ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

<u>תרגיל בית מעשי במבני נתונים- תיעוד סיבוכיות</u>
<u>מגישים ניר בורגר, אריאל אראבוב</u>

<u>פונקציות במחלקה $AVLTree$</u>

פונקציית $empty()$-
סיבוכיות של $O(1)$.
הפונקציה מבצעת 2 פעולות השוואה, הראשונה- בודקת אם השורש שווה ל-$null$, השנייה- האם השורש הוא צומת וירטואלי. שתי הפעולות הן קבועות ולכן בסה״כ נבצע 2 פעולות לכל היותר ולכן $2 = O(1)$.

פונקציית $search(int\ k)$-
סיבוכיות של $O(\log(n))$.
נשים לב שמדובר ב-2 פונקציות שבעזרתן אנו מבצעים חיפוש.
הפונקציה העיקרית היא <u>$generalSearch(int\ k)$-</u> בהינתן מפתח כלשהו, אם המפתח לא קיים בעץ הפונקציה מחזירה את מי שאמור להיות האבא של הצומת או במידה והצומת קיים את הצומת עצמו. פונקציה זו עוברת על מסלול ספציפי בעץ על מנת להגיע לצומת הדרוש. כפי שנלמד בהרצאות עומק העץ הוא לכל היותר $O(\log(n))$ ולכן גם סיור על מסלול ייקח לכל היותר $O(\log(n))$.
פונקציית המעטפת היא <u>$search(int\ k)$-</u> בפונקציה זו אנו מבצעים לכל היותר 4 פעולות קבועות של השוואות בין $keys$ עבור הצומת שקיבלנו ב-$generalSearch$. השוואות אלה מתבצעות בזמן קבוע ולכן סיבוכיות הפונקציה היא $O(1)$.
סה״כ נקבל שעבור $search(int\ k)$ הסיבוכיות הכוללת היא-
$$O(1) + O(\log(n)) = \max\{O(1), O(log(n))\} = O(\log(n))$$

פונקציית $insert(int\ k, String\ s)$-
סיבוכיות של $O(\log(n))$.
נעבור על כל אחת מהפונקציות שאיתן אנחנו מבצעים את ההכנסה ונחשב עבור כל אחת מהן את סיבוכיות הזמן שלה, לבסוף נחבר את כולם על מנת לחשב את הסיבוכיות הכוללת.
<u>$insert(int\ k, String\ s)$-</u> הפונקציה יוצרת $node$ חדש ומבצעת השמה לאבא שלו (מחושבת באמצעות $generalSearch$). הפונקציה בודקת מקרי קצה (ע״י השוואות והשמות)- אם העץ הוא ריק או אם הצומת כבר קיים בעץ. אם לא נכנסו למקרים האלה היא מבצעת את ההכנסה (באמצעות $insertNode$), מחשבת את מספר האיזונים ($rebalanceInsert$), מעדכנת את השדות הכללים של העץ ($updateTreeFields$) ומחזירה את מספר האיזונים שנדרשו.
<u>$generalSearch(int\ k)$-</u> הוסבר שפועלת ב-$O(\log(n))$.
<u>$insertNode(IAVLNode\ parent, IAVLNode\ node)$-</u> מחשבת איפה להכניס את הצומת החדשה (בן שמאלי או ימני), מאתחלת השדות של הצומת (פעולות קבועות) ומעדכנת אותם ובנוסף מעדכנת את השדות של האבא ($updateFeilds$), $O(1)$.
<u>$updateFields(IAVLNode\ node)$-</u> מבצעת עדכונים לכל השדות עבור צומת מסוים, פעולות קבועות שמתבצעות ב-$O(1)$. קורא ל-$updateSizeInTree, UpdateRankDifferenceInTree$.
<u>$updateTreeFields()$-</u> עדכון השדות של העץ ע״י השמה של הערכים של השורש, פעולות קבועות שמתבצעות ב-$O(1)$.
<u>$UdateRankDifferenceInTree(\ IAVLNode\ node)$-</u> מעדכן את ה- $rank\ difference$ של צומת מסוימת בהתאם לגבוה של הבנים שלו. מבצע חישובים קבועים ולכן הסיבוכיות שווה ל- $O(1)$.
<u>$rebalanceInsert(IAVL\ Node\ node)$-</u> הפונקציה מבצעת $rebalance$ רק עבור צמתים במסלול הספציפי שבו ירדנו על מנת להכניס את הצומת (הוסבר בהרצאות ובתרגול שאיזון רק עבור המסלול הזה מספיק על מנת להחזיר את העץ למצב תקין). אנחנו רצים על לולאה שבה כל עוד לא הגענו לשורש הפונקציה בודקת אם אנחנו באחד מהמקרים שעברנו עליהם בהרצאות ומבצעת קריאות ל-

ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

$promote, demote, rotateRight, rotateLeft$ בהתאם למקרה שאנחנו נמצאים בו כרגע. נשים לב שקיימים תנאים מיוחדים עבור $join$, שלא רלוונטיים למקרים של $insert$. מכיוון שאנחנו עולים במסלול הספציפי שבו עשינו את החיפוש איפה להכניס את הצומת (לכל היותר $O(\log(n))$) וכל פעולות ההשוואה והעדכון מתבצעים ב-$O(1)$ הסיבוכיות הכוללת של הפונקציה הוא $O(\log(n))$.

<u>$isLeagalRD\ (int[]\ rD)$-</u> פונקציה שבודקת האם עבור $rank\ difference$ נתון (עבור צומת מסוים) הוא תקין בהתאם להגדרות של עץ AVL. הפונקציה עוברת על לולאה בגודל קבוע (גודל המערך $leagalRD = 3$) ומבצעת פעולות השוואה בין הפרשים תקינים למערך שקיבלנו. הפונקציה מבצעת פעולות קבועות ללא תלות בגודל הקלט ולכן הסיבוכיות היא קבועה, $O(1)$.

<u>$rotateRight(IAVLNode\ parent)$-</u> הפונקציה מבצעת סיבוב ימינה ע״י השמה וחלוקה למקרים קבועים. בכל מקרה מתבצעים חישובים אריתמטיים ונעשה שינוי מצביעים. כל הפעולות קבועות ולכן הסיבוכיות לסיבוב בודד (ימינה או שמאלה) היא $O(1)$.

<u>$rotateLeft(IAVLNode\ parent)$-</u> דומה ל$rotateRight$ (עד כדי סימטריות) ולכן גם היא בסיבוכיות של $O(1)$.

<u>$prompte(IAVLNode\ node)$-</u> מעדכן את הגובה של צומת מסוים, פעולה אריתמטית אחת- $O(1)$.

<u>$demote(IAVLNode\ node)$-</u> מעדכן את הגובה של צומת מסוים, פעולה אריתמטית אחת- $O(1)$.

נחבר את כל הקריאות לפונקציות ב-$insert(int\ k, String\ s)$ על מנת לקבל את הסיבוכיות הכוללת של הפונקציה-

$$10 \cdot O(1) + 2 \cdot O(\log(n)) = \max\{O(1), O(log(n))\} = O(\log(n))$$

פונקציית $\boldsymbol{delete(int\ k)}$-

סיבוכיות של $O(\log(n))$.

נעבור על כל אחת מהפונקציות שאיתן אנחנו מבצעים את המחיקה ונחשב עבור כל אחת מהן את סיבוכיות הזמן שלה, לבסוף נחבר את כולם על מנת לחשב את הסיבוכיות הכוללת.

<u>$delete(int\ k)$-</u> הפונקציה מחפשת את הצומת של המפתח שאנחנו רוצים למחוק (ע״י קריאה ל$generalSearch$ - $O(\log(n))$), במידה והצומת לא קיים נחזיר -1. כעת נחלק למקרים ונבצע את המחיקה כפי שנלמד בהרצאות. נבדוק האם הצומת בינארי ($isBinary$) במידה וכן נחליף עם ה$successor$ ($binSuccessor$). אם הצומת הוא עלה ($isLeaf$) נבצע את $deleteALeaf$, נעדכן שדות ($updateFields$) ונבצע פעולות איזון- $demote, rebalanceDelete$. אם הצומת הוא צומת אונרי ($isUnary$), נמחק אותו ($deleteUnary$) ונבצע פעולת איזון. נחשב את $rebalanceRoot$ ונחזיר את מספר הפעולות.

<u>$isBinary(IAVLNode\ node)$-</u> בודק האם לצומת כלשהי יש שני בנים. גישה שמתבצעת ב-$O(1)$.

<u>$isUnary(IAVLNode\ node)$-</u> בודק האם אחד הצמתים של צומת כלשהי הוא צומת חיצוני. גישה שמתבצעת ב- $O(1)$.

<u>$isLeaf\ (IAVLNode\ node)$-</u> בודק האם שתי הצמתים של צומת מסוים הם צמתים חיצוניים. גישה שמתבצעת ב-$O(1)$.

<u>$binSuccessor(IAVLNode\ node)$-</u> הפונקציה רצה בלולאה כל עוד לא הגענו לעלה חיצוני, מכיוון שעומק העץ הוא לכל היותר $O(\log(n))$ גם הלולאה תבצע לכל היותר $O(\log(n))$ איטרציות. בתוך הלולאה הפונקציה מבצעת השוואות והשמות בסיבוכיות קבועה. לבסוף נחליף בין הצמתים ($switchNodes$) ונעדכן את השדות ע״י קריאה ל- $updateFields,\ \ updatesFields$ שכולן מתבצעות ב-$O(1)$. לכן הסיבוכיות של הפונקציה היא $O(\log(n))$.

<u>$switchNodes(IAVlNode\ node, IAVLNode\ succs)$-</u> הפונקציה מבצעת החלפה בין הצומת ליורש שלו כלומר, הפונקציה משנה ומעדכנת הצבעות של ארבעה צמתים לכל היותר. הפונקציה מבצעת מספר פעולות קבוע כמו עדכון שדות אשר מתבצעות ב- $O(1)$ ולכן הסיבוכיות של כל הפונקציה הוא $O(1)$.

<u>$deleteLeaf\ (IAVLNode\ node)$-</u> מבצעת את פעולת המחיקה של עלה מהעץ. הפונקציה מעדכנת שדות קבועים ומבצעת השמה של צמתים וירטואליים במידת הצורך. כל הפעולות מתבצעות ב-$O(1)$ ולכן זוהי הסיבוכיות של הפונקציה.

ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

$\underline{deleteUnary(IAVLNode\ node)}$- מבצעת את פעולת המחיקה של צומת אונרי מהעץ. הפונקציה מעדכנת שדות ומבצעת השמה של צמתים וירטואליים במידת הצורך. עבור הבן של הצומת שאותו מוחקים קוראת ל-$setParentChild$. כל הפעולות הינם קבועות ולכן הסיבוכיות היא $O(1)$.

$\underline{setParentChild(IAVLNode\ node, IAVLNode\ parent, IAVLNode\ child)}$- עבור צומת אונרי שמוחקים מעדכנת את ההורה והילדים של הצומת. הפונקציה מבצעת השמות והשוואות קבועים, גישה לשדות מתבצעת ב-$O(1)$, ולכן סיבוכיות הפונקציה היא $O(1)$.

$\underline{rebalanceDelete(IAVLNode\ node)}$- הפונקציה מבצעת $rebalance$ רק עבור צמתים במסלול הספציפי שבו ירדנו על מנת להכניס את הצומת (הוסבר בהרצאות ובתרגול שאיזון רק עבור המסלול הזה מספיק על מנת להחזיר את העץ למצב תקין). אנחנו רצים על לולאה שבה כל עוד לא הגענו לשורש הפונקציה בודקת אם אנחנו באחד מהמקרים שעברנו עליהם בהרצאות ומבצעת קריאות ל- $promote, demote, rotateRight, rotateLeft$, (מתבצעות כולן ב-$O(1)$) בהתאם למקרה שאנחנו נמצאים בו כרגע. מכיוון שאנחנו עולים במסלול הספציפי שבו עשינו את החיפוש איפה להכניס את הצומת (לכל היותר $O(\log(n))$) וכל פעולות ההשוואה והעדכון מתבצעים ב-$O(1)$ הסיבוכיות הכוללת של הפונקציה הוא $O(\log(n))$.

$\underline{rebalanceRoot()}$- מקרה קצה מיוחד שבו השורש לא מאוזן. הפונקציה מבצעת סדרת פעולות קבועות של השוואות והשמות וקוראת ל-$rebalanceDelete$ במקרה הצורך ועושה לכל היותר 2 איטרציות. לכן הסיבוכיות של הפונקציה היא $O(1)$.

נחבר את כל הקריאות לפונקציות ב-$insert(int\ k, String\ s)$ על מנת לקבל את הסיבוכיות הכוללת של הפונקציה-

$$8 \cdot O(1) + 3 \cdot O(\log(n)) = \max\{O(1), O(log(n))\} = O(\log(n))$$

פונקציית () min-

סיבוכיות של $O(1)$.

הפונקציה ניגשת לשדה min של השורש, שמצביע על הצומת המינימלית ומחזירה את הערך $info$ של הצומת, במידה והעץ ריק מחזיר $null$. הגישה וההחזרה מתבצעות ב-$O(1)$ ולכן הסיבוכיות גם היא $O(1)$.

פונקציית () max-

סיבוכיות של $O(1)$.

הפונקציה ניגשת לשדה max של השורש, שמצביע על הצומת המקסימלית ומחזירה את הערך $info$ של הצומת, במידה והעץ ריק מחזיר $null$. הגישה וההחזרה מתבצעות ב-$O(1)$ ולכן הסיבוכיות גם היא $O(1)$.

פונקציית $\boldsymbol{keysToArray}()$-

סיבוכיות של $O(n)$.

$\underline{keysToArray()}$- הפונקציה מאחלת מערכים, בודקת תנאים קבועים ומבצעת קריאה לפונקציה הרקורסיבית $keysToArrayRec$. לכן הסיבוכיות שלה היא קבועה, $O(1)$.

$\underline{keysToArrayRec(IAVLNode\ node, int[]\ array, int[]\ index)}$- הפונקציה מבצעת סיור על כל הצמתים האמיתיים. היא כל פעם קוראת לפונקציה עבור כל הצמתים בתת עץ השמאלי ועבור כל הצמתים עבור התת עץ הימני של צומת. כלומר, הפונקציה מבצעת n קריאות שבכל קריאה היא מבצעת מספר קבוע של פעולות ולכן הסיבוכיות היא $n \cdot 1 = O(n)$.

הפונקציה $keysToArray()$ קוראת ל- $keysToArrayRec()$ ולכן הסיבוכיות הכוללת של הפונקציה היא $O(n)$.

ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

פונקציית $infoToArray()$-
סיבוכיות של $O(n)$.
$-infoToArray()$ הפונקציה פועלת באופן דומה ל-$keysToArray$ (עד כדי גישה ל$values$, מתבצע בזמן קבוע) ולכן מתבצעת ב-$O(1)$.
$-infoToArrayRec(IAVLNode\ node, String[]\ array, int[]\ index)$ הפונקציה פועלת באופן דומה ל-$keysToArrayRec$ (עד כדי גישה ל$values$, מתבצע בזמן קבוע) ולכן מתבצעת ב-$O(n)$.
הפונקציה $infoToArray()$ קוראת ל- $infoToArrayRec()$ ולכן הסיבוכיות הכוללת של הפונקציה היא $O(n)$.

פונקציית $size()$-
סיבוכיות של $O(1)$.
הפונקציה ניגשת לשדה $size$ של השורש, שבו מעודכן מספר הצמתים בתת עץ של השורש. הגישה וההחזרה מתבצעות ב-$O(1)$ ולכן הסיבוכיות גם היא $O(1)$.

פונקציית $getRoot()$-
סיבוכיות של $O(1)$.
מחזירה את השורש של העץ, הגישה וההחזרה מתבצעות ב-$O(1)$ ולכן הסיבוכיות גם היא $O(1)$.

פונקציית $split(int\ x)$-
סיבוכיות של $O(\log(n))$.
הפונקציה מחפשת את הצומת עבורו אנחנו רוצים לבצע את הפיצול ($generalSearch - O(\log(n))$).
מבצעת השמות ואז נכנסת ללולאה מהצומת שמצאנו עד לשורש. בכל איטרציה היא בודקת תנאים בזמן קבוע ומבצעת $join$ עבור תתי העצים הרלוונטיים. הסיבוכיות של $join()$ כפי שנראה הוא בסיבוכיות של $O(|rank(T_2) - rank(T_1)| + 1)$. הלולאה מבצעת לכל היותר $O(\log(n))$ איטרציות (המסלול יהיה לכל היותר מעלה לשורש) ולכן הסיבוכיות של $split$ שווה ל-
$$O(\log(n)) + O(\log(n)) \cdot O(|rank(T_2) - rank(T_1)| + 1) = O(\log(n))$$

פונקציית $join(IAVLNode\ x, AVLtree\ t)$-
סיבוכיות של $O(|rank(T_2) - rank(T_1)| + 1)$.
תחילה נבדוק תנאים ונבצע השמות שמתבצעות בזמן קבוע. כעת נחלק ל-2 מקרים, נבדוק אם שני העצים ריקים, במידה וכן נאתחל משתנים ושדות בזמן קבוע אם רק אחד מהם ריק נבצע הכנסה ($insertNode = O(1), generalSearch = O(\log(n))$). נדגיש שדרישת הסיבוכיות נשמרת שכן גובה העץ הלא ריק הוא לכל היותר $O(\log(n))$, נחזיר את הערך הדרוש ונסיים. במידה ושני העצים אינם ריקים נבדוק תנאים בזמן בקבוע ונבצע
$joinRight, joinLeft = O(|rank(T_2) - rank(T_1)| + 1)$ בהתאם למקרה.

פונקציית $joinRight(AVLTree\ big, AVLTree\ small, IAVLNode\ x)$-
סיבוכיות של $O(|rank(T_2) - rank(T_1)| + 1)$.
תחילה נבצע השמות בזמן קבוע ונרוץ בלולאה עד שנגיע לצומת בעץ הגדול שעבורו $rank(Node\ in\ big\ Tree) \leq rank(small\ Tree)$. לאחר מכן בודקים תנאים ומבצעים השמות בזמן קבוע, לאחר מכן קוראים ל$rebalanceInsert$ שאמנם היא בסיבוכיות של $O(\log(n))$ אך מפני שמתחילים מהצומת שאותו הכנסנו שנמצא בגובה של העץ הקטן הסיבוכיות תהיה $O(|rank(T_2) - rank(T_1)| + 1)$. לאחר מכן נעדכן שדות בזמן קבוע ($updateTreeFields()$) ונחזיר את ההפרש שמחושב בזמן קבוע.

ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

פונקציית $joinLeft(AVLTree\ big, AVLTree\ small, IAVLNode\ x)$-
סיבוכיות של $O(|rank(T_2) - rank(T_1)| + 1)$.
מתבצע באופן סימטרי לחלוטין כמו $joinRight$ ולכן הסיבוכיות תהיה בהתאם, שווה ל- $O(|rank(T_2) - rank(T_1)| + 1$.

<u>פונקציות במחלקה *AVLNode*</u>

פונקציית $getKey()$-
מחזירה את הערך בשדה max של צומת מסוים, $O(1)$.

פונקציית $getValue()$-
בודקת אם צומת הוא וירטואלי. במידה ולא מחזירה את הערך של הצומת, אחרת מחזירה $null$. $O(1)$.

פונקציית $setLeft()$-
מעדכנת את התת עץ השמאלי של צומת, $O(1)$.

פונקציית $getLeft()$-
בודקת אם לצומת קיים תת עץ שמאלי. במידה ולא מחזירה $null$, אחרת מחזירה את התת עץ השמאלי. $O(1)$.

פונקציית $setRight()$-
מעדכנת את התת עץ הימני של צומת, $O(1)$.

פונקציית $getRight()$-
בודקת אם לצומת קיים תת עץ ימני. במידה ולא מחזירה $null$, אחרת מחזירה את התת עץ הימני. $O(1)$.

פונקציית $setParent()$-
מעדכנת את ההורה של הצומת, $O(1)$.

פונקציית $getParent()$-
בודקת אם לצומת קיים הורה. במידה ולא מחזירה $null$, אחרת מחזירה את ההורה. $O(1)$.

פונקציית $isRealNode()$-
בודקת אם הגובה של הצומת שווה ל -1. במידה וכן, מחזירה $true$, אחרת $false$, $O(1)$.

פונקציית $setHeight()$-
מעדכנת את הגובה של הצומת, $O(1)$.

פונקציית $getHeight()$-
מחזירה את השדה $rank$ של צומת, $O(1)$.

פונקציית $updateRankDifference()$-
מחשבת את ההפרש גבהים בין צומת לילדים שלו, $O(1)$.

ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

פונקציית $\boldsymbol{getRankDifference}()$**-**
מחזירה את השדה $rankDifference$ של הצומת, $O(1)$.

פונקציית $\boldsymbol{updateSize}()$**-**
מעדכן את ה$Size$ של הצומת להיות ה-$size$ של שני תתי העץ שלו 1+, $O(1)$.

פונקציית $\boldsymbol{getSize}()$**-**
מחזירה את השדה $size$ של הצומת, $O(1)$.

פונקציית $\boldsymbol{updateMin}()$**-**
בודקת אם הצומת הוא וירטואלי. במידה ולא תחזיר את השדה min של התת עץ השמאלי, אחרת תחזיר את הבן השמאלי, $O(1)$.

פונקציית $\boldsymbol{getMin}()$**-**
מחזירה את השדה min של הצומת, $O(1)$.

פונקציית $\boldsymbol{updateMax}()$**-**
בודקת אם הצומת הוא וירטואלי. במידה ולא תחזיר את השדה max של התת עץ הימני, אחרת תחזיר את הבן הימני, $O(1)$.

פונקציית $\boldsymbol{getMax}()$**-**
מחזירה את השדה max של הצומת, $O(1)$.

ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

<u>חלק עיוני שאלה 1</u>

סעיף א׳

מספר סידורי i	מספר חילופים במערך ממוין- הפוך	עלות החיפושים במיון AVL עבור מערך ממוין- הפוך	מספר חילופים במערך מסודר אקראית	עלות החיפושים במיון עבור מערך AVL מסודר אקראית
1	1,999,000	38,884	1,008,637	33,619
2	7,998,000	85,763	4,074,543	80,113
3	31,996,000	187,527	16,056,178	171,397
4	127,992,000	407,044	64,330,154	375,117
5	511,984,000	878,084	255,724,350	789,236

סעיף ב׳

<u>מספר החילופים במערך ממוין הפוך-</u>
נשים לב שעבור מערך ממוין הפוך מספר ה-$swaps$ שנצטרך לעשות הוא המקסימלי בהינתן מערך כלשהו. עבור האיבר הראשון במערך נצטרך לבצע $n-1$ $swaps$ על מנת שיגיע לאינדקס האחרון, כעת לאיבר הראשון במערך נצטרך לבצע $n-2$ החלפות כדי שיגיע לאינדקס האחד לפני האחרון וכך הלאה.
נשים לב שנקבל סכום של סדרה חשבונית-

$$\sum_{i=1}^{n} i = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n = \binom{n}{2}$$

<u>עלות החיפושים במיון AVL עבור מערך ממוין הפוך-</u>
עלות החיפושים כתלות ב-n היא $\Theta(n \cdot \log(n))$.
מפני שהמערך הוא ממוין הפוך כל צומת שנכניס יהיה הצומת המינימלי בעץ (עד לאותה הכנסה כולל).
לכן, המרחק בין הצומת שנכניס למקסימום יהיה כמות הצמתים בעץ עד עכשיו, נסמן אותו ב- i ולכן עלות ההכנסה לצומת יהיה $O(\log(i))$.
סה״כ נקבל שהעלות הכוללת של הכנסת כל האיברים לעץ שווה ל-

$$\sum_{i=1}^{n} \log(i) = \log\left(\prod_{i=1}^{n} i\right) = \log(n!) = \Theta(n \cdot \log(n))$$

המעבר האחרון הוכח בתרגול הראשון (שקופית 27).

סעיף ג׳

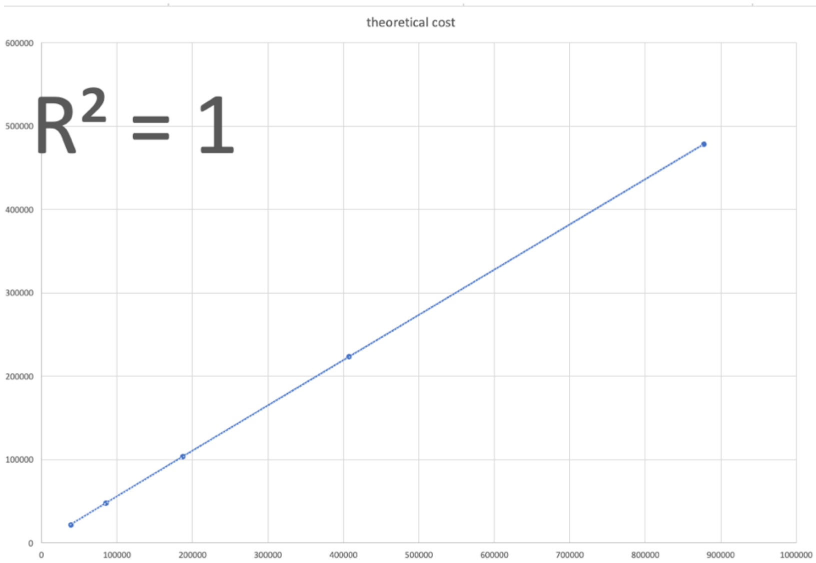
הערכים המתקבלים בסעיף א׳ מתאימים לניתוחים בסעיף ב׳, נראה זאת באמצעות הטבלאות והגרפים הבאים-

Sorted upside down array

i	num of swaps		cost of AVL search	
1	1999000		38884	
2	7998000		85764	
3	31996000		187524	
4	127992000		407044	
5	511984000		878084	

ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

		Linear transformation		
size of array	num of swaps	theoretical num of swaps	cost of AVL search	theoretical cost
2000	1999000	1999000	38884	21931
4000	7998000	7998000	85764	47863
8000	31996000	31996000	187524	103726
16000	127992000	127992000	407044	223452
32000	511984000	511984000	878084	478905



סעיף ד׳

נתון שמספר החילופים במערך מסוים h. נניח שאנחנו רוצים להוסיף צומת לעץ שכרגע קיימים בו $i-1$ צמתים. מספר החילופים במערך עבור הצומת ה- i מוגדר להיות h_i. נשים לב שאורך המסלול שצריך לעבור i בעת ההכנסה אל העץ בהכרח קטנה שווה ל- h_i+1. מכיוון שבעץ יש לכל היותר n צמתים במסלול (מתקיים רק בהכנסה האחרונה) בהכרח מספר הצמתים שנעבור דרכם יהיה קטן או שווה למספר החילופים שאנחנו נבצע במערך עצמו.

ולכן-

$$cost\ of\ search \leq \sum_{i=1}^{n} \log(h_i+1) = \log\left(\prod_{i=1}^{n}(h_i+1)\right) \leq \log\left(\left(\frac{h+n}{n}\right)^n\right) = n \cdot \log\left(\frac{h}{n}+1\right)$$

מעבר הראשון והאחרון נובעים מחוקי לוגים.

המעבר השני נובע מאי שוויון הממוצעים-

$$\prod_{i=1}^{n} h_i + n \leq \left(\frac{\sum_{i=1}^{n} h_i + n}{n}\right)^n = \left(\frac{h+n}{n}\right)^n$$

סה״כ נקבל שהחסם העליון לסיבוכיות שווה ל- $insertion\ sort = O\left(nlog\left(\frac{h}{n}+1\right)\right)$.

ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

<u>חלק עיוני שאלה 2</u>

סעיף א׳

מספר סידורי i	עלות $join$ ממוצע עבור $split$ אקראי	עלות $join$ מקסימלי עבור $slpit$ אקראי	עלות $join$ ממוצע עבור $split$ של איבר מקסימלי בתת העץ השמאלי	עלות $join$ מקסימלי עבור $split$ של איבר מקסימלי בתת העץ השמאלי
1	2.799	7	2.799	13
2	3.099	5	2.818	14
3	2.727	5	2.714	15
4	3	7	2.692	17
5	2.571	5	3	18
6	3.285	6	2.799	19
7	2.647	5	2.882	20
8	2.666	7	2.812	22
9	2.736	6	2.611	22
10	3.055	7	2.789	24

סעיף ב׳

<u>עלות $join$ ממוצע עבור $split$ על איבר אקראי-</u>
עפ״י האלגוריתם של $split$ אנחנו מתחילים מהצומת שנבחר ועולים ממנו עד שמגיעים לשורש. נשים לב שכמות פעולות ה-$join$ שנבצע היא כעומק הצומת-

$$(height\ of\ root - hight\ of\ node) = O(\log(n))$$

בכיתה הוכחנו (הרצאה 4 שקופית 71) כי עלות פעולות ה-$join$ בפעולת $split$ בודדת לא תלוי בצומת שעליו מבצעים את הפיצול ולכן העלות היא תמיד $O(\log(n))$.
נניח שנבחר באקראי לבצע פיצול על צומת בעומק k, זאת אומרת שנצטרך לבצע k פעולות $join$ על k עצים שנסמנם- $T_1, T_2, \dots, T_k$. עבור העצים האלה מתקיים $rank(T_1) \le rank(T_2) \le \cdots \le rank(T_n)$.
נשים לב ש- $rank(T_k) - rank(T_1) \le O(\log(n))$ (במקרה הגרוע בו T_1 הוא עץ ריק ו-T_k הוא תת עץ שלם של השורש נקבל שוויון). סה״כ נקבל שעלות כל פעולות ה-$join$ שווה ל-

$$O\left(\sum_{i=2}^{k} \left|rank(T_i) - rank\big(Join(T_1, \dots, T_{i-1})\big)\right| + 1\right) = O\left(\sum_{i=2}^{k} rank(T_i) - rank(T_{i-1}) + 1\right)$$

$$= O(rank(T_k) - rank(T_1) + k) = O(\log(n))$$

לכן עלות $join$ ממוצע עבור $split$ על צומת אקראי k שווה ל-

$$\frac{\sum join\ cost}{num\ of\ joins} = \frac{split\ cost}{number\ of\ joins} = \frac{O(rank(T_k) - rank(T_1) + k)}{depth\ of\ node} = \frac{O(\log(n))}{O(\log(n))} = O(1)$$

ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

<u>עלות $join$ ממוצע עבור $split$ על איבר מקסימלי בתת עץ השמאלי-</u>
ראינו שעלות $join$ ממוצע עבור $split$ על איבר אקראי היא $O(1)$. נשיב לב, שכפי שצוין חישוב זה לא היה תלוי בצומת ספציפי ולכן גם כאשר בוחרים את האיבר המקסימלי בתת עץ השמאלי העלות תהיה זהה ולכן תהיה שווה ל-$O(1)$.

גם עבור $split$ אקראי וגם עבור $split$ על איבר מקסימלי ניתוחי הסיבוכיות תואמים את התוצאות שקיבלנו בסעיף א׳-

size of array	average cost of join for random split	theoretical average cost of join for split (random or max)	average cost of join for split on max index of left sub tree
2000	2.799	1	2.799
4000	3.099	1	2.818
8000	2.727	1	2.714
16000	3	1	2.692
32000	2.571	1	3
64000	3.285	1	2.799
128000	2.647	1	2.882
256000	2.666	1	2.812
512000	2.736	1	2.611
1024000	3.055	1	2.789

סעיף ג׳
<u>עלות $join$ מקסימלי עבור $split$ על האיבר המקסימלי בתת עץ השמאלי-</u>
נשים לב שעובר הצומת הזו כל האיברים שגדולים ממנו הוא השורש וכל תת העץ הימני, שאר האיברים קטנים ממנו.
בפועלת ה-$split$ אנחנו עולים מהאיבר שעליו עושים $split$ עד השורש, במקרה שלנו עד השורש (לא כולל) אנחנו תמיד עולים שמאלה. מכיוון שאנחנו מפצלים עץ AVL תקין ועולים כל פעם שמאלה ה-$joins$ שנבצע יהיו על עצים שעבורם מתקיים $rank(T_{i+1}) - rank(T_i) \leq 1$ כאשר נגדיר את T_i, T_{i+1} להיות תתי עצים מהתת עץ השמאלי (שקטנים מהאיבר המקסימלי שעליו עושים $split$) שעליהם מבצעים את ה- $join$. לכן, כל פעולת $join$ שנבצע עבור איברים שקטנים מהמקסימלי בתת עץ השמאלי יהיו לכל היותר $O(1)$. במקרה הגרוע הפרשי ה-$rank$ יהיה שווה ל-1 והפונקציה תחזיר- $|rank(T_{i+1}) - rank(T_i)| + 1$.
בשביל לקבל את העץ שמכיל את כל האיברים שגדולים מהצומת המקסימלי בתת העץ השמאלי נעשה פעולת $join$ בודדת. נאחד את השורש עם עץ ריק, $rank = -1$, (תת העץ של הימני של הצומת המקסימלי בתת עץ השמאלי) ועם תת העץ הימני של העץ הנתון. עבור תת העץ הימני אשר נסמן אותו כ-T_R מתקיים-

$$rank(root) = O(\log(n)) \rightarrow rank(right\ son\ of\ root) = O(\log(n)) - 1\ or\ O(\log(n)) - 2 \rightarrow$$

$$rank(T_R) = O(\log(n))$$

ולכן סך העלות של ה-$join$ הזה שווה ל- $|\mathrm{O}(\log(\mathrm{n})) - (-1)| + 1 = O(\log(n))$.

סה״כ נקבל שעלות ה-$join$ מקסימלי עבור $split$ על איבר מקסימלי בתת עץ השמאלי שווה ל-
$O(1) + O(\log(n)) = O(\log(n))$

ניר בורגר, ת״ז- 313580920, יוזר במודל- nirborger
אריאל אראבוב, ת״ז- 209881531, יוזר במודל- arabov

התוצאות שקיבלנו אכן תואמים את הניתוח סיבוכיות-

size of array	max cost of join for split on max index of left sub tree	theoretical max cost of join for split on max index of left sub tree
2000	13	10.965
4000	14	11.965
8000	15	12.965
16000	17	13.965
32000	18	14.965
64000	19	15.965
128000	20	16.964
256000	22	17.965
512000	22	18.965
1024000	24	19.965