אלגוריתמים קומבינטוריים
סיכומים של תרגילי כיתה מסמסטרים קודמים בנושא
מיון ובעית הבחירה

1. סיכום אלגוריתמי המיון שנלמד:

האלגוריתם	זמן (גרוע)	זמן (ממוצע)	זמן (טוב)	זכרון נוסף	הנחות והערות
INSERTION-SORT	$O(n^2)$		$O(n)$	$O(1)$	מיון במקום
MERGE-SORT	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	
QUICK-SORT	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	מהר בפעל
HEAP-SORT	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	מיון במקום
COUNTING-SORT	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	$a_i \in \{1, 2, \ldots, k\}$
RADIX-SORT	$O(d(n+k))$	$O(d(n+k))$	$O(d(n+k))$	$O(n+k)$	d ספרות, k סימנים
BUCKET-SORT	$O(n \log n)$	$O(n)$	$O(n)$	$O(n)$	מפולג אחיד

2. INSERTION-SORT:

קלט: סדרה $a_1, a_2, \ldots, a_n$.

פלט: הסדרה ממוינת בסדר לא יורד: $a'_1, a'_2, \ldots, a'_n$ כך ש- $a'_1 \le a'_2 \le \ldots \le a'_n$.

(א) האלגוריתם: נניח שהסדרה כבר מופיע במערך $\mathrm{A}[1 \ldots n]$.

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
       ▷ Insert A[j] into the sorted sequence A[1...(j − 1)]
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

(ב) דוגמה: $A = [6, 2, 4, 3]$ או משהוא דומה.

(ג) ניתוח סיבוכיות ל- INSERTION-SORT. (מספר השוואות, מספר הזזות, אפשרות לשיפור מספר ההשוואות.)

3. MERGE-SORT.

(א) האלגוריתם: נניח שהסדרה כבר מופיע במערך $\mathrm{A}[1 \ldots n]$.

```
MERGE-SORT(A, p, r)
if p < r
    then q ← ⌊(p + r)/2⌋
         MERGE-SORT(A, p, q)
         MERGE-SORT(A, q + 1, r)
         MERGE(A, p, q, r)
```

(ב) דוגמה: $A = [6, 2, 4, 3]$.

(ג) ניתוח הסיבוכיות ל- MERGE-SORT מבוסס על נוסחת הנסיגה:

$$T(n) = \begin{cases} n + 2T(n/2), & n \ge 2 \\ 1, & n < 2 \end{cases}$$

שעבורו מתקיים $T(n) = \Theta(n \log n)$.

4. QUICK-SORT.

(א) האלגוריתם: נניח שהסדרה כבר מופיע במערך $A[1\dots n]$.

```
QUICK-SORT(A, p, r)
if p < r
    then q ← PARTITION(A, p, r)
         QUICK-SORT(A, p, q)
         QUICK-SORT(A, q + 1, r)
```

```
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
```

(ב) דוגמה: $A = [6, 2, 4, 3]$.

(ג) ניתוח הסיבוכיות ל- QUICK-SORT מבוסס על שיטות הסתברותיות.

(ד) האלגוריתם הוא מהר מאוד בפעל (קבוע נמוך למקרה הממוצע).

5. COUNTING-SORT:

(א) אלגוריתם זה עובד תחת ההנחה כי המספרים הם מספרים טבעיים חסומים על ידי k, כלומר מהקבוצה $\{1, 2, \dots, k\}$.

(ב) האלגוריתם: נניח שהסדרה כבר מופיע במערך $A[1\dots n]$.

```
COUNTING-SORT(A, B, k)
for i ← 1 to k
    do C[i] ← 0
for j ← 1 to length[A]
    do C[A[j]] ← C[A[j]] + 1
▷ C[i] is the number of times i appears as an entry in A.
for i ← 2 to k
    do C[i] ← C[i] + C[i − 1]
▷ C[i] is the now the number of elements ≤ i appearing in A.
for j ← length[A] downto 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]] − 1
```

(ג) האלגוריתם יציב בצורה הנ"ל. תיארנו בתרגיל גישה פשוטה יותר שאינה יציבה מתוך הנחה שאין מידע נלווה.

(ד) דוגמה כאשר $k = 6$: $A = [6, 5, 2, 4, 3, 6, 4, 1, 3, 2, 5, 4]$.

6. BUCKET-SORT:

(א) אלגוריתם זה עובד תחת ההנחה כי ממיינים מספרים המפולגים באופן אחיד בתחום מסוים (אך חזרות מותרות).

(ב) האלגוריתם: נניח שהסדרה כבר מופיע במערך $A[1 \ldots n]$.

```
BUCKET-SORT(A)
  ▷ We divide the allowed range into n equally sized consecutive buckets.
  for i ← 1 to length[A]
       do Place A[i] into the appropriate bucket.
  Sort each one of the n buckets (using heap or merge sort).
  Output the (sorted) contents of each bucket according to the order of the buckets.
```

(ג) דוגמה של מספרים טבעיים מפולגים בקבוצה $\{0, 1, 2, \ldots, 98, 99\}$:
$A = [78, 21, 49, 67, 89, 29, 12, 46, 81, 17]$.

(ד) ניתוח סיבוכיות (בשיטה הסתברותית) כאשר המיון הפנימי נעשה לפי MERGE-SORT או HEAP-SORT.

7. מיון יציב:

הגדרה: אלגוריתם מיון נקרא יתיב אם שני אברים שווי ערך (בהשוואה) רשומים בסדר המקורי בפלט. הגדרה זו משמעותית רק כאשר יש מידע נלווה.

דוגמה: נניח שממיינים אנשים לפי היום הולדת שלהם. (נניח לשם פשטות שרק מטפלים באנשים שנולדו בחודש ינואר, ובכך נכסוך הצורך לטפל בחודש, ונרשום רק את היום בחודש ינואר.)

הקלט: משה נולד ב- 5 (לינואר). רונית נולדה ב- 7 (לינואר). דן נולד ב- 5 (לינואר).

פלט אפשרי א': דן נולד ב- 5 (לינואר). משה נולד ב- 5 (לינואר). רונית נולדה ב- 7 (לינואר).

פלט אפשרי ב': משה נולד ב- 5 (לינואר). דן נולד ב- 5 (לינואר). רונית נולדה ב- 7 (לינואר).

אלגוריתם מיון שמחזיר פלט אפשרי א' אינו יציב, כי גם משה וגם דן נולדו ב 5 לינואר ולכן אמורים להופיע בסדרם בקלט במיון יציב. פלט אפשרי ב' הוא הפלט של אלגוריתם יציב על הקלט הזו.

8. RADIX-SORT:

(א) אלגוריתם זה עובד תחת ההנחה כי ממיינים מספרים (מלים) של d ספרות (אותיות) כאשר כל ספרה לוקח אחד מ- k ערכים. נמספר את הספרות מקטל לגדול.

(ב) האלגוריתם: נניח שהסדרה כבר מופיע במערך $A[1 \ldots n]$.

```
RADIX-SORT(A, d)
  for i ← 1 to d
       do Apply a stable sorting algorithm to sort A according to the digit i.
```

(ג) צריך להפעיל אלגוריתם מיון יציב: שמשמר את סדר האיברים המקורי ששוי ערך במיון הנוכחי.

(ד) דוגמה: $A = [839, 251, 782, 571, 199]$.

(ה) הסבר למה זה עובד: האלגוריתם מתיחס בסוף שלב i רק ל- i הספרות האחרונות - ולפיהם הוא ממיין נכון.

(ו) הסיפור ההיסטורי על למה פתחו את האלגוריתם... מכונות שימשו למיון כרטיסים... ואלגוריתם זו איננה דורשת הרבה ערימות ביניים של כרטיסים.

(ז) ניתוח סיבוכיות כאשר משתמשים ב- COUNTING-SORT למיון לפי כל ספרה.

9. ערימות (heaps) ו- HEAP-SORT:

(א) הגדרות: ערימה הוא מארך A אם שני פרמטרים: $length[A]$ (אורך המארך) ו- $heap\text{-}size[A]$ (מספר האיברים מהמארך שהם בערימה). מתיחסים לערימה כעץ בינארי מלא שהתאים מתואמים לצמתי העץ משמאל לימין שורה לאחר שורה. דורשים שהערך בצומת לא יהיה קטן מהערך באף צומת בתת עץ מתחתיו.

(ב) קוד:

```
PARENT(i)
   return ⌊i/2⌋

LEFT(i)
   return 2i

RIGHT(i)
   return 2i + 1

BUILD-HEAP(A)
heap-size[A] ← length[A]
for i ← ⌊length[A]/2⌋ downto 1
      do HEAPIFY(A, i)

HEAPIFY(A, i)
l ← LEFT(i)
r ← RIGHT(i)
if (l ≤ heap-size[A] and A[l] > A[i])
   then largest ← l
   else largest ← i
if (r ≤ heap-size[A] and A[r] > A[largest])
   then largest ← r
if largest ≠ i
   then exchange A[i] ↔ A[largest]
        HEAPIFY(A, largest)

HEAPSORT(A)
BUILD-HEAP(A)
for i ← length[A] downto 2
   do exchange A[1] ↔ A[i]
      heap-size[A] ← heap-size[A] − 1
      HEAPIFY(A, 1)
```

(ג) תרגיל: תהיה A ערימה עם n איברים. כתוב שיגרה שמוסיפה איבר ל- A (תוך שמירה על היותה של A ערימה) בעלת זמן ריצה $O(\log n)$. ניתן להניח שנשאר מקום בסוף המערך.

(ד) פתרון:

```
HEAP-INSERT(A, x)
heap-size[A] ← heap-size[A] + 1
i ← heap-size[A]
while (i > 1 and A[PARENT(i)] < x)
   do A[i] ← A[PARENT(i)]
      i ← PARENT(i)
A[i] ← x
```

10. תרגיל: הוכח שבמקרה הגרוע דרוש $\lceil 3n/2 \rceil - 2$ השוואות למצוא את המקסימום והמינימום של n מספרים.

11. פתרון: נרשום רשימת מועמדים לכל תפקיד (מקסימום ומינימום) ונמחוק מועמדים שמ-ועמדתם נפסלה: כאשר משווים את a_i ו- a_j התוצאה $a_i < a_j$ מוכיחה לנו ש- a_i אינו המקסימום וש- a_j אינו המינימם.

עכשיו נניח שאנחנו משחקים נגד יריב שנותן לנו את התשובות להשוואות בין זוגות אך מסתיר את הערכים עצמם וגם אינו חייב לקבוע את הערגים עד סוף התהליך. במשך המשחק, היריב רושם לעצמו את התשובות שכבר נתן, כך שלא יסתור את עצמו, וגם אחרי כל תשובה מיד מחשב את היחסים שנקבעים על ידי טרנציטיביות. לפני שמשיב תשובה - הוא יבדוק אם היחס כבר נקבע לפי התשובות הקודמות - ואם כן יענה לפי ההכרח. אם אין תשובה הכרחית - הוא נותן תשובה שמגלה לנו מינימום של מידע חדש. במקרה שטרנציטיביות קובע יחס - כבר ידוע לנו שהקטן בזוג אינו מקסימום והגדול בזוג אענו מינימום - ואין חידוש!

לפני תחילת התהליך כל אחד מהאיברים (a_i) יכול להיות המקסימום או המינימום, כלומר שיש n מועמדים לכל תפקיד. התהליך נגמר כאשר נשאר רק איבר יחיד כמועמד למקסימום ואיבר יחיד כמועמד למינימום. לכן ההשוואות חייבות להוכיח לנו ש- $n-1$ איברים אינם יכולים להיות המקסימום וש- $n-1$ איברים אינם יכולים להיות המינימום; סה"כ עלינו לשלל $2n-2$ אופציות.

ברור שאם נשווה את a_i ו- a_j כאשר כבר ידוע לנו ש- a_i אינו המקסימום (המינימום) ולא ידוע מידע על a_j היריב יענה ש- $a_i < a_j$ ($a_i > a_j$) כך שלגבי a_i לא קבלנו מידע חדש! (טענה דומה נכונה אם ידוע לנו מידע על a_j ולא על a_i.) כאשר ידוע מידע גם על a_i וגם על a_j היריב יענה בצורה שלא תסתר את הטרנציטיביות של התשובות הקודמות אבל שלכל היותר יפסול מועמדות אחת. לכן השוואת זוג שידוע עליהם מידע יכול לשלול לכל היותר אופציה אחת, במקרה הגרוע.

אם כבר ידוע ש- a_i אינו המקסימום וש- a_j אינו המינימום ואין יחס ביניהם לפי טרנציטיביות, אז היריב יענה $a_i < a_j$ ולא יגלה שום מידע שימושי חדש!!! המסקנה הוא שלא כדאי לעשות השוואה כזו!!!

ניתן לעשות (לכל היותר) $\lfloor n/2 \rfloor$ השוואות בין זוגות שלא ידוע עליהם מאומה - ומכל השוואה כזאת להוריד גם את מספר המועמדים למקסימום וגם את מספר המועמדים למינימום באחד. נניח שעשינו k השוואות כאלו - סה"כ שללנו $2k$ אופציות. כל השוואה בין זוג שיש מידע חלקי עליהם שולל לכל היותר אופתיה אחד (במקרה הגרוע), ולכן צריכים לפחות $(2n-2)-2k$ השוואות כאלו. סה"כ נעשה $k+(2n-2-2k) = 2n-k-2$ השוואת. נקבל מינימום עבור $k = \lfloor n/2 \rfloor$ עם

$$2n - \lfloor n/2 \rfloor - 2 = \lceil 3n/2 \rceil - 2$$

השוואות סה"כ.

חזרנו על ההוכחת החסם התחתון של $\Omega(n \log n)$ השוואות באלגוריתם מבוסס השוואות למיון n מספרים. (כעזרה לפתור את תרגיל בית 3 שאלה 1.) העיקר הוא שעץ בינארי (עם שורש) עם $n!$ עלים העומק שלו חייבת להיות לפחות $\log(n!) = \Omega(n \log n)$.

12. בעית הבחירה (selection problem): דיון על אלגוריתם החמישיות (ראה בספר Introduction to Algorithms של Cormen, Leiserson, and Rivest) בפרק 10.3: Selection in worst-case linear time.

מומלץ מאוד להשקיע קצת זמן בהבנת האלגוריתם והניתוח שלו - זאת אחת האלגוריתמים היותר מסובכים בקורס ואין זו המקום לדיון רציני בו.

13. תרגיל: נתונה פרמוטציה τ של המספרים $\{1, 2, \ldots, n\}$. נאמר של- τ יש חילוף (i, j)- אם $i < j$ ו- $\tau(i) > \tau(j)$. תן אלגוריתם בעל סיבוכיות $O(n \log n)$ המוצא את מספר החילופים של τ.
(רמז: וריאציה של אלגוריתם MERGE-SORT.)

14. פתרון:

ALT-MERGE-SORT(A, p, r)
$a \leftarrow 0; b \leftarrow 0; c \leftarrow 0;$
if $p < r$
  then $q \leftarrow \lfloor (p + r)/2 \rfloor$
    $a \leftarrow$ ALT-MERGE-SORT(A, p, q)
    $b \leftarrow$ ALT-MERGE-SORT$(A, q + 1, r)$
    $c \leftarrow$ ALT-MERGE(A, p, q, r)
return $(a + b + c)$

ALT-MERGE(A, p, q, r)
$i \leftarrow p$ $\triangleright$ Pointer into left part
$j \leftarrow q + 1$ $\triangleright$ Pointer into right part
$k \leftarrow p$ $\triangleright$ Pointer into temporary array
$s \leftarrow 0$ $\triangleright$ Number of swaps seen so far
$l \leftarrow q - (p - 1)$ $\triangleright$ Number of elements remaining in left part
while $(i \leq q$ **and** $j \leq r)$ **do**
  if $(A(i) \leq A(j))$
    then $\triangleright$ No swap here.
      $B(k) \leftarrow A(i)$
      $i \leftarrow i + 1$
      $k \leftarrow k + 1$
      $l \leftarrow l - 1$ $\triangleright$ Left side got smaller.
    else $\triangleright$ Here there are swaps as $A(i) > A(j)$.
      $B(k) \leftarrow A(j)$
      $j \leftarrow j + 1$
      $k \leftarrow k + 1$
      $s \leftarrow s + l$ $\triangleright$ How many new swaps?
while $(i \leq q)$ **do** $\triangleright$ Copy rest of left side.
  $B(k) \leftarrow A(i)$
  $i \leftarrow i + 1$
  $k \leftarrow k + 1$
while $(j \leq r)$ **do** $\triangleright$ Copy rest of right side.
  $B(k) \leftarrow A(j)$
  $j \leftarrow j + 1$
  $k \leftarrow k + 1$
for $k = p$ **to** r **do** $\triangleright$ Copy result back into A.
  $A(k) \leftarrow B(k)$
  $k \leftarrow k + 1$
return (s)