עמרי יקיר 318867199
מאיה רייטן 209085711

מבני נתונים - פרויקט מספר 1 - עץ דרגות

name: Omri Yakir, username: omriyakir, id: 318867199
name: Maya Raytan, username: mayaraytan, id: 209085711

חלק מעשי - תיעוד פונקציות

למחלקה AVLTree שדות:
- IAVLNode root
- IAVLNode max
- IAVLNode min
- private static final IAVLNode EXTERNAL_LEAF = AVLNode.EXTERNAL_LEAF

1. **<u>Public Boolean empty()</u>**
הפונקציה מחזירה אמת אם ערך השדה שורש הוא null אחרת שקר.
סיבוכיות זמן הריצה: $\boldsymbol{O(1)}$.

2. **<u>Public String search(int k)</u>**
הפונקציה מחזירה את ערך הvalue של הnode עם המפתח k , אם לא קיים מפתח היא מחזירה null.
הפונקציה עושה שימוש בnode_search.
סיבוכיות זמן הריצה: כמו node_search, כלומר $\boldsymbol{O(logn)}$.
   a) **Private IAVLNode node_search(IAVLNode x,int k)**
   הפונקציה ממומשת ע"פ הגרסה האיטרטיבית של Tree-Search.
   סיבוכיות הזמן הריצה: כפי שראינו בכיתה: $\boldsymbol{O(logn)}$.

3. **<u>Public int insert(int k ,String i)</u>**
הפונקציה מחפשת איבר בעל מפתח k.
- אם קיים איבר כזה, היא מחזירה את הערך השמור עבורו , אחרת תחזיר null.
- במידה והעץ ריק הפונקציה מעדכנת את שדה השורש להיות צומת בעלת מפתח וערך k ו i בהתאמה. ומעדכנת את שדה size של השורש להיות 1.
- אחרת : מכניסה לעץ את הצומת המתאים בעזרת tree_insert (סיבוכיות $O(logn)$), מעדכנת את גדלי העץ בעזרת update_sizes_till_root (סיבוכיות $O(logn)$), ועושה rebalance לעץ כדי שיהיה עץ AVL תקין בעזרת rebalance (סיבוכיות $O(logn)$).
אם האיבר שהכנסנו קטן מהמינימאלי או גדול מהמקסימאלי – הפונקציה מעדכנת בהתאם את שדות min,max של העץ.
סיבוכיות זמן הריצה: $O(3logn) = \boldsymbol{O(logn)}$.
הפונקציה עושה שימוש בפונקציות העזר:
   a) **private Boolean tree_insert(IAVLNode x, IAVLNode z)**
   הפונקציה מבצעת הכנסה של הצומת z לתוך תת העץ x במידה והכנסה זו חוקית. מחזירה אמת אם ההכנסה הייתה חוקית, אחרת שקר. הפונקציה ממומשת לפי האלגוריתם שראינו בכיתה. משתמשת בtree_position (סיבוכיות $O(logn)$). מעבר לכך עושה פעולות קבועות.
   b) **private IAVLNode tree_position(IAVLNode x,int k)**
   הפונקציה מחפשת צומת עם המפתח K בתת העץ x, ומחזירה את הצומת האחרונה בה נתקלה. אם תת העץ הוא null היא תחזיר null. הפונקציה ממומשת ע"פ האלגוריתם שראינו בכיתה. ראינו כי הסיבוכיות זמן שלה היא $O(logn)$.
   c) **private void update_sizes_till_root(IAVLNode x)**
   הפונקציה מקבלת צומת x. כל עוד הוא שונה מ-null, הפונקציה תקרא לפונקציה update_size עם צומת זה ולאחר מכן תקדם את x להיות ההורה שלו.
   סיבוכיות זמן הריצה: $O(logn)$ כיוון שמתבצע מעבר על גובה העץ.
      i. **private void update_size(IAVLNode x)**

הפונקציה מקבלת צומת x ומעדכנת לו את שדה הגודל בהתאם לכך:
-אם הצומת הוא עלה חיצוני: גודלו 1-.
-אם הצומת הוא עלה: גודלו 0.
אחרת: גודל הצומת שווה לגודל צמתי הבנים שלו ועוד 1.
בפונקציה זו מתבצעות פעולות בעלות קבועה כמו גישה לשדות והשמות ולכן סיבוכיות הפונקציה היא $O(1)$.

d) public int rebalance (IAVLNode p)

הפונקציה מקבלת צומת בעץ, p. ומבצעת עליה מבצעת את פעולות האיזון הדרושות כדי להכשיר את העץ לאחר ההכנסה, ומחזירה את מספר פעולות האיזון שבוצעו. הפונקציה תחילה בודקת עבור הצומת שהיא קיבלה את סוג הצומת שהיא (הפרש הדרגות עם הילדים שלה) בעזרת הפונקציה rank_type, ולאחר מכן בודקת אם היא צומת חוקית. במידה ולא היא מתקנת את צומת לפי הcases שראינו בכיתה. אם נדרש, ה-caseים הרלוונטיים יקראו לפונקציות: double_rotate_left ,double_rotate_right,rotate_left,rotate_right בהן יתבצעו הגלגולים כפי שנלמד בכיתה בסיבוכיות קבועה. בסיום case היא בודקת את הדרגה של ההורה וחוזר חלילה. לאחר כל case אנו סוכמים את מספר הrebalance_steps, אותם נחזיר בסיום התיקון. כמות פעולות התיקון הן לכל היותר $O(logn)$ כפי שראינו בכיתה , וכל תיקון מתבצע בזמן קבוע , לכן במקרה הגרוע הסיבוכיות היא $O(logn)$. הפונקציה משתמשת בפונקציה ספציפית לכל case ומבצעת את השינויים הדרושים כפי שראינו בכיתה, בסיבוכיות $O(1)$.
לכן הסיבוכיות הכוללת של rebalance היא: $O(logn)$.

.i private static String rank_type(IAVLNode x)

הפונקציה בודקת מה הפרש הגבהים של הקשתות של הצומת הרלוונטית ומחזירה מחרוזת בה האיבר הראשון הוא הפרש הגבהים של הקשת השמאלית והאיבר השני הוא הפרש הגבהים של הקשת הימנית. פונקציה זו פועלת בסיבוכיות קבועה כיוון שרק ניגשת לשדות.

.4 <u>Public int delete(int k)</u>

הפונקציה מחפשת את הצומת בעל מפתח k בעץ באמצעות המתודה node_search. אם צומת זה לא נמצא בעץ - הפונקציה מחזירה 1-. אחרת: הפונקציה שומרת את הקודם והעוקב של צומת זה.
הפונקציה תקרא לפונקציית העזר delete_rec והיא זו שתחשב את עלות ה-delete.
לסיום נבדוק אם הצומת שמחקנו הוא המקסימלי או המינימאלי בעץ ובהתאמה נעדכן את המקסימום בעץ לקודם של הצומת שמחקנו ואת המינימום לעוקב של הצומת שמחקנו. פעולות אלו בעלות $O(logn)$ כסיבוכיות של הפונקציות find_successor, find_predeccessor.
נחזיר את עלות ה-delete.
סיבוכיות זמן הריצה: $\boldsymbol{O(logn)}$ (כסיבוכיות פונקציות העזר: קודם, עוקב, delete_rec).

a) <u>Public int delete_rec(int k)</u>

הפונקציה מחפשת את הצומת בעל מפתח k בעץ באמצעות המתודה node_search. אם צומת זה לא נמצא בעץ - הפונקציה מחזירה 1-. אחרת:
-אם זהו האיבר היחיד בעץ: הפונקציה תשנה עץ זה לעץ ריק בעזרת המתודה setEmpty() שמשנה את המצביע של שורש העץ ל-null. הפונקציה תחזיר 0.
-אם האיבר שאנו רוצים למחוק הוא עלה: ראשית נבדוק האם צומת זה הוא עלה באמצעות המתודה (isLeaf(IAVLNode x. לאחר מכן נחליף אותו בעלה חיצוני (EXTERNAL_LEAF).
-אם האיבר שאנו רוצים למחוק הוא צומת אונרי בעל בן יחיד: נחליף אותו בבן השמאלי שלו (אם קיים) ואם לא אז בבן הימני שלו.
-אם לאיבר שאנו רוצים למחוק יש שני בנים: נמצא את היורש שלו ונמחק אותו באמצעות קריאה רקורסיבית לפונקציה זו והכנסת המפתח של היורש. כשחוזר לריצה המקורית של הפונקציה נחליף את הצומת שאנו רוצים למחוק ביורש שלו.
בסיום ההחלפות - נחשב את מספר פעולות האיזון הנדרשות באמצעות קריאה לפונקצייה rebalance. הפונקציה תחזיר את פעולות האיזון שנדרשו לביצוע פעולת המחיקה.

סיבוכיות זמן הריצה הכולל:
הפונקציות: setEmpty, isLeaf פועלות בעלות קבועה.
פונקציית rebalance פועלת ב-$O(logn)$ ב-w.c. פונקציית change_node פועלת ב-$O(logn)$ ב-w.c. הקריאות לפונקציות אלו מתבצעות אחת אחר השנייה ולכן הסיבוכיות הכוללת של delete היא $O(logn)$.

פונקציות עזר:

i. **private void setEmpty()**
הפונקציה מגדירה את השורש של העץ כ-null ובכך הופכת את העץ לריק.
סיבוכיות הזמן הריצה: $O(1)$.

ii. **private void change_node(IAVLNode x, IAVLNode y)**
הפונקציה מחליפה את צומת x לצומת y באמצעות שינוי מצביעים רלוונטיים. הפונקציה מגדירה את ההורים של x להיות ההורים של y, ואת הבנים של x להיות הבנים של y. פעולות אלו בסיבוכיות קבועה. בנוסף, פונקציה זו מבצעת קריאה ל-update_sizes_till_root ומכניסה כקלט את צומת y החדש. פונקציית שינוי גדלי הצמתים רצה ב-$O(logn)$ ב-w.c.
סיבוכיות: $O(logn)$.

iii. **private IAVLNode find_successor(IAVLNode x)**
הפונקציה מוצאת את האיבר העוקב של x לפי סדר המפתחות. הפונקציה פועלת לפי האלגוריתם שראינו בכיתה: אם ל-x יש בן ימני, הפונקציה נעזרת בפונקציית העזר min_node כדי למצוא את המינימום בתת העץ של הבן הימני של x. אחרת: הפונקציה תתקדם להורה כל עוד הבן הקודם הוא הצומת הימני של ההורה.
סיבוכיות: $O(logn)$.

- **private IAVLNode min_node(IAVLNode x)**
אם x ריק או צומת וירטואלי, הפונקציה תחזיר null. אחרת: הפונקציה מתקדמת אל הבן השמאלי כל עוד הוא קיים. לסיום הפונקציה תחזיר את הבן השמאלי ביותר בתת העץ של x. **סיבוכיות זמן הריצה:** $O(logn)$ ב-w.c.

iv. **private IAVLNode find_predeccessor(IAVLNode x)**
הפונקציה מוצאת את האיבר הקודם של x לפי סדר המפתחות. הפונקציה פועלת בסימטריה לפונקצייה find_successor.
סיבוכיות: $O(logn)$.

- **private IAVLNode max_node(IAVLNode x)**
אם x ריק או שאינו איבר אמיתי, הפונקציה תחזיר null. אחרת: הפונקציה מתקדמת אל הבן הימני כל עוד הוא קיים. לסיום הפונקציה תחזיר את הבן הימני ביותר בתת העץ של x.
סיבוכיות זמן הריצה : $O(logn)$ ב-w.c.

5. **<u>public String min()</u>**
אם העץ ריק- הפונקצייה תחזיר null, אחרת הפונקציה תחזיר את הערך של השדה min של העץ.
סיבוכיות זמן הריצה היא $O(1)$ כיוון שניגשים לשדה של העץ.

6. **<u>public String max()</u>**
אם העץ ריק- הפונקצייה תחזיר null, אחרת הפונקציה תחזיר את הערך של השדה max של העץ.
סיבוכיות זמן הריצה היא $O(1)$ כיוון שניגשים לשדה של העץ.

7. **<u>public int [] keysToArray()</u>**
הפוקנציה מחזירה מערך ממויין של מפתחות העץ. הפונקציה עושה שימוש בפונקציית העזר in_order_arr, אשר מחזירה מערך ממויין של הצמתים בעץ (לפי מפתחות), סיבוכיות (O(logn.

a) **protected IAVLNode [] in_order_arr(IAVLNode n)**
הפונקציה מקבלת צומת n ומחזירה מערך של הצמתים בתת העץ של n במעבר in-order , כלומר במקרה של עץ החיפוש שלנו, מערך ממוין לפי מפתחות. הפונקציה מאתחלת שני מערכים, arr

שישמור את צמתים בעץ לפי סדר in-order, והשני arr_stack שיתפקד כמחסנית. תאתחל מצביע x להיות n.המשתנים i וcnt יחזיקו את אינדקס האיבר האחרון בarr וarr_stack בהתאמה. כעת נרוץ בלולאה כל עוד מערך המחסנית אינו ריק או x הוא צומת אמיתית. עבור x שהוא צומת אמיתית נכניס אותו למערך המחסנית ונלך לבנו השמאלי, אחרת (כאשר x הוא עלה וירטואלי) נוציא את האיבר האחרון ממערך המחסנית, נחזיר אותו , ונעבור לבנו הימני. בצורה זו נעבור על כל צומת בעץ לכל היותר פעמיים (פעם ראשונה כשנכניס אותה למערך המחסנית ופעם שנייה שנכניס אותה למערך הצמתים) לכן סה"כ הסיבוכיות היא O(logn).

8. **public int [] infoToArray()**

הפונקציה עובדת בדומה לפונקציה keysToArray, רק שהיא מחזירה מערך ערכי הצמתים שממויין לפי גודל המפתחות. סיבוכיות הפונקציה O(logn) כסיבוכיות keysToArray.

9. **public int size()**

הפונקציה בודקת אם העץ ריק- אם כן תחזיק 0. אחרת תיגש לשדה size של שורש העץ ותחזיר אותו.
סיבוכיות זמן הריצה: : $O(1)$.

10. **public AVLTree[] split(int x)**

הפונקציה מקבלת מפתח של צומת מתוך העץ ומחזירה שני עצים , האחד של כל צמתי העץ בעלי מפתח קטן מ x והשני כל צמתי העץ בעלי מפתח גדול מ-x.
הפונקציה מאתחלת עץ big לצמתים הגדולים מ-x, ועץ small לצמתים הקטנים מ-x.
תחילה הפונקציה מוסיפה את תת העץ של הבן הימני ואת תת העץ של הבן השמאלי של הצומת עם המפתח x ו big ולsmall בהתאמה.
אחר כך היא מטפסת עד לשורש , ובכל עליית שלב , אם היא מגיעה מצד ימין נבצע פעולת join ל big עם מפתח הצומת אליה הגענו ועם תת העץ השמאלי של הצומת אליה הגענו. אם היא מגיע מצד שמאל נעשה אותו דבר באופן סימטרי.
לפני החזרת מערך העצים, הפונקציה מעדכנת את שדות ה-min,max של big,small להיות המינימום או המקסימום של העץ המקורי, או האיבר הקודם או העוקב של צומת x (צומת הפיצול). העדכון יתבצע בעלות $O(logn)$ בהתאם לחיפוש הקודם והעוקב של צומת x.
סיבוכיות זמן הריצה: ראינו בכיתה כי סיבוכיות זמן הריצה היא: $O(rank(t_k) - rank(t_1) + k) =$ $O(logn)$. האלגוריתם שביצענו זהה לאותו אחד שראינו בכיתה לכן הסיבוכיות נשארת.

11. **public int join(IAVLNode x, AVLTree t)**

הפונקציה בודקת את התנאים:
-אם שני העצים ריקים: נגדיר את השורש העץ כ-x, נגדיר את שדות min,max של העץ כ-x.
-אם אחד העצים ריק והשני לא- נחבר את x לעץ הלא ריק באמצעות פונקציות העזר join_x_to_t1, join_x_to_t2. נבדוק האם המפתח של העץ שאנחנו מחברים ל-x קטן ממנו או לא ובהתאמה נקרא לפונקציות העזר. בנוסף נעדכן את שדות ה-min,max של העץ בהתאמה להאם x גדול מהעץ הריק או קטן ממנו.
-אחרת: נבדוק האם גובה העץ הנוכחי גדול, קטן או שווה לגובה העץ t והאם המפתח של שורש העץ הנוכחי קטן או גדול מ-x בהתאם לכך נקרא לפונקציות העזר join_t1_is_higher ,join_t1_is_higher, join_same_height. פונקציות אלו מבצעות את החיבורים הרלוונטיים בין העצים לצומת x כפי שלמדנו בכיתה. נעדכן את שדה ה-min של העץ להיות שדה ה-min של העץ שהמפתחות שלו קטנים יותר ואת שדה ה-max של העץ שלנו להיות שדה ה-max של העץ שהמפתחות שלו גדולים יותר.
לסיום נחשב את הפרש הגבהים של העצים בתוספת 1 ונחזיר ערך זה.
הפעולות בפונקציה זו שאינן קוראות לפונקציות העזר בעלות **:** $O(1)$ ולכן:
סיבוכיות זמן הריצה: כעלות פונקציות העזר: $O(logn)$.
פירוט על פונקציות העזר:

a) **private void join_x_to_t1(IAVLNode x, AVLTree t1),**
private void join_x_to_t2(IAVLNode x, AVLTree t2)

פונקציה זו מחברת בין צומת x לעץ היחיד שאינו ריק. נקרא לפונקציה join_x_to_t1 אם

המפתחות בעץ שאינו ריק קטנים מהמפתח של x ובמקרה ההפוך נקרא לפונקציה join_x_to_t2.
פונקציות אלו מבצעת קריאה לפונקציות findNodeOnLeftByHeight,
findNodeOnRightByHeight אם מתחברים לעץ שהמפתחות שלו גדולים מ-x או קטנים
בהתאמה. בעזרת פונקציות אלו נחפש את הצומת בעץ שנדרש לחבר אליה את x. פעולה זו
בסיבוכיות $O(logn)$.
נבצע השמות לשדות ההורים והבנים של צמתים כדי לחבר את x לצומת החיבור ולהורה שלה.
נגדיר את שורש העץ הנוכחי שלנו כשורש t1 או t2 בהתאמה לפונקציות. פעולות אלו בסיבוכיות
קבועה משום שמדובר בשינוי מבציעים בלבד.
נעדכן את הגדלים של הצמתים לאחר החיבורים החדשים באמצעות הפונקציה
update_sizes_till_root החל מ-x ועד לשורש. פעולה זו בסיבוכיות $O(logn)$.
לסיום נבצע פעולות איזון באמצעות בפונקציה rebalance החל מההורה של x והלאה אם נדרש.
פעולות האיזון בסיבוכיות $O(logn)$.
סיבוכיות זמן הריצה: כלל הפעולות ב- $O(logn)$.

i. **private IAVLNode[] findNodeOnLeftByHeight(int k)**
private IAVLNode[] findNodeOnRightByHeight(int k)
פונקציות אלו מחזירות מערך עם הצומת הראשון מגובה קטן או שווה ל-k וההורה שלו
בעץ בצד השמאלי של העץ או הימני בהתאמה לשם הפונקציה. במהלך ריצת הפונקציות
מתבצע מעבר מהשורש ועד הצומת המתאים באמצעות מעבר על הבנים הימניים /
שמאליים של השורש. במקרה הגרוע הצומת שאנו מחפשים הוא שורש או עלה חיצוני
ולכן נבצע טיול על גובה העץ. כלומר:
סיבוכיות זמן הריצה: $O(logn)$.

b) **private void join_t2_is_higher(AVLTree t1, IAVLNode x, AVLTree t2)**
private void join_t1_is_higher(AVLTree t1, IAVLNode x, AVLTree t2)
private void join_same_height(AVLTree t1, IAVLNode x, AVLTree t2)
פונקציית join המקורית תזין את הקלטים לפונקציות אלו כך שמתקיים:
$t1.root.key < x.key < t2.root.key$. הקריאה לכל פונקציה תתבצע לפי הפרשי הגבהים
של העצים t2 ו-t1. אם הגבהים שלהם זהים- נחבר את x לשורשים של t1 ו- t2, נעדכן את הגובה
של x ונבצע פעולות איזון על x.
אם t2 גבוה מ-t1: נחפש את צומת החיבור באמצעות findNodeOnLeftByHeight בעלות
$O(logn)$. צומת החיבור יהיה האיבר הראשון ברשימה המוחזרת. צומת זה יהיה הבן הימני של x.
הבן השמאלי של x יהיה t1. ההורה של x יהיה ההורה המקורי של צומת החיבור. שינויי המצביעים
בעלות קבועה. נגדיר את שורש העץ הנוכחי להיות השורש של t2. נעדכן את גבהי הצמתים החל
מ-x ועד השורש באמצעות הפונקציה update_sizes_till_root בעלות $O(logn)$.נבצע פעולות
איזון באמצעות הפונקציה rebalance החל מההורה של x ועד השורש אם נדרש.
אם t1 גבוה מ-t2 נקרא לפונקציה join_t1_is_higher שפועלת באופן סימטרי לפונקציה
שמתוארת לעיל.
סיבוכיות זמן הריצה: $O(logn)$.

12. **public IAVLNode getRoot()**
הפונקציה מחזירה את השדה root של העץ.
סיבוכיות זמן הריצה: : $O(1)$.

<u>המחלקה AVLNode:</u>
למחלקה שדות left,right,parent מטיפוס IAVLNode, שמייצגים את הבן השמאלי , הימני והאב של הצומת
בהתאמה.
שדות rank, key, size מטיפוס int שמייצגים את גובה ,מפתח, וגודל הצומת בהתאמה.
שדה info שמייצג את ערך הצומת.
למחלקה שני בנאים, בנאי ראשון, ריק, איתו מאתחלים את ה-EXTERNAL_LEAF.
בנאי שני, אשר מקבל ערך מפתח וערך מחרוזת ומעדכן את שדות העץ בהתאמה, מעדכנת את גודל העץ להיות 1,
ואת שני הבנים של הצומת להיות עלים וירטואליים.

1. **Public int getkey():**
הפונקציה מחזירה את המפתח של הצומת. אם הוא צומת וירטואלי תחזיר 1.
סיבוכיות: O(1).
2. **Public String getValue():**
הפונקציה מחזירו את ה-info של הצומת או null, אם זה צומת ויראטולי.
סיבוכיות: O(1).
3. **Public void getLeft():**
הפונקציה מחזירה את הבן השמאלי של הצומת, או null אם אין כזה.
סיבוכיות: O(1).
4. **Public void getRight():**
הפונקציה מחזירה את הבן הימני של הצומת, או nullאם אין כזה.
סיבוכיות: O(1).
5. **Public Boolean isRealNode():**
הפונקציה תחזיר True אם הצומת הוא אמיתי.
סיבוכיות: O(1).
6. **Public int getHeight():**
הפונקציה מחזירה את גובה הצומת, ומחזירה 1- עבור בן וירטואלי.
סיבוכיות: O(1).

<u>פונקציות שהוספנו</u>:

7. **Public void setLeft(IAVLNode node):**
הפוקנציה מעדכנת את ערך הבן השמאלי של הצומת להיות node.
סיבוכיות: O(1).
8. **Public void setRight(IAVLNode node):**
הפוקנציה מעדכנת את ערך הבן הימני של הצומת להיות node.
סיבוכיות: O(1).
9. **Public IAVLNode getparent():**
הפונקציה מחזירה את צומת האב של הצומת שלנו.
סיבוכיות O(1).
10. **Public void SetParent(IAVLNode node):**
הפונקציה מעדכנת את ערך האב של הצומת שלנו להיות node.
סיבוכיות: O(1).

תוספות למנשק:

1. **Public void setHeight(int height):**
הפוקנציה מעדכנת את גובה הצומת.
סיבוכיות: O(1).
2. **Public void setSize(int s):**
הפוקנציה מעדכנת את ערך הגודל של הצומת להיות s.
סיבוכיות: O(1).
3. **Public int getSize():**
הפונקציה מחזירה את ערך הגודל של הצומת.
סיבוכיות: O(1).
4. **Public void promote():**
הפונקציה מעלה את ערך הגובה של הצומת ב-1.
סיבוכיות: O(1).
5. **Public void demote():**
הפונקציה מורידה את ערך הגובה של הצומת ב-1.
סיבוכיות: O(1).
6. **Public Boolean isLeaf():**
הפונקציה בודקת אם הצומת היא עלה , ואם כן מחזירה true, אחרת מחזירה false.
סיבוכיות: O(1).

חלק ניסויי/תיאורטי

__שאלה 1:__

א.

מספר סידורי i	מספר החילופים במערך ממוין- הפוך	עלות החיפושים במיון AVL עבור מערך ממוין הפוך	מספר חילופים במערך מסודר אקראית	עלות החיפושים במיון AVL עבור מערך מסודר אקראי
1	1999000	38884	981987	33815
2	7998000	85764	4002906	78228
3	31996000	187524	16010421	167956
4	127992000	407044	64077931	365300
5	511984000	878084	255445483	808819

ב. עבור מערך הפוך מתקיים כי לכל i, $j > i$, $a[i] > a[j]$, כלומר מספר החילופים הוא כמספר הזוגות של האינדקסים השונים במערך:

$$\binom{n}{2} = \frac{n!}{2!\,(n-2)!} = \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \boldsymbol{O(n^2)}$$

בכל הכנסה של איבר מהרשימה לעץ, האיבר הנכנס יהיה הקטן בעץ. באלגוריתם ההכנסה שמשתמש בחיפוש מהאיבר המקסימלי, נטפס מצומת המקסימום עד שנגיע לצומת האחרון שהמפתח שלו גדול מהמפתח של האיבר שרוצים להכניס. בהגעה לצומת זה נעשה הכנסה כפי שאנו מכירים בעץ AVL רגיל. בגלל שהאיבר שנכניס הוא הקטן בעץ, נגיע תמיד לשורש ולכן עלות העלייה עד לצומת זה היא $logk$ עבור k- גודל העץ, ועלות החיפוש גם היא $logk$.

סה"כ נקבל כי עלות החיפושים היא:

$$\Omega(nlogn) = n(logn - log2) = nlog\left(\frac{n}{2}\right) \le 2\sum_{k=\frac{n}{2}}^{n} logk \le \sum_{k=0}^{n} 2logk \le 2nlogn$$

$$= O(nlogn)$$

לכן **עלות החיפושים הכוללת:** $\boldsymbol{\theta(nlogn)}$

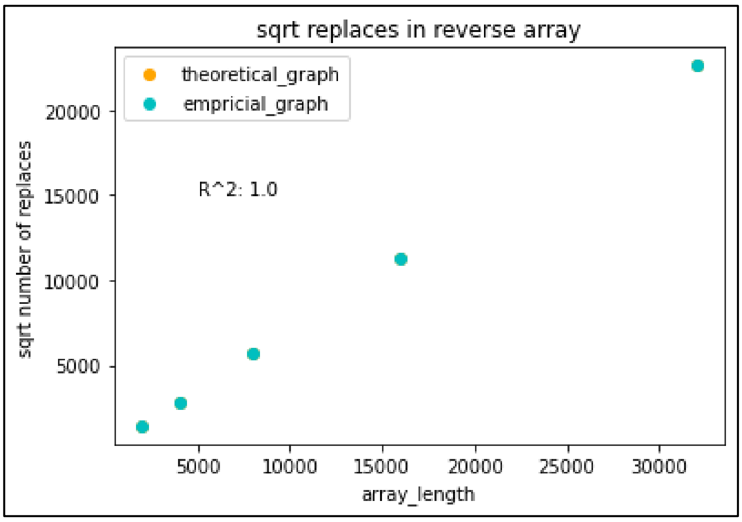
ג. הערכים בטבלה מתאימים לניתוח מסעיף ב'.

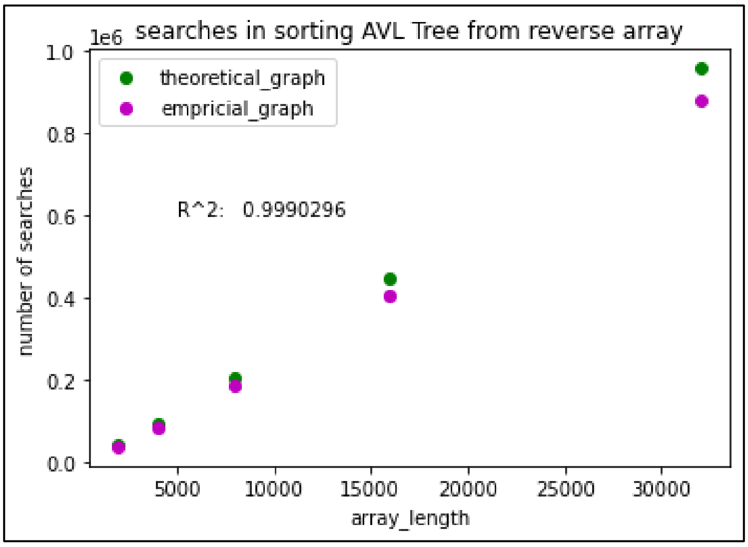
עבור נמספר החילופים במערך ממוין הפוך:

הגרף הכתום הוא גרף של החישוב $\binom{n}{2}$ לאחר טרנספורמציית שורש.

הגרף הכחול הוא גרף של התוצאות מהטבלה לאחר טרנספורמציית שורש.

קיבלנו שהגרף התיאורטי התלכד עם הגרף האמפירי כלומר אכן קיימת התאמה בין התיאוריה לתוצאות בפועל.





עלות החיפושים במיון AVL עבור
מערך ממוין הפוך:
הגרף הירוק הוא הגרף $2nlogn$
עבור n אורך הרשימה.
הגרף הורוד הוא התוצאות
מהטבלה.
ניתן לראות התאמה.

ד. כפי שהסברנו בסעיף ב', במהלך חיפוש עבור איבר באינדקס i נבצע עבודה בסיבוכיות $O\left(log(h_i)\right)$. מינימום עבודה תתקבל עבור $h_i = 0$ ועלות העבודה היא 1.

$$f(n,h) = \sum_{i=1}^{n} max\left(1, log(h_i)\right) \overset{\forall i\ h_i \geq 1}{\lesssim} n + \sum_{i=1}^{n} log(h_i) = n + log\left(\prod_{i=1}^{n} h_i\right)$$

$$= n + log\left(\sqrt[n]{\prod_{i=1}^{n} h_i}\right) \underset{\substack{inequality\ of\ arithmetic \\ and\ geometric\ means}}{\lesssim}$$

$$\leq n + n(log(h+1) - logn) = \boldsymbol{O\left(max\left(n(log(h+1) - logn)\right), n\right)}$$

הנחנו כי $\forall i\ h_i \geq 1$ כיוון שהתוספת של לכל היותר חילוף אחד עבור כל אינדקס לא תשנה את הסיבוכיות הסופית.

שאלה 2:

א.

מספר סידורי i	עלות join ממוצע עבור split אקראי	עלות join מקסימלי עבור split אקראי	עלות join ממוצע עבור split של האיבר מקסימלי בתת העץ השמאלי	עלות join מקסימלי עבור split של האיבר מקסימלי בתת העץ השמאלי
1	2.5555555555555554	6	2.888888888888889	12
2	2.3636363636363638	4	2.1666666666666665	13
3	2.9	6	2.5833333333333335	14
4	2.5	6	2.6923076923076925	15
5	2.3846153846153846	4	2.6153846153846154	16
6	2.6666666666666665	4	2.6666666666666665	18
7	3	8	2.411764705882353	19
8	2.75	7	2.764705882352941	20
9	2.6	11	2.4444444444444446	22
10	2.45	7	2.526315789473684	23

ב. <u>עלות join ממוצע עבור split לצומת אקראי:</u>

נסמן את הצומת ב-t_1 . נסמן את גובה הצומת ב- h. גודל העץ t יהיה n.

ראינו בכיתה כי עלות פעולת ה-split היא: $O(rank(t_k) - rank(t_1) + k)$

כאשר $t_1, \ldots, t_k$ אלו הם הצמתים שנעשה עליהם split. t_k הוא השורש.

לכן נבצע סה"כ k פעולות join.

עלות פעולה ממוצעת היא: $O\left(\frac{rank(t_k) - rank(t_1) + k}{k}\right)$

$rank(t_k) = O(logn)$ גובה העץ

$rank(t_1) = h$

$k = logn - h$

נציב ונקבל:

$$O\left(\frac{rank(t_k) - rank(t_1) + k}{k}\right) = O\left(\frac{logn - h + logn - h}{logn - h}\right) = O(2)$$

לכן נקבל גם כי העלות הממוצעת של פעולות ה-join עבור הצומת המקסימלי בתת העץ השמאלי של השורש היא גם $O(2)$ שכן זהו מקרה פרטי ולא תלוי ב-n או ב-h.

ניתן לראות כי התוצאות המפורטות בטבלה מתיישבות עם ניתוח הסיבוכיות התאורטי.

ג. במהלך פעולת ה-split בנינו שני עצים- עץ שמכיל את האיברים שקטנים מצומת הפיצול, נסמנו ב-small ועץ שמכיל את האיברים שגדולים מצומת הפיצול, נסמנו ב-big.

ראינו שעלות פעולת join בודד היא $O\left(rank(t_f) - rank(t_i) + 1\right)$.

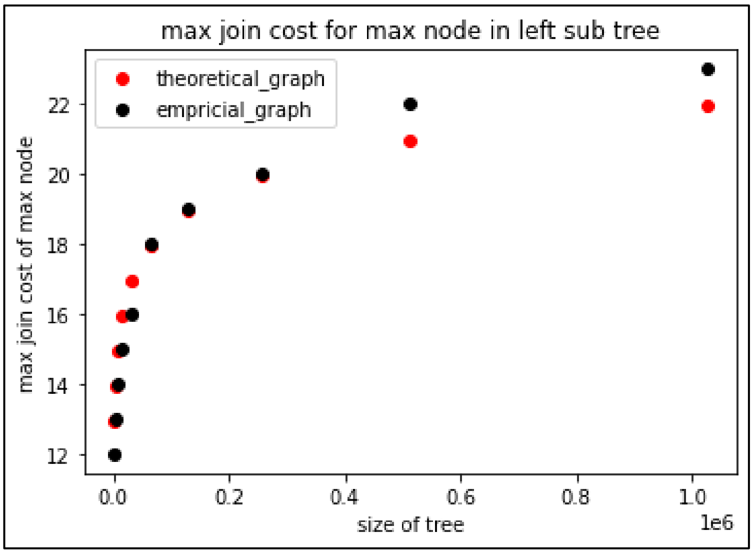
נניח שצומת הפיצול מגובה logn-k, נבצע k פעולות join עד שנגיע לשורש.

במהלך k-1 פעולות ה-join הראשונות בכל פעולה נאחד את small עם תת העץ השמאלי של הצומת שאליו הגענו. בכל פעולה כזו, small הוא תת העץ הימני של הצומת אליה הגענו (למעט צומת הפיצול) ולכן הפרשי הגבהים ביניהם הוא לכל היותר 2. כלומר עלות פעולת ה-join היא לכל היותר 3.

בפעולת ה-join ה-k, בהגיענו לצומת השורש, big עדיין ריק. כעת נרצה לחבר את תת העץ הימני של השורש שהוא בגובה $O(logn)$ עם big. ולכן עלות פעולת ה-join היא:

$O(logn - 0 + 1) = O(logn)$, זוהי פעולת ה-join שעלותה מקסימלית.

ניתן לראות כי התוצאות האמפיריות המפורטות בטבלה מתיישבות עם הניתוח התיאורטי בגרף הבא:



ד. עלות ה-join המקסימלי עבור צומת אקראי היא מספר הטיפוסים הרצופים כלפי הורה לאותו כיוון במהלך פעולת ה-split. בלי הגבלת הכלליות העץ מלא, אחרת הסיבוכיות לא תשתנה באופן משמעותי.

כאשר עולים באותו הכיוון k פעמים רצוף, אחד העצים (big או small מסימוני סעיף ג') לא מתעדכן. בסיום רצף העליות לאותו כיוון, כאשר נבצע עלייה לכיוון השני, הבדלי הגבהים של העץ מבין big ו-small שלא עדכנו לבין תת העץ שנרצה לחבר אליו יהיה k+1 ולכן עלות פעולת ה-join תהייה $O\left((k+1)+1\right)$. מכאן נסיק שעלות פעולת ה-join היא תלויה במספר העליות הרצופות לאותו כיוון.

האקראיות היא אחידה על פני בחירת מבנה העץ ובחירת האיבר בתוך העץ. לכן נוכל לבצע רדוקציה לבעיה הסתברותית שקולה:

עבור צומת שהפרש הגבהים שלו מהשורש הוא h, נרצה למצוא את מספר העליות הרצופות לאותו כיוון. נסמן ב-0 המשך עלייה לאותו הכיוון וב-1 את שינוי כיוון העלייה.

נרצה למצוא את הרצף המקסימלי של אפסים מתוך h עליות.

ניתן לתאר את המיקום של צומת אקראי שהפרש הגבהים שלו מהשורש הוא h כפרמוטציה מתוך מחרוזת של h ביטים.

טווח הערכים שנוכל לקבל תעבור הרצף המקסימלי של אפסים הוא בין 0 ל-h.

נראה כי קיים חסם עליון בסיבוכיות $O(logh)$:

ההסתברות לקבל רצף של לפחות $mlogh$ אפסים (עבור m חיובי) ממיקום מסוים במחרוזת היא: $p = \frac{1}{2}^{mlogh} = h^{-m}$. קיימים פחות מ-h מיקומים פוטנציאלים כאלו, לכן סך ההסתברות לרצף מסוג זה היא חסומה על ידי $p*h = h*\frac{1}{2}^{mlogh} = h*n^{-m} = h^{-m+1}$.

תוחלת עלות ה-join המקסימלי חסומה על ידי:

$$E(h) = (1-p)*mlogh + p*n \le 1*mlogh + h^{-m+1}$$

$$= 1*mlogh + \frac{1}{h^{m-1}} \overset{m\ge1}{\lesssim} mlogh + 1 \to \boldsymbol{E(h) = O(logh)}$$

ההסתברות ליפול ברמה h היא $\frac{2^h}{n}$. נכפיל הסתברות זו בתוחלת שחישבנו:

$$\sum_{h=0}^{logn} \frac{2^h}{n} * E(h) \leq \frac{1}{n} \sum_{h=0}^{logn} 2^h * c * logh \leq \frac{c}{n} \sum_{h=0}^{logn} 2^h * logh \overset{sum\ from\ last\ to\ first}{\triangleq}$$
$$= \frac{c}{n} * 2^{logn} * (log(logn) + 2^{-1}log(logn - 1) + 2^{-2}log(logn - 2) \dots)$$
$$\leq c * 2 * log(logn) = \boldsymbol{O\big(log(logn)\big)}$$