הערה: השתמשנו בשיטת ה-Balance Factor כדי לאזן את העץ.
איזנו חישבנו את מספר פעולות האיזון כפי שנלמד בכיתה, השתמשנו ב-BF רק כדי לסווג את המקרים.
הגדרת Balance Factor: BF הוא גובה תת העץ הימני פחות גובה תת העץ השמאלי.

מחלקת AVLNode

תיאור כללי: מחלקה פנימית המייצגת צומת בעץ AVL ומממשת את המנשק IAVLNode.

שדות:

שם	תיאור
int key	המפתח של הצומת בעץ
String value	מחרוזת - הערך/המידע בצומת
int height	אורך המסלול הכי ארוך מצומת לעלה
int size	מספר הצמתים בתת העץ ששורשו בצומת
AVLNode parent	האב של הצומת
AVLNode left	הבן השמאלי של הצומת
AVLNode right	הבן הימני של הצומת

הבנאים:

```
public AVLNode() {
```

בנאי ריק שתפקידו לבנות צומת וירטואלית.
המפתח והגובה יאותחלו לערך 1-.
הגודל יאותחל לערך 0.
שאר השדות יאותחלו לערך null.

```
public AVLNode(int key, String value) {
```

הבנאי זורק AssertionError במידה והערך של המפתח שלילי.
לאחר מכן מעדכן את השדות key, value בקלט.
מתבצעת השמה של צמתים וירטואליים בילדים.
בהורה מתבצעת השמה ל-null.
הגובה מעודכן ל0, והגודל ל1.

פונקציות:

שם	פרמטרים	טיפוס החזרה	תיאור
getKey		int	מחזירה את המפתח של הצומת
getValue		String	מחזירה את הערך של הצומת (מחרוזת)
setValue	String value	void	מעדכנת את הערך של הצומת
setLeft	IAVLNode node	AVLNode	מעדכנת את הבן השמאלי של הצומת וגם מעדכנת את האבא של הבן השמאלי החדש.
getLeft		AVLNode	מחזירה מצביע לבן השמאלי של הצומת
setRight	IAVLNode node	AVLNode	מעדכן את הבן הימני של הצומת וגם מעדכנת את האבא של הבן הימני החדש.
getRight		AVLNode	מחזירה מצביע לבן הימני של הצומת
setParent	IAVLNode node	AVLNode	מעדכנת את האב של הצומת
getParent		AVLNode	מחזירה מצביע לאב של הצומת
setHeight	height int	int	מעדכנת את הגובה של הצומת
getHeight		int	מחזירה את הגובה של הצומת
setSize	int size	int	מעדכנת את גודל תת העץ ששורשו בצומת
getSize		int	מחזירה את גודל תת העץ ששורשו בצומת
isRealNode		Boolean	מחזירה שקר אם הצומת וירטואלית, אחרת אמת.

מחלקת AVLTree

תיאור כללי: מחלקה המממשת עץ AVL. כמו כן, העץ הוא Finger Tree, כלומר מחזיק מצביעים למינימום ולמקסימום.

שדות:

שם	תיאור
AVLNode root	המפתח של הצומת בעץ
AVLNode min	הצומת בעלת המפתח המינימלי
AVLNode max	הצומת בעלת המפתח המקסימלי

הבנאים:

```
public AVLTree() {
```

בנאי ריק המאתחל עץ ריק.
השורש מאותחל לצומת וירטואלית. המינימום והמקסימום מאותחלים ל- null.

```
public AVLTree(AVLNode root) {
```

בנאי המקבל צומת מאתחל עץ ששורשו, המינימום שלו והמקסימום שלו מתעדכנים בהתאם לתת עץ ששורשו מתחיל בצומת שקיבלנו בעזרת הפונק' minNode,maxNode.

פונקציות שהתבקשנו לממש:

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
1	empty		boolean	מחזירה אמת אם העץ ריק, אחרת שקר	$O(1)$

תנאי קדם: אין.

אופן פעולה: אם השורש הוא null מחזיר שקר. קוראת לפונקציה isRealNode כדי לבדוק אם השורש הוא צומת וירטואלי.

ניתוח סיבוכיות: isRealNode היא מסיבוכיות $O(1)$ מכיוון שהיא מכילה בדיקה אחת בלבד, לכן גם הפונקציה שלנו היא מסיבוכיות זו.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
2	search	int k	String	הפונקציה מחזירה את הערך עבור המפתח K. אם אין מפתח כזה בעץ נחזיר null	$O(\log n)$

תנאי קדם: אין

אופן פעולה: הפונקציה משמשת כמעטפת לפונקציה הרקורסיבית searchRec אשר מקבלת שורש של תת עץ, ומפתח אשר מבצעת חיפוש בינארי בעץ ומחזירה null אם הצומת לא נמצאה.

ניתוח סיבוכיות: אנחנו מבצעים חיפוש בינארי ולכן סיבוכיות הזמן היא $O(\log n)$

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
3	insert	int k String s	int	הכנסת איבר בעל ערך s ומפתח k לעץ, אם המפתח לא קיים הפונקציה מחזירה את מספר פעולות האיזון שנדרשו בסה"כ בשלב תיקון העץ על מנת להשלים אתהפעולה. אם קיים איבר בעל מפתח k בעץ הפונקציה מחזירה 1- ולא מתבצעת הכנסה.	$O(\log\ n)$

תנאי קדם: אין

אופן פעולה: יוצרים צומת חדשה עם הערכים הרצויים וקוראים לפונק עזר מספר 0 שמקבלת צומת ומחזירה את מספר פעולות האיזון שנדרשו בסה"כ בשלב תיקון העץ על מנת להשלים את הפעולה. אם קיים איבר בעל מפתח k בעץ הפונקציה מחזירה 1- ולא מתבצעת הכנסה.

ניתוח סיבוכיות: בניית הצומת היא $O(1)$, פונק העזר היא $O(\log n)$. לכן סה"כ הסיבוכיות היא $O(\log n)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
4	delete	int k	int	מחיקת איבר בעל המפתח k בעץ, אם הוא קיים. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו בסה"כ בשלב תיקון העץ על מנת להשלים את הפעולה.	$O(\log n)$

				אם לא קיים איבר בעל המפתח k בעץ הפונקציה מחזירה 1-.	

<u>תנאי קדם:</u> אין

<u>אופן פעולה:</u> תחילה מתבצעת בדיקה האם הצומת קיים בעזרת פונקציית עזר מספר 1 treePosition בסיבוכיות $O(\log n)$. האם הצומת לא קיימת מוחזר 1-. אחרת, מתבצעת מחיקה של הצומת, איזונה וספירת מספר פעולות האיזון באמצעות פונקציית העזר מספר 3 *rebalance* בסיבוכיות $O(\log n)$. כמו כן מתבצעות מספר בדיקות בסיבוכיות קבועה כדי לבדוק אם אנו נמצאים באחד המקרים שנלמדו בהרצאה אשר לא דורשים איזונים שנעשים בשיטת ה-*BF* ומחזירים עבורם את מספר האיזונים שנדרשו לפי הנלמד בהרצאה.
כמו כן, מתבצעת תחזוקה של השדות *min, max* ע"י בדיקה האם השדה אותו מחקנו הוא המינימום או המקסימום. לאחר מכן מתבצעת גישה למינימום/מקסימום החדש ע"י פונקציית העזר *minNode/maxNode* בסיבוכיות $O(\log n)$ ועדכון השדות.

<u>ניתוח סיבוכיות:</u> כלל הפעולות מתבצעות אחת לאחר השנייה. לכן סיבוכיות הזמן היא:

$$O(\log n) + O(1) + O(\log n) + O(\log n) = O(\log n)$$

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
5	min		String	מחזירה את ערך (info) האיבר בעץ בעל המפתח המינימלי, או null בעץ ריק.	$O(1)$

<u>תנאי קדם:</u> אין

<u>אופן פעולה:</u> בדיקה אם העץ ריק באמצעות פונקציה isRealNode, ואם הוא אכן ריק מוחזר null. אחרת מחזירים את ערך שדה min .

<u>ניתוח סיבוכיות:</u> כלל הפעולות מתבצעים בסיבוכיות קבועה $O(1)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
6	max		String	מחזירה את ערך (info) האיבר בעץ בעל המפתח המקסימלי, או null בעץ ריק.	$O(1)$

באופן דומה ל- min.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
7	keysToArray		int[ ]	הפונקציה מחזירה מערך ממוין המכיל את כל המפתחות בעץ, או מערך ריק אם העץ ריק.	$O(n)$

<u>תנאי קדם:</u> אין

<u>אופן פעולה:</u> הפונקציה משמשת מעטפת לפונקציה הרקורסיבית keysToArrayRec. מאותחל מערך ריק ואז מתבצעת קריאה לפונקציה הרקורסיבית עם המערך הריק, מערך בגודל 1 שמכיל את האינדקס הבא אליו יש להכניס במערך. הפונקצייה keysToArrayRec מבצעת סריקת in order על העץ ומוסיפה בכל שלב את המפתח הנוכחי למערך.

<u>ניתוח סיבוכיות:</u> מתבצעת סריקת in order כאשר בכל צומת מתבצעת עבודה קבועה של הוספה לסוף מערך וקידום אינדקס. לכן הסיבוכיות הכוללת היא: $O(n)$ כפי שנלמד בכיתה.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
8	infoToArray		String[ ]	הפונקציה מחזירה מערך מחרוזות המכיל את כל המחרוזות בעץ, ממוינות עפ"י סדר המפתחות.	$O(n)$

באופן דומה ל-keysToArray אך כאן מוסיפים למערך את הערך של הצומת ולא את המפתח.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
9	size		int	הפונקציה מחזירה את מספר האיברים בעץ.	$O(1)$

<u>תנאי קדם:</u> אין

<u>אופן פעולה:</u> הפונקציה קוראת לפוקנציית getSize של השורש, המחזירה את הערך השמור בשדה הגודל שלו.

<u>ניתוח סיבוכיות:</u> מתבצעת גישה לשדות וחישובים בסיבוכיות קבועה : $O(1)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
10	split	int x	AVLTree[ ]	הפונקציה מקבלת מפתח x שנמצא בעץ. על הפונקציה להפריד את העץ ל-2 עציAVL כאשר המפתחות של האחד גדולים מ-x ושל השני קטנים מ-x.	$O(\log n)$

<u>תנאי קדם:</u> x קיים בעץ, העץ לא ריק.

<u>אופן פעולה:</u> נאתחל שני עצים: אחד שיכיל את כל המפתחות שגדולים מ-x השני שמכיל את כל המפתחות שקטנים מ-x. תחילה נשים בעץ הגדולים את תת העץ הימני של x ובקטנים את תת העץ השמאלי של x. לאחר מכן נעלה למעלה כלפי השורש כל פעם כאשר כל פעם אנחנו בודקים מאיזה כיוון עלינו. אם עלינו מימין, משמע שעלינו מתת עץ שערכיו גדולים מ-x ולכן נוסיף אותו לעץ המתאים. אחרת נוסיף את תת העץ לעץ שערכיו קטנים מ-x.

כמו כן, במהלך ביצוע פעולות ה-*join* של תתי העצים, מעודכנים המקסימום והמינימום של תתי העצים.

ניתוח סיבוכיות: כפי שנלמד בכיתה, סיבוכיות זמן הריצה של הפונקציה היא $O(\log n)$

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
11	join	IAVLNode x AVLTree t	int	הפונקציה מקבלת צומת x ועץ t שכל ה-keys שלהם קטנים, או שכולם גדולים,מה-keys של העץ הנוכחי שביחס אליו קראנו ל-join. על הפונקציה לאחד את x,t לעץ הנוכחי כפי שמומש בהרצאה. הפונקציה מחזירה את עלות של הפעולה (הפרשי גבהי העצים + 1)	$O(\log n)$

תנאי קדם: המפתח של x מקיים:
$keys(t) < x < keys(this)\ or\ keys(this) < x < keys(t)$.
כמו כן, t יכול להיות ריק.

אופן פעולה: אם t ריק נכניס את x לעץ ונסיים.
אחרת, נבדוק איזה עץ מכיל מפתחות גדולים יותר מ-x, ומי מכיל מפתחות קטנים יותר מ-x. כמו כן, נבדוק איזה עץ גבוה יותר. נסביר כעת את אופן הפעולה במידה והעץ עם המפתחות הגדולים יותר גבוה יותר (המקרה השני סימטרי). נרוץ על הדופן השמאלית של העץ עד שנמצא את הצומת הראשון (נסמנו b) שגובהו כגובה העץ הנמוך יותר או נמוך ב-1. נחבר אותו כבן ימני של x ואת שורשו של העץ הנמוך היותר כבנו השמאלי. ההורה של x יהיה האב של b. כעת נבצע איזון החל מ-x.
בנוסף, אנחנו מבצעים תחזוקה לשדות המינימום והמקסימום ע"י השמה של המקסימום של העץ של הגדולים מ-x למקסימום, ובאופן דומה השמה של המינימום למינימום של עץ הקטנים מ-x.

ניתוח סיבוכיות: כפי שנלמד בכיתה, סיבוכיות זמן הריצה של join היא $O(\log n)$. תחזוקת שדות המקסימום והמינימום דורשת גישה לשדות בלבד ולכן היא בזמן קבוע. כלומר סה"כ זמן הריצה הוא $O(\log n)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
12	getRoot		AVLNode	הפונקציה מחזירה את השורש של העץ.	$O(1)$

תנאי קדם: אין

אופן פעולה: מחזירה null אם העץ ריק, אחרת מחזירה את השורש.

ניתוח סיבוכיות: זמן קבוע: $O(1)$.

פונקציות עזר:

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות

0	insert	AVLNode x	int	הכנסת הצומת לעץ, אם המפתח לא קיים הפונקציה מחזירה את מספר פעולות האיזון שנדרשו בסה"כ בשלב תיקון העץ על מנת להשלים את הפעולה. אם קיים איבר בעל מפתח k בעץ הפונקציה מחזירה 1- ולא מתבצעת הכנסה.	$O(\log\ n)$

<u>**תנאי קדם:**</u> אין

<u>**אופן פעולה:**</u> אם העץ ריק מעדכנים את הצומת להיות השורש, המינימום והמקסימום של העץ, בנוסף מחזירים אפס (סיבוכיות קבועה). אחרת אנחנו קוראים לפונקציית עזר מספר 4 treeInsert (סיבוכיות $O(\log n)$), המנסה להכניס את הצומת לעץ, ובמידה שלא הצליחה מחזירה שקר. אם *treeInsert* החזירה שקר מחזירים 1-. אחרת, מבצעים איזון לעץ החל מהצומת שהוכנסה כפי שנלמד בהרצאה ע"י פונקציית העזר מספר 3 *rebalance* (סיבוכיות $O(\log n)$). כמו כן סופרים את מספר האיזונים שבוצעו ומחזירים אותם. במהלך פעולות האיזון מתבצעות השוואות וגישה לשדות בסיבוכיות $O(1)$. כמו כן, מתבצעת בדיקה אם הכנסנו ערך גדול יותר מהמקסימום או קטן יותר מהמינימום ועדכון שדות בהתאם בזמן קבוע.

<u>**ניתוח סיבוכיות:**</u> כלל הפעולות מתבצעות אחת לאחר השנייה. לכן סיבוכיות הזמן היא:

$$O(1) + O(\log n) + O(\log n) +\ O(1) = O(\log n)$$

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
1	treePosition	int k AVLNode x	AVLNode x	מחפש את הצומת בעץ ומחזיר את הצומת האחרון בו נתקלנו בחיפוש. אם העץ ריק יוחזר צומת פיקטיבי.	$O(\log n)$

<u>**תנאי קדם:**</u> אין

<u>**אופן פעולה:**</u> מתבצע חיפוש באופן דומה לחיפוש בעץ, אך הפעם שומרים משתנה temp שמכיל את הצומת הקודם שבדקנו. אם המפתח שחיפשנו לא נמצא מחזירים את temp שהוא למעשה הצומת אשר אמור להיות האב העתידי של – x.

<u>**ניתוח סיבוכיות:**</u> בדומה לחיפוש בעץ במקרה הגרוע נרד לכל אורך העץ, ולכן הסיבוכיות היא $O(\log n)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
2	rotateRight	AVLNode x	void	מבצעת סיבוב קשת ימינה	$O(1)$

				באופן שנלמד בהרצאה	

<u>תנאי קדם:</u> x אינו null ולא וירטואלי.

<u>אופן פעולה:</u> הפונקציה מחליפה את המצביעים של הצומת עם הבן השמאלי שלו (אם הוא קיים) ומעדכנת את הגבהים והגדלים שלהם.

<u>ניתוח סיבוכיות:</u> החלפת המצביעים ועדכון הגבהים כוללים פעולות אריתמטיות והשמות בסיבוכיות $O(1)$.

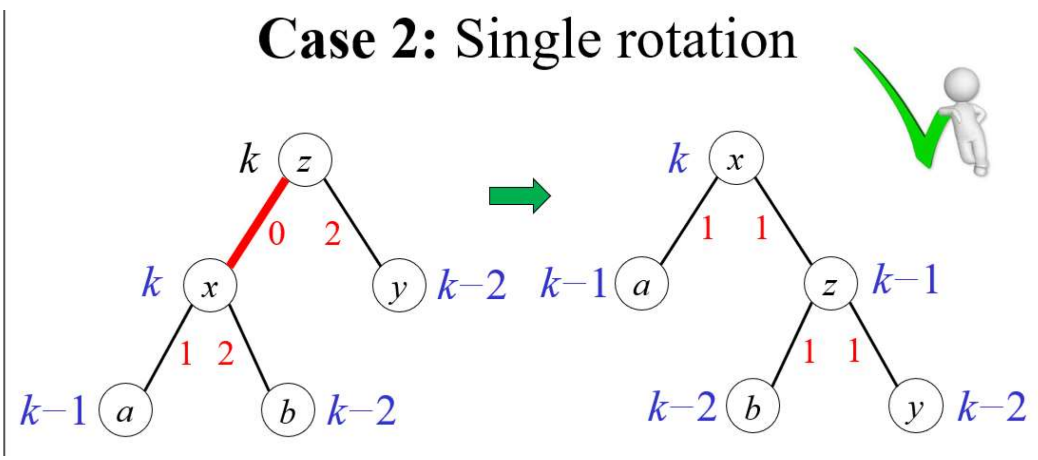
RotateLeft מנהגת באופן דומה.**

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
3	rebalance	AVLNode x	int	מבצעת איזון לעץ החל מהצומת x כפי שנלמד בהרצאה. מחזירה את מספר פעולות האיזון שביצענו.	$O(\log n)$

<u>תנאי קדם:</u> אין

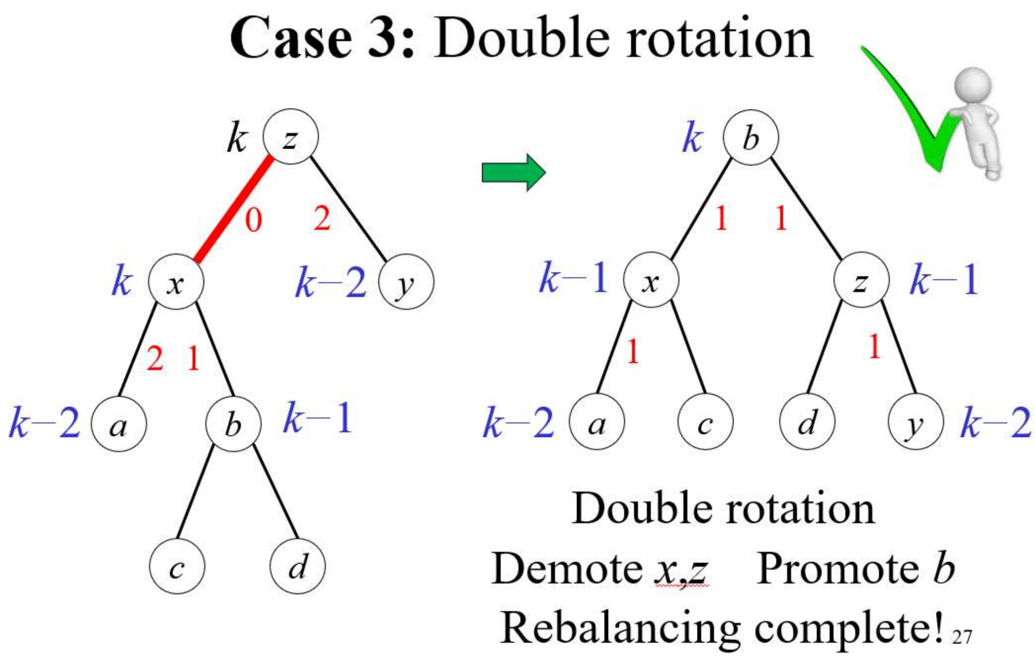
<u>אופן פעולה:</u> *פעולות ה- promote, demote הקשורות לסיבובים מבוצעות בפונקציות הסיבוב.

מתבצעת בדיקה של ה-Balance Factor ואז מתבצע איזון לעץ לפי המקרים הבאים (המקרים שבהם אנו מטפלים הם מקרים הדורשים סיבובים ו-promote/demote שנוצרו כתוצאה מסיבובים. שאר המקרים מטופלים בפונקציית insert/delete בהתאם למקרה)

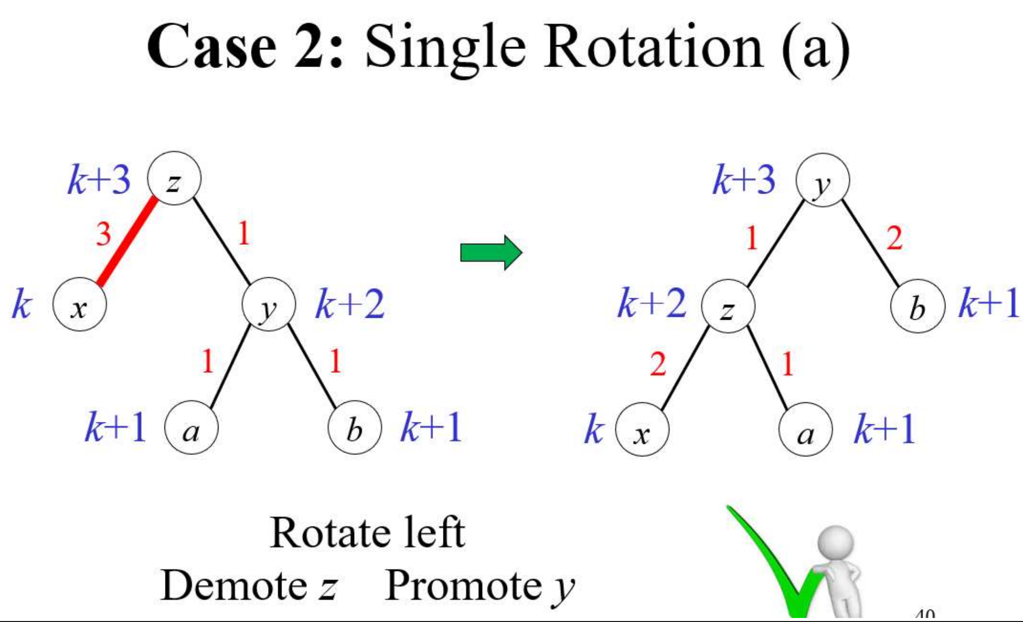


- במקרה זה BF של Z הוא 2, ו-BF של x הוא 1. נבצע סיבוב אחד ימינה ונחזיר 2 (פעולת סיבוב + demote ל-Z).

במקרה הסימטרי ה-BF של השורש הוא 2- ו-BF של הבן הימני של השורש הוא 1-.

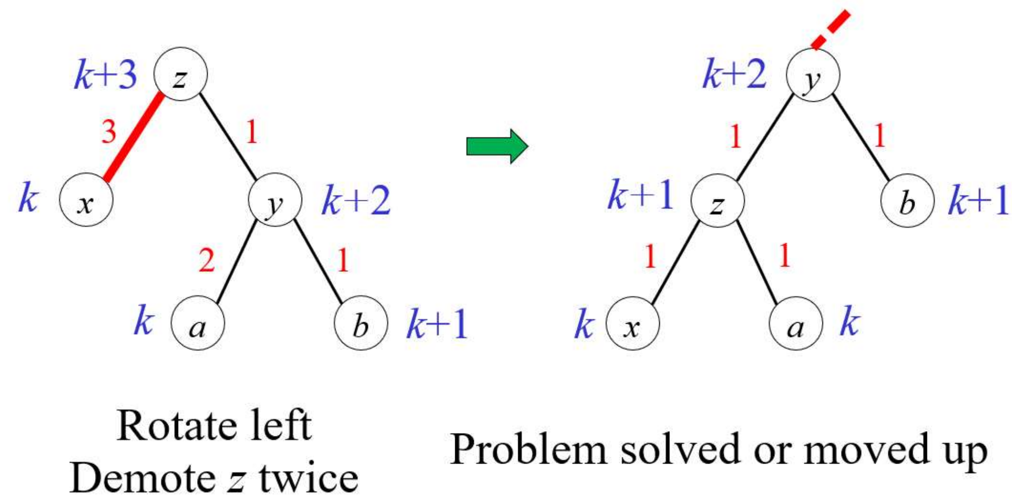


- ה-BF של Z הוא 2, ושל X ‎-1. נבצע סיבוב שמאלה על הקשת בין X ל-Z, ואז סיבוב ימינה על Z. נחזיר 5 עבור הפעולות הרשומות למעלה.
המקרה הסימטרי: BF של השורש הוא ‎-2, ושל הבן הימני שלו הוא 1.



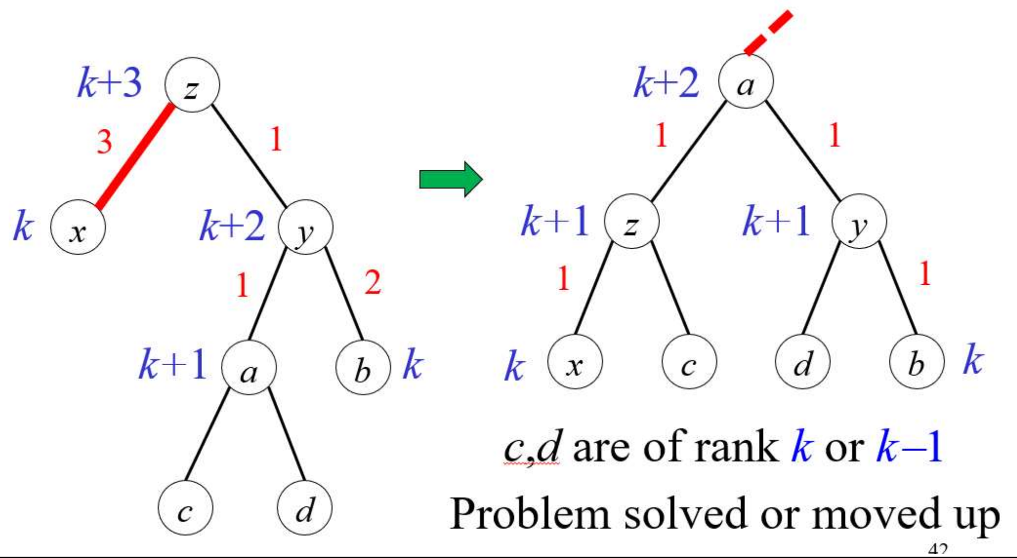
- BF של Z הוא ‎-2, ושל y הוא 0. נבצע פעולת סיבוב שמאלה.
נחזיר 3 עבור הסיבוב, ושינויי הדרגה.
במקרה הסימטרי ה-BF של השורש שווה 2, ושל הבן השמאלי שלו 0.

Case 3: Single Rotation (b)



- BF של Z הוא 2- ושל הבן הימני שלו הוא 1-. נבצע סיבוב שמאלה.
נחזיר 2 עבור הסיבוב וה-demote הכפול.
במקרה הסימטרי BF של השורש הוא 2, ושל הבן השמאלי שלו 1.

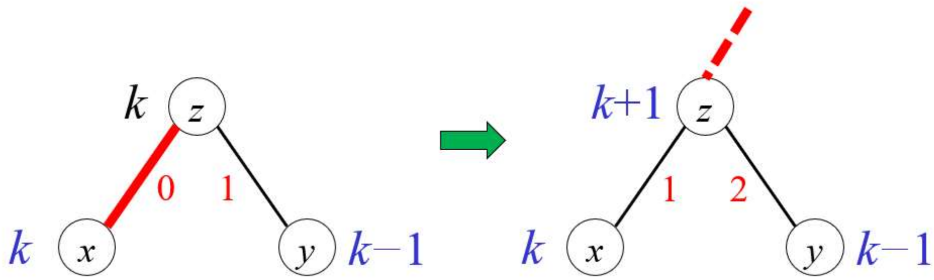
Case 4: Double Rotation



- BF של Z הוא 2-, ושל y הוא 1. במקרה זה נבצע סיבוב ימינה על הקשת yz ואז סיבוב שמאלה על הקשת za. נחזיר 5 עבור שני הסיבובים ושלושה שינויי דרגה.
במקרה הסימטרי BF של השורש הוא 2, ושל הבן השמאלי שלו הוא 1-.

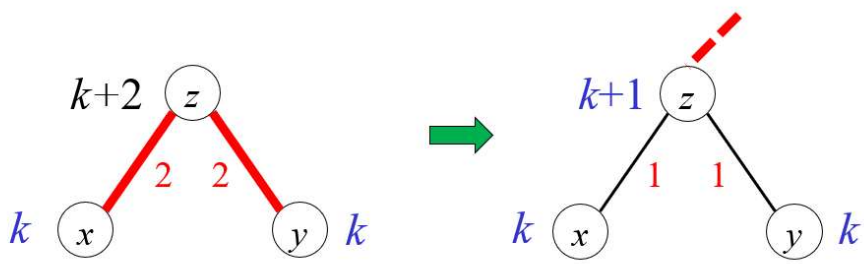
<u>מטופל בפונקציה insert:</u>

Case 1: Promote



:delete מטופל בפונקציה

Case 1: Demote



ניתוח סיבוכיות: כפי שנלמד בכיתה במקרה הגרוע מתבצע מעבר על כל גובה העץ כאשר בכל רמה מתבצעות פעולות העלאה בדרגה, הורדה בדרגה וסיבובים בעלות $O(1)$. לכן, סה"כ סיבוכיות זמן הריצה תהיה $O(\log n)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
4	treeInsert	AVLNode toInsert	boolean	מבצעת הכנסה פיזית של הצומת לעץ במידה והוא לא קיים בעץ ומחזירה אמ"מ הצומת הוכנס.	$O(\log n)$

<u>תנאי קדם:</u> אין

<u>אופן פעולה:</u> מתבצעת קריאה ל-treePostion כדי למצוא את האב של הצומת אותו יש להכניס. אם הערך המוחזר בעל מפתח זהה לערך שאותו רצינו להכניס מחזירים שקר. אחרת, בודקים את יחס הסדר בין הצמתים ולפי התוצאה בוחרים אם הצומת המוכנס יהיה בין ימני או שמאלי.

<u>ניתוח סיבוכיות:</u> treePosition היא בסיבוכיות לוגריתמית. שאר הפעולות בסיבוכיות קבועה. לכן סה"כ סיבוכיות זמן הריצה היא $O(\log n)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
5	isRightChild	AVLNode parent AVLNode child	boolean	מחזירה אמת אמ"מ child הוא הבן הימני של parent.	$O(1)$

<u>תנאי קדם:</u> אין

<u>אופן פעולה:</u> אם ההורה null מחזירים אמת (כדי למנוע שגיאת זמן ריצה). אחרת מבצעים בדיקה אם אכן הילד בן ימני של ההורה.

<u>ניתוח סיבוכיות:</u> מתבצעות פעולות השוואה בלבד ולכן הסיבוכיות היא $O(1)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
6	successor	AVLNode x	AVLNode	מחזיר את ה successor (הצומת בעלת המפתח המינימלי שגדול מהמפתח של x). במידה ו-x הוא המקסימום הפונקציה תחזיר צומת וירטואלית.	$O(\log n)$

<u>תנאי קדם:</u> x הינו צומת אמיתי ולא null.

<u>אופן פעולה:</u> אם יש ל-X בן ימני נרד לתת העץ הימני ונמצא את המינימום. אחרת, נעלה אב אחד למעלה באופן איטרטיבי עד כאשר נגיע לצומת שעלינו אליו מבן שמאלי.

<u>ניתוח סיבוכיות:</u> לכל היותר נעלה את כל גובה העץ, בכל רמה יש לנו השוואות וגישה לשדות בלבד, ולכן הסיבוכיות תהיה $O(\log n)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות

7	updateSizes	AVLNode x	void	מעדכן את גודל כל האבות של x עד השורש.	$O(\log n)$

<u>תנאי קדם:</u> X אינו צומת וירטואלי או null.

<u>אופן פעולה:</u> מתחילים ב-x וקוראים באופן איטרטיבי לאב, כאשר בכל שלב מעדכנים את גודל הצומת לפי סכום הגדלים ילדיו + 1.

<u>ניתוח סיבוכיות:</u> בכל רמה מבצעים פעולות אריתמטיות, גישה לשדות והשמות ב-$O(1)$. לכל היותר נעבור על כל גובה העץ ולכן הסיבוכיות היא $O(\log n)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
8	updateHeights	AVLNode x	void	מעדכן את גובה הצומת.	$O(\log n)$

<u>תנאי קדם:</u> X אינו צומת וירטואלי או null.

<u>אופן פעולה:</u> באופן דומה ל- updateSizes, אך הפעם מעדכנים את הגובה להיות למקסימום בין גבהי תתי העצים + 1.

<u>ניתוח סיבוכיות:</u> $O(\log n)$ בדומה ל-updateSizes.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
9	getBF	AVLNode x	int	מחשב את ה- balance factor של הצומת (הגדרת BF נמצאת בתחילת הקובץ)	$O(1)$

<u>תנאי קדם:</u> X אינו צומת וירטואלי או null.

<u>אופן פעולה:</u> מחזירה את הפרש הגבהים של תת העץ השמאלי ותת העץ הימני.

<u>ניתוח סיבוכיות:</u> גישה לשדות ופעולות אריתמטיות ב-$O(1)$.

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
10	minNode	AVLNode x	AVLNode	מחזירה מצביע לצומת בעל המפתח המינימלי בעץ ששורשו x.	$O(\log n)$

<u>תנאי קדם:</u> X אינו צומת וירטואלי או null.

<u>אופן פעולה:</u> מתחילים בשורש ויורדים שמאלה עד לעלה השמאלי ביותר וזהו המינימום. זה המינימום בגלל תכונת עץ החיפוש אשר אומרת כי הבן השמאלי של צומת קטן ממנו.

<u>ניתוח סיבוכיות:</u> נרד לאורך גובה כל העץ ונבצע עבודה קבועה בכל רמה, לכן סיבוכיות זמן הריצה היא $O(\log n)$

מס"ד	שם	פרמטרים	טיפוס החזרה	תיאור	סיבוכיות
11	maxNode	AVLNode x	AVLNode	מחזירה מצביע לצומת בעל המפתח המקסימלי בעץ ששורשו x.	$O(\log n)$

באופן דומה ל-*minNode* אך הפעם נרד ימינה לעלה הימני ביותר.

חלק ניסויי/תאורטי

שאלה 1:

א.

מספר סידורי i	מספר חילופים במערך ממוין-הפוך	עלות חיפושים במיון AVL עבור מערך ממוין-הפוך	מספר חילופים במערך מסודר אקראית	עלות החיפושים במיון AVL עבור מערך מסודר אקראי
1	1999000	38884	978661	33648
2	7998000	85764	3965238	72805
3	31996000	187524	15981443	169703
4	127992000	407044	63789026	369651
5	511984000	878084	256979891	791776

ב. **מספר החילופים:**

עבור כל מספר k כך ש-k נמצא במערך מתקיים כי כל המספרים הקטנים ממנו נמצאים מימינו, ואין מימינו מספרים גדולים ממנו. ישנם $k-1$ מספרים קטנים ממנו, ולכן מספר החילופים עבור כל k הוא $k-1$. לכן מספר החילופים עבור כל המערך יהיה:

$$\sum_{k=1}^{n} k-1 = \frac{n(n-1)}{2} = \frac{1}{2}(n^2-1) = \Theta(n^2)$$

כאשר השוויון הראשון משמאל הוא לפי הנוסחה לסכום סדרה חשבונית.

עלות חיפוש:

בכל הכנסה אנו מכניסים איבר שקטן מהמינימום הנוכחי של העץ. לכן עבור המפתח ה-k שאנו מכניסים עולים ויורדים את גובה העץ הנוכחי פעמיים. לפי הנלמד בכיתה, גובה עץ *AVL* הוא לוגריתמי במספר הצמתים. נסכום את כלל הפעולות:

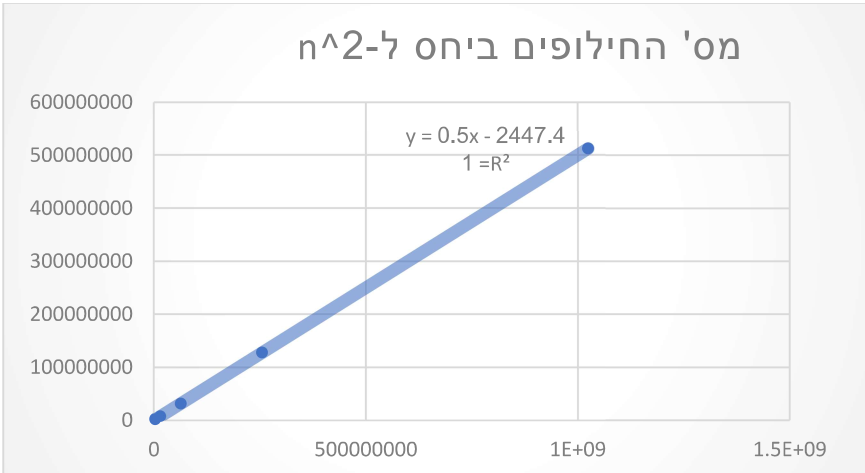
$$\sum_{k=1}^{n} 2\log k = 2\sum_{k=1}^{n} \log k = \ 2\,(\log n + \log n-1 + \dots + \log 1) = 2\log(n \cdot (n-1)\ldots 1) = 2\log(n!) = \Theta(n\log n)$$

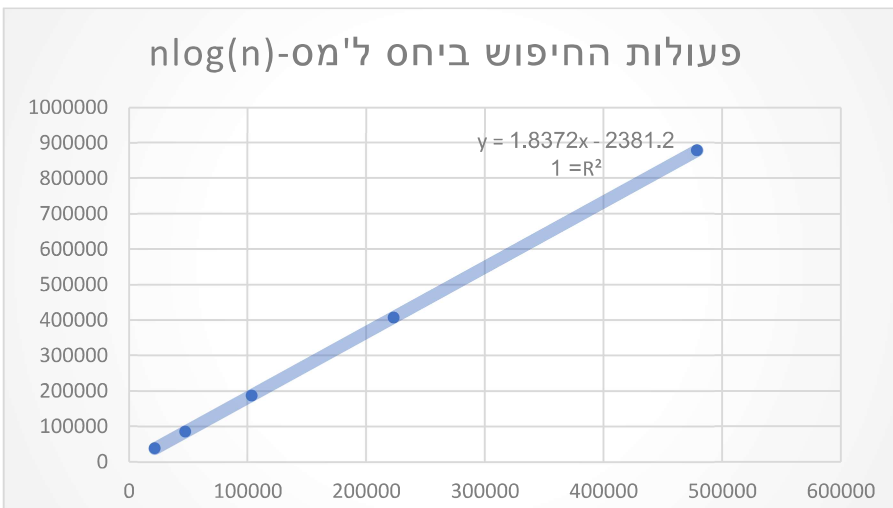
כאשר המעבר האחרון נלמד בכיתה.

ג.

i	n	n^2	nlog(n)	מס חילופים	עלות חיפוש
1	2000	4000000	21931.57	1999000	38884
2	4000	16000000	47863.14	7998000	85764
3	8000	64000000	103726.3	31996000	187524

407044	127992000	223452.5	2.56E+08	16000	4
878084	511984000	478905.1	1.02E+09	32000	5





כפי שניתן לראות בקווי המגמה קיימת התאמה בין הנתונים בסעיף א' לניתוח בסעיף ב' משום שמדד ה- R^2 הינו 1.

ד. נסמן: מספר האיברים הגדולים מהאיבר ה-i הנמצאים משמאלו במערך =: h_i. לפי ההגדרה מתקיים כי $h = h_1 + h_2 + \cdots + h_n$. למעשה $h_i + 1$ הוא ה Rank – שלי בעת הכנסתי (האינדקס שלי במערך הממוין עד כה). כלומר אם $h_i = 2$ אז בעת הכנסתי היו שני איברים גדולים ממני ואני הייתי האיבר השלישי בגודלו. למעשה נבצע פעולת חיפוש של איבר בשיטת Finger Tree , וכפי שנלמד בכיתה סיבוכיות זמן הריצה של פעולה זו היא $O(\log(h_i + 1) + 1)$ מכיוון ש: (Rank(i) = $h_i + 1$).

הסבר לסיבוכיות זו: נתחיל את החיפוש בצומת המקסימלית ונעלה למעלה עד שנגיע לצומת הכי גבוהה שהערך שלה גדול שווה לX. לאחר מכן נחפש בעזרת חיפוש בינארי בתת עץ השמאלי את הצומת X. בזמן העלייה נעלה לכל היותר $c \cdot \log(h_i + 1)$ צמתים ובירידה נרד לכל היותר $c \cdot \log(h_i + 1) + 1$ צמתים עקב מבנה עץ הAVL.

כעת, נסכום את כל עלות כל החיפושים בהכנסות ונקבל:

$$\sum_{i=1}^{n} O(\log(h_i+1)+1) = O(\sum_{i=1}^{n}\log(h_i+1))+1 = n+\ O(\log(\prod_{i=1}^{n}(h_i+1)))$$
$$= n+\ O\left(nlog\left(\sqrt[n]{\prod_{i=1}^{n}(h_i+1)}\right)\right) \leq$$
$$\leq n+\ O\left(n\log\left(\frac{h+n}{n}\right)\right) = n+\ O(n\log(\frac{h}{n}+1))$$
כאשר המעבר האחרון על פי א"ש הממוצעים.
נשים לב כי גם במקרה הטוב ביותר (הכנסת מערך ממוין) נבצע n פעולות. לכן סה"כ סיבוכיות זמן הריצה היא :

$$O(n+\mathrm{n}\log(\frac{h}{n}+1))$$

ה. התבקשנו בסעיף ד' לתת חסם עליון בלבד מכיוון שלא ניתן לקבוע חסם Θ אחיד לכל סדרה של הכנסות. נוכיח: נבחר $h=\frac{n}{2}\log n$. כעת נראה כי יש שתי חלוקות של h_i עבורן נקבל חסמי Θ שונים כאשר n שואף לאינסוף.

הסדרה הראשונה: תחילה נכניס את $\frac{n}{2}-\log n$ האיברים הקטנים ביותר בסדר ממוין. לאחר מכן נכניס את $\frac{n}{2}$ האיברים הגדולים ביותר. לסיום, נכניס את $\log n$ האיברים שנותרו. ל- $n-\log n$ האיברים הראשונים שהכנסנו מתקיים $h_i=0$. לאיברים האחרים מתקיים:

$$h_{n-\log n+1}=\cdots=h_n=\frac{n}{2}$$

לכן מתקיים:

$$\log(\prod_{i=1}^{n}(h_i+1))+n=\log\left(\left(\frac{n}{2}+1\right)^{\log n}\right)+n=\log n\log\left(\frac{n}{2}+1\right)+n=\Theta(\mathrm{n})$$

כאשר מכפלת הלוגריתמים נבלעת בסיבוכיות מכיוון שהיא קטנה אסימפטומטית מסיבוכיות לינארית.

הסדרה השנייה: תחילה נכניס $\frac{n}{2}-\log n$ האיברים הקטנים בסדר ממוין. לאחר מכן נכניס את שאר האיברים. לכן עבור $\frac{n}{2}$ האיברים הראשונים נקבל כי $h_i=0$, ולכל שאר האיברים נקבל כי $h_i=\log n$. לכן מתקיים:

$$\log(\prod_{i=1}^{n}(h_i+1))+n=\log\left((\log n+1)^{\frac{n}{2}}\right)+n=\frac{n}{2}\log(\log n+1)+n=\Theta(\mathrm{n}\log\log n)$$

כלומר מצאנו שתי סדרות עם חסמי Θ שונים, כנדרש.

שאלה 2:

א.

מס"ד	עלות join **ממוצע** עבור split **אקראי**	עלות join **מקסימלי** עבור split **אקראי**	עלות join **ממוצע** עבור split של **איבר** מקסימלי בתת העץ השמאלי	עלות join **מקסימלי** עבור split של **איבר** מקסימלי בתת העץ השמאלי
1	2.375	4	2.4545	13
2	2.6363	5	2.6666	14
3	2.7272	10	2.6666	16
4	2.7692	5	2.8461	17
5	2.4666	8	2.6428	18
6	2.375	7	2.9285	19
7	2.4444	7	2.4705	20
8	2.3592	5	2.7058	21
9	2.4375	6	2.4	22

24	2.7894	5	2.5263	10

ב. אנו מתחילים מהאיבר המקסימלי ועולים שמאלה תוך כדי ביצוע פעולות join לתתי העץ השמאליים. מכיוון שמדובר בעץ AVL מאוזן נסמן את תת גבהי העץ הימני והשמאלי ב- Tr, Tl בהתאמה.מתקיים:

$$|Tr - Tl| \leq 1$$

כלומר, עלות כל join היא 1 או 2. בעלייה בתת העץ השמאלי נבצע $\log n$ פעולות. לאחר מכן, כשנגיע לשורש נבצע join בין עץ רק לכל תת העץ הימני של העץ המקורי ופעולה זו תעלה לנו $\log n$.

לכן נקבל שמתקיים:

$$2 = \frac{\log n + \log n}{\log n} \leq Avergae\ Cost \leq \frac{2\log n + \log n}{\log n} = 3$$

קיבלנו כי העלות הממוצעת היא $O(1)$.

כעת ננתח את הסיבוכיות עבור המקרה האקראי:

נבצע את הפעולה split על צומת אקראי x שגובהו h.

סך פעולות הsplit שנבצע= log(n)- h. זאת עקב מבנה עץ הAVL .

נסמן את גובה תתי העצים שקטנים מא ב $h_1, h_2, \dots h_i$. ואת גובה תתי העצים שגדולים מא ב $H_1, H_2, \dots H_K$.

כאשר $H_m \geq H_n, h_m \geq h_n$ עבור $n < m$

נסמן את סכום פעולות הjoin על תתי העצים שקטנים מא בS_1 ואת סכום פעולות הjoin על תתי העצים שגדולים מא בS_2 עלות פעולות הjoin הכוללת היא $S = S_2 + S_1$.

$$S_1 \leq (\cancel{h_2} - h_1 + 1) + (\cancel{h_3} - \cancel{h_2} + 1) + (\cancel{h_4} - \cancel{h_3} + 1) \dots + (\cancel{h_{i-1}} - \cancel{h_{i-2}} + 1) + (h_i - \cancel{h_{i-1}} + 1)$$

$$S_2 \leq (\cancel{H_2} - H_1 + 1) + (\cancel{H_3} - \cancel{H_2} + 1) + (\cancel{H_4} - \cancel{H_3} + 1) \dots + (\cancel{H_{i-1}} - \cancel{H_{i-2}} + 1) + (H_i - \cancel{H_{i-1}} + 1)$$

$$\rightarrow S \leq h_i + H_K - h_1 - H_1 + [\log(n) - h] \leq h_i + H_K + [\log(n) - h] - 2(h-2) \leq 2(\log(n) + 1) + [\log(n) - h] - 2(h-2) = 3\log(n) - 3h + c$$

נסמן את ממוצע פעולת ה join ב A.

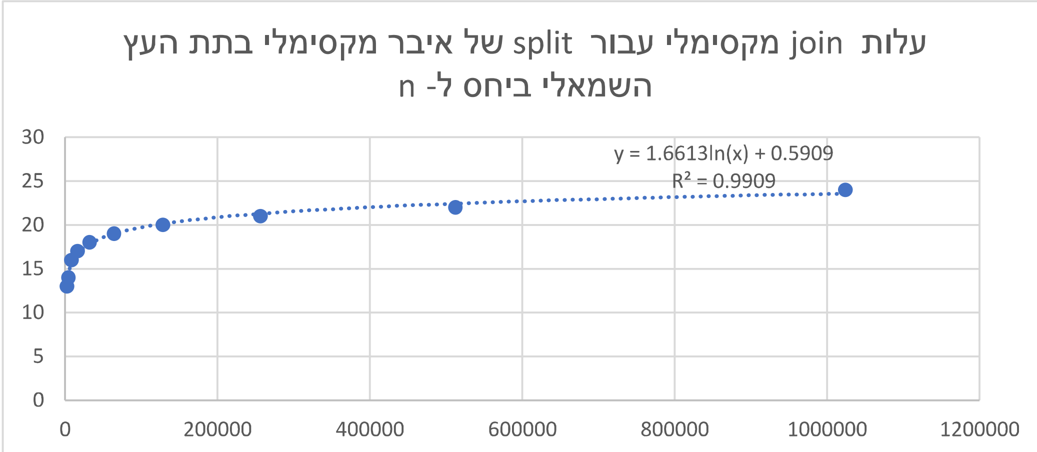
$$\mathrm{A} = \frac{\mathrm{S}}{\log(\mathrm{n}) - h} \leq \frac{3\log(n) - 3h + c}{\log(\mathrm{n}) - h} = 3 + \frac{c}{\log(\mathrm{n}) - h} = O(1)$$

בנוסף נשים ♡ שכל פעולת join תעלה לנו לפחות 1 לכן $\mathrm{A} \geq 1$.

הניתוח התאורטי אכן מתיישב עם התוצאות, ניתן לראות שבכל הניסויים קיבלנו עלות ממוצעת בעלות קבועה שווה בקירוב ל-2.5.

ג. בסעיף הקודם הסברנו את תהליך ביצוע פעולת ה-*split* המתחילה באיבר המקסימלי בתת העץ השמאלי. כפי שהסברנו עד להגעה לשורש, עלות כל *join* היא $O(1)$. כשנגיע לשורש נבצע join בין עץ רק לכל תת העץ הימני של העץ המקורי ופעולה זו תעלה לנו $\log n$. לאחר פעולה זאת נסיים את הפיצול. לכן עלות ה-*join* המקסימלי הינו כשנגיע לשורש נבצע join בין עץ רק לכל תת העץ הימני של

העץ המקורי ופעולה זו תעלה לנו $\theta(\log n)$.



ניתן לראות כי הניתוח התאורטי מתיישב עם התוצאות ואכן קיבלנו עלות לוגריתמית.

ד. ראשית, ה*join* המקסימלי האפשרי עבור עץ בגודל *n* הוא $m = O(logn)$ (מכיוון שהעלות המקסימלית שיכולה להיות היא כאשר נבצע את הפעולה בין תת עץ ריק לבין תת העץ הימני\השמאלי של השורש שגובהו כ- $\log n$.

התוחלת של עלות *join* מקסימלי עבור *split* אקראי קטנה מהעלות המקסימלית האפשרית ל*join* בכללי. כלומר התוחלת של עלות *join* מקסימלי עבור *split* אקראי קטנה מ*m* ולכן היא גם $O(logn)$.

נמצא חסם הדוק יותר:

נשים לב שתוחלת עלות join מקסימלית עבור split אקראי קטן שווה לתוחלת רצף האחדים\ אפסים הארוך ביותר עבור רשימה של $\log n$ ביטים (רשימה כגובה העץ).
נסביר זאת בכך ש 0 מסמל עלייה שמאלה, ו 1 מסמל עליה ימינה. שגם הם מתפלגים בצורה אחידה.

בכך שאראה בטענת עזר שתוחלת רצף האחדים\אפסים הארוך ביותר עבור רשימה באורך n היא $O(logn)$ נסיק ש**תוחלת עלות join מקסימלית עבור split אקראי היא** $O(loglogn)$.

הוכחת טענת העזר: נסתכל על רצפים באורך $2\log n$.

הסיכוי שנקבל ממקום מסוים רצף כזה הוא $2 \cdot 0.5^{(2\log_2 n)} = \frac{2}{n^2}$. יש לכל היותר n מקומות בהם הרצף יכול להתחיל ולכן ההסתברות שרצף כזה קיים היא

$p(\text{Sequence of length } 2logn \;\; exists) \leq n \cdot \frac{2}{n^2} = \frac{2}{n}$.

נסמן ב- X את הרצף הכי ארוך ברשימת n ביטים:

$$\mathbb{E}(X) \leq p(\text{Sequence of length } 2logn \; doesnt \;\; exists) \cdot 2\log_2 n + p(\text{Sequence of length } 2logn \;\; exists) \cdot n$$
$$\leq 1 \cdot 2\log_2 n + \frac{2}{n} \cdot 2 \leq 3\log_2 n$$