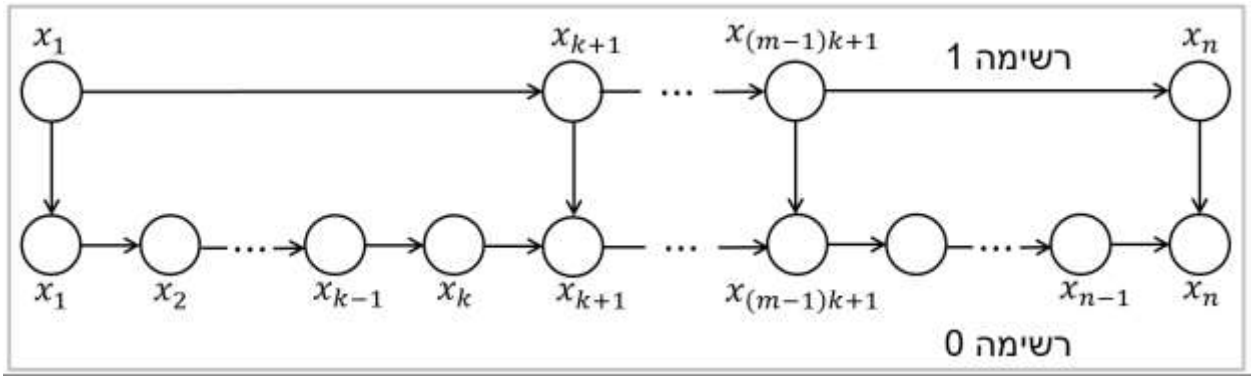
שאלה 1 (40 נק')

בשאלה זו עוסקים בחיפוש ברשימות מקושרות, ומתעלמים לחלוטין מהכנסת ומחיקת איברים.

חברת "הרשימה המקושרת" בנתה רשימה מקושרת שאיבריה $x_1, x_2, \dots, x_n$ ממוינים משמאל לימין בסדר עולה כמתואר בציור. נקרא לה רשימה 0. מניחים שכל האיברים שונים זה מזה.

על מנת להאיץ את זמן החפוש מהנדס א' הציע להוסיף רשימה מקושרת כלהלן (ראו ציור):

1. מחלקים את רשימה 0 ל $m = \lceil n/k \rceil$ קבוצות. $\lceil n/k \rceil$ הראשונות בגודל k, ואחרונה בגודל $n - k(m-1)$ במדה ו n אינו מתחלק ב k.
2. בונים רשימה מקושרת בגודל $m + 1$ שנקראת רשימה 1, של האיברים הראשונים בכל אחת מ m הקבוצות, כשהאיבר האחרון הוא x_n.
3. כל איבר ברשימה 1 מצביע לאיבר התואם ברשימה 0.



א. (10 נק') מחפשים איבר x. הציעו את אלגוריתם החיפוש היעיל ביותר במבנה הנתונים החדש.
<u>רמז</u>: השתשמו בשתי הרשימות.

פתרון

1. אם $x < x_1$ האיבר לא ברשימה.
2. עוברים על רשימה 1 עד ש $x \in \left[x_{jk+1}, x_{(j+1)k+1}\right]$, $0 \le j \le m-1$. (בטווח האחרון האיבר האחרון הוא x_n).
3. אם $j \nexists$ כך ש $x \in \left[x_{jk+1}, x_{(j+1)k+1}\right]$, אז $x > x_n$ והאיבר לא ברשימה.
4. ברשימה 0 עוברים על הטווח $\left[x_{jk+1}, x_{(j+1)k+1}\right]$ עד ש x נמצא.

<u>הערה</u>: כאשר ב 2 מוצאים את $\left[x_{jk+1}, x_{(j+1)k+1}\right]$ ניתן לבדוק מיד האם $x = x_{jk+1}$ או $x = x_{(j+1)k+1}$, אם כי מבחינת סיבוכיות זמן הריצה אין לזה שום השפעה.

ב. (5 נק') לאלגוריתם שהצעתם ב א' חשבו באופן מלא ומדויק את זמן החיפוש $T(n,k)$ במקרה הגרוע ביותר.

פתרון
צעד 2 דורש $O(n/k)$ זמן חיפוש וצעד 3 דורש $O(k)$ זמן חיפוש. סה"כ $T(n,k) = O(k) + O(n/k)$.

ג. (5 נק') מהו k שנותן זמן חיפוש מינימלי בסעיף ב'.
פתרון
את $T(n,k)$ גוזרים ומשווים לאפס $dT(n,k)/dk = 0$ ומקבלים $k^2 = n$, ולכן $k = \sqrt{n}$.

מהנדס ב' הציע להרחיב את הרעיון של א' ולבנות "מגדל" של רשימות מקושרת כלהלן:

1. מחלקים את רשימה 0 ל $m = \lceil n/k \rceil$ קבוצות. $\lfloor n/k \rfloor$ הראשונות בגודל k, ואחרונה בגודל $n - k(m-1)$ במדה ו n אינו מתחלק ב k.
2. בונים רשימה מקושרת בגודל $m+1$ שנקראת רשימה 1, של האיבר הראשון בכל אחת מ m הקבוצות, כשהאיבר האחרון הוא x_n.
3. כל איבר ברשימה 1 מצביע לאיבר התואם ברשימה 0.
4. אם $m > k$ אז מציבים $n = m$, רשימה 1 הופכת ל רשימה 0, וחוזרים לצעד 1.

ד. (5 נק') הציעו את אלגוריתם החיפוש היעיל ביותר האפשרי במבנה הנתונים החדש.
<u>רמז</u>: חשבו ראשית את מספר הרשימות המקושרות (גובה ה"מגדל").
פתרון
מאחר ורשימה 1 קטנה בפקטור k ביחס לרשימה 0, והתהליך מתחיל ברשימה באורך n ומסתיים ברשימה באורך לכל היותר k, מספר הרשימות הינו $\lceil \log_k n \rceil$.
מתחילים ברשימה העליונה ויורדים כל פעם לרשימה התחתונה עד שמגיעם לרשימה 0 ושם מוצאים את x.

ה. (5 נק') בהנחה שאיבר קיים ברשימה, חשבו באופן מלא ומדויק את זמן החיפוש $T(n,k)$ במקרה הגרוע ביותר.
פתרון
מאחר ומספר הרשימות המקושרות ("קומות") הינו $\Theta(\log_k n)$ וגודל קבוצה שעוברים עליה בכל רשימה אינו עולה על k, $T(n,k) = \sum_{i=1}^{\lceil \log_k n \rceil} O(k) = O(k \log_k n)$.

ו. (10 נק') יהי k מספר קבוע שאינו תלוי ב n. האם זמן החיפוש האסימפטוטי במבנה שהציע מהנדס ב' גרוע, שווה או טוב יותר מאשר בעץ חיפוש בינארי מאוזן? נמקו והוכיחו באופן מלא את תשובתכם.
פתרון
בעץ בינארי מאוזן זמן החיפוש הוא $O(\log_2 n)$.
אם k קבוע ואינו תלוי ב n אז מתקיים $O(k \log_k n) = O(\log_k n)$.

כמו כן $O(\log_2 n) = \log_2 k\, O(\log_k n) = O(\log_k n)$.
ולכן זמן החיפוש האסימפטוטי שווה.

<u>שאלה 2 (40 נק')</u>

נתון מערך A בגודל n של מספרים שלמים שונים זה מזה.
במערך פזורים k מספרים זוגיים בסדר שרירותי ובמקומות שרירותיים, ואילו שאר $n-k$ המספרים האחרים הם אי-זוגיים ומופיעים במערך כשהם כבר ממויינים בסדר עולה.
יהי $k = \lceil n/\log n \rceil$.

א. (20 נקודות) הציגו אלגוריתם למיון A בסבוכיות זמן $O(n)$.
<u>רמז</u>: הפרידו את A לשני תתי מערכים של זוגיים ואי זוגיים.

פתרון

עוברים על כל המערך A בזמן $O(n)$ ומפרידים אותו לשני תתי מערכים A_O ו A_E.
A_O באורך $n-k$ שמכיל את המספרים האי זוגיים, ומנתוני הבעיה הוא כבר ממוין בסדר עולה.
A_E באורך k שמכיל את המספרים הזוגיים ואינו ממוין.

נמיין את A_E בזמן $O(k\log k)$, למשל ע"י אלגוריתם MERGE-SORT , ובהצבת k נקבל $O\big((n/\log n)\log(n/\log n)\big) = O(n)$.

כעת A_O ו A_E ממוינים בסדר עולה, וניתן במעבר ליניארי על שניהם במקביל, כמו שלב ה merge ב MERG-ESORT למזגם בזמן $O(n)$ למערך A ממוין כולו.

סכום כל הזמנים הנ"ל $O(n)$.

רוצים למיין מערך A בגודל n . מהנדסי חברת "הממין" הציעו אלגוריתם מיון חדש שנקרא BST-SORT ופועל ע"י באופן הבא:
1. מעבר על איברי A, והכנסת כל איבר לתוך BST.
2. שליפה ומחיקה חוזרת ונישנית n פעמים של האיבר הקטן ביותר ב BST ומילוי מחדש של A החל מהמקום 0 והלאה.

ב. (10 נקודות) חשבו באופן מנומק מלא ומדוייק את זמן המיון למקרים הגרוע ביותר והטוב ביותר לכל אחד מהשלבים 1 ו 2 של BST-SORT.

פתרון

א. המעבר על A לכשעצמו לוקח $\Theta(n)$ זמן בלא קשר ל BST.
הכנסה ל BST במקרה הגרוע ביותר לוקחת $\Theta(n)$ זמן, ולכן בניית כל ה BST לוקחת $\Theta(n^2)$ זמן.
הכנסה ל BST במקרה הטוב ביותר לוקחת $O(\log n)$ זמן, ולכן בניית כל ה BST לוקחת $\Theta(n\log n)$ זמן.

ב. מציאת מינימום ומחיקתו מ BST במקרה הגרוע ביותר לוקחת $\Theta(n)$ זמן, ולכן ריקון כל ה BST לוקח $\Theta(n^2)$ זמן.
מציאת מינימום ומחיקתו מ BST במקרה הטוב ביותר לוקחת $\Theta(\log n)$ זמן, ולכן ריקון כל ה BST לוקח $\Theta(n \log n)$ זמן.
המילוי חזרה של A לכשעצמו לוקח $\Theta(n)$ זמן בלא קשר ל BST.

ג. (10 נקודות) השוו בין הביצועים האסימפטוטיים של BST-SORT ו QUICK-SORT ביחס לזמן וזיכרון הנדרשים למקרים הגרוע ביותר והטוב ביותר. מלאו את הטבלה המצורפת.

		BST-SORT	QUICK-SORT
Run Time	Worst-case		
	Best-case		
Memory	Worst-case		
	Best-case		

פתרון

		BST-SORT	QUICK-SORT
Run Time	Worst-case	$\Theta(n^2)$	$\Theta(n^2)$
	Best-case	$\Theta(n \log n)$	$\Theta(n \log n)$
Memory	Worst-case	$\Theta(n)$	$\Theta(n)$
	Best-case	$\Theta(n)$	$\Theta(n)$

<u>שאלה 3 (40 נק')</u>

א. (10 נקודות) נתונה המשואה $F(n) = F(n-1) + F(n-2)$, $F(0) = 0$, $F(1) = 1$.
צריך להוכיח ש $F(n) \geq (3/2)^{n-2}$.

פתרון

באינדוקציה. עבור $n = 0$ ו $n = 1$ מתקיים ע"פ הגדרה. עבור $n = 2$ מתקיים
$F(2) = F(1) + F(0) = 1 \geq (3/2)^{2-2}$.
נניח $F(i) \geq (3/2)^{i-2}$ לכל $0 \leq i \leq n-1$. אזי
$$F(n) = F(n-1) + F(n-2) \geq (3/2)^{i-3} + (3/2)^{i-4}$$
$$\geq (3/2)^{i-2}(2/3 + 4/9) \geq (3/2)^{i-2}$$

חברת ייצור צריכה לבצע n עבודות $\{a_1, a_2, \dots, a_n\}$.

העבודות a_i , $1 \le i \le n$, מתוזמנות מראש ומוגדרות ע"י הפרמטרים (s_i, f_i, v_i) כאשר s_i - זמן התחלת העבודה, f_i - זמן סיום העבודה, v_i - הרווח לחברה מביצוע העבודה.
העבודות מסודרות מראש כך ש $f_1 \le f_2 \le \cdots \le f_n$.
לחברה מכונה אחת בלבד. המכונה מסוגלת לבצע בזמן נתון עבודה אחת בלבד, כלומר ניתן לבצע את העבודות a_i ו a_j , $i < j$, אם ורק אם $f_i \le s_j$ (אין חפיפה בין התזמונים שלהן).
מטרת החברה להשיג רווח מקסימלי מביצוע העבודות, כלומר להחליט אילו עבודות יבוצעו.
מהנדס הייצור של החברה החליט לפתור את הבעיה באופן הבא:
להוסיף עבודה מלאכותית (s_0, f_0, v_0): a_0 כאשר $s_0 = f_0 = 0$ וכן $v_0 = 0$.
להצמיד לכל a_i פרמטר p_i המוגדר כדלהלן:

$$p_i = \max_{0 \le k < i} \{f_k \le s_i\}, \ 1 \le i \le n$$

כלומר a_k קודם הכי מאוחר שניתן לביצוע כך שאפשר לבצע גם את a_i .
מהנדס הייצור הציע את האלגוריתם הבא שלדעתו יחשב את הרווח מקסימלי

```
JOB-OPT(i)
    if i == 0 return 0
    return max{v_i + JOB-OPT(p_i) , JOB-OPT(i − 1)}
```

ב. (10 נקודות) האם JOB-OPT אכן מבצע חישוב נכון? נמקו הייטב את תשובתכם.

פתרון

ראשית, הרקורסיה אכן מסתיימת משום שבהגדרה $p_i < i$ ובכל קריאה האינדקס i יורד ולבסוף מתאפס.

שנית, כל תת פתרון של פתרון אופטימלי חייב בעצמו להיות כזה.

שלישית, הפרמטר $i-1$ המועבר לרקורסיה מבטיח שכל תתי הבעיות האפשריות $a_1, a_2, \dots, a_i, 1 \le i \le n$ אכן נפתרות באופן אופטימלי.

ג. (10 נקודות) הוכיחו שזמן הריצה של JOB-OPT הוא $T(n) = \Omega((3/2)^n)$ (בלא קשר לב').

<u>רמז והנחיה לפתרון</u>:

השתמשו בבעיית תזמון שעבורה $p_i = i - 2$, $2 \le i \le n$. אין צורך לבנות דוגמא באופן מפורש ומספיק להניח שקיימת בעיה כזאת.
עשו שימוש בתוצאה של סעיף א'.

פתרון

בעית התזמון המוצעת גורמת לכך שפתרון JOB-OPT(i) מחייב פתרון JOB-OPT$(i-1)$ ו JOB-OPT$(i-2)$.

נסמן $T(n)$ את זמן החישוב עבור בעיה JOB-OPT(n) ונקבל

$T(n) = T(n-1) + T(n-2)$, $T(0) = 0$, $T(1) = 1$.

הבעיה אכן נפתרה בסעיף א' כנדרש.

ד. (10 נקודות) הציעו אלגוריתם שפותר את הבעיה בזמן פולינומיאלי הטוב ביותר שניתן. מספיק שהאלגוריתם יחשב את הערך של הרווח המקסימלי, ואין צורך שירשום מיהן העבודות שנבחרו לביצוע.

<u>הנחיה</u>: רישמו באופן מפורש משוואה שמחשבת את הרווח המקסימלי. לאחר מכן רישמו PSEUDO CODE והוכיחו את זמן החישוב של הקוד שרשמתם.

פתרון

לבעיה ישנה התכונה שכל תת פתרון של הפתרון האופטימלי חייב להיות אופטימלי גם כן.

יהי c_i הרווח המקסימלי שניתן להשיג מתזמון כל העבודות a_k , $1 \le k \le i$. מתקיים

$$c_i = \max\{v_i + c_{p_i}, c_{i-1}\}$$

והפתרון המתבקש הוא בתכנות דינאמי.

```
NEW–JOB–OPT
    // let C[1..n] be array of costs
    for i = 1 to n
        c_i = max{v_i + C[p_i], C[i − 1]}
    return C[n]
```

זמן החישוב הוא $\Theta(n)$ ולכן גם אופטימלי.