פתרון טוב לממן 12 בקורס 20407 2019א

שאלה 1

סעיף ב פשוט ולעניין

לכל צומת שאינו עלה השגרה תבדוק האם הוא ברמה זוגית או אי-זוגית. אם הוא ברמה זוגית, היא תבדוק האם ערך הצומת גדול או שווה משל שני בניו וארבעת נכדיו (אם קיימים), ואם הוא ברמה אי-זוגית, היא תבדוק האם ערך הצומת קטן או שווה משל שני בניו וארבעת נכדיו (אם קיימים). אם הבדיקה חיובית, היא תמשיך לצומת הבא בעץ (שאינו עלה), ואם היא שלילית, היא תסתיים מיד ותחזיר FALSE. אם השגרה הגיעה עד הצומת האחרון שאינו עלה וכל הבדיקות שהיא ביצעה חיוביות, היא תחזיר TRUE.

<u>IsAlternatingHeap(A)</u>

```
 1. n ← length[A]
 2. for i ← 1 to floor(n/2)
 3.     if floor(lg(i)) is even
 4.         if A[i] < A[2i] or A[i] < A[2i+1] or
 5.            A[i] < A[4i] or A[i] < A[4i+1] or
 6.            A[i] < A[4i+2] or A[i] < A[4i+3]
 7.               then return FALSE
 8.     else     // odd level
 9.         if A[i] > A[2i] or A[i] > A[2i+1] or
10.            A[i] > A[4i] or A[i] > A[4i+1] or
11.            A[i] > A[4i+2] or A[i] > A[4i+3]
12.               then return FALSE
13. return TRUE
```

נכונות האלגוריתם: שמורת הלולאה: לפני האיטרציה ה-i המערך A[1 .. i-1] מהווה ערימה לסירוגין חוקית.

<u>אתחול:</u> לפני האיטרציה הראשונה i=1 והמערך הוא A[1 .. 0]. זהו מערך ריק, ולכן תכונת הערימה לסירוגין מתקיימת באופן ריק.

<u>תחזוקה:</u> באיטרציה ה-i-ית, אם A[i] נמצא ברמה הזוגית והוא מפר את תכונת הערימה לסירוגין עבור צומת ברמה זוגית, הרי שכל העץ אינו ערימה לסירוגין חוקית ומוחזר False (שורות 3-7). אם A[i] נמצא ברמה האי-זוגית והוא מפר את תכונת הערימה לסירוגין עבור צומת ברמה אי-זוגית, הרי שכל העץ אינו ערימה לסירוגין חוקית ומוחזר False (שורות 8-12). אם A[i] אינו מפר את תכונת הערימה לסירוגין, אזי לפי שמורת הלולאה A[1 .. i-1] הוא ערימה לסירוגין חוקית ולכן גם A[1 .. i] הוא ערימה לסירוגין חוקית.

<u>סיום:</u> ביציאה מהלולאה, אם הוחזר False הרי שנמצאה הפרה בעץ ולכן העץ אינו ערימה לסירוגין חוקית, ואם הוחזר True הרי שהמערך הוא כעת A[1 .. floor(n/2)] וזו ערימה לסירוגין חוקית לפי שמורת הלולאה. נשארו רק העלים שנמצאים החל מהמקום ה-floor(n/2)+1, אבל הם כולם בנים של צמתים ששייכים ל-A[1 .. floor(n/2)] ולכן כבר נבדקו באיטרציות קודמות. אף אחד מהם לא מפר את תכונת הערימה לסירוגין (כי אחרת הלולאה הייתה מחזירה False), ולכן כל העץ מהווה ערימה לסירוגין חוקית.

סיבוכיות: בשגרה זו יש לולאה אחת (שורות 2-13) שמבוצעת לכל היותר $\lfloor n/2 \rfloor$ פעמים, ובגוף הלולאה יש מספר קבוע של פעולות. לכן סיבוכיות השגרה היא $O(n)$.

גישה אחרת

סעיף ב':

רעיון האלגוריתם: כל "מסלול" מהשורש ועד לאחד העלים, בגלל תנאי הערמה, יהיה "כלוא" בין ערכי המינימום והמקסימום ההולכים ומתקרבים. ניתן להמחיש ע"י הגרף הבא:

ז"א בשורש הערמה יהיה הערך המקסימלי, לאחר מכן הערך המינימלי, לאחר מכן ערך שחייב להיות ביניהם, לאחר מכן ערך שחייב להיות בין הערך המינימלי הראשון והערך המקסימלי השני וכן הלאה....

האלגוריתם שומר כל הזמן את הערך המינימלאי והמקסימלי שהצומת הנוכחי בעץ יכול לקבל, ובודק אם הצומת הנוכחי אכן נמצא בטווח האפשרי. לאחר מכן, לפי רמת הצומת (זוגי או אי זוגי), האלגוריתם "מעדכן", את ערכי המינימום או המקסימום בהתאמה, ובודק בצורה רקורסיבית את 2 הבנים שלו – אם הם גם בטוחח הנכון, ושומרים על מבנה הערמה, זוהי ערמה חוקית.

ניקח למשל מסלול כלשהו מהשורש ועד לאחד העלים (האיבר הראשון (10) הוא ברמה ה0 של העץ, רמה זוגית):

4 ← 5 ← 2 ← 8 ← 2 ← 10

בקריאה לשגרה נציב "זקיפים" שיהוו מינימום ומקסימום- אינסוף ומינוס אינסוף. נבדוק – האם 10 נמצא בין מינוס אינסוף לאינסוף? כן, ולכן האלגוריתם ימשיך לבדוק בצורה רקורסיבית את שאר העץ. מכיוון ש10 הוא ברמה הזוגית, וכל הצאצאים שלו צריכים להיות קטנים ממנו – תתבצע הקריאה הרקורסיבית כאשר המינימום הוא מינוס אינסוף, והמקסימום הוא 10. במידה ויהיה מספר גבוהה מ10 בהמשך העץ – זוהי לא ערמה חוקית.

נדגים את השלב הבא, להמחשת המינימום: האם המספר 2 הוא בין מינוס אינסוף לבין 10? כן, ולכן תתבצע קריאה רקורסיבית לבדוק את המשך העץ, כשערכי הצמתים חייבים להיות בין 10 לבין 2. וכן הלאה.

השגרה תעצר כאשר יש קריאה ל Null - ואז היא תחזיר שהעץ תקין. (אם בדרך אחד הצמתים לא היה תקין, FALSE היה חוזר מקודם).

תיאור האלגוריתם:

1. בדיקה – האם הערך של הצומת הנוכחי הוא בין המינימום והמקסימום? אם לא, החזר FALSE
2. עדכן את המקסימום או המינימום, לפי רמת העץ (זוגי או אי זוגי, בהתאמה)
3. בצע קריאה רקורסיבית לבדוק את הבן השמאלי ואת הבן הימני בעץ

האלגוריתם:

```
is_valid_heap(arr, i, max, min, level_type):
    if i >= length(arr):  // not a node on the heap, a Null
        return True
    if arr[i] < min or arr[i] > max: // illegal value
        return False
    else:
        // update legal ranges
        if level = EVEN:
            max<- arr[i]
        else: // level =ODD
            min <- arr[i]
        level_type <- NOT level_type // change level type. if odd change to
                                     // even, if even change to odd
        i <- i+1
        return is_valid_heap(arr, get_left_son(i), max, min, level_type)
                and is_valid_heap(arr, get_right_son(i), max, min, level_type)
```

הקריאה לשגרה תתבצע באמצעות הפקודה:

`is_valid_heap(array, 0, ∞, -∞, True)`

כאשר אינסוף ומינוס אינסוף משמשים כזקיפים.

ניתוח נכונות:

טענה: השגרה `is_valid_heap` בודקת האם ערמה בגודל n היא ערמה תקינה. ערמה תקינה היא ערמה שבה צומת ברמה זוגית מכיל ערך גדול או שווה משל כל צאצאיו, וכל צומת ברמה אי זוגית מכיל ערך קטן או שווה משל כל צאצאיו. (יש לשים לב שהקריאות הרקורסיביות לא בודקות את רמת העץ, אלא מקבלות אותו כפרמטת שמועבר בקריאה לשגרה).

בדיקה עבור n=0:

בקריאה הראשונה לשגרה i=0, ולכן בשורה 3 השגרה תחזיר TRUE. עץ ריק הוא ערימה חוקית, ולכן עבור n=0 השגרה מבצעת את המוטל עליה.

בדיקה עבור n=1:

הבדיקה בשורה 4 תבדוק האם הערך שנמצא בערמה נמצא בגבולות המותר לו. (בין MIN לMAX). זה נכון עבור כל עץ בעל שורש בלבד (הוא ערמה תקינה). מכיוון שהשורש הוא ברמה זוגית (0), MAX יעודכן לערך שבA[0]. (אם level_type היה אי זוגי, min היה מעודכן, וכך היה נוצר טווח שרק בו הערכים בעלים היו חוקיים). יתבצעו 2 קריאות רקורסיביות לשני הבנים של השורש. מכיוון שאין לו בנים, בשורה 3 יוחזר TRUE עבור שניהם. מכאן שהשגרה תחזיר TRUE.

הנחת האינדוקציה: נניח שהשגרה נכונה עבור ערמות קטנות ממש מn, ונוכיח עבור ערמה בגודל n.

הוכחה: השגרה תבדוק האם הערך שבשורש הוא בין אינסוף לבין מינוס אינסוף. זה כמובן נכון, ולכן תעדכן את max ובכך "תנמיך" את הגבול העלין של הערכים המותרים בעץ.

תתבצע קריאה רקורסיבית לבנים, עם המינימום והמקסימום החדשים. הפעם, level_type יהיה אי זוגי. ע"פ הנחת האינדוקציה, הקריאות הרקורסיביות יחזירו אם הערמה תקינה – וכך גם יוחזר מהשגרה.

ניתוח סיבוכיות (ע"פ השלבים בתיאור האלגוריתם):

1. Θ(1)
2. Θ(1)
3. 2T(n/2)

סה"כ:

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(1)$$

$$a = 2\,; b = 2\,; n^{(\log_2 2)} = n\,; f(n) = \Theta(1)$$

קיים קבוע ε (למשל ε=0.5) כך ש:

$$f(n) = \Theta(1) = O(n^{1-0.5}) = O(n^{0.5})$$

ולכן, לפי משפט 1 של שיטת האב:

$$\mathrm{T}(n) = \Theta(n)$$

סעיף ג1 השורש

סעיף ג 2 אחד הבנים של השורש, הקטן שבהם

סעיףג3

הוספת ערך חדש: מתכונות הערימה לסירוגין נובע כי ערכו של כל צומת ברמה זוגית גדול מערך אביו וקטן מערך סבו, וברמה אי-זוגית – קטן מערך אביו וגדול מערך סבו. לפיכך, כשנוסיף ערך חדש לסוף הערימה נבדוק האם ערכו נמצא בין ערך אביו לערך סבו. אם כן – קיבלנו ערימה לסירוגין חוקית וסיימנו. אחרת – אם הערך החדש גדול גם מערך אביו וגם מערך סבו, הוא יחליף מקומות עם הגדול מביניהם, ואם הערך החדש קטן גם מערך אביו וגם מערך סבו, הוא יחליף מקומות עם הקטן מביניהם. לאחר מכן נחזור על התהליך על הערך החדש במקומו החדש ונמשיך עד שנקבל ערימה לסירוגין חוקית.

סעיף ג4

החלפת ערך השורש: תחילה נבדוק מי הגדול ביותר מבין שני בניו וארבעת נכדיו של השורש (אם קיימים). אם מבין שישה צמתים אלה הגדול ביותר הוא נכד של השורש, נבדוק האם הנכד גדול מהשורש ואם כן – נחליף מקומות ביניהם, ואם הנכד במקומו החדש קטן מאביו – נחליף מקומות גם ביניהם. נפעיל באופן רקורסיבי תהליך זה על השורש במקומו החדש עד שנקבל ערימה לסירוגין חוקית. במקרה שהגדול מבין ששת הצמתים שהוזכרו קודם הוא בן של השורש, נחליף מקומות ביניהם במקרה שהבן גדול מהשורש.

שאלה 2

סעיף א

השגרה *HEAP-EXTRACT-MAX* עצמה לא מבצעת השוואות, אלא מחליפה את ערך השורש בערך העלה האחרון, מוחקת את אותו עלה וקוראת ל-*HEAPIFY*.
השגרה *HEAPIFY* רצה על מסלול מהשורש החדש לעלה כלשהו, ובכל צומת מבצעת שתי השוואות. אורך מסלול כזה הוא $\lfloor \lg(n-1) \rfloor$ או $\lfloor \lg(n-1) \rfloor - 1$ (כגובה העץ).
לכן, במקרה הגרוע מתבצעות $2\lfloor \lg(n-1) \rfloor$ פעולות השוואה בין איברי הערימה, שהן בקירוב $2\lg n$ השוואות.

בסדר

הסיבה ל"בערך" המופיע בשאלה היא כי ייתכן ולצומת בסוף יש רק עלה אחד.

ראשית נראה כי בכל דרך מן שורש לתחתית הערימה קיימים ceiling(log(n)) כאשר n הוא מספר איברי הערימה.
נראה כי בכל קריאה לפונקציה MAX-HEAPIFY(ללא התחשבות ברקורסיה) מתבצעות 2 השוואות של ערכים מהערימה, השוואה לבן השמאלי והשוואה של הגדול מהאינדקס הנוכחי והבן השמאלי לבן הימני.
כעת עלינו להוכיח כי בשימוש בפונקציה HEAP-EXTRACT-MAX הפונקציה MAX-HEAPIFY נקראת בערך
log(n) פעמים(כלומר עבור כל איבר בדרך מסוימת מהשורש לתחתית הערימה).

הפונקציה HEAP-EXTRACT-MAX מקטינה את גודל הערימה באחד ומעבירה את האיבר בסוף הערימה להיות הראשון וקוראת ל-MAX-HEAPIFY עבור אינדקס 1. נשים לב כי עבור האיבר האחרון בערימה קיימת דרך אחת לפחות עבורה הפונקציה תיקרא log(n) פעמים.
ידוע כי האיבר האחרון קטן או שווה לכל מי שמעליו בערימה, לכן במקרה הגרוע ביותר הפונקציה תיקרא ceiling(log(n))-1 פעמים ותתקבל ערימה לאיבר האחרון יהיה את אותם אבות קדמונים (כלומר לא רק ישירים) חוץ מהשורש שנמחק מהערימה. כעת אם היה לאיבר האחרון אח שמאלי לפני המחיקה תתבצע השוואה תיקראה הפונקציה שוב, כלומר ceiling(log(n)) ותתבצע השוואה איתו. אחרת תסתיים הפונקציה.
בנוסף, קיימים קלטים עבורם האיבר האחרון יגיע לתחתית הערימה אך עם אבות קדמונים שונים משהיו לו לפני המחיקה והפונקציה MAX-HEAPIFY תקרא ceiling(log(n)).
לסיכום, הוכחתי כי במקרה הגרוע ביותר הפונקציה MAX-HEAPIFY נקראת ceiling(log(n)) ובכל פעם מתבצעות 2(באחרונה עבור קלטים מסוימים 1) השוואות ולכן מתבצעות בערך log(n)2 השוואות.

סעיף ב

הרעיון בשגרה הוא להוציא את הצומת האחרונה, אחר כך למצוא את המסלול שעובר בכל שלב בבן הגדול ביותר; את המסלול הזה צריך "לדחוף למעלה" כדי להוציא את השורש. נשמור את המסלול במערך, (שיהיה ממויין הפוך משום שאנו יורדים בערימה) ובמערך הממוין הזה נחפש בחיפוש בינארי את המקום בו ניתן "לשלב" את העלה שהסרנו כלומר האינדקס הכי גדול שהתא בו קטן שווה מהערך.

Binary-Search’ (A, low, high, val)

```
if high = low
        then return high
mid ← $\left\lfloor \frac{low+high}{2} \right\rfloor$
if A[mid] > val
        then return Binary-Search’(A, mid+1, high, val)
        else return Binary-Search’(A, low, mid, val)
```

Heap-Extract-Max’ (A)

```
val ← A[heap-size[A]]
heap-size[A] ← heap-size[A] -1
h ← 1
i ← 1
while Left(h)<heap-size[A]
        do      if Right(h) < heap-size[A] and A[Right(h)] > A[Left(h)]
                        then   Path[i] ← r
                               h ← Right(h)
                        else   Path[i] ← l
                               h ← Left(h)
                B[i] ←A[h]
                i ← i+1
j ← Binary-Search’(B,1, i-1, val)
h←1
from k←1 to j
        do      A[h]←B[i]
                if Path[i] = r
                        then h ← Right(h)
                        else h←Left(h)
```

```
A[h] ← val
from k ← j+1 to i-1
    do    if Path[i] = r
              then h ← Right(h)
              else h ← Left(h)
          A[h] ← B[i]

```

אורך הלולאה הראשונה הוא כאורך מסלול מהשורש לעלה, כלומר $\lfloor \lg(n-1) \rfloor$ במקרה הגרוע, וזה גם גודל המערך *B*.

סיבוכיות החיפוש הבינארי היא $\Theta(\lg n)$, וזהו גם מספר ההשוואות שמתבצעות שכן בכל איטרציה מתבצעת השוואה אחת. אבל משום שגודל המערך הוא לוגריתמי לגודל העץ בעצם מתבצעות $\lg \lg n$ השוואות. בהמשך השגרה אין השוואות בין איברי הערימה, לכן מתבצעות כ$\lg n + \lg \lg n + O(1)$ השוואות כנדרש.

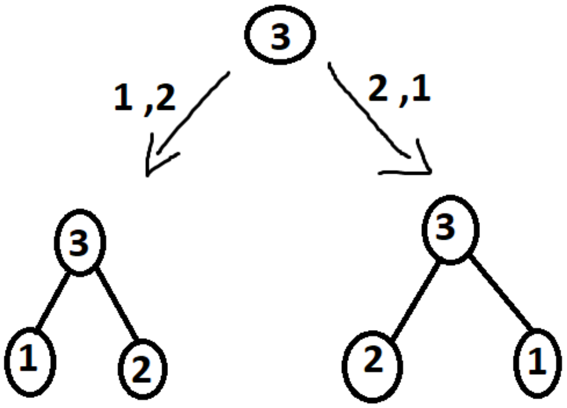
סעיף ג

Max-Heapify לוקחת את הבן השמאלי באופן קבוע כאשר שני הבנים זהים והשורש שונה.

ניקח לדוגמא את הערימה $[1,1,0,0,1,0]$
נכניס 6 לערימה באמצעות השגרה המופיעה בספר ונקבל $[6,1,1,0,1,0,0]$
עתה נבצע הוצאה של 6 ונקבל $[1,1,1,0,0,0]$
הערימות שהתקבלה לאחר הכנסה והוצאה לא זהה לערימה המקורית.

בעץ בו אין ערכים זהים – אין הבדל

סעיף ד מבנה העץ יכול להיות שונה. דוגמה נגדית:



סעיף ה

שימוש ב*BUILDHEAP* עדיף. בספר הראו שסיבוכיות הפעולה היא $O(n)$.

אראה ששימוש חוזר ונשנה בהוספות הוא $\Theta(n \lg n)$.

סיבוכיות הוספה לערימה בגודל n הוא $\Theta(\lg n)$. לכן כאן יש כ:

$\sum_{i=1}^{n} \lg n$ פעולות.

סכום של לוגריתמים הוא לוגריתם של מכפלה, לכן מתקיים:

$\sum_{i=1}^{n} \lg n = \lg n!$.

ולפי מדריך הלמידה בשאלה 5-ב מתקיים:

$$\sum_{i=1}^{n} \lg n = \Theta(n \lg n)$$

כלומר סיבוכיות הוספות חוזרות לערימה גדולה מסיבוכיות הפעולה *BUILDHEAP*.

שאלה 3

סעיף א

קלט: מערך של מספרים טבעיים

פלט: המערך ממויין כך שכל המספרים האי זוגיים מופיעים לפני הזוגיים במערך.

נשנה במקצת את האלגוריתם partition של מיון מהיר (עמוד 122 בספר). האלגוריתם עובד על אותו הרעיון – נתקדם לאורך המערך, עם 2 אינדקסים – אינדקס i שיצביע לקצה המספרים האי זוגיים, ואינדקס j- שיצביע לאיברים שעוד לא בדקנו. הריצה תסתיים כאשר בדקנו את כל האיברים. נקבל מערך, שבו באינדקים 1 עד i מופיעים כל האי זוגיים, ומi+1 ועד לסוף המערך מופיעים הזוגיים.

האלגוריתם:

```
odds_and_evens(array):
    i = 0
    for j <- 1 to length(array):
        if array[j] mod 2 is not 0: // odd number
            i = i + 1
            exchange array[i] <--> array[j]
```

השינויים מהשגרה partition הם שאין איבר ציר (אין צורך בו), והתנאי הוא אחר. מאחר ואלו שינויים שלא משנים את סדר הגודל, זמן הריצה הוא זהה – Θ(n) בזמן ו Θ(1) סיבוכיות מקום.

סעיף ב3ב1

נניח כי מדובר באלגוריתם quicksort לא אקראי.

בכל שלב נסמן את איבר הציר בכחול, את האיברים הגדולים באדום, ואת האיברים הקטנים בירוק.

[1,3,5,7,9,2,4,6,8,10]

[1,3,5,7,2,4,6,8,9,10]

[1,3,5,2,4,6,7,8,9,10]

[1,3,2,4,5,6,7,8,9,10]

[1,2,3,4,5,6,7,8,9,10]
[1,2,3,4,5,6,7,8,9,10]
וסיימנו.

סעיף 3ב2

בקריאה הראשונה לאלגוריתם המיון, שגרת החלוקה מחלקת את תת המערך לאיזור המכיל n-1 איברים ואזור המכיל 0 איברים. זה קורה מכיוון שA[n] ישמש כאיבר הציר, ובגלל נתוני הקלט A הנתונים בשאלה, הוא האיבר הכי גבוהה במערך.

מלבד הקריאה הראשונה, בכל קריאה למיון, שגרת החלוקה מחלקת את תת המערך לאיזור המכיל n-2 איברים ואזור המכיל איבר אחד.

נסביר: לאחר כל קריאה לחלוקה, תת המערך הנוכחי מכיל את כל האיברים מ1 ועד לאיבר הציר הקודם. מכיוון שתת המערך הימני מורכב מאיברים זוגיים וממויינים בסדר עולה, איבר הציר הבא שיבחר קטן ב2 מאיבר הציר הקודם. בתת המערך הנוכחי, יש רק איבר אחד שגדול יותר מאיבר הציר – זהו האיבר שבין איבר הציר הנוכחי לבין איבר הציר הקודם. (למשל, אם איבר הציר הקודם היה 12, אז כל האיברים בתת המערך הנוכחי קטנים ממש מ12. איבר הציר שיבחר יהיה 10 (כי הוא הזוגי הקרוב ביותר ל12). בתת המערך הנוכחי, רק 11 גדול יותר מ10).

בסיום החלוקה, יעבור האיבר הגדול יותר לצד ימין, ושאר המערך ישאר ללא שינוי בצד שמאל. הקריאה הבאה לחלוקה תיצור שוב איזור (מצד ימין) של איבר אחד (בדוגמא ממקודם, רק האיבר עם 11 יהיה בתת המערך הזה), ואיזור (מצד שמאל) של n-2 איברים: n האיברים המקוריים, בלי איבר הציר, ובלי האיבר הבודד שהיה גדול ממנו.

וכן הלאה במשך כל הקריאות הרקורסיביות. בכל שלב יהיה רק איבר אחד שגדול יותר מאיבר הציר, ולכן החלוקה תיצור שוב איזור בגודל 1 ואיזור בגודל n-2.

הפעלת מיון מהיר על מערך בגודל 0 (בפעם הראשונה) או 1 (בפעמים הבאות) חוזרת בלי שום שינוי, ולכן זמן הריצה הוא $\Theta(1)$. החלוקה לוקחת בכל פעם $\Theta(n)$, ולכן זמן הריצה יהיה:

$$T(n) = T(n-2) + \theta(1) + \theta(n)$$

נחשב בעזרת שיטת האיטרציה:

$$\begin{aligned}
&\mathrm{T}(n) = T(n-2) + \theta(1) + \theta(n) \\
&= T(n-4) + 2 * \theta(1) + \theta(n) + \theta(n-2) \\
&= T(n-6) + 3 * \theta(1) + \theta(n) + \theta(n-2) + \theta(n-4) \\
&in\ the\ i_{th} iteration: \ldots = T(n - 2 * i) + i * \theta(1) + \sum_{k=0}^{i-1} n - k * 2
\end{aligned}$$

נעצור כאשר תת המערך השמאלי יהיה גם הוא עם איבר 1:

$$\begin{aligned}
&n - 2 * i = 1 \\
&i = \frac{n-1}{2}
\end{aligned}$$

נציב:

$$\begin{aligned}
T(n) &= T(1) + \frac{n-1}{2} * \theta(1) + \sum_{k=0}^{\frac{n-1}{2}-1} n - k * 2 \\
&= \theta(1) + \theta(n) + [n + (n-2) + (n-4) + \cdots + 5 + 3]
\end{aligned}$$

זהו סכום של סדרה חשבונית שעולה בכל פעם ב2, ולכן:

$$\theta(1)+\theta(n)+\frac{(n+3)*\frac{n-1}{2}}{2}=$$
$$\theta(1)+\theta(n)+\frac{(n+3)(n-1)}{4}=\theta(1)+\theta(n)\theta(1)+\theta(n)+\theta(n^2)=\theta(n^2)$$

ז"א שסיבוכיות הזמן היא $\theta(n^2)$.

שאלה 4 סעיף א

הביטוי (M-n)/n   הוא המרחק הממוצע בין כל איברי הסדרה. (נשים לב שיש n+1 איברים, ולכן יש n "מרחקים" בין האיברים). הביטוי |x-y| הוא מרחק בין שני איברים כלשהם. יכולים להיות 2 מקרים:

1. המרחק בין כל האיברים בסדרה שווה. במקרה כזה הטענה נכונה, כי מרחק בין כל שני איברים שווה לממוצע המרחקים.
2. יתכן ויש 2 איברים שהמרחק ביניהם גדול ממוצע המרחקים. אם קיימים 2 איברים כאלו, הרי שבהכרח קיימים גם 2 איברים אחרים שהמרחק ביניהם קטן מהממוצע (לא יכול להיות שכל המרחקים גדולים מהממוצע)(זה נקרא עיקרון שובך היונים) . ולכן הטענה נכונה.

שאלה 4 סעיף ב

הסבר על האלגוריתם:

ראשית נמצא את M ואת m, (מקסימום ומינימום), ונסדר את המערך כך שהמינימום יהיה בהתחלה והמקסימום בסוף.

נמצא את החציון במערך, בעזרת שגרת SELECT. בסיום הריצה, כל המערך מחולק כך שכל האיברים הגדולים מהחציון בחצי המערך הימני, וכל השאר בחצי השמאלי. החציון הוא המינימום של תת המערך הימני, והמקסימום של תת המערך השמאלי. נמצא את ממוצע המרחקים שבחצי המערך השמאלי. אם ממוצע תת המערך **גדול** מהממוצע הכללי – אזי בתת המערך העליון (כלל החציון בעצמו) בוודאי ימצאו 2 איברים שמרחקם זה מזה קטן מהממוצע (לפי ההסבר בסעיף א').   אם ממוצע תת המערך **קטן** מהממוצע הכללי – אז 2 האיברים שמרחקם קטן מהממוצע נמצא בתת המערך התחתון (גם הוא כולל החציון עצמו, כי יתכן שהחציון הוא אחד משני האיברים x וy). אם הממוצע שווה – אז ב2 תתי המערכים ישנם 2 ערכים שמרחקם קטן או שווה לממוצע, ולכן נבחר לחפש בתת הסדרה התחתונה בצורה שרירותית. לאחר מכן נקרא בצורה רקורסיבית על תת המערך שבו ימצאו x וy כנדרש.

השגרה תעצור כאשר יש מערך של 2 איברים – ואז הם האיברים X וY הנדרשים.

סיבוכיות:

findXY :

שורות 3 + 4: $\Theta(n)$ כל אחת, בסך הכל $\Theta(n)$. כל שאר השורות $\Theta(1)$

getXY:

בכל קריאה רקורסיבית, נחתך המערך בחצי. מציאת החציון היא $\Theta(n)$, וכל שאר הפעולות בזמן קבוע. לכן, הסיבוכיות היא נוסחת הנסיגה:

$$T(n) = T\left(\frac{n}{2}\right) + \text{n}$$
$$a = 1; b = 2; n^{\lg_2 1} = n^0 = 1$$

קיים קבוע ε=0.5 כך ש Θ(n)=Ω(n^0.5), ובנוסף עבור c=0.5 מתקיים ש $1 * \frac{n}{2} \le \frac{1}{2} * n$, ולכן לפי מקרה 3 של שיטת האב, קיבלנו ש T(n)= Θ(n).

נחבר את הסיבוכיות, קיבלנו Θ(n)+ Θ(n)= Θ(n).

האלגוריתם:

```
findXY(array):
    len <- length_of(array) // length of Array
    max <- select(array, 1, len, len) // finding the maximum and placing it in
                                      // array[len]
    min <- select(array, 1, len, 1) // finding the minimum and placing it in
                                    // array[1]
    avg_distnc <- (max-min)/len-1 // note that n in the question is number of
                                  // "distances" in the array, not the number
                                  //of elements in the array
    return getXY(array, 1, len, len, avg_distnc)


getXY(array, min_index, max_index, len, avg_distnc):
    if (len = 2): // array size of 2
        return [array[min_index] , array[max_index]] // x and y found
    else:
        // find median
        med_index <- $\left\lfloor \frac{\text{min\_index+max\_index}}{2} \right\rfloor$
        med <- select(array, min_index, max_index, $\left\lfloor \frac{len+1}{2} \right\rfloor$ )
        // at this point, all the values above med are in the right side, and
        // all the values below med are in the left side
        lower_half_len <- med_index - min_index + 1
        lower_half_avg <- ((med - array[min_index])/(lower_half_len - 1))
        if lower_half_avg <= avg_distnc: // x and y are in the lower sub array
            return getXY(array, min_index, med_index, lower_half_len, avg_distnc)
        else:
            higer_half_len <- max_index - med_index + 1
            return getXY(array, med_index, max_index, higer_half_len, avg_distnc)
```

שאלה 5 א

נניח בשלילה שקיימים יותר מ4 מספרים החוזרים על עצמם יותר מ $\left\lfloor\frac{n}{5}\right\rfloor$ פעמים. למשל, קיימים 5 מספרים כאלה. נחשב מהו המספר המינימלי של מספרים שקיימים בS.

המספרים מופיעים **יותר** מ$\left\lfloor\frac{n}{5}\right\rfloor$ פעמים, ז"א לפחות $\left\lfloor\frac{n}{5}\right\rfloor + 1$ פעמים. ניתן לחסום את הביטוי $\left\lfloor\frac{n}{5}\right\rfloor$ ע"י הביטוי $\frac{n-(5-1)}{5} = \frac{n-4}{5}$, לפי נוסחא 3.7 בספר. נחשב:

$$5 * (\left\lfloor\frac{n}{5}\right\rfloor + 1) \geq 5 * \left(\frac{n-4}{5} + 1\right) = n - 4 + 5 = n + 1$$

קיבלנו שסכום המספרים ב S חייבים להיות גדול או שווה מn+1, וזה לא נכון כי נתון שב S יש n מספרים. לכן הנחת השלילה נפלה, והוכחנו את הטענה.

סעיף ב

הסבר: קיבלנו מערך בין n מספרים. לצורך ההסבר, נניח שהמערך ממויין. (אין צורך למיין את המערך בפועל). עכשיו, נחלק את המערך ל5 קבוצות שוות (עד כדי 1). בכל קבוצה, יש n/5 איברים. אם איבר כלשהו מופיע **יותר** מn/5 פעמים במערך – אז בוודאי שהוא מופיע ביותר מקבוצה אחת.

לכן, אם נמצא את האיברים בנקודות החיבור בין הקבוצות, ז"א האיברים שמחלקים את המערך ל5 קבוצות שוות – האיברים שמצאנו הם מועמדים פוטנציאליים להיות אותם 4 איברים שאנחנו מחפשים, שמופיעים **יותר** מn/5 פעמים במערך.

ז"א, אנחנו צריכים למצוא את האיברים שערכי המיקום שלהם הם:

$$\left\lfloor\frac{n}{5}\right\rfloor, 2 * \left\lfloor\frac{n}{5}\right\rfloor, 3 * \left\lfloor\frac{n}{5}\right\rfloor, \ 4 * \left\lfloor\frac{n}{5}\right\rfloor$$

ניתן למצוא את ערכי המיקום בלי למיין.

לאחר שנמצא את ערכי המיקום האלו, נקבל מערך בן 4 איברים שמכיל את ערכי המיקום הנ"ל. בנוסף, זהו מערך ממויין (מכיון שהקריאה ל SELECT עושה גם PARTITION ומחלקת את המערך לפי ערכי המיקום, ונדאג לסדר את הקריאות הרקורסיביות ככה שיחזירו מערך ממויין). בנוסף, המערך המקורי **לא** ממויין, אבל הוא מחולק לאיזורים לפי ערכי המיקום.

לאחר מכן, נותר לעשות 2 דברים:

- לוודא שאכן המספרים שנמצאו מופיעים במערך יותר מ $\left\lfloor\frac{n}{5}\right\rfloor$ פעמים.
- להסיר את הכפולים מהמערך

נעשה את זה באמצעות מעבר על כל ערכי המיקום שמצאנו. עבור כל ערך מיקום, נבדוק כמה פעמים הוא מופיע באיזור שמלפניו ובאיזור שאחריו. אם מצאנו יותר חזרות מאשר $\left\lfloor\frac{n}{5}\right\rfloor$, זה הערך שאנחנו מחפשים. בנוסף, נבדוק גם מול ערך המיקום הבא בסדר – אם הוא אותו הערך, נמחק אותו מהרשימה כי הוא מופיע פעמיים. (כדי לבדוק אם כל ערך מופיע פעמיים, מכיון שמערך ערכי המיקום ממויין, נצטרך לבדוק רק כל איבר אל מול האיבר העוקב).

האלגוריתם:

1. נמצא את 4 ערכי המיקום בעזרת QUANT (מדריך הלמידה שאלה ו-13. האלגוריתם מוצא את ערך המיקום האמצעי, ואז קורא רקורסיבית ל2 הצדדים – נדאג לסדר את התוצאות בצורה שתוציא מערך ממוין. השגרה תמצא את ערך המיקום האמצעי ותחלק את המערך על פיו. לפני שערך המיקום הנמצא יתווסף למערך שיוחזר מהשגרה, תתבצע קריאה רקורסיבית שמאלה (לערכים הקטנים). לאחר מכן ערך המיקום הנוכחי יתווסף למערך, ולאחר מכן תתבצע קריאה רקורסיבית ימינה. בצורה כזאת נקבל מערך ממויין, ללא שינוי בזמני הריצה)

2. עבור כל ערך, נבדוק:
   - א. אם הוא שווה לערך שמעליו – נוציא את הערך מהמערך הסופי
   - ב. אחרת, נספור כמה פעמים הערך מופיע, באיזור המערך שמתחת ומעל לערך המיקום. (לדוגמה, אם $\left\lfloor \frac{n}{5} \right\rfloor = 7$, נספור 7 ערכים מעל ו7 ערכים מתחת לערך המיקום שמצאנו).
   - ג. אם הערך מופיע פחות פעמים מ$\left\lfloor \frac{n}{5} \right\rfloor$ – נוציא אותו מהמערך הסופי.
3. נחזיר את המערך שנותר

```
find5(array, n):
    optionals <- QUANT(array, n, 5) // array of optionals numbers
    // validating the numbers
    location <- floor(n/5)
    for i from 1 to 4:
        // duplicats check
        if i is not 4:
            if optionals[i] = optionals[i+1]:
                remove optionals[i] from optionals
                 continue from top of the loop
        // occurrences check
        counter <- 0
        for j from (i-1)*location+1 to (i+1)*location:
            if array[j] = optionals[i]:
                counter <- counter +1
        if counter <= location:
            remove optionals[i] from optionals

    return optionals
```

סיבוכיות (לפי הסעיפים שהוגדרו למעלה):

1. סיבוכיות של $\Theta(n\lg5)=\Theta(n)$ לפי הספר. לשינוי שביצענו על מנת לקבל מערך ממויין אין השפעה על זמני הריצה (אלא רק סדר שונה של קריאות)
2. את הלולאה נבצע 4 פעמים:
   - א. $\Theta(1)$
   - ב. עבור כל ערך, נשווה מול $\left\lfloor \frac{n}{5} \right\rfloor * 2$ ערכים. ערכים (תת המערכים שמצדדי ערך המיקום).
   - ג. $\Theta(1)$

אם נסכם את הכל, זה יוצא $\Theta(n)$.

סעיף ג'

ע"פ אותו העיקרון של סעיף ב', יש לכל היותר k-1 איברים שמופיעים במערך יותר מ$\left\lfloor \frac{n}{k} \right\rfloor$ פעמים.
האלגוריתם יעבוד באופן זהה לאלגוריתם בסעיף ב':

1. נמצא את k-1 ערכי המיקום בעזרת QUANT (מדריך הלמידה שאלה 13-ו. האלגוריתם מוצא את ערך המיקום האמצעי, ואז קורא רקורסיבית ל2 הצדדים – נדאג לסדר את התוצאות בצורה שתוציא מערך ממוין. השגרה תמצא את ערך המיקום האמצעי ותחלק את המערך על פיו. לפני שערך המיקום הנמצא יתווסף למערך שיוחזר מהשגרה, תתבצע קריאה רקורסיבית שמאלה (לערכים הקטנים). לאחר מכן ערך המיקום הנוכחי יתווסף למערך, ולאחר מכן תתבצע קריאה רקורסיבית ימינה. בצורה כזאת נקבל מערך ממויין, ללא שינוי בזמני הריצה)
2. עבור כל ערך, נבדוק:
   ד. אם הוא שווה לערך שמעליו – נוציא את הערך מהמערך הסופי
   ה. אחרת, נספור כמה פעמים הערך מופיע, באיזור המערך שמתחת ומעל לערך המיקום. (לדוגמה, אם $\left\lfloor \frac{n}{k} \right\rfloor = 7$, נספור 7 ערכים מעל ו7 ערכים מתחת לערך המיקום שמצאנו).
   ו. אם הערך מופיע פחות פעמים מ$\left\lfloor \frac{n}{k} \right\rfloor$ – נוציא אותו מהמערך הסופי.
3. נחזיר את המערך שנותר

סיבוכיות הזמן:

1. סיבוכיות של $\Theta(n\lg k)$ לפי הספר. לשינוי שביצענו על מנת לקבל מערך ממויין אין השפעה על זמני הריצה (אלא רק סדר שונה של קריאות)
2. את הלולאה נבצע k-1 פעמים:
   ד. $\Theta(1)$
   ה. עבור כל ערך, נשווה מול $\left\lfloor \frac{n}{k} \right\rfloor * 2$ ערכים. ערכים (תת המערכים שמצדדי ערך המיקום).
   ו. $\Theta(1)$

בסך הכל נקבל:

$$T(n) = \theta(nlgk) + (k-1) * \left(2 * \left\lfloor \frac{n}{k} \right\rfloor\right) =$$
$$= \theta(nlgk) + 2k * \frac{n}{k} - 2 * \frac{n}{k} = \theta(nlgk) + \theta(n) - \theta\left(\frac{n}{k}\right) =$$
$$= \theta(nlgk)$$

סיימנו.