מבני נתונים ומבוא לאלגוריתמים – ממ"ן 13

שאלה 1

א. מהנתון שטווח המספרים הוא בין $0 \to n\log n$, נוכל להשתמש במיון בסיס בזמן ליניארי.
לפי עמוד 118 במדריך הלמידה, נקבל שמספר הספרות הנחוץ לייצוג כל ספרה בטווח זה בבסיס n הוא:

$$\log_n n\log n \le \log_n n^2 = 2$$

כלומר לכל היותר 2 ספרות. לכן סיבוכיות זמן הריצה של מיון בסיס במקרה זה הוא במקרה הגרוע: $\Theta(d+k)=\Theta(2+n)=\Theta(n)$.

כשיש לנו מערך ממוין, נוכל באמצעות לולאה פשוטה שעוברת במקרה הגרוע פעם אחת על כל איבר, לקבל האם יש שני איברים שסכומם z.
נפעל ע"י שיטת המאזניים, כלומר נתחיל מהאיבר הראשון והאחרון במערך, קרי מינימום ומקסימום. נבדוק את סכומם. אם הסכום קטן מ־z, נקדם את האינדקס של האיבר השמאלי (מינימלי), אם גדול יותר, נוריד את האינדקס של האיבר הימני (מקסימלי).
נמשיך כך עד שנגיע לזוג מספרים שסכומם z, או שהאינדקסים יצטלבו, ואז נסיק שלא קיים זוג מספרים כזה.

פסאודוקוד:

```
FIND-PAIR(S,z):
RADIX-SORT(S, 2)
left <- 1
right <- length[S]

while left < right:
  sum <- S[left]+S[right]
  if sum < z:
    left = left+1
  else if sum > z:
    right = right-1
  else:  // sum = z
    return TRUE

return FALSE
```

בכל איטרציה אנו מקדמים את אחד האינדקסים לכיוון האחר, ולכן לאחר מספר סופי של איטרציות הם יפגשו והלולאה תעצור. לכן סיבוכיות זמן הריצה של הלולאה היא $\Theta(n)$ במקרה הגרוע. כאמור, גם הסיבוכיות של מיון בסיס במקרה שלנו היא $\Theta(n)$, ולכן סה"כ הסיבוכיות של האלגוריתם היא $\Theta(n)$ במקרה הגרוע, כנדרש.

האלגוריתם עובד נכון מפני שכשהמערך ממוין, אנו יודעים שע"י קידום left בהכרח נקבל סכום גבוה מהאיטרציה הקודמת, וכאשר נחסר 1 מ־right נקבל סכום קטן מהאיטרציה

הקודמת. לכן אנו שמים את z כציר, ומאזנים את האינדקסים עד שנגיע למספר הקרוב ביותר אליו (במטרה כמובן להגיע ל־z עצמו).

ב. על מנת למצוא שלישיית מספרים שסכומם z, נפעל באופן דומה לשגרה בסעיף א'. נמיין באותו אופן את המערך, אך הפעם נעבור על כל איברי המערך משמאל לימין (מלבד השניים האחרונים), ובכל איטרציה כזו נעשה לולאה דומה ללולאה שבסעיף א', כלומר לפי שיטת המאזניים.

פסאודוקוד:

```
FIND-TRIPLETS(S,z):
RADIX-SORT(S, 2)

for i = 1 to length[S]-2:
    left <- i+1
    right <- length[S]

  while left < right:
      sum <- S[i]+S[left]+S[right]
      if sum < z:
           left = left+1
      else if sum > z:
          right = right-1
      else:  // sum = z
          return TRUE

return FALSE
```

נשים לב כי אין צורך בלולאה הראשונה לעבור על שני האיברים האחרונים, שכן אז לא ישארו לנו מספיק איברים לבניית שלישייה.
הנכונות של אלגוריתם זה דומה לנכונות לאלגוריתם בסעיף א', שכן אנו בכל פעם בוחרים באיבר גדול יותר מהקודם, ואז מבצעים את אותה לולאה על שאר האיברים הגדולים ממנו.
לכן אם קיים z כזה, בוודאי שנפגוש בו.

המיון מתבצע ב־$\Theta(n)$, כל איטרציה של הלולאה החיצונית מתבצעת ב־$\Theta(n)$ (לפי סעיף א'), ואנו מבצעים $n-2=\Theta(n)$ איטרציות. לכן סה"כ הסיבוכיות: $\Theta(n)+n\Theta(n)=\Theta(n^2)$, כנדרש.

ג. על מנת למצוא רביעיית מספרים שסכומם z, נצטרך להשתמש במקום נוסף.

נגדיר מערך pairs, שיכיל את בכל תא סכום של שני איברים שונים ב־S. סיבוכיות הריצה של מערך כזה היא $\Theta(n^2)$.

לכל תא במערך pairs, נשמור במערך נוסף key1 באותו אינדקס את האינדקס של האיבר הראשון ב־A שהשתתף בסכום, ובמערך key2 באותו אינדקס את האינדקס של האיבר השני ב־A.

להמחשת הרעיון: נניח שבתא $pairs[i]$ יש 10 שנוצר ע"י האיברים 3 ו־7 ב־A שנמצאים באינדקסים 1 ו־2 בהתאמה (מפני שכל האיברים שונים, אין כפל משמעות), אז: $key1(i) = 1, key2(i) = 2$.

נשים לב שטווח הערכים במערך pairs תלוי ב־A, ומפני שהטווח ש־A הוא בין 0 ל־nlgn, אז הטווח הכי רחב שיכול להיות ב־pairs הוא בין 0 ל־$nlgn + nlgn - 1$.

לכן נוכל לבצע מיון בדומה לסעיף א' בזמן לינארי, ומפני שאורך המערך pairs הוא $\Theta(n^2)$, אז המיון יתבצע בזמן $\Theta(n^2)$.

בזמן המיון של pairs עלינו לדאוג לעדכן את האינדקסים במערכים key1, key2, וזה יתבצע בזמן קבוע לכל החלפה בתהליך המיון ב־pairs, ולכן זה לא יפגע אסימפטוטית במיון.

כעת נוכל לפעול בדומה לסעיף א', רק על המערך pairs, רק שהפעם אם נמצא זוג (רביעיית איברים שסכומם z, נצטרך לבדוק ע"י מערכי key שאין איבר שמשתתף פעמיים בסכום.

```
FIND-FOURPLETS(S, pairs, key1, key2, z):
left <- 1
right <- length[pairs]

while left < right:
  sum <- pairs [left]+ pairs [right]
  if sum < z:
    left = left+1
  else if sum > z:
    right = right-1
  else:  // sum = z
    if keys1(left) != keys2(left) and keys1(right) != keys2(right)
        return TRUE

return FALSE
```

שאלה 2

נתון כי התפלגות הנקודות אחידה, לכן ננסה להשתמש במיון דלי.

נשים לב כי הזווית בחצי הימני של העיגול היא π. נחלק את הזווית ל־n חלקים שווים ע"י קרניים מראשית הצירים כך שזווית כל חלק תהיה $\frac{\pi}{n}$. הזווית של כל חלק i בין 0 ל־n היא: $\alpha_i = i * \frac{\pi}{n} - \frac{\pi}{2}$ (שכן הקרן הראשונה היא $-\frac{\pi}{2}$).
אז מתקיים $-\frac{\pi}{2} \le \alpha_i \le \frac{\pi}{2}$ ולכן: $0 \le \frac{\mathrm{\alpha_j}}{pi} + \frac{1}{2} \le 1$, ולפי חישוב זה נקבל את המספר בקטע $[0,1]$ שמתאים לכל נקודה, ולכן נוכל להשתמש במיון דלי.
נבחר n דליים, כך שכל דלי יכיל את הנקודות שה־θ שלהן בין α_{i-1} ל־α_i.
מפני שמתקיים: $\tan(\theta_i) = x_i/y_i$, אז:

$$\theta_i = \arctan(\tan(\mathrm{\theta_i}\ )) = \arctan(\frac{x_i}{y_i})$$

נשתמש במיון דלי דומה למה שיש בספר הלימוד, כך שנכניס את הנקודה לאינדקס:

$$n * \left(\frac{\arctan\left(\frac{x_1}{y_1}\right)}{n} + \frac{1}{2}\right)$$

החישוב מתבצע בזמן קבוע, ולכן לא שינינו את זמן הריצה של אלגוריתם המיון, ולכן כפי שמוכח בספר סיבוכיות זמן הריצה היא $\Theta(n)$ במקרה הגרוע.

שאלה 3

א. לכל איבר באינדקס i במערך P, נעבור על כל האיברים לפניו במערך, ונספור כמה מהם קטנים או שווים לו. כשניתקל באיבר גדול ממנו, נצא מהלולאה ונשים באינדקס i של S את מספר האיברים שספרנו.

```
SEQUENCE(P, S):

for i=length[P] to 1:
   count <- 1
   for j=i to 1:
     if P[i]>=P[j]
       count = count + 1
     else:  // sum = z
        break

   S[i]<- count
```

במקרה הגרוע (מערך ממוין), בכל פעם נרוץ מאינדקס i עד תחילת המערך, ולכן הסיבוכיות היא ריבועית, כלומר $\Theta(n^2)$ כנדרש.

ב. נשפר את האלגוריתם בסעיף א' ע"י שימוש במחסנית.
נשים לב כי אם $P[i] \leq P[i+1]$ אז $P[i+1]$ גדול או שווה גם מכל האיברים ש־$P[i]$ גדול או שווה מהם. נכניס למחסנית את האינדקס שערכו עודכן במערך S, וכשנעדכן את המחסנית אז נבדוק האם אינדקס i גדול מ־top(ST), אם כן אז הוא גדול מכל האינדקסים הגדולים מ־top(ST) ואם הוא קטן מ־top(ST) אז נוכל למצוא את הרצף ע"י חישוב: $i - \mathrm{top}(\mathrm{ST})$.

```
SEQUENCE(P, S):

init stack ST
push(ST, 1)
S[1] <- 1

for i=2 to length[P]:
   while not empty(ST)  P[i]>=P[top(ST)]:
     pop(ST)
   S[i] <-max(1, i-top(B))

   push(ST, i)
```

נתחת את זמן הריצה: הלולאה החיצונית מבצעת $n-1$ איטרציות, והלולאה הפנימית מבצעת לכל היותר n איטרציות. נשים לב כי הלולאה הפנימית יכולה להתבצע כל עוד יש איברים במחסנית, ומפני שאנו מבצעים פעולת push לכל היותר n פעמים, הרי בכל ריצת התוכנית (ולא רק עבור כל איטרציה של הלולאה החיצונית), לא יכולות להתבצע יותר מ־n איטרציות של לולאת ה־while. לכן סה"כ הסיבוכיות היא: $n-1+n=\Theta(n)$, כנדרש.

שאלה 4

א.

```
PUSH-LEFT(D, x):
if head(D) = tail(D)+1 or head(D)=1 and tail(D)=length(D):
    error "overflow"

If head(D)=1
    head(D) <- length(D)
else:
    head(D) <- head(D) -  1

D[head(D)] <- x
```

```
PUSH-RIGHT(D, x):
if head(D) = tail(D)+1 or head(D)=1 and tail(D)=length(D):
    error "overflow"

If tail(D)=length(D)
    tail(D) <- 1
else:
    tail(D) <- tail(D) +  1

D[tail(D)] <- x
```

```
POP-LEFT(D, x):
if head(D) = tail(D):
    error "underflow"

x <- D(head(D)]

If head(D)=length(D):
    head(D) <- 1
else:
    head(D) <- head(D) +  1

return x
```

```
POP-RIGHT(D, x):
if head(D) = tail(D):
    error "underflow"

x <- D(tail(D)]

If tail(D)=1:
    tail(D) <- length(D)
else:
    tail(D) <- tail(D) -  1

return x
```

ב. ננצל את ערכי הגיבוב כך: כאשר נסרוק את הרשימה ונחפש מפתח מסוים x, נחשב קודם את $h(x)$, ואז עבור כל איבר k ברשימה נבדוק קודם האם $h(x) = h(k)$. אם לא, אז אנו יודעים בוודאות ש: $x \neq k$ ונוכל להמשיך הלאה, אם כן, רק אז נבצע את הבדיקה $x = k$. כאמור, המפתחות עצמם הם מחרוזת תווים ארוכה, ולכן בדיקה האם שתי מחרוזות שוות אורכת $\min(length(x), length(k))$. בדיקה האם שני ערכי גיבוב זהים תלויה בטווח של הפונקציה, ומפני שההנחה בטבלת גיבוב שמספר האיברים בטבלה קטן מאוד (<<) ממספר האיברים בתחום, נקבל כי ההשוואה של ערכי הגיבוב תהיה מהירה יותר משמעותית.

שאלה 5

א. נתחיל מהסבר מדוע האלגוריתם לא בהכרח ממיין.
אם פונקציית הגיבוב לא מחלקת את האיברים לפי גודלם, כלומר לא מתקיים בהכרח שהאיברים שנכנסו לאינדקס i בטבלה יהיו גדולים מהאיברים באינדקסים $j < i$.
משהבנו זאת, קל לבנות דוגמה נגדית – נשתמש בפונקציית גיבוב שלא שומרת על הסדר הנ"ל.

נגדיר: $h(x) = k\ mod\ 2$, ונבחר לדוגמה $U = [1,2,3,4]$. אז $n = 4, m = 2$.
נחשב את טבלת הגיבוב (שלב ראשון באלגוריתם):

$$h(1) = 1, \qquad h(2) = 0, \qquad h(3) = 1, \qquad h(4) = 0$$

כלומר בתא 0 נמצאים האיברים 2,4, ובתא 1 נמצאים 1,3.
בשלב השני נמיין אותם, ובשלב השלישי נצרף אותם לפי הסדר, אז נקבל את הרצף: 2,4,1,3.
מובן כי הוא לא ממוין, ולכן מצאנו דוגמה שבה האלגוריתם לא ממיין.

ב. נמצא את זמן הריצה הממוצע של האלגוריתם. לפי הנתון פונקציית הגיבוב נותנת גיבוב אחיד ופשוט, ולכן היא מתבצעת בזמן קבוע. אז להכניס n מפתחות לטבלה זה $\Theta(n)$.
בשלב השני, נבצע m מיונים מהירים על n מפתחות. כידוע, זמן הריצה של מיון מהיר במקרה הממוצע הוא $\Theta(n\lg n)$, מכיוון שפונקציית הגיבוב מפזרת באופן אחיד, אז יש בממוצע $\frac{n}{m}$ איברים בכל תא, לכן מיון כל תא מתבצע בזמן $\Theta(\frac{n}{m}\lg\frac{n}{m})$.
יש m תאים, לכן סה"כ זמן המיונים: $\Theta\left(m * \frac{n}{m}\lg\frac{n}{m}\right) = \Theta(\mathrm{n}\lg\frac{n}{m})$.
את השלב השלישי באלגוריתם (צירוף) מבצעים ב־$\Theta(n)$, ולכן סך הסיבוכיות של האלגוריתם:

$$\Theta\left(n + n + \mathrm{n}\lg\frac{n}{m}\right) = \Theta(\mathrm{n}\lg\frac{n}{m})$$

בפרט, אם $n = m$, אז נקבל: $\Theta(n + n + n\lg 1) = \Theta(\mathrm{n})$.

ג. נמצא את זמן הריצה של האלגוריתם במקרה הגרוע.
בדומה לסעיף ב', הכנסת האיברים וצירופם בסוף לוקחת $\Theta(n)$ לכל אחד.
ההבדל הוא במיון המהיר, שכידוע זמן הריצה הגרוע שלו הוא $\Theta(n^2)$.
כמו כן, במקרה הגרוע פונקציית הגיבוב תפזר את כל האיברים לתא אחד, כלומר יהיה תא i שבו נמצאים n איברים, ולכן סך עלות המיון תהיה $\Theta(n^2)$.
לכן נחשב מחדש את הסיבוכיות:

$$\Theta(n + n^2 + n) = \Theta(n^2)$$

ד. ננסה לבנות מצב בו התנאים בשאלה מתקיימים.
מסעיף ב' נובע שבכדי להשיג יעילות לינארית, עלינו לבחור $n = m$.
נתון שהאיברים מתפלגים באופן אחיד, לכן לכל איבר x מקיים: $0 \le x \le n^3 - 1$, כלומר:

$$0 \le \frac{x}{n^3} < 1$$

לכן לפי עמוד 193 בספר הלימוד, נוכל להשתמש בפונקציית הגיבוב: $h(x) = \left\lfloor \frac{x}{n^3} * m \right\rfloor$
מפני שבחרנו $m = n$, נקבל: $h(x) = \left\lfloor \frac{x}{n^3} * n \right\rfloor = \left\lfloor \frac{x}{n^2} \right\rfloor$.
נשים לב כי פונקציית הגיבוב מקיימת את התנאי ההכרחי למיון שמצאנו בסעיף א', שהוא שפונקציית הגיבוב מפזרת את האיברים לפי גודלם. כלומר, איבר $x > y$ בהכרח יהיה בטבלת הגיבוב באינדקס גדול או שווה לאינדקס ש־y נמצא בו.
לכן לאחר שנמיין את כל האיברים בכל תא, כאשר נצרף את האיברים בשלב השלישי

של האלגוריתם, נקבל רשימה ממוינת.
כפי שהראנו בסעיף ב', במקרה הממוצע האלגוריתם יתבצע ב: $\Theta(n)$, בהתאם לדרישה.

אין סתירה בין מקרה זה לסעיף א', שכן כאמור בסעיף א' הראינו שהאלגוריתם לא ממיין כל עוד פונקציית הגיבוב לא מקיימת את התנאי שהוזכר לעיל. משבחרנו פונקציה שכן מקיימת תנאי זה, הרי שאין סתירה.