שאלה 1 (20 נק')

In this question, we discuss heaps implemented as binary trees (not as arrays). There are two heaps h_1 and h_2. In the first heap there are 2^k-1 nodes (k>1). In the second heap the number of nodes is smaller than the number of nodes in the first heap but it is bigger than 2^{k-1}-1.

Describe, in words, an efficient algorithm (no need for pseudocode) that merges the two heaps into one heap. The algorithm's running time should be O(k).

נדון בערמות הממומשות כעצים בינריים (ולא במערך). נתונות שתי ערמות h_1 ו-h_2. בערמה הראשונה 2^k-1 איברים (k>1), ובערמה השנייה מספר האיברים קטן ממספר האיברים בערמה הראשונה אך גדול מ- 2^{k-1}-1 איברים.

תארי אלגוריתם (אין צורך לרשום פסידוקוד) יעיל למיזוג שתי הערמות לערמה אחת. זמן הריצה של האלגוריתם יהיה O(k).

שתי הערמות באותו גובה. בראשונה כל הרמות מלאות ובשנייה לא. ניקח את העלה האחרון מהערמה השנייה, הוא יהפוך לשורש – הערמה הראשונה משמאלו ומה שנשאר מהשנייה מימינו. נבצע ערמום (גלגול כלפי מטה) לשורש ונקבל ערמה. גובה הערמה הוא O(k), וזה גם זמן הריצה.

שאלה 2 (20 נק')

Write (in pseudocode) a recursive algorithm BUILD-TREE (A,p,r), that gets a sorted array A and using the array elements, from index p to index r, builds a balanced search tree.

Assume that new(x) creates a new node with key x and three null pointers: parent, left and right. new(x) is already implemented and you do not have to implement it yourself.

The running time of the algorithm should be O(n).

The algorithm must not be longer than 15 lines. Every line will contain one instruction. The pseudocode must be indented.

כתבי (בפסידוקוד) אלגוריתם רקורסיבי בשם BUILD-TREE(A,p,r), אשר יבנה מאיברי מערך מסודר A , מאינדקס p עד אינדקס r, עץ חיפוש בינרי **מאוזן**.

הניחי שהפונקציה new(x) יוצרת צמת חדש בעל מפתח x, ושלושה מצביעים ל null: left, right ו-parent. הפונקציה כבר ממומשת ואינך צריכה לממשה.

זמן הריצה של האלגוריתם צריך להיות $O(n)$.

האלגוריתם לא יכיל יותר מ-15 שורות. כל שורה תכיל הוראה אחת. האלגוריתם יהיה מעומד ומוזח כנדרש.

```
BUILD-TREE(A,p,r)
if p>r
   then return null
q←⌊(p+r)/2⌋
t←new(A[q])
t.left←BUILD-TREE(A, p, q-1)
t.right←BUILD-TREE(A, q+1, r)
if t.left≠null
   then t.left.parent←t
if t.right≠null
   then t.right.parent←t
return t
```

שאלה 3 (16 נקודות)

An operation f() is implemented on a data structure. This operation is executed n times.

The running time of the i'th call of f() is:

O(i) : if i is a power of 2.

O(lgi): if i is a power of 3

O(1): otherwise.

What is the amortized running time of f()? Prove your answer using the aggregation method.

על מבנה נתונים מוגדרת פעולה f(). הפעולה מבוצעת n פעמים.

זמן הריצה של הקריאה ה-i ל-f() היא:

O(i) אם i הוא חזקה של 2.

O(lgi) אם i הוא חזקה של 3.

לכל ערך אחר של i זמן הריצה של f() הוא O(1).

מה זמן הריצה האמורטי של f()? הוכיחי את תשובתך בעזרת שיטת הצבירה.

נשתמש בשיטה הראשונה ונסכם את כל זמני הריצה של n פעולות (המחובר הראשון לפעולות שמספרן הסידורי חזקה של 2, המחובר השני לפעולות שמספרן הסידורי חזקה של 3 והמחובר השלישי לשאר הפעולות):

$$\sum_{i=0}^{\lfloor \log_2 n \rfloor} 2^i + \sum_{i=0}^{\lfloor \log_3 n \rfloor} i + (n - \log_2 n - \log_3 n) < 2n + (\log_3 n)^2 + n = O(n)$$

ולכן זמן הריצה לשיעורין של כל אחת מ-n הפעולות הוא O(1).

שאלה 4 (20 נקודות)

Design a data structure that stores items with keys. The keys are uniformly distributed integers in the range of [0…10^k-1]. The following operations are defined on the data structure:

a. push(DS,x) – insert item x to DS
b. search (DS,k) – search and return an item with key x.
c. pop(DS) – remove from DS the last item that was inserted into it.

The average running time of every operation should be O(1).

Explain how the data structure is implemented. Show that the operations' running times are as required.

דביר, עבור פעולת search האם לא צריך לציין מה קורה אם האיבר המבוקש לא נמצא במבנה הנתונים.

עלייך לתכנן מבנה נתונים שיאחסן איברים בעלי מפתחות שהם מספרים שלמים המפוזרים בצורה אחידה בטווח [0…10^k-1]. על מבנה הנתונים מוגדרות הפעולות הבאות:

א. push (DS, x) – הכנסת האיבר x למבנה DS.
ב. search (DS, k) – חיפוש והחזרה של איבר בעל המפתח k.
ג. pop (DS) – מחיקת האיבר האחרון שנכנס למבנה DS.

סיבוכיות זמן הריצה של כל הפעולות צריך להיות O(1) בממוצע.

הסבירי איך ימומש המבנה ואיך תתבצע כל פעולה כך שתעמוד בתנאי זמן הריצה.

מבנה הנתונים יורכב ממחסנית ומטבלת גיבוב.
בהכנסה נכניס לטבלה ולמחסנית. זמן ההכנסה למחסנית הוא O(1), בכל מקרה, וזמן ההכנסה לטבלת גיבוב הוא O(1) בממוצע.
חיפוש יעשה על טבלת הגיבוב – O(1) בממוצע.

מחיקה תתבצע על-ידי הוצאה מהמחסנית, חיפוש האיבר בטבלה ומחיקתו. הוצאה מהמחסנית מתבצעת ב-O(1), בכל מקרה, ומחיקה מטבלת גיבוב מתבצעת ב-O(1) בממוצע.

שאלה 5 (24 נקודות)

Write true/false and explain.

a. When analyzing amortized running time, it doesn't matter which of the possible three methods you will use, the amortized running time will be the same.
b. The running time of an efficient algorithm that prints the five smallest keys in a minimum heap is O(1).
c. A rotation is performed on a node in an AVL tree (with unique keys). The preorder scan of the tree before the rotation must be different from the preorder scan of the tree after the scan.

לכל משפט כתבי אם הוא נכון או לא נכון. נמקי בקצרה.

א. כאשר ננתח זמן ריצה לשיעורין, לא משנה באיזו משלוש השיטות נשתמש, נקבל את אותו זמן ריצה.
ב. זמן הריצה של אלגוריתם יעיל שמקבל ערמת מינימום ומדפיס את חמשת המפתחות הקטנים ביותר בערמה הוא O(1).
ג. מבוצעת רוטציה על צמת בעץ AVL שכל איבריו שונים זה מזה. הסריקה בסדר תחילי של העץ לפני הרוטציה שונה, בהכרח, מהסריקה בסדר תחילי של העץ אחרי הרוטציה.

א. לא נכון. אפילו באותה שיטה ניתן לקבל זמנים שונים, למשל, על-ידי בחירת פונקציות פוטנציאל שונות.
ב. נכון. 5 האיברים הקטנים ביותר הם בין 31 האיברים הראשונים בערמה. נמיין אותם ונמצא את הקטנים ביותר. מכיוון שמספר האיברים בהם נחפש (31) הוא קבוע (לא תלוי בגודל הערמה) אז זמן הריצה הוא קבוע.
ג. נכון. אם בוצעה רוטציה ימינה על X, אז בנו השמאלי של X (נקרא לו Y) הופך להיות אביו של X. X הופך להיות בנו הימני של Y. אם כך, לפני הרוטציה X קדם ל-Y בסדר תחילי, ואחרי הרוטציה Y קודם ל-X.