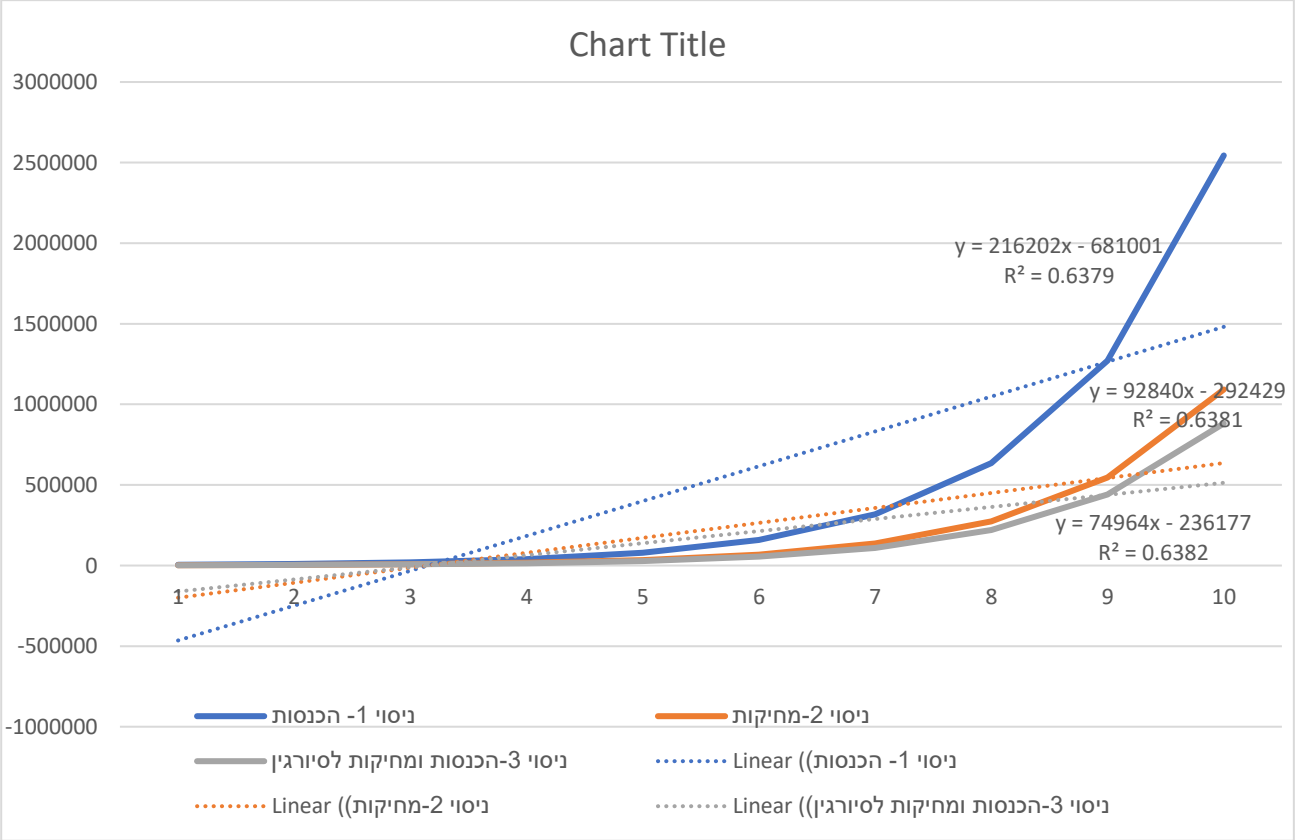
חלק תיאורטי

שאלה 1:

1)

מספר סידורי	ניסוי 3-הכנסות ומחיקות לסיורגין	ניסוי 2-מחיקות	ניסוי 1- הכנסות
1	1678	2121	4926
2	3339	4186	9893
3	6922	8579	19904
4	13797	17010	39642
5	27268	34029	79468
6	55140	68178	158862
7	109372	136498	317583
8	220755	273243	635572
9	441548	545805	1270953
10	881439	1092272	2544299

2) כפי שרואים בטבלה להלן שלושת הניסויים רצים בזמן לינארי כתלות באורך הקלט.



שאלה 2:

1)

max join cost for max in left subtree split	average join cost for max in left subtree split	max join cost for random split	average join cost for random split
11	2	4	1.8
12	2.09	4	1.181
13	2.166	4	1.7272
15	2.153	5	1.454
16	2.23	4	1.846
17	2.285	4	1.928
18	2.333	3	1.411
19	2.533	4	2.117
20	2.055	4	1.722
22	2.052	4	1.928

2)לפי התוצאות הנ"ל אנחנו רואים שבממוצע עבור פעולת split פעולת ה-concat נעשית בממוצע על שני עצים שהפרש הגבהים שלהם הוא לכל היותר 3 שזו העלות של ה-concat שהיא בסיבוכיות של O(1) ובמקרה הגרוע ביותר נעשה concat על כל גובה כלומר סיבוכיות הזמן של split תהיה log(n)*O(1). נשים לב שזוהי סיבוכיות הזמן שהיינו מצפים לה בעת ניתוח הסיבוכיות למעלה זה מה שיצא. והניתוח הזה מתאים לשני המקרים של איבר רנדומלי או של איבר ספיציפי.

3) אם נסתכל על תוצאות join מקסימלי נשים לב שהסיבוכיות היא O(logn) אם לא היינו מקבלים כך(ומקבלים w(logn) היה סותר את ניתוח סיבוכיות הקוד שלנו. אבל מכיוון שה-join הממוצע מתבצע ב-O(1) ולפי ניתוח ה-ammortized time זה לא קורה הרבה פעמים. לכן התוצאות תואמות את הניתוח שלנו.

שאלה 3:

1)

מספר פעולות האיזון בממוצע

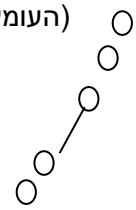
עץ AVL סדרה חשבונית	עץ ללא מנגנון איזון סדרה חשבונית	עץ AVL סדרה מאוזנת	עץ ללא מנגנון איזון סדרה מאוזנת	עץ AVL סדרה אקראית	עץ ללא מנגנון איזון סדרה אקראית
2.974	499.5	0.994	0.994	2.485	2.321
2.986	999.5	0.997	0.997	2.495	2.5275
2.989	1499.5	0.99766	0.99766	2.474	2.441
2.9925	1999.5	0.9985	0.9985	2.522	2.476
2.9938	2499.5	0.999	0.999	2.488	2.4496
2.9945	2999.5	0.998833	0.998833	2.4685	2.4911
2.995	3499.5	0.999	0.999	2.478	2.447
2.996	3999.5	0.99925	0.99925	2.483	2.478
2.9963	4499.5	0.999444	0.999444	2.472	2.385
2.9967	4999.5	0.9995	0.9995	2.4982	2.4612

עומק הצומת המוכנס בממוצע

עץ AVL סדרה חשבונית	עץ ללא מנגנון איזון סדרה חשבונית	עץ AVL סדרה מאוזנת	עץ ללא מנגנון איזון סדרה מאוזנת	עץ AVL סדרה אקראית	עץ ללא מנגנון איזון סדרה אקראית
8.977	499.5	7.987	7.987	8.77	12.183
9.9765	999.5	8.982	8.982	9.7205	13.6955
10.635	1499.5	9.639	9.639	10.363	13.262
10.97625	1999.5	9.97925	9.97925	10.7765	15.2972
11.3618	2499.5	10.6344	10.3644	11.0864	14.446
11.634	2999.5	10.637	10.637	11.323	15.098
11.829	3499.5	10.381	10.381	11.572	14.157
11.976	3999.5	10.977	10.977	11.733	17.276
12.179	4499.5	11.181	11.181	11.951	14.766
12.3617	4499.5	11.3631	11.3631	12.0872	16.0567

נסתכל על הכנסות להתחלה(סדרה חשבונית):

בעץ ללא איזון נצפה שמספר פעולות האיזון בממוצע יהיה שווה לעומק הצומת המוכנס בממוצע שהוא ½(n-1)n (כאשר n הוא מספר הצמתים בעץ) וזאת מכיוון שבעץ כזה פעולות האיזון מורכבות רק משינוי גובה ואין רוטציות כך כשמכניסים צומת להתחלה תמיד נקבל שהעומק של הצומת המוכנס הוא עומק של צומת האבא ועוד אחד כך שהעומק של צומת השורש הוא 0 נמשיך בהכנסות עד שיתקבל עץ בגובה n-1 שנראה כך: (העומק של השורש 0 , בנו 1 , נכדו 2 וכך הלאה) כך שנקבל שסכום העומקים בעץ הוא:



$$\sum_{i=0}^{n-1} \mathrm{i} = \frac{1}{2} * (\mathrm{n}-1) * \mathrm{n}.$$

ובגלל שאין רוטציות כשמכניסים צומת לעץ בהתחלה הגובה של שאר הצמתים יגדל ב-1 ובסוף נקבל שהשורש הוא בעל גובה n-1,בנו בעל גובה n-2 וכך הלאה עד שנגיע לעלה שנמצא בגובה 0. לכן בסוף מס' פעולות האיזון : $\sum_{i=0}^{n-1} \mathrm{i} = \frac{1}{2} * (\mathrm{n}-1) * \mathrm{n}$

לכן בסה"כ אנחנו מקבלים את התוצאות שציפינו.

בעץ AVL אנחנו מצפים לאיזור ה-3 פעולות איזון (ראינו בכיתה) ומכיוון שמדובר בעץ AVL נצפה שהעומק של כל צומת יהיה בערך logn (n מס' הצמתים). כמובן נצפה להפרש גדול במספר פעולות האיזון בין שני סוגי העצים (כי באחד אין רוטציות). ובהסתכל על התוצאות נראה כי הן תואמות לציפיות שלנו.

נסתכל על הכנסות מאוזנות :

בעץ ללא איזון : בגלל אופן הכנסת האיברים (העץ שנקבל הוא תמיד כמעט מושלם או מושלם) לכן תמיד נקבל שגובה העץ הוא מינימלי כלומר $floor(logn)$. ועבור עץ כזה לא נצפה להרבה פעולות איזון עבור כל הכנסה. בנוסף, נצפה שעומק כל צומת בממוצע יהיה logn כי העץ כמעט מושלם או מושלם.

בעץ AVL היינו מצפים שעץ AVL יתנהג כמו העץ בלי פעולות האיזון כי שניהם מוכנסים מאוזנים (כלומר אין צורך לבצע רוטציות בעץ AVL). לכן בסה"כ התוצאות תואמות את הציפיות.

נסתכל על הכנסות אקראיות:

היינו מצפים לפי מה שלמדנו בכיתה שיהיה הפרש משמעותי בין שני סוגי העצים בעת הכנסות אקראיות אבל להפתעתינו קיבלנו שמספר פעולות האיזון בממוצע הן כמעט זהה. לעומת זאת בעומק היינו מצפים שבעץ ללא רוטציות העומק הממוצע יהיה גדול יותר ואכן קיבלנו תוצאות כאלו כאשר ההפרש אינו עצום אבל ניתן לזיהוי.

פונקצייה empty:

מחזירה אם העץ ריק או לא. מתבצעת פעולת השוואה אחת בבדיקת השורש . כלומר סיבוכיות הזמן היא O(1)

פונקציית retrieve:

מתודות עזר:

Tree_select(T,k):מחזירה את האיבר שה-rank שלו הוא k.היא מתבצעת בצורה רקורסיבית ובמקרה הגרוע ביותר המסלול הארוך ביותר שנצטרך לעבור הוא כאורך העץ (בעץ AVL הוא log(n)).ישנן גם פעולות השמות והשוואות שנעשות בסיבוכיות זמן קבועה לכן סה"כ סיבוכיות הזמן של הפונקציה הוא O(log(n)).

retrieve(i): מחזירה את האיבר שנמצא באינדקס ה-i. נשים לב שהאיבר שנמצא באינדקס ה-i הוא מ-rank i+1 לכן הפונקציה מחזירה את פונקציית העזר Tree_select(i+1) . לכן סה"כ סיבוכיות זמן : O(log(n)).

פונקצייה insert :

מתודות עזר :

*update_height : מעדכנת את הגובה עבור כל צומת , מתבצעות פעולות השוואה והשמות בזמן קבוע לכן סיבוכיות זמן הריצה היא O(1) .

*update_size:מעדכנת את שדה ה-size עבור כל צומת, מתבצעות פעולות השוואה והשמה בזמן קבוע לכן סיבוכיות הזמן היא O(1).

*compute_BF:מעדכנים את שדה ה- balance factor עבור כל צומת. נשים לב שהוספנו שדה height לכל צומת בעץ שמעודכן בפונקציית insert.לכן בפונקצייה compute_BF השליפה של המידע לגבי הגובה מתבצעת בזמן קבוע וכל פעולות ההשוואה גם כן לכן סה"כ סיבוכיות זמן הריצה היא O(1).

*Rotate_left/right:פונקציות המבצעות גלגולים מתי שצריך לפי הכללים מהשיעור.בפונקציות אלו מבצעים פעולות השמה ושינויי מצביעים בנוסף לזה יש פעולות השוואה ופעולות אלה מתבצעות בסיבוכיות זמן קבועה.בנוסף לכך מעודכנים שדות הגובה וה-size באמצעות מתודות העזר שפירטנו עליהם קודם ומתבצעות בסיבוכיות זמן קבועה.לכן סה"כ סיבוכיות הזמן היא O(1).

*insert(i,s):פעולת השמת איבר חדש בעץ avl באינדקס i בעל ערך s. לפני הכניסה ללולאת ה- while שעולה במעלה העץ עד השורש מתבצעות פעולות השוואה והשמה בסיבוכיות זמן קבועה ובנוסף אנחנו משתמשים בפונקציה empty בעלת סיבוכיות זמן קבועה ובנוסף , מתבצעות לכל היותר שתי קריאות לפונקציה retrieve שסיבוכיות הזמן שלה היא O(log(n)). ניכנס ללולאת ה-while :בנוסף לפעולות ההשמות וההשוואות שמתבצעות בזמן קבוע אנחנו משתמשים בפונקציות עזר Rotate_left/right ו-update_height (סיבוכיות זמן קבועה).הלולאה מתבצעת לכל היותר כאורך העץ (שהוא log(n) בעץ AVL) לכן סה"כ סיבוכיות הזמן של הלולאה הוא O(log(n)).

סה"כ סיבוכיות הזמן של הפונקציה insert: O(log(n)).

פונקצייה Delete(i):

מתודות עזר:

*השתמשנו בפונקציות update_height,update_size,compute_BF,Rotate_left/right שמעדכנות את שדות הגובה,size,balance factor ומבצעות גלגולים. עליהם הסברנו בדף הקודם וסיבוכיות הזמן שלהם קבועה.

*delete_leaf: מוחקת את הצומת אם הוא עלה,מתבצעות פעולות השמה והשוואות בלבד לכן מתבצעת בסיבוכיות זמן קבועה.

*delete_1child:מוחק את הצומת אם יש לן בן אחד בלבד, מתבצעות פעולות השמה והשוואות בלבד לכן מתבצעת בסיבוכיות זמן קבועה.

*delete_2child: מוחק את הצומת אם יש לו שני בנים,בנוסף לפעולות ההשמות וההשוואות מתבצעת פעולת retrieve בסיבוכיות זמן O(log(n)). לכן סה"כ: O(log(n)).

*לפני הכניסה ללולאת ה-while שעולה מהמקום המתאים עד השורש: השתמשנו בהשמות והשוואות שמתבצעות בסיבוכיות זמן קבועה, ובנוסף השתמשנו בפונקציה retrieve לכל היותר 3 פעמים שמתבצעת בסיבוכיות זמן של O(logn(n)). בנוסף השתמשנו בפונקציות העזר delete_leaf/1child/2child שעולות O(1) זמן.

*בתוך לולאת ה-while: בנוסף לפעולות ההשמות וההשוואות שמתבצעות בסיבוכיות זמן קבועה אנו משתמשים בפונקציות עזר Rotate_left/right ו-update_height שכפי שהסברנו מתבצעות בסיבוכיות זמן של O(1). הלולאה מתבצעת לכל היותר כאורך העץ(שהוא log(n) בעץ AVL).לכן סה"כ סיבוכיות זמן של הלולאה היא O(log(n)).

סה"כ סיבוכיות זמן של הפונקציהdelete:O(log(n)).

פונקציית first:

מחזירה את האיבר הראשון ברשימה.

נשים לב שעבור כל עץ הוספנו את התכונה first כך שבכל הכנסה או מחיקה שדה- first מעודכן לכן עלות השליפה היא O(1).

פונקציית last:

מחזירה את האיבר האחרון ברשימה.

נשים לב שעבור כל עץ הוספנו את התכונה last כך שבכל הכנסה או מחיקה שדה- last מעודכן לכן עלות השליפה היא O(1).

פונקציית listToArray:

ביצענו טיול in-order כלומר התחלנו מתת העץ השמאלי , השורש ואז תת העץ הימני.טיול in-order על העץ נעשה ב-O(n) כי אנחנו עוברים על כל קשת פעמיים (יורדים ואז עולים) לכן סה"כ סיבוכיות הזמן היא O(n).

פונקציית length:

מחזירה את אורך הרשימה.

נשים לב שעבור כל עץ הוספנו את התכונה length כך שבכל הכנסה או מחיקה שדה- length מעודכן לכן עלות השליפה היא O(1).

פונקציית split(i):

מתודות עזר:

Find_first/last: שתי פונקציות שאחת מוצאת את האיבר הכי שמאלי(הראשון) והכי ימני(האחרון) בעץ נתון.כל אחת מהפונקציות הולכת כל הזמן שמאלה\ימינה עד שנגיע לבן ורטואלי. סיבוכיות הזמן של הפונקציות האלו הן לכל היותר כאורך העץ O(h).

Is_right/left: מחזירה true אם הצומת היא בן ימני.שמאלי. סיבוכיות זמן O(1).

Update_fields: מבצעת עדכון לשדות ה-first/last/length/root עבור עץ נתון . מתבצעים אך ורק שינוי מצביעים לכן סיבוכיות זמן קבועה של O(1).

*לפני הכניסה ללולאת ה-while : אנחנו מטפלים במקרי הבסיס(הספליט הוא מהאינדקס הראשון או האחרון או שהאיבר שנמצא באינדקס הנתון הוא השורש). בנוסף להשמות ,ההשוואות ופונקציות העזר שמתבצעות בסיבוכיות זמן קבועה אנחנו משתמשים בפונקציות delete ו-retrieve שכבר הסברנו שסיבוכיות הזמן שלהן היא O(log(n)). לכן סה"כ סיבוכיות הזמן לפני הכניסה ללולאה היא O(log(n)).

*בתוך הלולאה : אנחנו בונים את תתי העצים bottom-up עבור כל תת עץ פעולת האיחוד concat לוקחת זמן של O(הפרש הגבהים של תתי העצים אותם אנחנו מאחדים).לכן הסיבוכיות הכללית היא סכום האיחודים, נשים לב שהאירוע של הפרש גדול בין הגבהים אינו קורה הרבה לכן קיים איזון בין הרבה איחודים גדולים למעט איחודים זולים וסה"כ נקבל (בנוסף להשמות,השוואות ופונקציות העזר שלוקחים זמן קבוע): $O(\sum|height(ti) - height(concat(t1, \dots, ti-1)| + 1)$ כאשר הסכום רץ מ-2 עד k כאשר k מסמל את תת העץ ה-k. ולפי ההרצאה נקבל סיבוכיות זמן של O(log(n)).

לכן סה"כ סיבוכיות הזמן של split היא:O(log(n)) .

פונקציית concat(lst):

מתודות עזר:

Update_fields: מבצעת עדכון לשדות ה-first/last/length/root עבור עץ נתון . מתבצעים אך ורק שינוי מצביעים לכן סיבוכיות זמן קבועה של O(1).

*מטפלים קודם כל במקרי הבסיס(אחד או שני העצים ריקים) והם מבוצעים בסיבוכיות זמן קבועה.

*לפני הכניסה ללולאת ה-while (שמבצעת גלגולים) מתבצעות פעולות של השמות,השוואות ופונקציית העזר update_fields לאחר מכן ניכנס ללואת while שבה אנחנו מחפשים את האיבר הראשון ברשימה שאנחנו רוצים לשרשר שהגובה שלו קטן\שווה לגובה של העץ שלנו. לכן סיבוכיות הזמן היא O(h). לאחר מכן מתבצעות פעולות השוואה,השמה ופונקציית העזר שכולן כאמור לוקחות זמן קבע. אנחנו מטפלים בשני המקרים שהעץ השני גדול\שווה לראשון או שהעץ השני קטן מהעץ הראשון).לאחר מכן ניכנס ללולאת while שמבצעת את האיזון בעזרת גלגולים : בנוסף לפעולות ההשמות וההשוואות שמתבצעות בזמן קבוע אנחנו משתמשים בפונקציות עזר Rotate_left/right ו-update_height (סיבוכיות זמן קבועה).הלולאה מתבצעת לכל היותר כאורך העץ (שהוא log(n) בעץ AVL) לכן סה"כ סיבוכיות הזמן של הלולאה הוא O(log(n)).

נקבל שסה"כ סיבוכיות הזמן של הפונקציה concat היא O(log(n)).

פונקציית search(val):

אנחנו בונים מערך מתאים מהרשימה בעזרת הפונקציה listToArray שמימשנו מקודם והיא לוקחת בסה"כ סיבוכיות זמן של O(log(n)), לאחר מכן אנחנו עוברים על המערך כדי למצוא את האינדקס של האיבר בעל הערך המתאים לכן סיבוכיות הזמן היא ככל היותר כאורך הקלט כלומר O(n).