תיעוד פרויקט מעשי עצי AVL

סטודנט 1: ת"ז-209130269, שם-תומר סרוסי, שם משתמש-sarusi
סטודנט 2: ת"ז-209327337, שם-יונתן כהן, שם משתמש-yonatancohn

AVLNode

AVLNode היא מחלקה המממשת צמתים עבור עצי AVL.

שדות:

- **int key** – שדה המכיל את מפתח הצומת.
- **String info** – שדה המכיל את המידע של הצומת.
- **IAVLNode left** – שדה המכיל מצביע לצומת שהיא הבן השמאלי של הצומת הנוכחי.
- **IAVLNode right** – שדה המכיל מצביע לצומת שהיא הבן הימני של הצומת הנוכחי.
- **IAVLNode parent** – שדה המכיל מצביע לצומת שהיא ההורה של הצומת הנוכחי.
- **int height –** גובה תת העץ שהצומת הנוכחי הוא השורש שלו.
- **int size –** גודל תת העץ שהצומת הנוכחי הוא השורש שלו.

בנאים:

- **public AVLNode(int key, String info) –** בנאי היוצר צומת עם המפתח key שהתקבל והמידע info שהתקבל. אם המפתח הינו 1- אז הצומת היא וירטואלי, ולכן גובהו יהיה 1-. אחרת, הצומת הוא עלה, ולכן לצומת נוצרים בנים וירטואלים, גובהו יהיה 0, וגודל תת העץ שהוא השורש שלו הוא 1.
סיבוכיות הבנאי היא $O(1)$, מכיוון שהבנאי מבצע מספר קבוע של פעולות בעלות סיבוכיות זמן קבועה.
- **public AVLNode()** – בנאי היוצר צומת וירטואלי תוך שימוש בבנאי הנ"ל.
סיבוכיות הבנאי היא $O(1)$, מכיוון שהבנאי מבצע פעולה בעלת סיבוכיות זמן קבועה.

מתודות:

- **public int getKey() –** מחזיר את ערך המפתח של הצומת.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **public String getInfo() -** מחזיר את המידע של הצומת.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.

- **public IAVLNode getLeft()** - מחזיר מצביע לבן השמאלי של הצומת.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **public void setLeft(IAVLNode node)** – מעדכן את מצביע הבן השמאלי של הצומת ל-node, ומעדכן את size.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעות פעולות בעלות סיבוכיות זמן קבועה.
- **public IAVLNode getRight()** - מחזיר מצביע לבן הימני של הצומת.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **public void setRight(IAVLNode node)** – מעדכן את מצביע הבן הימני של הצומת ל-node, ומעדכן את size.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעות פעולות בעלות סיבוכיות זמן קבועה.
- **public IAVLNode getParent()** - מחזיר מצביע להורה של הצומת.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **public void setParent(IAVLNode node)** – מעדכן את מצביע ההורה של הצומת ל-node.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **public boolean isReadNode()** – מחזיר אמת אם הצומת אינה וירטואלית, כלומר אם ערך מפתח הצומת אינו 1-. אחרת מחזיר שקר.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **public int getHeight()** – מחזיר את הגובה של הצומת.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **public void setHeight(int height)** – מעדכן את גובה הצומת להיות height.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **public int getSize()** – מחזיר את גודל תת העץ שהצומת היא השורש שלו.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **public void calculateSize()** - מעדכן את גודל תת העץ שהצומת היא השורש שלו בעזרת שדות ה-size של בניו.

סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.

AVLTree

AVLTree היא מחלקה המממשת עצי AVL.

שדות:

- **IAVLNode root** – מצביע לצומת שהוא שורש העץ.
- **IAVLNode min** – מצביע לצומת שערך המפתח שלו הוא המינימלי.
- **IAVLNode max** – מצביע לצומת שערך המפתח שלו הוא המקסימלי.

מתודות:

- **public boolean empty()** – מחזיר אמת אמ"מ העץ ריק, כלומר אם המצביע לשורש הוא null.
  סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעות פעולות בעלות סיבוכיות זמן קבועה.
- **public String search(int k)** – מחזיר את המידע של הצומת בעל המפתח k, במידה והצומת קיים בעץ. אחרת, אם לא קיים צומת בעל מפתח k בעץ, מחזיר 1-.
  המתודה מבצעת קריאה למתודה searchRec ומחזירה את המידע של הצומת המוחזר ממנה.
  סיבוכיות המתודה היא $O(\log(n))$, מכיוון שמתבצעת קריאה לפעולה בעלת סיבוכיות זמן $O(\log(n))$ ועוד פעולות בסיבוכיות זמן קבועה.
- **private IAVLNode searchRec(int k, IAVLNode node)** – המתודה מבצעת חיפוש רקורסיבי לערך המפתח k מהצומת node, לפי חיפוש רקורסיבי של עץ בינארי. כלומר, אם מפתח הצומת הנוכחי הוא k אז הצומת מוחזר, אם מפתח הצומת הנוכחי גדול מ-k אז מתבצעת קריאה רקורסיבית של המתודה עבור הבן השמאלי של הצומת, ואחרת אם מפתח הצומת הנוכחי קטן מ-k אז מתבצעת קריאה רקורסיבית של המתודה עבור הבן הימני של הצומת. במידה ולא קיים בעץ צומת עם מפתח k, יוחזר צומת וירטואלי במיקום בו אמור היה להיות צומת בעל מפתח k.
  סיבוכיות המתודה היא $O(\log(n))$, מכיוון שהמתודה מבצעת פעולות בסיבוכיות זמן קבועה, ומבצעת קריאה רקורסיבית על עצמה עבור כל הצמתים בין node ובין עלי העץ, ולכן מספר הקריאות האלו חסום על ידי גובה העץ, שהוא $\log(n)$, כידוע עבור עצי AVL.

- **public int insert(int k, String i)** – מכניס צומת חדש בעל מפתח k ומידע i לעץ, תוך שמירה על תכונות עץ AVL, כלומר על איזון העץ, ועל כך שהעץ הוא עץ בינארי, ובתנאי שלא קיים בעץ צומת עם מפתח k. אחרת, אם קיים צומת עם מפתח k, יוחזר 1-.
המתודה מבצעת קריאה למתודה insertNode ומחזירה את הערך המוחזר ממנה.
סיבוכיות המתודה היא $O(\log(n))$, מכיוון שמתבצעת קריאה לפעולה בעלת סיבוכיות זמן $O(\log(n))$ ועוד פעולות בסיבוכיות זמן קבועה.
- **private int insertNode(IAVLNode node)** – מבצע הכנסה של הצומת לעץ לפי עקרונות הכנסה לעץ בינארי, ולאחר מכן מבצע איזון לעץ לפי עקרונות איזון עצי AVL, ובתנאי שלא קיים בעץ צומת עם מפתח k. אחרת, אם קיים צומת עם מפתח k, יוחזר 1-.
הפעולה מחזירה את כמות פעולות האיזון שמתבצעות בעת איזון העץ.
הפונקציה מבצעת זאת ע"י קריאה לsearchRec, balanceFromNode, ופעולות נוספות בעלות סיבוכיות זמן קבועה, ומחזירה את הערך שמתקבל מ-balanceFromNode.
סיבוכיות המתודה היא $O(\log(n))$, מכיוון שמתבצעות 2 קריאות לפעולות בעלות סיבוכיות זמן $O(\log(n))$ ועוד פעולות בסיבוכיות זמן קבועה.
- **public int delete(int k)** – המתודה מוחקת את הצומת בעץ בעל המפתח k, במידה וקיים. אחרת, במידה ולא קיים צומת כזה, יוחזר 1-.
המתודה מוצאת את הצומת באמצעות searchRec, ובמידה והצומת הנמחק הוא צומת הmin או הmax, הפעולה מעדכנת את הצמתים האלו בעזרת הפעולות predecessor/successor בהתאמה.
לבסוף המתודה מבצעת קריאה למתודה deleteNode ומחזירה את הערך המוחזר ממנה.
סיבוכיות המתודה היא $O(\log(n))$, מכיוון שמתבצעות קריאות לפעולות בעלות סיבוכיות זמן $O(\log(n))$ ועוד פעולות בסיבוכיות זמן קבועה.
- **private int deleteNode(IAVLNode node)** – המתודה מוחקת את הצומת מהעץ לפי עקרון מחיקה מעצים בינאריים, מאזנת את העץ לפי עקרון איזון עצי AVL, ומחזירה את מספר פעולות האיזון שמתבצעות במהלך איזון העץ. במידה והצומת node הוא צומת בעל 2 בנים, הפעולה מחליפה את הערך והמידע של node בערך והמידע של העוקב שלה, ולאחר מכן מבצעת קריאה רקורסיבית של עצמה עבור הצומת העוקב של הצומת הנוכחי, שהוא בהכרח צומת עם לכל היותר בן יחיד. הפעולה מחזירה את הערך המוחזר מbalanceFromNode.
סיבוכיות המתודה היא $O(\log(n))$, מכיוון שמתבצעת לכל היותר קריאה

רקורסיבית אחת, ובנוסף מתבצעות כמות פעולות קבועה של פעולות בעלות סיבוכיות לכל היותר $O(\log(n))$.

- **private int balanceFromNode(IAVLNode nodeToBalanceFrom, boolean cameFromJoin)** – המתודה מאזנת את העץ לפי עקרונות איזון עצי AVL, ע"י השוואת הפרשי ה-ranks של בני הצומת וביצוע פעולות איזון בצורך. במידה והפעולה נקראת ע"י join, כלומר cameFromJoin=true, אז המתודה מבצעת בדיקות נוספות לווידוא איזון העץ. בנוסף המתודה מעדכנת את השדות size של כל הצמתים בעץ הנדרשים לעדכון. המתודה מחזירה את כמות פעולות האיזון שבוצעו במהלך איזון העץ.
סיבוכיות המתודה היא $O(\log(n))$, מכיוון שהפעולה מבצעת מספר קריאות קבוע של פעולות בעלות סיבוכיות $O(\log(n))$ (הפעולה updateNodeSize היא הפעולה היחידה בעלת סיבוכיות זמן לא קבועה, והיא נקראת לכל היותר פעמיים), ועוד $O(\log(n))$ פעולות בעלות סיבוכיות זמן קבועה, שכן המתודה עוברת על הצמתים מהצומת nodeToBalanceFrom עד השורש, וזה חסום ע"י גובה העץ, $\log(n)$.
- **public String min()** – המתודה מחזירה את המידע של הצומת בעל המפתח המינימלי. אם העץ ריק מוחזר null.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **private void updateMin()** – המתודה מחפשת את הצומת בעל המפתח המינימלי, ומעדכנת את השדה min להצביע לצומת הזה.
סיבוכיות המתודה היא $O(\log(n))$, שכן המתודה עוברת על הצמתים בענף השמאלי ביותר של העץ, שגובהו חסום ע"י גובה העץ, $\log(n)$, ועבור כל צומת המתודה מבצעת מספר קבוע של פעולות בעלות סיבוכיות זמן קבועה.
- **public String max()** – המתודה מחזירה את המידע של הצומת בעל המפתח המקסימלי. אם העץ ריק מוחזר null.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **private void updateMax()** – המתודה מחפשת את הצומת בעל המפתח המקסימלי, ומעדכנת את השדה max להצביע לצומת הזה.
סיבוכיות המתודה היא $O(\log(n))$, שכן המתודה עוברת על הצמתים בענף הימני ביותר של העץ, שגובהו חסום ע"י גובה העץ, $\log(n)$, ועבור כל צומת המתודה מבצעת מספר קבוע של פעולות בעלות סיבוכיות זמן קבועה.
- **public int[] keysToArray()** – המתודה מחזירה מערך ממוין שמכיל את כל ערכי מפתחות הצמתים שבעץ. פעולה זו מתבצעת ע"י קריאה למתודה keyToArrayRec.

סיבוכיות המתודה היא $O(n)$, שכן היא מבצעת קריאה למתודה בעלת סיבוכיות זמן של $O(n)$.

- **private int keyToArrayRec(IAVLNode node, int curPos, int[] output)** – המתודה מעדכנת רקורסיבית את המערך output להיות מערך ממוין המכיל את כל ערכי מפתחות הצמתים שבעץ, ומחזירה את המיקום הנוכחי במערך עבור מימוש הרקורסיביות. ראשית המתודה קוראת לעצמה עבור הבן השמאלי של הצומת node, אם קיים. לאחר מכן המתודה מוסיפה את המפתח של node למערך, ולבסוף המתודה קוראת לעצמה עבור הבן הימני של הצומת node, אם קיים. תנאי עצירת הרקורסיה הוא אם הצומת node הוא עלה, ובמצב זה המפתח של הצומת node מוכנס למערך.
  סיבוכיות המתודה היא $O(n)$, שכן המתודה עוברת על כל הצמתים בעץ, ועבור כל צומת היא מבצעת מספר קבוע של פעולות בעלות סיבוכיות זמן קבועה.
- **public String[] infoToArray()** – המתודה מחזירה מערך שמכיל את המידע של הצמתים בסדר ממוין לפי מפתחות הצמתים שבעץ. פעולה זו מתבצעת ע"י קריאה למתודה infoToArrayRec.
  סיבוכיות המתודה היא $O(n)$, שכן היא מבצעת קריאה למתודה בעלת סיבוכיות זמן של $O(n)$.
- **private int infoToArrayRec(IAVLNode node, int curPos, String[] output)** – המתודה מעדכנת רקורסיבית את המערך output להיות מערך המכיל את המידע של הצמתים בעץ בסדר ממוין לפי מפתחות הצמתים, ומחזירה את המיקום הנוכחי במערך עבור מימוש הרקורסיביות. ראשית המתודה קוראת לעצמה עבור הבן השמאלי של הצומת node, אם קיים. לאחר מכן המתודה מוסיפה את המידע של node למערך, ולבסוף המתודה קוראת לעצמה עבור הבן הימני של הצומת node, אם קיים. תנאי עצירת הרקורסיה הוא אם הצומת node הוא עלה, ובמצב זה המידעשל הצומת node מוכנס למערך.
  סיבוכיות המתודה היא $O(n)$, שכן המתודה עוברת על כל הצמתים בעץ, ועבור כל צומת היא מבצעת מספר קבוע של פעולות בעלות סיבוכיות זמן קבועה.
- **public int size()** – הפעולה מחזירה את כמות הצמתים בעץ. בפרט, אם העץ ריק, הפעולה מחזירה 0.
  סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.
- **public IAVLNode getRoot()** – הפעולה מחזירה את שורש העץ, במידה וקיים. אם העץ ריק, מוחזר null.

סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעת פעולה בעלת סיבוכיות זמן קבועה.

- **public AVLTree[] split(int x)** – המתודה מפצלת את העץ הנוכחי ומחזירה מערך בעל 2 עצי AVL, כאשר העץ הראשון במערך מכיל את כל הצמתים בעץ בעלי מפתח קטן מ-x, והעץ השני במערך מכיל את כל הצמתים בעץ בעלי מפתח גדול מ-x.
סיבוכיות המתודה היא $O(\log(n))$, מכיוון שהיא קוראת למספר קבוע של פעולות בעלות סיבוכיות $O(\log(n))$, ועוד $O(\log(n))$ פעולות בעלות סיבוכיות זמן קבועות.
- **private void disconnectFromParent(IAVLNode node)** – המתודה מנתקת את הצומת node מההורה שלו. כלומר, המתודה משנה את ה-parent של node להצביע ל-null, ומשנה את המצביע של ההורה של node אל node להיות מצביע אל צומת וירטואלי.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעות מספר קבוע של פעולות בעלות סיבוכיות זמן קבועות.
- **public int join(IAVLNode x, AVLTree t)** – המתודה מקבלת צומת x בעל מפתח k ועץ AVL t, כך שכל מפתחות הצמתים ב-t קטנים/גדולים מ-k וכל מפתחות הצמתים בעץ הנוכחי גדולים/קטנים מ-k בהתאמה. המתודה מאחדת את 2 העצים ואת הצומת x לעץ הנוכחי, תוך שמירה על תכונות של עץ AVL.
סיבוכיות המתודה היא $O(|this.height - t.height| + 1)$, מכיוון שמתבצעות כמות פעולות בעלות סיבוכיות זמן קבוע כגודל הפרשי גבהי העצים.
- **private void promote(IAVLNode node)** – המתודה מעלה את הדרגה של הצומת node באחד. מכיוון שהעץ הוא AVL, דרגת העץ היא גובה העץ, ולכן המתודה מעלה את גובה העץ במקום.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעות מספר קבוע של פעולות בעלות סיבוכיות זמן קבועות.
- **private void demote(IAVLNode node)** – המתודה מפחיתה את הדרגה של הצומת node באחד. מכיוון שהעץ הוא AVL, דרגת העץ היא גובה העץ, ולכן המתודה מפחיתה את גובה העץ במקום.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעות מספר קבוע של פעולות בעלות סיבוכיות זמן קבועות.
- **private void rotateLeft(IAVLNode node)** – המתודה מבצעת סיבוב לשמאל מהצומת node, לפי הגדרת סיבוב לשמאל בעץ.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעות מספר קבוע של פעולות בעלות סיבוכיות זמן קבועות.

- **private void rotateRight(IAVLNode node)** – המתודה מבצעת סיבוב לימין מהצומת node, לפי הגדרת סיבוב לימין בעץ.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעות מספר קבוע של פעולות בעלות סיבוכיות זמן קבועות.
- **private IAVLNode successor(IAVLNode node)** – המתודה מחזירה את העוקב של הצומת node. כלומר, המתודה מחזירה את הצומת בעלת המפתח המינימלי הגדול מהמפתח של node.
סיבוכיות המתודה היא $O(\log(n))$, מכיוון שהמתודה עוברת על צמתים הנמצאים באותו ענף עם node, ולכן כמות הצמתים עליהם המתודה עוברת חסומה ע"י גובה העץ, $\log(n)$, ועל כל צומת מתבצע מספר קבוע של פעולות בעלות סיבוכיות זמן קבועה.
- **private IAVLNode predecessor(IAVLNode node)** – המתודה מחזירה את הקודם של הצומת node. כלומר, המתודה מחזירה את הצומת בעלת המפתח המקסימלי הקטן מהמפתח של node.
סיבוכיות המתודה היא $O(\log(n))$, מכיוון שהמתודה עוברת על צמתים הנמצאים באותו ענף עם node, ולכן כמות הצמתים עליהם המתודה עוברת חסומה ע"י גובה העץ, $\log(n)$, ועל כל צומת מתבצע מספר קבוע של פעולות בעלות סיבוכיות זמן קבועה.
- **private void setParentSon(IAVLNode parent, IAVLNode son, boolean isLeft)** – המתודה מעדכנת את ההורה של son להיות מצביע לparent, ואת הבן השמאלי/ימני של parent להיות מצביע לson, בהתאם לisLeft.
סיבוכיות המתודה היא $O(1)$, מכיוון שמתבצעות מספר קבוע של פעולות בעלות סיבוכיות זמן קבועות.
- **private void updateNodesSize(IAVLNode node)** – המתודה מעדכנת את ערכי השדות size של כל הצמתים מהצומת node ועד לשורש העץ.
עדכון השדות מתבצע לפי השדות size של הבנים של כל צומת.
סיבוכיות המתודה היא $O(\log(n))$, מכיוון שהמתודה עוברת על כל הצמתים מ-node ועד לשורש, וכמות הצמתים הזו חסומה על ידי גובה העץ, $\log(n)$, ועל כל צומת מתבצעות כמות קבועה של פעולות בעלות סיבוכיות זמן קבועה.

מענה חלק ניסויי בפרויקט מעשי עצי AVL

שאלה מספר 1

סעיף א'

מספר סידורי i	מספר חילופים במערך ממוין-הפוך	עלות החיפושים במיון AVL עבור מערך ממוין-הפוך	מספר החילופים במערך מסודר אקראית	עלות החיפושים במיון AVL עבור מערך מסודר אקראית
1	1999000	36884	986808	31309
2	7998000	81764	3970260	74194
3	31996000	179524	15963709	158845
4	127992000	391044	64134963	358660
5	511984000	846084	256245267	789710

סעיף ב'

ראשית נחשב את מספר החילופים באופן תאורטי במערך ממוין-הפוך:
נשים לב שבמערך ממוין הפוך, ולכן איבר במיקום ה-i גדול מכל האיברים שבאים אחריו, ובמהלך המיון הוא יבצע חילוף עם איברים אלה, כלומר הוא יבצע $n-i$ חילופים. לכן סכום החילופים יהיה בדיוק:

$$\sum_{i=1}^{n} n-i = \frac{n*(a_1+a_n)}{2} = \frac{n*(n-1+0)}{2} = \frac{n*(n-1)}{2} = \frac{n^2-n}{2}$$

כעת נחשב את עלות החיפושים באופן תאורטי במערך ממוין-הפוך:
נשים לב שבכל הכנסה, החיפוש מתחיל מהאיבר המקסימלי לפי ההגדרה, ובנוסף האיבר המוכנס קטן יותר משאר איברי העץ מכיוון שהמערך ממוין-הפוך, ולכן מיקומו לאחר ההכנסה יהיה מיקום האיבר המינימלי. כדי להגיע מהשורש אל המינימום או אל המקסימום יש לעבור על הענף הימני או על הענף השמאלי של העץ בהתאמה, ולכן כדי להגיע אל הצמתים האלה מהשורש יש לעבור על כמות צמתים השווה לגובה העץ. לכן, כדי להגיע מצומת המקסימום למינימום יש לעבור על כמות צמתים השווה לגובה העץ כדי להגיע מהמקסימום לשורש ואז לעבור שוב על כמות צמתים השווה לגובה העץ כדי להגיע מהשורש אל המינימום, ובסה"כ נעבור על כמות צמתים השווה לפעמיים גובה העץ. כפי שנלמד בכיתה, גובה עץ AVL עם i איברים הוא $\boldsymbol{\Theta(\log(i))}$, ולכן כמות הצמתים עליהם נעבור בחיפוש של כל הכנסה היא $\boldsymbol{2*\Theta(\log(i))=\Theta(\log(i))}$, כאשר i זה כמות הצמתים בעץ בזמן ההכנסה. לכן סה"כ עלויות החיפושים של ההכנסות היא:

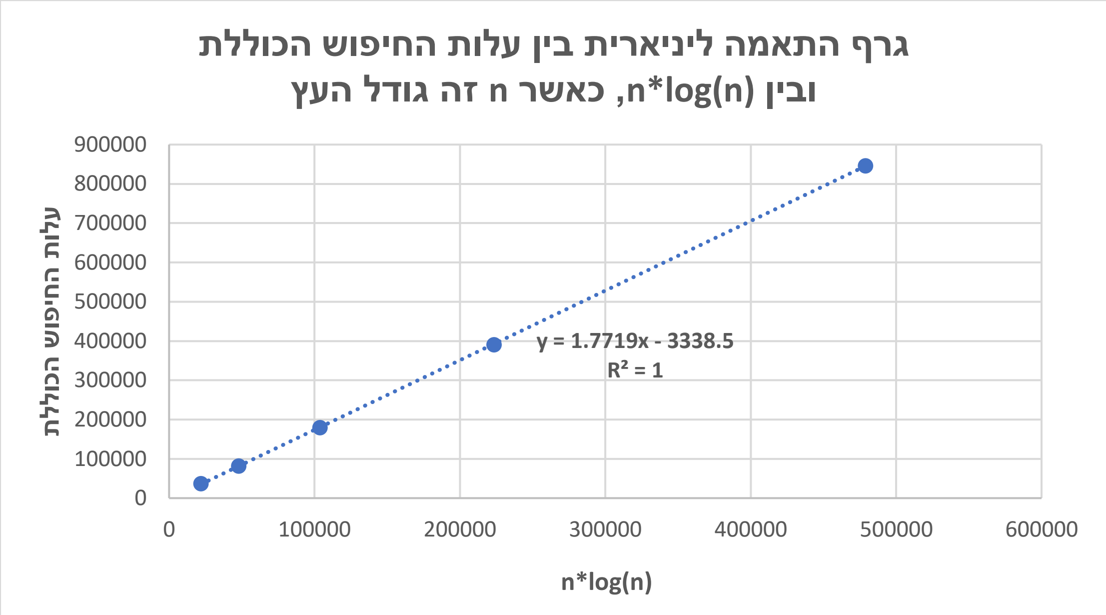
$$\sum_{i=1}^{n} \boldsymbol{\Theta}(\log(\boldsymbol{i})) = \boldsymbol{\Theta}\left(\log\left(\prod_{i=1}^{n} \boldsymbol{i}\right)\right) = \boldsymbol{\Theta}(\log(\mathbf{n}!)) = \boldsymbol{\Theta}(\boldsymbol{n}\log(\boldsymbol{n}))$$

כאשר המעבר הראשון נובע מחוקי לוגריתמים, המעבר השני נובע מיידית מהגדרת עצרת, והמעבר השלישי הוכח בתרגול.

<u>סעיף ג'</u>

ניתן לראות שכמות ההחלפות שהתקבלה בטבלה עבור כל i מתאימה ישירות לנוסחה שקיבלנו בסעיף ב', כאשר $\boldsymbol{n = 1000 * 2^{i}}$.

עבור עלויות החיפוש, נחשב את $\boldsymbol{n\log(n)}$ עבור כל i ונבצע את ההתאמה הליניארית הבאה:



ניתן לראות שיש התאמה ליניארית לפי מדד R^2, ולכן עלות החיפוש הכוללת היא אכן $\boldsymbol{\Theta(n\log n)}$.

<u>סעיף ד'</u>

נניח שהאיבר ה-i במערך דורש h_i החלפות עם האיברים שבאים לפניו במערך. נסמן את סכום כל החילופים במערך ב-h, ומתקיים $\boldsymbol{h = \sum_{i=1}^{n} h_i}$. אם דרושים h_i החלפות בעץ ה-AVL יש h_i איברים שגדולים מהאיבר ה-i, ולכן קיים תת עץ מינימלי המכיל את h_i איברים אלו אליו יוכנס האיבר ה-i. תת העץ הזה הוא בגובה $\boldsymbol{\log(h_i)}$, ולכן עלות ההכנסה d_i של האיבר ה-i תהיה לכל היותר $\boldsymbol{O(\log(h_i))}$.
לכן עלות כל ההכנסות d תהיה:

$$d=\sum_{i=1}^{n} d_i=\sum_{i=1}^{n} O(\log(h_i))=O\left(\log\left(\prod_{i=1}^{n} h_i\right)\right)=O\left(n*\log\left(\sqrt[n]{\prod_{i=1}^{n} h_i}\right)\right)$$
$$\le O\left(n*\log\left(\frac{\sum_{i=1}^{n} h_i}{n}\right)\right)=O\left(n*\log\left(\frac{h}{n}\right)\right)$$

כאשר המעבר הראשון נובע מהגדרת d, המעבר השני נובע מחישוב עלות d_i, המעברים השלישי והרביעי נובעים מחוקי לוגריתמים, המעבר החמישי נובע מאי-שוויון הממוצעים, והמעבר השישי נובע מהגדרת h.

בנוסף עלות הסריקה $in-order$ של העץ היא $O(n)$, ולכן בסה"כ נקבל שסיבוכיות הפעולה $insertionsort$ היא $O\left(n*\log\left(\frac{h}{n}\right)+n\right)$.

שאלה מספר 2

סעיף א'

מספר סידורי i	עלות join ממוצע עבור split אקראי	עלות join מקסימלי עבור split אקראי	עלות join ממוצע עבור split של האיבר המקסימלי בתת העץ השמאלי	עלות join מקסימלי עבור split של האיבר המקסימלי בתת העץ השמאלי
1	2.5555	7	2.6	12
2	2.4545	4	2.4545	13
3	3.2	6	2.8333	14
4	3	8	2.2666	16
5	2.5	8	2.5714	16
6	2.5333	9	2.4	18
7	2	5	2.5	19
8	2.625	7	2.625	21
9	2.9375	8	2.421	21
10	2.75	8	2.8333	23

סעיף ב'

בביצוע פעולת ה-$split$ מתבצעים כמות פעולות $join$ כגודל עומק האיבר עליו מתבצע ה-$split$, שכן יש לבצע $join$ על כל צומת הנמצאת בענף מהשורש לצומת עליה מתבצע ה-split. בנוסף העלות של כל פעולות ה-$join$ שמתבצעות ב-$split$ היא עלות פעולת ה-$split$, כלומר העלות היא $O(\log(n))$, כפי שנלמד בהרצאה.

עבור $split$ של האיבר המקסימלי בתת העץ השמאלי, כמות פעולות ה-$join$ שמתבצעות היא גובה העץ, כלומר $O(\log(n))$, שכן האיבר המקסימלי בהכרח נמצא

בתחתית עץ ה-AVL. לכן, העלות הממוצעת של $join$, שהיא העלות של כל פעולות ה-$join$ חלקי כמות פעולות ה-$join$, היא $\mathbf{\Theta(1)} = \frac{\mathbf{O(\log(n))}}{\mathbf{O(\log(n))}}$.

עבור $split$ של איבר רנדומלי בעץ, כמות פעולות ה-$join$ שמתבצעות שווה לעומק האיבר הרנדומלי, שזה חסום על ידי גובה העץ, כלומר $\mathbf{O(\log(n))}$. לכן העלות הממוצעת של ה-$join$ חסומה על ידי $\mathbf{\Theta(1)} = \frac{\mathbf{O(\log(n))}}{\mathbf{O(\log(n))}}$.

כפי שניתן לראות בטבלה, תוצאות המדידות מתיישבות בהתאם עם הניתוח התאורטי, שכן הגדלת כמות האיברים בעץ לא משנה את העלות הממוצעת בשני המקרים.

סעיף ג'

ניזכר שעלות פעולת $join$ היא הפרשי גבהי העצים עליהם מתבצע ה-$join$.

עבור $split$ של האיבר המקסימלי בתת העץ השמאלי, ה-$join$ המקסימלי יהיה בין תת העץ הימני של שורש עץ ה-AVL, ובין עץ האיברים הגדולים מהמקסימום, שהוא עץ ריק שכן עד לשורש לא קיימים איברים שגדולים מהמקסימום הנ"ל. הגובה של עץ ריק הוא 0, והגובה של תת העץ הימני של השורש הוא $\mathbf{\log(n) - 1}$ או $\mathbf{\log(n) - 2}$, שכן גובה העץ הוא $\mathbf{\log(n)}$ וזה עץ AVL, ולכן עלות פעולת ה-$join$ בין 2 העצים האלו היא $\mathbf{\log(n) - 1 - 0 + 1}$ או $\mathbf{\log(n) - 2 - 0 + 1}$, או במילים אחרות $\mathbf{\Theta(\log(n))}$.

עבור $split$ של איבר רנדומלי בעץ, נשים לב שה-$join$ בעל העלות המקסימלית שמתבצע תחת $split$ בעץ AVL הוא $join$ של תת עץ של השורש (שכן זה תת העץ בעל הגובה הגדול ביותר בעץ) יחד עם עץ ריק (שזה עץ בעל הגובה הקטן ביותר). מקרה כזה קורה למשל ב-$split$ של האיבר המקסימלי של תת העץ השמאלי, כפי שמתואר לעיל. כאמור, העלות של ה-$join$ המקסימלי במקרה הזה היא $\mathbf{\Theta(\log(n))}$. לכן העלות המקסימלית עבור $join$ שמתבצע ב-$split$ על איבר כלשהו בעץ חסומה ע"י $\mathbf{\Theta(\log(n))}$, כלומר העלות היא $\mathbf{O(\log(n))}$.