ← p − 1
3.   for j ← p to r − 1
4.        if A[j] ≤ x
5.             i ← i + 1
6.             exchange A[i] ↔ A[j]
7.   exchange A[i + 1] ↔ A[r]
8.   return i + 1
```



החלוקה – הדגמה דינאמית

```
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] ↔ A[j]
7.    exchange A[i + 1] ↔ A[r]
8.    return i + 1
```



בסיום החלוקה:

- x נמצא במקומו הסופי.
- ייתכן שאחד האזורים ריק.

הוכחת נכונות של אלגוריתמים

כדי להוכיח שאלגוריתם עובד נכון (כלומר לכל קלט חוקי מפיק את הפלט הרצוי),

עלינו להוכיח 2 דברים:

1. **עצירה** – לכל קלט חוקי, האלגוריתם עוצר לאחר מספר סופי של פעולות
2. **נכונות הפלט** – לכל קלט חוקי, כאשר האלגוריתם עוצר, הפלט שהוא מספק הוא הפלט המצופה

נדגים כעת הוכחת נכונות של Quick-Sort.

דוגמה נוספת להוכחת נכונות (של מיון בועות) נמצאת לקראת סוף המצגת.

הוכחת נכונות של מיון מהיר - עצירה

טענה 1:
Quick-Sort עוצר (מסתיים לאחר זמן סופי).

הוכחה:

- Partition בודאי עוצר, שכן הוא מבצע מספר סופי של איטרציות שכל אחת סופית, ועוד מספר קבוע של פעולות.
- תנאי העצירה של Quick-Sort הוא מערך בגודל 1. בכל קריאה רקורסיבית גודל הבעיה קטן ממש (לפחות ב- 1) לכן לאחר מספר סופי של שלבי רקורסיה (שכל אחד סופי) נגיע לתנאי העצירה.

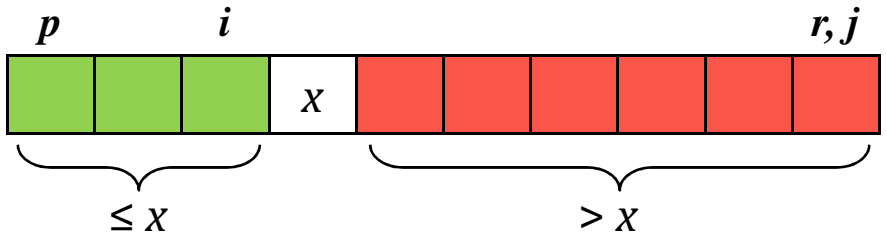
מ.ש.ל. (טענה 1)

הוכחת נכונות של מיון מהיר - החלוקה

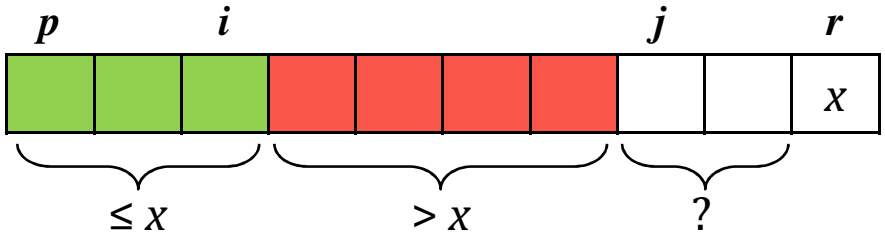
```
Partition(A, p, r)
1.   x ← A[r]
2.   i ← p − 1
3.   for j ← p to r − 1
4.        if A[j] ≤ x
5.             i ← i + 1
6.             exchange A[i] ↔ A[j]
7.   exchange A[i + 1] ↔ A[r]
8.   return i + 1
```

טענה 2: בסיום החלוקה,

- x נמצא במקומו הסופי
- משמאלו איברים שאינם גדולים ממנו
- מימינו איברים שגדולים ממנו



לשם הוכחת טענה 2 ננסח ונוכיח תכונה שמתקיימת בתחילת כל איטרציה של הלולאה בשורות 3-6.
תכונה כזו מכונה **שמורת לולאה** (loop invariant).



שמורת הלולאה בשורות 3-6:

בתחילת כל איטרציה, לכל אינדקס k במערך:

א. אם $p \le k \le i$ אז $A[k] \le x$

ב. אם $i+1 \le k \le j-1$ אז $A[k] > x$

ג. $A[r] = x$

הוכחת נכונות של מיון מהיר - החלוקה

```
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] ↔ A[j]
7.   exchange A[i + 1] ↔ A[r]
8.   return i + 1
```

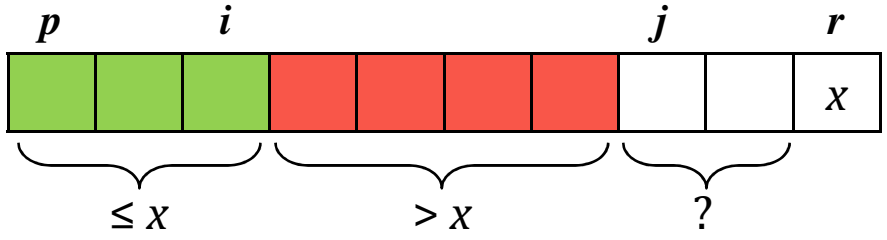
שמורת הלולאה בשורות 3-6:

בתחילת כל איטרציה, לכל אינדקס k במערך:

א. אם $p \le k \le i$ אז $A[k] \le x$

ב. אם $i+1 \le k \le j-1$ אז $A[k] > x$

ג. $A[r] = x$



הוכחת שמורת הלולאה (באינדוקציה על מספר האיטרציה):

בסיס: בתחילת האיטרציה הראשונה, א'+ב' מתקיימים באופן ריק, ג' מתקיים בגלל שורה 1.

צעד: נניח ששמורת הלולאה התקיימה בתחילת איטרציה m כלשהי, ונוכיח שהיא מתקיימת גם בתחילת האיטרציה ה- $m+1$.

אם $A[j] > x$ אז רק מקדמים את j, לכן תכונה ב' ממשיכה להתקיים, ואין פגיעה בתכונות א' ו- ג'.

אם $A[j] \le x$ אז בזכות ההחלפה בשורה 6 עדיין $A[i] \le x$ (עם ה- i החדש) לכן תכונה א' ממשיכה להתקיים, ועדיין $A[j-1] > x$ (עם ה- j החדש) לכן תכונה ב' ממשיכה להתקיים, ואין פגיעה בתכונה ג'.

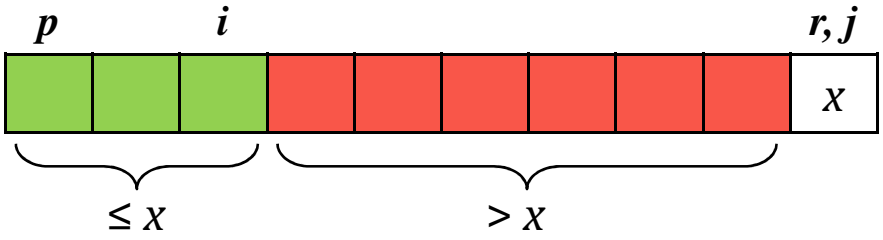
סיום הלולאה: בסוף האיטרציה האחרונה $j=r$, כלומר $A[p..i] \le x$ וגם $A[i+1..r-1] > x$ וגם $A[r] = x$.

הוכחת נכונות של מיון מהיר - החלוקה

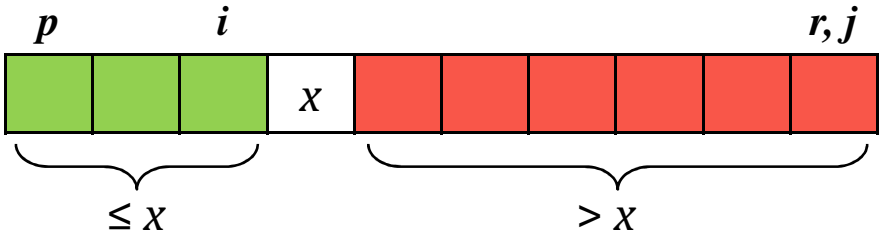
```
Partition(A, p, r)
1.   x ← A[r]
2.   i ← p − 1
3.   for j ← p to r − 1
4.        if A[j] ≤ x
5.             i ← i + 1
6.             exchange A[i] ↔ A[j]
7.   exchange A[i + 1] ↔ A[r]
8.   return i + 1
```

המשך הוכחת טענה 2

כאמור בשקף הקודם, מהוכחת שמורת הלולאה נובע שהמצב בסיום הלולאה הוא כזה:



מהחלפה בשורה 7 נובעת נכונות טענה 2:



מ.ש.ל. (טענה 2)

הוכחת נכונות של מיון מהיר

```
Quick-Sort (A, p, r)
1.   if  p < r
2.          q ← Partition(A, p, r)
3.          Quick-Sort (A, p, q−1)
4.          Quick-Sort (A, q+1, r)
```

טענה 3

Quick-Sort $(A, 1, n)$ ממיין את A.

הוכחה (באינדוקציה על n):

בסיס: עבור $n=1$ האלגוריתם נכון (מדוע?).

צעד: נניח ש- Quick-Sort ממיין נכון מערכים בגודל $k < n$.

לפי טענה 2, בסיום שורה 2 מתקיים שלכל $p \le k_1 \le q\text{-}1$ ולכל $q\text{+}1 \le k_2 \le r$: $A[k_1] \le A[q] < A[k_2]$.

לפי הנחת האינדוקציה, בשורות 3 ו- 4 מתבצע מיון נכון של שני תת-מערכים: $A[p..q\text{-}1]$ ו- $A[q\text{+}1..r]$.

משני הנ"ל נובע, שבסיום Quick-Sort המערך A ממוין.

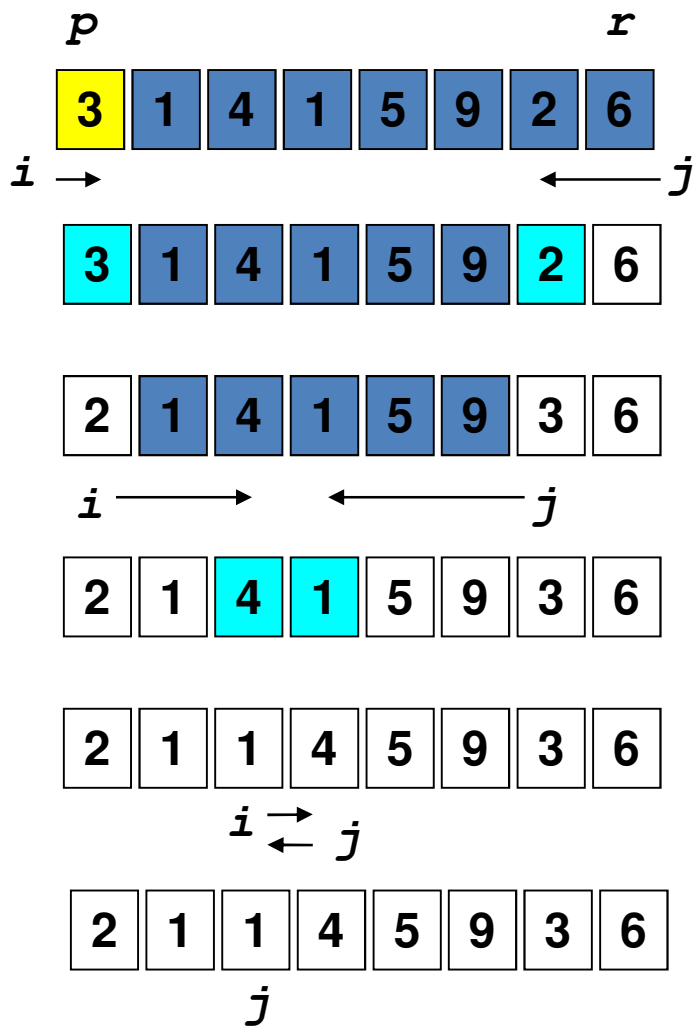
מ.ש.ל. (טענה 3)

בזאת הסתיימה הוכחת הנכונות של מיון מהיר, QED.

גרסה אחרת של חלוקה

החלוקה המקורית שחיבר HOARE:

```
Hoare-Partition (A, p, r)
1.    x ← A[p]
2.    i ← p − 1
3.    j ← r + 1
4.    while TRUE
5.        repeat  j ← j − 1
6.        until   A[j] ≤ x
7.        repeat  i ← i + 1
8.        until   A[i] ≥ x
9.        if  i < j
10.           exchange  A[i] ↔ A[j]
11.       else return  j
```



בסיום החלוקה:

- האם בהכרח x נמצא במקומו הסופי?
- האם ייתכן שאחד האזורים ריק?
- איזה שינוי צריך לבצע ב- Quick-Sort?

ראו בעיה 7-1 בספר הלימוד.

ניתוח זמן ריצה

סיבוכיות הזמן של החלוקה (לא משנה איזו גרסה) היא $\Theta(n)$.

- <u>המקרה הגרוע</u>

כאשר החלוקה <u>בכל שלב</u> של הרקורסיה היא בלתי מאוזנת באופן הקיצוני ביותר:

- בגרסה הראשונה של החלוקה: איזור אחד בגודל 0 והשני בגודל n-1
- בגרסת HOARE ?

זה קורה כאשר איבר הציר הוא המינימום או המקסימום של תת-המערך, בכל שלב.

<u>תרגיל</u>: הדגימו את ריצתו של מיון מהיר על מערך ממוין, ועל מערך ממוין הפוך.

$$T(n) = T(n-1) + T(0) + \Theta(n) = \Theta(n^2)$$

- <u>המקרה הטוב</u>

כאשר החלוקה <u>בכל שלב</u> של הרקורסיה היא מאוזנת: חלוקה לשני חצאים בערך.

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

ואם החלוקה היא ביחס α:(1- α) עבור $0<\alpha<1$?

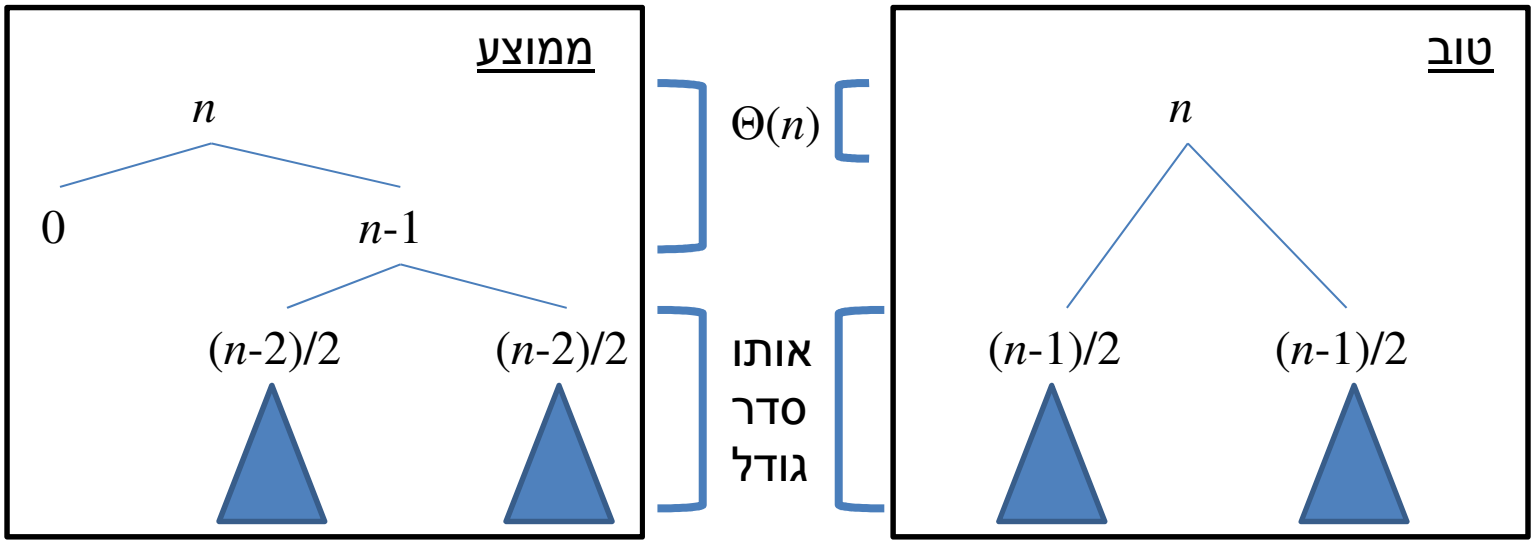
זמן ריצה ממוצע

- <u>זמן ריצה ממוצע</u>

<u>טענה</u>: זמן הריצה הממוצע של מיון מהיר הוא $\Theta(n\log n)$.

מה פירוש זמן ריצה ממוצע (או תוחלת זמן ריצה)?
ממוצע זמני הריצה של אלגוריתם על פני כל הקלטים האפשריים שלו בגודל n.

את הטענה הנ"ל לא נוכיח (הוכחה בספר הלימוד), אבל נסביר אותה אינטואיטיבית:
בריצה כלשהי של מיון מהיר ישנה תערובת של חלוקות "גרועות" וחלוקות "טובות".
נניח לשם פשטות שיש רק חלוקות גרועות <u>ביותר</u> וטובות <u>ביותר</u>, ושהן מופיעות <u>לסירוגין</u> בזו אחר זו. אז זמן הריצה הממוצע שקול לזמן הריצה במקרה הטוב:



אלגוריתמים אקראיים

- אלגוריתם אקראי הוא אלגוריתם שעושה שימוש במחולל מספרים אקראיים (random-number generator).

- היתרון של אלגוריתם אקראי לעומת אלגוריתם דטרמיניסטי (לא אקראי) הוא חסינות בפני "יריב מרושע" שמכיר את פרטי האלגוריתם ויכול לייצר קלט שיהווה מקרה גרוע.

- בפועל רוב סביבות התכנות מציעות מחולל מספרים פסידו-אקראיים – כלומר אלגוריתם דטרמיניסטי (לא אקראי) שמחזיר מספרים שנראים סטטיסטית אקראיים מספיק.

- קיימים שני סוגים של אלגוריתמים אקראיים:

לאס וגאס: האלגוריתם תמיד מחזיר תוצאה נכונה, אבל הבחירות האקראיות משפיעות על זמן הריצה והוא עלול להיות גדול (מיון מהיר האקראי הוא כזה).

מונטה קארלו: זמן הריצה לא מושפע מהבחירות האקראיות, אבל האלגוריתם עלול להחזיר תוצאה שגויה (ההסתברות לכך חסומה – לכל היותר ½)

מיון מהיר האקראי

```
Random-Partition (A, p, r)
1.   i ← Random (p, r)
2.   exchange A[r] ↔ A[i]
3.   return Partition (A, p, r)

Random-Quick-Sort (A, p, r)
1.   if  p < r
2.         q ← Random-Partition(A, p, r)
3.         Random-Quick-Sort (A, p, q−1)
4.         Random-Quick-Sort (A, q+1, r)
```

למיון מהיר האקראי אותם סדרי גודל של זמני ריצה.

כאן אפשר לדבר גם על זמן ריצה ממוצע, כאשר <u>הממוצע הוא על פני ההגרלות שמבצע האלגוריתם (על אותו קלט)</u>.

<u>שאלה</u>: מהי ההסתברות להתרחשות המקרה הגרוע?

הסיכוי שבשלב הראשון ייבחר המינימום/מקסימום כציר הוא $2/n$.

בשלב השני: $2/(n\text{-}1)$

בשלב ה- i: $2/(n\text{-}i\text{+}1)$

סה"כ הסיכוי לכך שבכל שלב ייבחר המינימום/מקסימום הוא $2^{n-1}/n!$.

יציבות של מיון

עוד תכונה לפיה ניתן להשוות בין אלגוריתמי מיון היא היציבות (stability).

נאמר שמיון היא יציב (stable) אם הוא שומר על סדר יחסי של איברים בעלי מפתחות זהים.

3 1 3’ 0 5 3’’ → 0 1 3 3’ 3’’ 5

האם מיון מהיר הוא יציב?

- מה לגבי מיון מיזוג? מיון בועות? מיון הכנסה? ...

הגבלת עומק מחסנית הרקורסיה של מיון מהיר

כאמור מיון מהיר הוא מיון במקום (in-place) – כלומר $\Theta(1)$ איברים מאוחסנים בכל רגע מחוץ למערך הקלט.

אבל זהו אלגוריתם רקורסיבי – משאבי הזיכרון שהוא דורש תלויים גם במחסנית הרקורסיה.

מהו עומק הרקורסיה של מיון מהיר:

- במקרה הגרוע?
- במקרה הטוב?
- בממוצע?

ניתן להקטין את עומק מחסנית הרקורסיה, באמצעות טכניקה שנקראת **"העלמת רקורסיית זנב"**.

- רקורסיית זנב הוא מצב בו הפעולה האחרונה שמבצע אלגוריתם היא קריאה רקורסיבית.
- ניתן להחליף קריאה זו באיטרציה.

העלמת רקורסית זנב (tail recursion)

בעייה 7-4 בספר הלימוד

נתבונן בגרסה הבאה של מיון מהיר, בה הקריאה הרקורסיבית השנייה על החלק הימני הוחלפה באיטרציה של לולאה:

```
Quick-Sort'(A, p, r)
1.    while p < r
2.        do  ► partition and sort left sub-array
3.            q ← Partition (A, p, r)
4.            Quick-Sort'(A, p, q − 1)
5.            p ← q + 1
```

הערה:

בקומפיילרים מודרניים מימוש של העלמת רקורסיית זנב (tail recursion) נעשה אוטומטית.

א. נמקו מדוע גרסה זו עובדת נכון.

ב. תארו מקרה שבו עומק מחסנית הרקורסיה הוא $\Theta(n)$.

ג. הציעו שינוי ל- QuickSort' כך שעומק מחסנית הרקורסיה יוגבל במקרה הגרוע ל- $\Theta(\log n)$, ללא פגיעה בסיבוכיות הזמן של האלגוריתם.

העלמת רקורסית זנב (tail recursion)

<u>פתרון</u>

א. למעשה Quick-Sort ו- Quick-Sort' עושות בדיוק אותו דבר:

שניהם מתחילים מאותה חלוקה, ואז ישנה קריאה רקורסיבית על $A[p..q\text{-}1]$.

אח"כ Quick-Sort מבצעת קריאה רקורסיבית על $A[q+1..r]$.

Quick-Sort' במקום זאת מציבה $p \leftarrow q + 1$ ואח"כ קוראת לעצמה עם $A[p..r]$.

כך ששתי השגרות מטפלות באותם תת-מערכים ובאותו סדר בדיוק.

ב. אם בכל פעם הציר הוא המקסימום אז Quick-Sort' תקרא לעצמה בכל פעם עם תת-מערך שגודלו קטן ב- 1. במצב כזה יהיו $\Theta(n)$ קריאות רקורסיביות:

Quick-Sort'(A, 1, n)

Quick-Sort'(A, 1, n-1)

Quick-Sort'(A, 1, n-2)

…

Quick-Sort'(A, 1, 1)

הדבר מתרחש כאשר המערך A כבר ממוין (בסדר עולה).

העלמת רקורסית זנב (tail recursion)

<u>פתרון</u>

ג. הרעיון הוא לבצע את הקריאה הרקורסיבית ל- Quick-Sort' בכל איטרציה עם תת-המערך הקטן יותר:

```
Quick-Sort''(A, p, r)
1.    while p < r
2.          ► partition and sort the small sub-array first
3.          q ← Partition (A, p, r)
4.          if  q – p < r – q
5.                Quick-Sort''(A, p, q – 1)
6.                p ← q + 1
7.          else  Quick-Sort''(A, q + 1, r)
8.                r ← q – 1
```

גודל התת-מערך בכל קריאה הוא לכל היותר מחצית מהגודל הנוכחי, ולכן מספר הקריאות (ומכאן שעומק המחסנית) הוא $\Theta(\log n)$ במקרה הגרוע.

זמן הריצה של המיון לא נפגע, שכן אותן חלוקות נעשות ואותם תתי-מערכים מטופלים (רק אולי בסדר שונה כעת).

דוגמה נוספת להוכחת נכונות - מיון בועות

נדגים הוכחת נכונות על מיון בועות:

```
Bubble-Sort(A, n)
1.   for i ← 1 to n
2.       for j ← n downto i+1
3.           if A[j] < A[j-1]
4.               exchange A[j]  □  A[j-1]
```

1. <u>עצירה</u>: ברור שהאלגוריתם עוצר כי כל לולאה מבצעת מספר סופי של איטרציות ובכל איטרציה מספר סופי של פעולות.

2. <u>נכונות הפלט</u>: ננסח תכונות שמתקיימת בתחילת כל אחת מהלולאות (שמורות לולאה):

<u>שמורת הלולאה החיצונית</u>: בהתחלת כל איטרציה של לולאת ה- **for** שבשורות 1-4, התת-מערך $A[1..i\text{-}1]$ מכיל את $i\text{-}1$ האיברים הקטנים ביותר ב- A בסדר ממוין.

<u>שמורת הלולאה הפנימית</u>: בהתחלת כל איטרציה של לולאת ה-**for** שבשורות 2-4, האיבר $A[j]$ הוא האיבר המינימלי בתת-מערך $A[j..n]$

נכונות של מיון בועות – לולאה פנימית

```
Bubble-Sort(A, n)
1.    for  i ← 1 to n
2.         for  j ← n  downto  i+1
3.              if  A[j] < A[j-1]
4.                   exchange A[j]  □  A[j-1]
```

שמורת הלולאה הפנימית: בהתחלת כל איטרציה של לולאת ה-**for** שבשורות 2-4,

האיבר $A[j]$ הוא האיבר המינימלי בתת-מערך $A[j..n]$

אתחול: בתחילת האיטרציה הראשונה $j = n$; התת-מערך $A[j..n]$ מכיל איבר אחד בלבד, לכן שמורת הלולאה מתקיימת.

תחזוקה: נניח ששמורת הלולאה הפנימית מתקיימת בהתחלת איטרציה כלשהי, כלומר $A[j]$ הוא האיבר המינימלי בתת-מערך $A[j..n]$; אחרי ההשוואה בשורה 3, אם $A[j] < A[j-1]$, מתבצעת גם ההחלפה שבשורה 4; בסוף האיטרציה הזו (ובהתחלת האיטרציה הבאה), $A[j-1]$ הוא האיבר המינימלי בתת-מערך $A[j-1..n]$, כלומר שמורת הלולאה מתקיימת גם בתחילת האיטרציה הבאה.

סיום: אחרי ביצוע האיטרציה האחרונה $j = i$; האיבר $A[i]$ הוא האיבר המינימלי בתת-מערך $A[i..n]$.

נכונות של מיון בועות – לולאה חיצונית

```
Bubble-Sort(A, n)
1.    for i ← 1 to n
2.        for j ← n downto i+1
3.            if A[j] < A[j-1]
4.                exchange A[j] □ A[j-1]
```

שמורת הלולאה החיצונית: בהתחלת כל איטרציה של לולאת ה- **for** שבשורות 1-4,

התת-מערך $A[1..i-1]$ מכיל את $i-1$ האיברים הקטנים ביותר ב- A בסדר ממוין.

אתחול: לפני האיטרציה הראשונה, $i=1$ והתת-מערך $A[1..i-1]$ ריק, לכן שמורת הלולאה מתקיימת באופן ריק.

תחזוקה: נניח ששמורת הלולאה החיצונית מתקיימת בהתחלת האיטרציה ה-i, כלומר התת-מערך $A[1..i-1]$ מכיל את $i-1$ האיברים הקטנים ביותר בסדר ממוין; כפי שראינו, הלולאה הפנימית מעבירה את האיבר המינימלי של התת-מערך $A[i..n]$ ל-$A[i]$; לכן, בסוף האיטרציה ה-i (בהתחלת האיטרציה ה-$(i+1)$) התת-מערך $A[1..i]$ מכיל את i האיברים הקטנים ביותר בסדר ממוין, כלומר שמורת הלולאה מתקיימת גם לפני האיטרציה ה-$(i+1)$.

סיום: אחרי ביצוע האיטרציה ה-$(n-1)$, התת-מערך $A[1..n-1]$ מכיל את $n-1$ האיברים הקטנים ביותר בסדר ממוין; אבל אז, $A[n]$ הוא האיבר הגדול ביותר, לכן כל המערך $A[1..n]$ הוא עכשיו ממוין.

מ.ש.ל.

שאלות חזרה

1. עבור מערך שכל איבריו שונים זה מזה, איזה ערך של q מחזירה Partition כאשר איבר הציר $A[r]$ הוא המקסימום? וכאשר הוא המינימום?

2. הדגימו את ריצתו של מיון מהיר על מערך ממוין, שכל איבריו שונים זה מזה. מהי סיבוכיות זמן הריצה במקרה זה?

3. ודאו שאתם מבינים: במיון מהיר, החלוקה של המערך בכל שלב היא דינמית, כלומר: בכל שלב תיתכן חלוקה שונה. זאת בניגוד למיון מיזוג, בו החלוקה היא סטטית (תמיד לשני חצאים).

תשובות לשאלות חזרה

1. כאשר הוא המקסימום, Partition תחזיר את הערך n (כאשר n הוא גודל המערך).
   כאשר הוא המינימום, היא תחזיר את הערך 1.

2. בכל שלב תתבצע חלוקה לאזור שמאלי בגודל n-1, ולאזור ימני ריק. הסיבוכיות $\Theta(n^2)$.

תרגילים

תרגילים נוספים

1. נשנה את שורה 1 באלגוריתם Quick-Sort ל:

1. **if** $r\text{-}p \geq k$

כלומר, לא ממיינים תתי-מערכים בגודל k או פחות.

במקרה כזה אומרים שהפלט של Quick-Sort הוא מערך "כמעט ממוין עם שגיאה בגודל k".

נתחו את סיבוכיות הזמן של אלגוריתם זה במקרה הגרוע ובמקרה הטוב, כתלות ב- n וב- k.

2. הדגימו את ריצתו של מיון מהיר על מערך $A[1..n]$ הממוין בסדר הפוך, שכל איבריו שונים זה מזה. מהי סיבוכיות זמן הריצה במקרה זה?

3. הדגימו את ריצת Partition ואת ריצת Hoare-Partition על מערך שכל מפתחותיו שווים זה לזה.
מהם זמני הריצה של Quick-Sort על מערך שכל מפתחותיו שווים, עם כל אחת מגרסאות החלוקה?

פתרון 1

מקרה גרוע: $\Theta(n^2 - k^2)$.
מקרה טוב: $\Theta(n \log(n/k))$.

פתרון 2

בשלב הראשון איבר הציר הוא המינימום: האיבר הראשון והאחרון מתחלפים, וממשיכים עם תת המערך $A[2..n]$.
בשלב השני איבר הציר הוא המקסימום: החלוקה לא משנה דבר וממשיכים עם $A[2..n\text{-}1]$.
השלב השלישי דומה לראשון, והרביעי לשני וכך הלאה.

כללית – בכל שלב מתקבלת חלוקה שאחד האזורים שלה ריק.
זמן הריצה הוא כמו במקרה הגרוע - $\Theta(n^2)$.