

חלק תיאורטי-שמות המגשים:

תומר רודניצקי (20662787820) - rudnitzky

עידו רוזינר (209617000) - idorosiner

להלן תיעוד AVLNode:

פונקציה	תיאור	חישוב סיבוכי ת זמן
getLeft()	הפונקציה מחזירה את הבן השמאלי של הצומת. מוגדר להיות None במקרה והצומת וירטואלית. – עידו רוזינר	$O(1)$
getRight()	הפונקציה מחזירה את הבן הימני של הצומת. מוגדר להיות None במקרה והצומת וירטואלית. – עידו רוזינר	$O(1)$
getParent()	הפונקציה מחזירה את ההורה של הצומת. מוגדר להיות None במקרה והצומת וירטואלית. – עידו רוזינר	$O(1)$
getValue()	הפונקציה מחזירה את הערך של הצומת. מוגדר להיות None במקרה והצומת וירטואלית. – עידו רוזינר	$O(1)$
getHeight()	הפונקציה מחזירה את הגובה של הצומת. מוגדר להיות 1- במקרה והצומת וירטואלית. – עידו רוזינר	$O(1)$
getSize()	הפונקציה מחזירה את הגודל של הצומת. מוגדר להיות 0 במקרה והצומת וירטואלית. – עידו רוזינר	$O(1)$
getBF()	הפונקציה מחזירה את ה-balance factor של הצומת. – עידו רוזינר	$O(1)$
setLeft(node)	הפונקציה מקבעת את הבן השמאלי של הצומת להיות node. – עידו רוזינר	$O(1)$
setRight(node)	הפונקציה מקבעת את הבן הימני של הצומת להיות node. – עידו רוזינר	$O(1)$
setParent(node)	הפונקציה מקבעת את ההורה של הצומת להיות node. – עידו רוזינר	$O(1)$
setValue(value)	הפונקציה מקבעת את הערך של הצומת להיות value. – עידו רוזינר	$O(1)$
setHeight(h)	הפונקציה מקבעת את הגובה של הצומת להיות h. – עידו רוזינר	$O(1)$
setSize(s)	הפונקציה מקבעת את הגודל של הצומת להיות s. – עידו רוזינר	$O(1)$
setBF(bf)	הפונקציה מקבעת את ה-balance factor של הצומת להיות bf. – עידו רוזינר	$O(1)$
isRealNode()	הפונקציה מחזירה True אם הצומת היא לא צומת וירטואלית (הגובה שונה מ-1) ואחרת False. – עידו רוזינר	$O(1)$
computeBF()	הפונקציה מחשבת ומחזירה את ה-balance factor של הצומת. כלומר, מחשבת ומחזירה $1 + height(leftSubtree) - (1 + height(rightSubtree))$ – עידו רוזינר	$O(1)$
computeHeight()	הפונקציה מחשבת ומחזירה את הגובה של הצומת. כלומר, מחשבת ומחזירה	$O(1)$

	$1 + \max\{height(leftSubtree), height(rightSubtree)\}$ – עידו רוזינר	
computeSize()	הפונקציה מחשבת ומחזירה את הגודל של הצומת. כלומר, מחשבת ומחזירה את מספר הצמתים שקטנים שווים לה – $1 + size(leftSubtree) + size(rightSubtree)$ – עידו רוזינר	$O(1)$

להלן תיעוד AVLTreeList:

פונקציה	תיאור	חישוב סיבוכיות זמן
getRoot()	הפונקציה מחזירה את השורש של העץ. מוחזר AVLNode(None) במקרה והעץ וירטואלי. – עידו רוזינר	$O(1)$
getMax()	הפונקציה מחזירה את הצומת המקסימלית בעץ. מוחזר None במקרה והעץ וירטואלי – עידו רוזינר	$O(1)$
getMin()	הפונקציה מחזירה את הצומת המינימלית בעץ. מוחזר None במקרה והעץ וירטואלי. – עידו רוזינר	$O(1)$
getSize()	הפונקציה מחזירה את גודל העץ - מספר הצמתים הלא וירטואליים שנמצאים בעץ. מוחזר 0 במקרה והעץ וירטואלי. – עידו רוזינר	$O(1)$
setRoot(node)	הפונקציה מקבעת את השורש של העץ להיות node. – עידו רוזינר	$O(1)$
setMax(node)	הפונקציה מקבעת את הצומת המקסימלית בעץ להיות node. – עידו רוזינר	$O(1)$
setMin(node)	הפונקציה מקבעת את הצומת המינימלית בעץ להיות node. – עידו רוזינר	$O(1)$
setSize(s)	הפונקציה מקבעת את הגודל של העץ להיות s. – עידו רוזינר	$O(1)$
empty()	הפונקציה מחזירה True אם הרשימה (העץ) ריקה (עץ וירטואלי) ואחרת False. – עידו רוזינר	$O(1)$
retrieve(i)	הפונקציה מחזירה את ערך האיבר במקום ה-i אם קיים, אחרת היא מחזירה None. – תומר רודניצקי	הפונקציה קוראת ל- $treeSelect(self, i+1)$, שרצה ב- $O(\log i)$, וכל שאר הפעולות קורות ב- $O(1)$. לכן הפונקציה רצה ב- $O(\log i)$.
insert(i, val)	הפונקציה מכניסה איבר בעל ערך val לרשימה	תחילה הפונקציה קוראת ל- $treeSelect(i)$ שרצה בסיבוכיות $O(\log i)$.

	במקום ה-i במידה וקיימים לפחותו איברים ברשימה. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו בשלב תיקון העץ על מנת לשמר את תכונת האיזון. – תומר רודניצקי	לאחר מכן הפונקציה קוראת ל- <code>rebalanceTree(node)</code> בסיבוכיות $O(\log n)$. מכיוון שכל שאר הפעולות קורות ב- $O(1)$ הפונקציה רצה בסיבוכיות $O(\log n)$.
<code>delete(i)</code>	הפונקציה מוחקת את האיבר במקום ה-i ברשימה, אם הוא קיים. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו בשלב תיקון העץ על מנת לשמר את תכונת האיזון. אם לא קיימים מספיק איברים ברשימה הפונקציה מחזירה -1. – עידו רוזינר	תחילה הפונקציה קוראת ל- <code>treeSelect(i)</code> שרצה בסיבוכיות $O(\log i)$ או לפונקציה <code>successor(x)</code> שרצה בסיבוכיות $O(\log n)$. לאחר מכן הפונקציה קוראת ל- <code>delete_node(node_to_del)</code> ול- <code>rebalanceTree(node)</code> , שרצות בסיבוכיות $O(\log n)$, אחד אחרי השני. מכיוון שכל שאר הפעולות קורות ב- $O(1)$, הפונקציה רצה בסיבוכיות $O(\log n)$.
<code>delete_node(node_to_del)</code>	הפונקציה מוחקת את הצומת <code>node_to_del</code> מהעץ ומחזירה את הצומת שממנה נתחיל את פעולת האיזון של העץ. – עידו רוזינר	אם <code>node_to_del</code> בעל שני ילדים אמיתיים הפונקציה קוראת ל- <code>successor(node)</code> שרצה בסיבוכיות $O(\log n)$, לכל היותר $O(1)$ פעמים. מכיוון שכל שאר הפעולות בפונקציה קורות ב- $O(1)$ הפונקציה רצה בסיבוכיות $O(\log n)$.
<code>first()</code>	הפונקציה מחזירה את ערך האיבר הראשון ברשימה, או None ברשימה ריקה. – עידו רוזינר	$O(1)$
<code>last()</code>	הפונקציה מחזירה את ערך האיבר האחרון ברשימה, או None ברשימה ריקה. – עידו רוזינר	$O(1)$
<code>listToArray()</code>	הפונקציה מחזירה מערך המכיל את הערכים של איברי הרשימה (העץ) לפי סדר האינדקסים (in-order), או מערך ריק אם הרשימה ריקה. – עידו רוזינר	הפונקציה קוראת לפונקציה <code>inorder(node, L)</code> שרצה בסיבוכיות $O(n)$. מכיוון שכל שאר הפעולות קורות ב- $O(1)$, הפונקציה רצה בסיבוכיות $O(n)$.
<code>inorder(node, L)</code>	הפונקציה מכניסה לרשימה L את הערכים של איברי הרשימה (העץ) לפי סדר האינדקסים. הפונקציה עושה זאת על	ההכנסה מבוצעת כמו באלגוריתם in-order שנלמד בשיעור (במקום הדפסה נוסף את הערך של הצומת למערך L) ולכן הפונקציה רצה בסיבוכיות $O(n)$.

	ידי ריצה in-order על איברי העץ. – עידו רוזינר	
length()	הפונקציה מחזירה את מספר האיברים ברשימה. – עידו רוזינר	$O(1)$
sort()	הפונקציה מסדרת את איברי הרשימה בסדר עולה ומחזירה רשימה (עץ) חדשה. – תומר רודניצקי	<p>תחילה הפונקציה קוראת ל- <code>listToArray(self)</code>, שרצה בסיבוכיות $O(n)$. לאחר מכן הפונקציה מפרידה את כל האיברים שהם None מהאיברים שהם לא None בסיבוכיות $O(n)$. לאחר מכן הפונקציה קוראת ל- <code>merge_sort(lst)</code> שרצה בסיבוכיות $O(n \cdot \log n)$. אחר כך הפונקציה קוראת ל- <code>buildTreeFromList(lst, first, last)</code> שרצה בסיבוכיות $O(n)$ ושומרת את הצומת שהתקבלה כשורש של AVLTreeList חדש. לבסוף הפונקציה מעדכנת את השדות של ה-AVLTreeList החדש. לשם כך הפונקציה קוראת ל- <code>max_node(node)</code> ו- <code>min_node(node)</code> אשר רצות בסיבוכיות $O(\log n)$. סה"כ נקבל כי הפונקציה <code>sort(self)</code> רצה בסיבוכיות $O(n \cdot \log n)$.</p>
permutation()	הפונקציה מחזירה רשימה (עץ) חדשה המכילה את אותם האיברים של הרשימה (העץ) הנוכחי (self) בסדר אקראי. – עידו רוזינר	<p>תחילה הפונקציה קוראת ל- <code>listToArray(self)</code> שרצה בסיבוכיות $O(n)$. לאחר מכן הפונקציה קוראת לפונקציה <code>shuffle(lst)</code>, שרצה בסיבוכיות $O(n)$. אחר כך הפונקציה קוראת ל- <code>buildTreeFromList(lst, first, last)</code> שרצה בסיבוכיות $O(n)$ ושומרת את הצומת שהתקבלה כשורש של AVLTreeList חדש. לבסוף הפונקציה מעדכנת את השדות של ה-AVLTreeList החדש. לשם כך הפונקציה קוראת ל- <code>max_node(node)</code> ו- <code>min_node(node)</code> אשר רצות בסיבוכיות $O(\log n)$. סה"כ נקבל כי הפונקציה <code>permutation(self)</code> רצה בסיבוכיות $O(n)$.</p>
concat(lst)	הפונקציה מקבלת רשימה ומשרשר אותה אל סוף הרשימה הנוכחית.	במקרה שבו <code>lst</code> או <code>self</code> ריקים הפונקציה מחברת את העצים (אם יש מה לחבר) ומסיימת לרוץ

	<p>הפונקציה מחזירה את הערך המוחלט של הפרש הגבהים של עצי ה-AVL שמוזגו. – עידו רוזינר</p>	<p>– פעולה זו קוראת בסיבוכיות $O(1)$. אם האורך של אחד העצים הוא 1 הפונקציה תבצע insert לאיבר של העץ בעל האיבר היחיד אל תוך העץ השני ותסיים את הריצה בסיבוכיות $O(\log n)$. אחרת, הפונקציה שומרת את האיבר המקסימלי של self וקוראת ל--delete(self.length() – 1) בסיבוכיות $O(\log n)$. כעת, הפונקציה משווה בין הגובה של העצים (לאחר המחיקה) ומחברת אותם בהתאם: אם הגובה של self שווה לגובה של lst הפונקציה קוראת ל-joinSelfSame(self, x, lst) שרצה בסיבוכיות $O(1)$. אם הגובה של self גדול מהגובה של lst הפונקציה קוראת ל-joinSelfBigger(self, x, lst) שרצה בסיבוכיות $O(\log n)$. אם הגובה של self קטן מהגובה של lst הפונקציה קוראת ל-joinSelfSmaller(self, x, lst) שרצה בסיבוכיות $O(\log n)$. לבסוף הפונקציה תעדכן את השדות של העץ (בסיבוכיות $O(1)$) ותקרא ל-rebalanceTree(node) כאשר node הוא הצומת המקסימלית שמחקנו ושמרנו מקודם – בסיבוכיות $O(\log n)$. סה"כ הפונקציה רצה בסיבוכיות $O(\log n)$.</p>
joinSelfBigger(x, lst)	<p>הפונקציה מחברת את העץ ל-self דרך הצומת x. – עידו רוזינר</p>	<p>הפונקציה מניחה כי גובה העץ self גדול מגובה העץ lst ומחברת את העצים דרך הצומת x כנלמד בשיעור (הפונקציה Join מהשיעור) – בסיבוכיות $O(\log n)$</p>
joinSelfSmaller(x, lst)	<p>הפונקציה מחברת את העץ ל-self דרך הצומת x. – עידו רוזינר</p>	<p>הפונקציה מניחה כי גובה העץ self קטן מגובה העץ lst ומחברת את העצים דרך הצומת x כנלמד בשיעור (הפונקציה Join מהשיעור) – בסיבוכיות $O(\log n)$</p>
joinSelfSame(x, lst)	<p>הפונקציה מחברת את העץ ל-self דרך הצומת x. – עידו רוזינר</p>	<p>הפונקציה מניחה כי גובה העץ self שווה לגובה העץ lst ומחברת את העצים דרך הצומת x כנלמד בשיעור (הפונקציה Join מהשיעור). נשים לב כי הפעם אין</p>

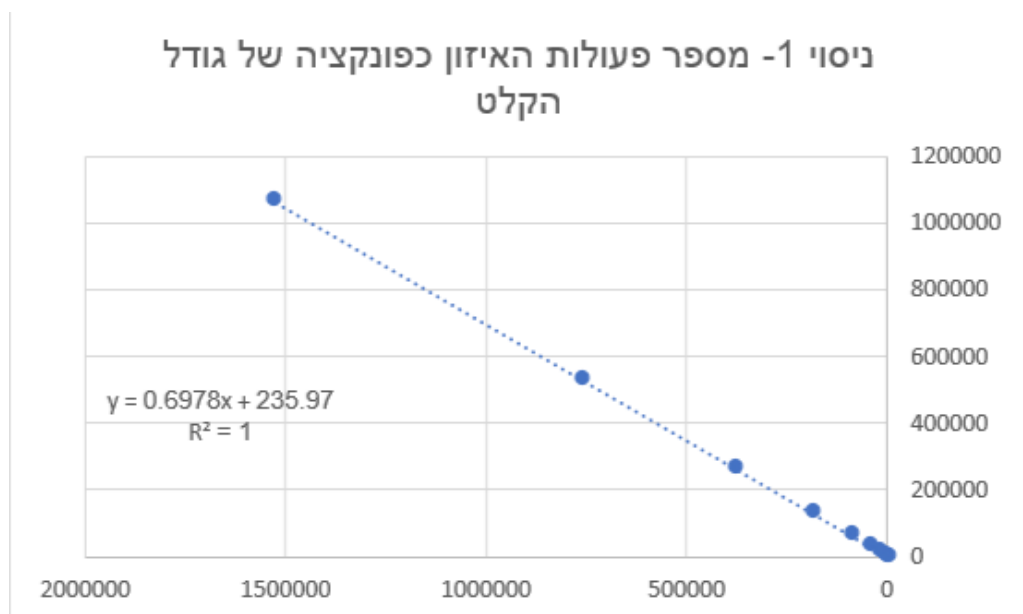
		צורך לרדת במורד אף עץ או לאזן יותר מצומת אחת בעץ, ולכן הפונקציה רצה בסיבוכיות $O(1)$
search(val)	הפונקציה מחזירה את האינדקס הראשון ברשימה בו מופיע הערך val או -1 אם לא קיים אינדקס כזה. – עידו רוזינר	תחילה הפונקציה קוראת ל- listToArray(self) – סיבוכיות $O(n)$. ולאחר מכן הפונקציה רצה בלולאת for על איברי הרשימה – סיבוכיות $O(n)$ סה"כ הפונקציה רצה בסיבוכיות $O(n)$.
rotateRight(node)	הפונקציה מבצעת סיבוב ימינה על node. – תומר רודניצקי	הפונקציה פועלת כמעט כמו שנלמד בהרצאה (לאחר המודיפיקציה של הוספת השדה size ו-height לצמתים). ההבדל היחיד הוא הבדיקה של האם node הוא השורש של העץ. אם node הוא השורש של העץ נעדן את השורש להיות הבן השמאלי של node – סיבוכיות $O(1)$. סה"כ הפונקציה רצה בסיבוכיות $O(1)$.
rotateLeft(node)	הפונקציה מבצעת סיבוב שמאלה על node. – תומר רודניצקי	כמעט כמו שנלמד בהרצאה (לאחר המודיפיקציה של הוספת השדה size ו-height לצמתים). ההבדל היחיד הוא הבדיקה של האם node הוא השורש של העץ. אם node הוא השורש של העץ נעדן את השורש להיות הבן הימני של node – סיבוכיות $O(1)$. סה"כ הפונקציה רצה בסיבוכיות $O(1)$.
rebalanceTree(node)	הפונקציה מאזנת את העץ ומעדכנת את השדות כמו שנלמד בהרצאה. – תומר רודניצקי	הפונקציה זהה לחלק של האיזון של העץ של הפונקציה Delete(D,x) שנלמדה בכיתה (לאחר המודיפיקציה של כל צומת קיים שדה size) – סיבוכיות $O(\log n)$.
treeSelect(i) treeSelectHelper(node, i)	הפונקציות מחזירות את האיבר ה-i הכי קטן בעץ כמו שנלמד בכיתה (לאחר המודיפיקציה של התחלת החיפוש מהאיבר המינימלי). – תומר רודניצקי	הפונקציות זהות לפונקציית Select(D, k) שנלמדה בכיתה (לאחר המודיפיקציה שמתחילים את החיפוש בעץ מהצומת המינימלית) – סיבוכיות $O(\log i)$.
max_node(node)	הפונקציה מחזירה את האיבר המקסימלי בתת-עץ שבו node הוא השורש. – תומר רודניצקי	הפונקציה יורדת מהשורש של התת עץ אל הצומת המקסימלית שלו במסלול ישיר ולכן הסיבוכיות היא $O(\log n)$.

min_node(node)	הפונקציה מחזירה את האיבר המינימלי בתת-עץ שבו node הוא השורש. – תומר רודניצקי	הפונקציה יורדת מהשורש של התת עץ אל הצומת המינימלית שלו במסלול ישיר ולכן סיבוכיות היא $O(\log n)$
predecessor(x)	הפונקציה מחזירה את האיבר הקודם ברשימה. כלומר, אם x הוא האיבר במקום ה-i הפונקציה תחזיר את האיבר ה-i-1. – תומר רודניצקי	הפונקציה זהה לפונקציית Predecessor(D, x) שנלמדה בשיעור - סיבוכיות $O(\log n)$.
successor(x)	הפונקציה מחזירה את האיבר הבא ברשימה. כלומר, אם x הוא האיבר במקום ה-i הפונקציה תחזיר את האיבר ה-i+1. – עידו רוזינר	הפונקציה זהה לפונקציית Successor(D, x) שנלמדה בשיעור - סיבוכיות $O(\log n)$.
buildTreeFromList(lst, left, right)	הפונקציה מקבלת רשימה של ערכים ומחזירה צומת שמהווה שורש לעץ AVLTreeList הנוצר מהרשימה. – עידו רוזינר	כל פעם הפונקציה מבצעת שתי קריאות רקורסיביות על שני חצאים שונים של הרשימה (אנו עושים זאת באמצעות מצביעים כך שלא תיוסף סיבוכיות) ולכן עומק עץ הרקורסיה יהיה $\log n$ כאשר בכל קריאה מתבצעת $O(1)$ עבודה. על כן, סיבוכיות הפונקציה היא $2^{\log n} = O(n)$
merge_sort(lst) merge(lst1, lst2)	הפונקציות מחזירות רשימה מסודרת בסדר עולה. – תומר רודניצקי	הקוד זהה לקוד של mergeSort שלמדנו בקורס מבוא מורחב למדעי המחשב – $O(n \cdot \log n)$
Shuffle(lst)	הפונקציה "מבלגנת" את הרשימה באופן אקראי in-place. – עידו רוזינר	הפונקציה עוברת על הרשימה בלולאות for וכל פעם מחליפה בין שני איברי הרשימה - $O(n)$

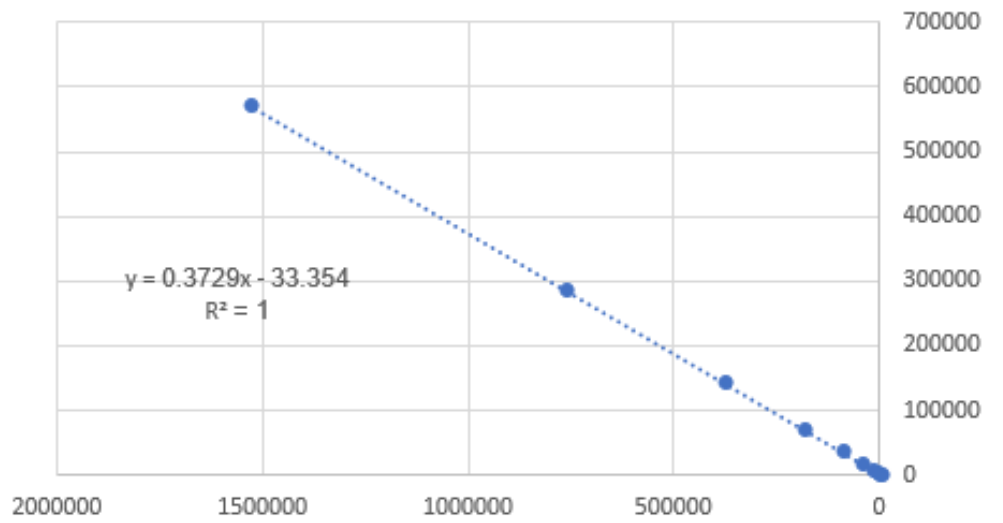
1.1. להלן טבלה המתארת את מספר פעולות האיזון שנדרשו כדי לתקן את העץ בכל ניסוי:

i	ניסוי 1- הכנסות	ניסוי 2- מחיקות	ניסוי 3- הכנסות ומחיקות לסירוגין
1	2146	1146	1830
2	4130	2143	3699
3	8332	4478	7367
4	16817	8933	14625
5	33467	17812	29600
6	67365	35793	58789
7	134828	71788	117405
8	268134	142998	236183
9	537312	286055	471642
10	1071498	572806	942261

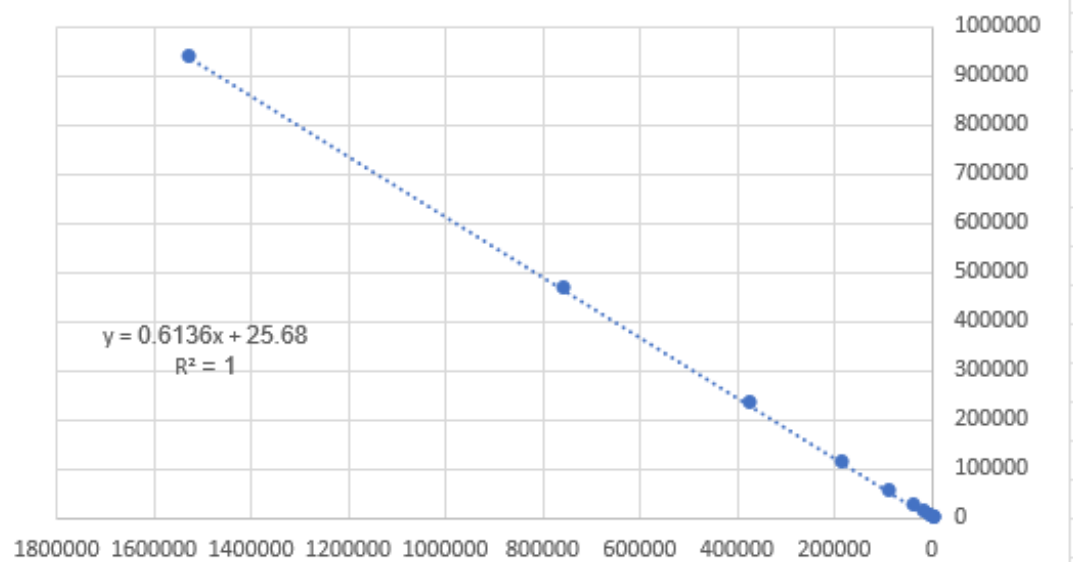
2.1. להלן תרשימים המתארים את תוצאות הניסוי בצורה גראפית. הוספנו קו מגמה, מדד R^2 ומשוואה המתארת בקירוב את הקו. מכל אלה המסקנה היא כי הביטוי האסימפטומטי הוא $O(n)$ בשלושת הניסויים:



ניסוי 2- מספר פעולות האיזון כפונקציה של גודל הקלט



ניסוי 3- מספר פעולות האיזון כפונקציה של גודל הקלט



2. להלן שלושת הטבלאות המתארות את זמן הריצה בממוצע עבור עץ AVL, רשימה מקושרת ועבור מערך:

i	זמן ריצה- עץ AVL הכנסות להתחלה	זמן ריצה- רשימה מקושרת הכנסות להתחלה	זמן ריצה- מערך הכנסות להתחלה
1	3.664356e-05	5.654738e-07	3.295837e-07
2	2.923434e-05	6.545347e-07	5.545469e-07
3	3.192972e-05	6.878794e-07	7.467864e-07
4	3.123943e-05	6.564767e-07	9.565846e-07
5	3.225311e-05	6.747463e-07	1.113233e-06
6	3.487298e-05	6.657457e-07	1.515454e-06
7	3.123936e-05	6.754744e-07	1.724939e-06
8	3.286947e-05	6.336785e-07	1.835562e-06
9	3.112393e-05	6.112436e-07	1.964126e-06
10	3.133183e-05	6.868746e-07	2.021410e-06

i	זמן ריצה- עץ AVL הכנסות אקראיות	זמן ריצה- רשימה מקושרת הכנסות אקראיות	זמן ריצה- מערך הכנסות אקראיות
1	4.129876e-05	1.775654e-05	8.469857e-07
2	3.700732e-05	3.748905e-05	8.965450e-07
3	3.774336e-05	7.969767e-05	1.193485e-06
4	4.012432e-05	9.021049e-04	1.327493e-06
5	4.498348e-05	1.156784e-04	1.344458e-06
6	4.744344e-05	1.331895e-04	1.230485e-06
7	4.425390e-05	1.567349e-04	1.446758e-06
8	5.345783e-05	1.936695e-04	1.546281e-06
9	4.745946e-05	2.188605e-04	1.748480e-06
10	5.449583e-05	2.324536e-04	1.645678e-06

מספר סידורי – i	זמן ריצה- עץ AVL הכנסות בסוף	זמן ריצה- רשימה מקושרת הכנסות בסוף	זמן ריצה- מערך הכנסות בסוף
1	2.802437e-05	5.812048e-07	2.241384e-07
2	2.923845e-05	6.458395e-07	1.934423e-07
3	3.001652e-05	6.835368e-07	1.748259e-07
4	3.168567e-05	6.412367e-07	1.539534e-07
5	3.432853e-05	6.503696e-07	1.934423e-07
6	3.023855e-05	6.436063e-07	1.493735e-07
7	3.359357e-05	6.130633e-07	1.495953e-07
8	3.004939e-05	6.543698e-07	1.495560e-07
9	3.583897e-05	6.663460e-07	1.535794e-07
10	3.668567e-05	6.346602e-07	1.569639e-07

לפני תחילת הניסוי, היינו מצפים כי התוצאות האמיתיות יהיו דומות לאלה שיצאו בסופו של דבר. ברשימה מקושרת, הכנסות לתחילתה ולסופה מתבצעות ב $O(1)$ (זוהי רשימה עם מצביע לסוף) ולכן התקבלו תוצאות מהירות יותר מאשר בהכנסות לסוף ולתחילת עץ AVL. מנגד, בהכנסות אקראיות ברשימה מקושרת "נאלץ" לעבור על חלק גדול מהרשימה בכל הכנסה ($O(n)$ במקרה הגרוע) ולכן זמן הריצה היה איטי יותר. נשאלת השאלה מדוע במערך התקבלו התוצאות הטובות ביותר בכמעט כל המקרים? ובכן, המימוש של פייתון למערך משודרג ויעיל, ככה שקשה להתחרות בו. בהכנסות אקראיות למשל, כשהמערך מתמלא, למדנו בהרצאה כי למרות שמדי פעם ייווצר מערך חדש הגדול פי $1 + \alpha$ (הערך הדיפולטי של α מוגדר להיות $1/8$ בפייתון) מהמערך הישן, דבר שיוביל לפעולה בגודל $O(n)$, סיבוכיות זמן ה-amortized תהיי $O(1)$.

הערה: במידה והיינו ממשים רשימה מקושרת ללא מצביע לסוף, אנו סבורים כי זמן הריצה היה האיטי ביותר בהכנסות לסוף, שכן במצב כזה בכל הכנסה היינו נאלצים לעבור על כל איבר ואיבר ברשימה ($O(n)$ ולא במקרה הגרוע, אלא תמיד!).