מגישי העבודה:

שם – יניב מועלם
ת"ז – 209127612
שם משתמש – yanivmualem

שם – עדן דאיה
ת"ז – 315285759
שם משתמש – edendaya

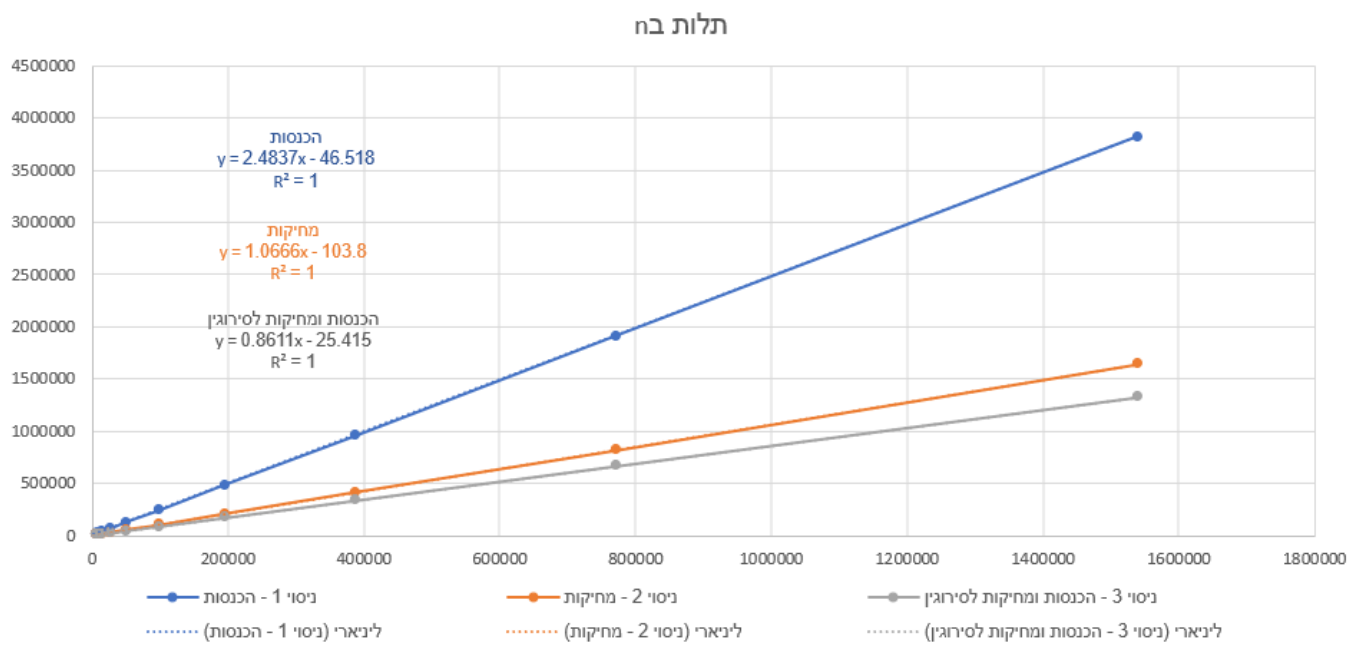
חלק תיאורטי

שאלה 1

.1

i	n	ניסוי 1 - הכנסות	ניסוי 2 - מחיקות	ניסוי 3 - הכנסות ומחיקות לסירוגין
1	3000	7406	3227	2675
2	6000	14894	6462	5088
3	12000	29594	12810	10195
4	24000	59581	25447	20964
5	48000	119315	51045	40620
6	96000	238495	102272	82730
7	192000	476630	204726	165857
8	384000	953675	409880	330569
9	768000	1907475	817846	661075
10	1536000	3814856	1638777	1322711

.2 הביטוי האסימפטוטי התואם עבור שלושת הניסויים הוא f(n)=O(n) –



שאלה 2

הכנסות לתחילת מבני הנתונים -

i	עץ AVL הכנסות להתחלה	רשימה מקושרת הכנסות להתחלה	מערך הכנסות להתחלה
1	2.86625E-05	6.65983E-07	6.62804E-07
2	3.12915E-05	3.77496E-07	6.6487E-07
3	3.26065E-05	4.62161E-07	7.01904E-07
4	3.43952E-05	4.98056E-07	8.63791E-07
5	3.46766E-05	3.98477E-07	1.22655E-06
6	3.48325E-05	5.53767E-07	1.34198E-06
7	3.46551E-05	4.82037E-07	1.55122E-06
8	3.58882E-05	4.31061E-07	1.68659E-06
9	3.66545E-05	1.93663E-06	4.42929E-07
10	3.57721E-05	2.7E-06	4.77378E-07

עבור הכנסה של איברים בהתחלה, למדנו כי מדובר בעבודה של O(logn) בעצי AVL, O(1) ברשימות מקושרות וO(n) בarray, וזאת ביחס למקרה הממוצע של פעולה זו.
בהתאם לכך ציפינו שזמני הריצה של הרשימה המקושרת יהיו הטובים ובניגוד לכך שזמני הריצה בעבור המערך יהיו הגרועים ביותר. בעוד כי זמן הריצה הממוצע של הרשימה המקושרת היה הקצר ביותר, קיבלנו שזמן הריצה הממוצע של עץ הAVL היה ארוך בהשוואה לזמן הממוצע שקיבלנו עבור מערך.
סיבה אפשרית לפער זה הוא בכך שהמערך ממומש על ידי פייתון ועשוי לפעול בצורה יעילה יותר מהמצופה.

הכנסות אקראיות למבני הנתונים -

i	עץ AVL הכנסות אקראיות	רשימה מקושרת הכנסות אקראיות	מערך הכנסות אקראיות
1	2.87048E-05	7.061E-05	1.32036E-06
2	3.20396E-05	0.000139874	1.127E-06
3	3.27632E-05	0.000221756	1.19389E-06
4	3.32258E-05	0.00029233	1.16297E-06
5	3.44329E-05	0.000364871	1.2325E-06
6	3.48507E-05	0.000454885	1.32932E-06
7	3.58395E-05	0.000563768	1.42508E-06
8	3.65938E-05	0.000641058	1.57724E-06
9	3.70558E-05	0.000730604	1.6983E-06
10	3.67181E-05	0.000832946	1.79408E-06

במקרה זה נצפה שזמן הריצה הממוצע יהיה זהה עבור הרשימות המקושרות והמערך, שכן אנחנו יודעים שסיבוכיות הזמן לפעולה זו היא O(n-k+1) כאשר k הוא האינדקס שנבחר באקראי להכנסת האיבר. מנגד נצפה לזמן ריצה טוב יותר עבור עץ הAVL שכן אנחנו יודעים שהסיבוכיות עבור פעולה זו היא O(logn). ניתן לראות שאמנם עץ הAVL רץ בזמן מהיר יותר מאשר הרשימה המקושרת, אך המערך עובד בזמן המהיר ביותר בניגוד לציפיות.
סיבה אפשרית למימוש המהיר של המערך הוא בכך שמדובר במימוש יעיל של שפת התוכנה פייתון.

הכנסות לסוף מבני הנתונים -

i	עץ AVL הכנסות לסוף	רשימה מקושרת הכנסות לסוף	מערך הכנסות לסוף
1	2.94412E-05	2.50273E-05	7.83602E-08
2	3.05643E-05	4.0961E-05	3.31799E-07
3	3.28606E-05	5.997E-05	2.21358E-07
4	3.29369E-05	7.8701E-05	1.77701E-07
5	3.41881E-05	9.71214E-05	1.33832E-07
6	3.52156E-05	0.000118177	1.10944E-07
7	3.42217E-05	0.000139131	1.01657E-07
8	3.53206E-05	0.000159614	8.3069E-08
9	3.58993E-05	0.000179565	7.39098E-08
10	3.62936E-05	0.000199437	7.89007E-08

בעבור סיבוכיות הזמן להכנסה זו, אנחנו יודעים שהסיבוכיות עבור עץ AVL היא O(logn), בעבור מערך היא O(1) ובעבור רשימה מקושרת הוא O(n). בהתאם לכך ציפינו לקבל כי המערך יעבוד בצורה המהירה ביותר, לאחר מכן עץ הAVL ולכך שהרשימה המקושרת תסיים בזמן הרב ביותר. ניתן לראות כי אכן התקבלו התוצאות להן ציפינו. המערך רץ באופן המהיר ביותר, לאחריו עץ הAVL ולבסוף הרשימה המקושרת.

תיעוד הקוד

מחלקת AVLNode

מחלקה המייצגת צומת בעץ מסוג AVL. המחלקה כוללת את השדות הבאים:
Value – ערכה של הצומת
Left/Right – מצביעים עבור הבנים (השמאלי והימני בהתאמה).
Parent – מצביע עבור ההורה של הצומת.
Height – גובה הצומת.
Size – גודל תת העץ של הצומת. ערך זה כולל את הצומת עצמו.
is_virtual – שדה המציין האם הצומת וירטואלי או לא. שדה זה מאותחל כFalse.

פונקציות המחלקה

findingRoot – פונקציית עזר, אשר בהינתן צומת תחזיר את שורש העץ אליו הוא שייך.

- אופן פעולה: הפונקציה מטפסת במעלה העץ החל מהצומת הנתון כל עוד קיים הורה לצומת הנבדק. לבסוף יוחזר הצומת האחרון אליו היא הגיעה שהוא שורש העץ.
- סיבוכיות: O(logn) . במקרה הגרוע הפונקציה תקבל צומת שעומקו כעומק העץ (עץ AVL).

fixingTree – פונקציה רקורסיבית אשר בהינתן צומת מתקנת את העץ החל ממנו וממשיכה מעלה. התיקון מתבצע על ידי שימוש בגלגולים ותחזוקה של השדות size וheight. הפונקציה מחזירה את כמות הגלגולים ותיקוני הגבהים שנעשו במהלך הפעלתה.

- אופן פעולה: הפונקציה עורכת את התיקונים הבאים לפי הסדר –
תיקון הHeight – הפונקציה קוראת לפונקציית העזר fixingHeight. פונקציית העזר מחשבת ראשית את גובהו התקין של הצומת (הערך המקסימלי מבין גבהי הבנים שלו + 1). פונקציית העזר בודקת האם גובהו של הצומת שווה לגובהו התקין. במידה ולא, ערכו יעודכן והפונקציה תספור את השינוי בגובה.

תיקון הSize - הפונקציה קוראת לפונקציית העזר fixingSize. פונקציית העזר מחשבת את גודלו התקין של תת עץ הצומת (סכום גודלם של תתי העצים + 1 עבור הצומת עצמו). במידה וגודל תת העץ שונה מזה שחושב פונקציית העזר תעדכן אותו.

תיקון ערך הBF – הפונקציה בודקת האם ערך הBF של הצומת תקין. במידה ולא היא תקרא לאחת מפונקציות העזר שתבצע את הגלגול הדרוש לצורך התיקון (rightRotate, leftRotate , leftThenRightRotate , rightThenLeftRotate). פונקציות עזר אלו מבצעות את הגלגולים המתאימים על ידי מנגנון שינוי המצביעים שנלמד בהרצאה, ולבסוף מחזירות את מספר הגלגולים שבוצעו.

לאחר ביצוע כל התיקונים עבור הצומת הנוכחי, הפונקציה תעבור לקריאה הרקורסיבית לעצמה עם ההורה של אותו הצומת.

- סיבוכיות: O(logn) כגובה העץ. בכל קריאה רקורסיבית לפונקציה נעשית O(1) עבודה בסך הכל. fixingHeight וfixingSize מבצעות פעולות אריתמטיות, גישה ועדכון לשדות כך שהן מבצעות O(1) עבודה. פונקציות הגלגולים מבצעות שינויים במצביעים ולכן גם הן מבצעות בסך הכל O(1) עבודה. בכל קריאה רקורסיבית נעשית קריאה לפונקציה עבור צומת ההורה ולכן במספר הגרוע נבצע קריאות כגובה העץ.

Successor – פונקציה אשר בהינתן צומת מוצאת את הצומת העוקב לו.

- אופן הפעולה- הפונקציה בודקת ראשית האם לצומת יש בן ימני. במידה וכן היא תטייל אליו וממנו תטייל שמאלה ככל הניתן כאשר הצומת האחרון הוא הצומת העוקב. במידה ולצומת הנתון אין בן ימני הפונקציה תטייל כלפי מעלה עד אשר תגיע לפנייה הראשונה ימינה, שם תיעצר על הצומת שיתקבל ואותו תחזיר.
- סיבוכיות- O(logn) כגובה העץ. במקרה הגרוע הפונקציה תטייל מסלול לכל אורך העץ.

Predecessor – פונקציה סימטרית לפונקציית ה**Successor** למציאת הצומת בעל הערך הקודם לצומת הנתון. הסיבוכיות אם כן זהה.

מחלקת AVLTreeList

מחלקה זו מממשת את פעולות הADT שהגדרנו בהרצאות על ידי מימוש העקרונות של עצי AVL.
במחלקה זו מתוחזקים השדות הבאים:
Root- מצביע עבור שורש העץ.
firstNode- מצביע עבור הצומת שמייצג האיבר הראשון ברשימה.
lastNode – מצביע עבור הצומת שמייצג את האיבר האחרון ברשימה.
Size – גודל העץ, כולל השורש.

פונקציות המחלקה:

Select – פונקציה אשר בהינתן מספר K מחזירה את הצומת מדרגה זו, על ידי מימוש הפעולה Tree-Select שנלמדה בהרצאה.

- אופן פעולה: הפונקציה מתחילה את פעולתה על שורש העץ ובודקת את דרגתו בעץ, על ידי חישוב גודל תת העץ השמאלי שלו + 1 עבורו. הפונקציה מגדירה את דרגתו כr. במידה ודרגתו שווה לK הפונקציה תחזיר את הצומת הנוכחי. במידה ודרגתו גבוהה מK הפונקציה תעבור לחיפוש בתת העץ השמאלי שלו על ידי בדיקת הבן השמאלי. במידה ודרגתו נמוכה מK הפונקציה תעבור לחפש בתת העץ הימני של הצומת על ידי בדיקת הבן הימני וחיפוש הדרגה שערכה k-r.
- סיבוכיות: O(logn) כגובה העץ. בכל איטרציה של הפונקציה מתבצעות פעולות כגון פעולות אריתמטיות וגישה לשדות שהן O(1). במקרה הגרוע הפונקציה תטייל משורש העץ כל הדרך למטה ולכן במקרה זה נקבל מספר איטרציות בסך O(logn).

Insert – פונקציה המקבלת ערך val ואינדקס i. הפונקציה מכניסה את הערך הנתון באינדקס המבוקש ומחזירה את מספר פעולות האיזון שנדרשו על מנת לתקן את העץ כך שיעמוד בתנאי הAVL.

- <u>אופן פעולה:</u> ראשית הפונקציה יוצרת איבר AVLNode חדש עם הערך val. הפונקציה קוראת לפונקציית העזר addingVirtualNodes שיוצרת עבורו בנים וירטואליים. כעת הפונקציה מבחינה בין מקרים שונים –
  עבור רשימה ריקה, הפונקציה תוסיף את הצומת כשורש העץ ותחזיר 0.
  עבור רשימה באורך i הפונקציה תמצא את האיבר האחרון ברשימה בעזרת הפונקציה retrieveNode, ותגדיר את הצומת החדש כבנו הימני, ותקשר את המצביע lastNode עבור הצומת החדש.
  בכל מקרה אחר, הפונקציה תחפש את האיבר שנמצא כרגע באינדקס i על ידי קריאה ל retrieveNode. במידה ולאיבר זה אין בן שמאלי, הפונקציה תוסיף את הצומת החדש כבנו השמאלי. במידה ולאיבר זה יש בן שמאלי, הפונקציה תמצא את קודמו של האיבר על ידי קריאה לPredecessor ותוסיף אותו כבנו הימני.
  במידה וi=0 הפונקציה תעדכן את השדה firstNode כך שיצביע עבור הצומת החדש.
  לאחר פעולות אלו הפונקציה תתקן את העץ החל מההורה של הצומת החדש כל הדרך למעלה אל השורש, ותעדכן את שורש העץ על ידי קריאה לfindingRoot .
  לבסוף הפונקציה תעדכן את גודל העץ על ידי קריאה לsetSize ותחזיר את מספר התיקונים שנעשו על מנת לאזן את העץ.
- <u>סיבוכיות:</u> O(logn). יצירת הצומת וקריאה לפונקציה addingVirtualNodes נעשות ב(1)O.
  לאחר מכן הפונקציה קוראת לפונקציית העזר retrieveNode שקוראת לפונקציה select אשר רצה בזמן O(logn). הקריאה לפונקציה Predecessor גם היא מתרחשת בזמן של O(logn).
  שינוי המצביעים על מנת להגדיר את הצומת החדש כבנו של אחד האיברים בעץ הוא O(1).
  לבסוף, תיקון העץ ומציאת השורש על ידי קריאה לפונקציות fixingTree וfindingRoot בהתאמה הוא O(logn) עבור כל אחת מהן. עדכון גודל העץ על ידי setSize נעשה בזמן O(1).
  נקבל כי הסיבוכיות הכוללת של Insert היא O(logn).

Delete – פונקציה המקבלת אינדקס i ומוחקת את האיבר הנמצא באינדקס זה ברשימה. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו על מנת לשמור על העץ כAVL.

- <u>אופן פעולה-</u> ראשית הפונקציה בודקת האם העץ הינו ריק. במידה וכן היא תחזיר -1 כדרוש.
  במידה והרשימה מכילה איבר אחד בלבד (האיבר שרוצים למחוק) אזי הפונקציה מעדכנת את השדות firstNode וlastNode לערך None.
  בכל מקרה אחר, הפונקציה תחפש את האיבר שנמצא כעת במיקום הi על ידי קריאה לפונקציית העזר retrieveNode (נסמנו לצורך התיעוד כx). כעת הפונקציה תבחין בין המקרים השונים –
  במידה ול-x ישנם שני בנים, הפונקציה תמצא את העוקב שלו על ידי קריאה לSuccessor והחלפת x בו על ידי העתקת הערך שלו. לבסוף הפונקציה תמחק פיזית את הצומת העוקב מהעץ.
  במידה ול-x אין שני בנים, הפונקציה תמחק פיזית מהעץ את x עצמו.
  הפונקציה משנה את המצביעים בהתאם למקרה שבו הצומת שעתיד להימחק הוא בן שמאלי או ימני של אביו.
  לבסוף הפונקציה מתקנת את העץ החל מאב הצומת שהזזנו כלפי מעלה, מעדכנת את השורש על ידי קריאה לפונקציה findingRoot. במידה והפונקציה מחקה את האיבר הראשון או האחרון, היא תקרא לretrieve שתחזיר את האיבר המתאים ותעדכן את השדות firstNode או lastNode בהתאמה. הפונקציה תעדכן את גודל העץ על ידי קריאה לsetSize ותחזיר את מספר התיקונים שנדרשו כדי לאזן את העץ.
- סיבוכיות- O(logn). שינויי המצביעים ועדכון השדות נעשים בזמן של O(1). פונקציית העזר retrieveNode קוראת לפונקציה Select שרצה בזמן O(logn). הפונקציה Successor רצה גם היא בזמן O(logn), וכך גם הפונקציות fixingTree, findingRoot וretrieve. עדכון גודל העץ על ידי setSize נעשה בזמן O(1). סך הסיבוכיות אם כך היא O(logn).

first- פונקציה המחזירה את האיבר ברשימה שהאינדקס שלו הוא אפס.

- אופן פעולה: הפונקציה בודקת האם השדה firsNode ריק. במידה וכן יוחזר None ובמידה ולא היא תחזיר את ערכו.
- סיבוכיות: O(1) .גישה לאיבר בזיכרון, בדיקות על האיבר והחזרת ערך.

last- פונקציה המחזירה את האיבר ברשימה שהאינדקס שלו הוא הגבוה ביותר.
- הפונקציה בודקת האם השדה lastNode ריק. במידה וכן יוחזר None ובמידה ולא היא תחזיר את ערכו.
- סיבוכיות: O(1) .גישה לאיבר בזיכרון, בדיקות על האיבר והחזרת ערך.

listToArray – הפונקציה מחזירה מערך של אברי הרשימה לפי הסדר שלהם.
- אופן פעולה: הפונקציה מאתחלת מערך ומבצעת סיור InOrder בעץ. כל צומת עליו הפונקציה עוברת יוכנס לפי הסדר למערך.
- סיבוכיות: O(n). סיור InOrder בעץ הוא O(n) במהלכו כל איבר שנכנס לרשימה הוא O(1).

Length – פונקציה המחזירה את אורך הרשימה.
- אופן פעולה: הפונקציה בודקת את איבר השורש. אם הוא קיים, היא תחזיר את גודלו. במידה והוא לא קיים יוחזר 0.
- סיבוכיות: O(1). קריאה למשתנים ובדיקות עליהם.

getRoot – פונקציה המחזירה את שורש העץ.
- אופן פעולת הפונקציה: הפונקציה הפונה לשדה השמור root ומחזירה אותו.
- סיבוכיות: O(1). קריאה למשתנים והחזרתם.

Search- פונקציה המקבלת ערך val ובודקת האם הוא קיים בעץ. במידה וכן יוחזר אינדקס האיבר המתאים, ובמידה ולא יוחזר -1.
- אופן פעולת הפונקציה: הפונקציה יוצרת מערך של אברי הרשימה בעזרת קריאה לפונקציה listToArray ולאחר מכן עוברת איבר איבר במערך ובודקת האם ערכו שווה לערך המבוקש. במידה וכן יוחזר האינדקס המתאים ובמידה ולא הפונקציה תחזיר -1.
- סיבוכיות: O(n). קריאה לפונקציה listToArray שהיא O(n). במקרה הגרוע נעבור על כל האיברים ברשימה המכילה n איברים ולכן סך העבודה של כל הפונקציה היא O(n).

Join – פונקציית עזר המקבלת רשימה lst וצומת מקשר המהווה bridge. הפונקציה משרשרת את הרשימה הנוכחית עם רשימת הקלט באופן כזה שכל אברי הרשימה הנוכחית יופיעו ראשונים בסדרם המקורי, לאחר מכן הצומת המקשר, ולאחר מכן אברי הרשימה lst לפי הסדר שלהם.
- אופן פעולת הפונקציה: הפונקציה מבחינה בין מקרים ופועלת בהתאם –
  במידה והרשימה lst ריקה, הצומת המקשר יוכנס לרשימה המקורית.
  במידה והרשימה lst אינה ריקה, הפונקציה בודקת מי מבין שני העצים גבוה יותר. בתיאור זה נתייחס בלי הגבלת הכלליות למצב בו lst גבוה יותר –
  הפונקציה תתחיל לטייל מטה מהשורש של lst לבן השמאלי בכל פעם באופן רקורסיבי, ותיעצר כאשר גובה תת העץ שהצומת עליו עומדת משרה כבר קטן מגובה העץ self.
  צומת זה יסומן כx. צומת זה ינותק מצומת האב שלו ובמקומו ימוקם הצומת המקשר bridge.
  הפונקציה תעדכן את הבן השמאלי של הצומת המקשר להיות השורש הישן של self ואת הבן הימני שלו להיות x.
  לאחר מכן העץ יעבור חיווט כך ששורשו יעודכן להיות השורש של lst והמצביע לאיבר הגדול ברשימה יצביע כעת על האיבר הגדול ביותר בlst.
  במידה וlst היה נמוך יותר, ההתקדמות הייתה נעשית לכיוון הבנים הימניים (הפעם על העץ הנוכחי), ובסיום התהליך לא היה צורך בשינוי המצביע לשורש.
  הפונקציה תחפש את השורש החדש במידה והשתנה ומבצעת תיקון על העץ מהצומת bridge כלפי מעלה.
  לבסוף הפונקציה תעדכן את גודל העץ על ידי קריאה לsetSize .
- סיבוכיות: O (Δh). הפונקציה מטיילת מטה כהפרש הגבהים שבין שני העצים, ונעצרת כאשר גובה תת העץ אליו הגיעה שווה לגובה תת העץ השני ((Δh) O עבודה).

לאחר מכן מבצעים שינוי במצביעים שפועל בזמן קבוע.
בשלב הבא בודקים האם השתנה השורש. בדיקה זו נעשית מהצומת bridge ועד השורש, ולכן גם היא נעשית בסיבוכיות O (Δh) כעומק הצומת bridge.
לבסוף ישנה קריאה לפונקציית העזר fixingtree לתיקון העץ שמבצעת פעולות בזמן קבוע מbridge ועד השורש ולכן לוקחת גם היא O (Δh).
נקבל אם כך כי הסיבוכיות הכוללת של הפונקציה היא O (Δh).

concat- פונקציה המקבלת שתי רשימות ומאחדת אותן בצורה כזו שאיברי הרשימה self יופיעו בסדרם המקורי ראשונים, ולאחר מכן יופיעו איברי הרשימה lst גם הם בסדרם המקורי. לבסוף הפונקציה תחזיר את הפרשי הגבהים של העצים שחוברו בתהליך.

- אופן פעולה: ראשית הפונקציה תטפל במקרי הקצה. במידה והיא מקבלת שתי רשימות ריקות היא תחזיר את הערך 0.
במידה והרשימה lst היא רשימה ריקה, יוחזר גובה העץ self + 1.
במידה והרשימה self ריקה, המצביעים ישונו בצורה כזו שכעת יצביעו על הרשימה החדשה, ולאחר מכן יוחזר גובה הרשימה + 1.
במידה ומקרי הקצה לא מתרחשים, הפונקציה תאחזר את האיבר האחרון ברשימה self ותשמור אותו. נגדיר צומת זה כx. הפונקציה תקרא לפונקציית העזר getHeight על מנת לחשב את הפרשי הגבהים. הפרש זה נשמר כמשתנה h. לבסוף הפונקציה מוחקת את הצומת x מהרשימה וקוראת לפונקציה join עבור שתי הרשימות כאשר הצומת המקשר הינו x.
- סיבוכיות: O (logn). ראשית הפונקציה מטפלת במקרי הקצה (החזרת ערכים בזמן קבוע). פעולת האחזור אותה ניתחנו עולה כO(logn) וכך גם פעולת המחיקה.
הקריאה לפונקציה join עולה כO(Δh) כאשר Δh חסום על ידי logn. לאחר מכן מבוצעות פעולות אריתמטיות שהן O(1). נקבל שהסיבוכיות הכוללת של הפונקציה אם כך היא O (logn).

mergeSort - פונקציה הממיינת את איברי המערך בסדר עולה.
אופן פעולת הפונקציה: הפונקציה יוצרת שני תתי מערכים של המערך המקורי על ידי חלוקת המערך המקורי לשתיים, וממיינת כל אחד ממהמערכים בנפרד, רקורסיבית.

- סיבוכיות: O(nlogn). המערך מתחלק כל פעם לשניים ובכל רמה בעץ הסיבוכיות יש O(N) עבודה שכן משווים בין כל האיברים. לפיכך נוסחת הנסיגה המתאימה למימוש הרקורסיבי היא T(n) = 2T(n/2) + O(n) והפתרון שלה הוא O(nlogn).

sort - פונקציה הממיינת את איברי הרשימה בסדר עולה באמצעות mergeSort ומחזירה עץ חדש ממויין.

- אופן פעולת הפונקציה: הפונקציה יוצרת מערך של אברי הרשימה בעזרת קריאה לפונקציה listToArray ולאחר מכן קוראת לפונקציה mergeSort על המערך. אחר כך, הפונקצייה עוברת בלולאה על המערך הממויין ומכניסה את איברי המערך לעץ חדש.
- סיבוכיות: O(nlogn). קריאה לפונקציה listToArray שהיא O(n). לאחר מכן, קריאה לפונקציה mergeSort שהיא O(nlogn). ולבסוף הכנסה של n איברים לרשימה כאשר כל הכנסה היא O(logn) ובסה"כ O(nlogn).

permutation - פונקציה המחזירה עץ חדש שמכיל את אותם האיברים מהעץ המקורי, בסדר אקראי.

- אופן פעולת הפונקציה: הפונקציה יוצרת מערך של אברי הרשימה בעזרת קריאה לפונקציה listToArray. לאחר מכן הפונקצייה עוברת בלולאה על המערך ובכל פעם בוחרת באקראי בעזרת random את האינדקס של האיבר הבא שנוסיף למערך ומוסיפה אותו לעץ. לאחר ההוספה האיבר נמחק מהמערך, ולבסוף מוחזר העץ החדש.
- סיבוכיות: O(nlogn). קריאה לפונקציה listToArray שהיא O(n). לאחר מכן הכנסה ומחיקה של n איברים לרשימה כאשר כל הכנסה או מחיקה היא O(logn) ובסה"כ O(nlogn).