תרגיל בית מעשי 1: חלק תיאורטי

תומר מילדוורט | 316081355 | Tomer Mildworth | mildworth
ליאור בודנר | 207702861 | Lior Bodner | liorbodner

תיעוד חיצוני

בבניית התרגיל, השתדלנו לחלק את האחריויות של כל מחלקה באופן הגיוני ומופרד ובמחשבה על שימוש הגיוני על ידי לקוח שזר לתרגיל. את מימוש העץ חילקנו ל-2 מחלקות כפי שניתנו בקובץ השלד: AVLNode ו-AVLTree. בכתיבת הקוד, נסינו להקפיד על כתיבה אבסטרקטית וקריאה ככל הניתן: כתיבת מתודות כלליות ופירוקן למספר פונקציות עזר, בחירת שמות משתנים ברורים ובהירים והצמדות לחומר הנלמד בכיתה ובתרגולים.

נתחיל בניתוח המחלקה AVLNode.

צומת - AVLNode

<u>מאפייני המחלקה</u>

- key (int) – מפתח לצומת. ערך מספרי שלם שייחודי לכל צומת בעץ. ערך דיפולטי None.
- value (Any) – ערך כלשהו לצומת. מידע מכל סוג שמצורף לצומת. ערך דיפולטי None.
- left (AVLNode) – בן שמאלי. מצביע על אובייקט צומת אחר. ערך דיפולטי None.
- right (AVLNode) – בן ימני. מצביע על אובייקט צומת אחר. ערך דיפולטי None.
- parent (AVLNode) – הורה. מצביע על אובייקט צומת אחר. ערך דיפולטי None.
- height (int) – גובה בעץ. מספר שלם גדול או שווה ל-(1-) (לצומת דמה). ערך דיפולטי 0.
- size (int) – גודל תת העץ המושרש מהצומת. מספר שלם גדול או שווה ל-0. ערך דיפולטי 1.

שימוש נוסף ומעט שונה במחלקה הוא צומת דמה (dummy_node, נקרא גם צומת וירטואלי). צומת דמה מוגדר להיות צומת עם key שהוא None, בגובה 1-, עם ערך size של 0 וכל המצביעים שלו לצמתים אחרים (left, right, parent) גם הם None. השימוש בצמתי דמה מקל על פעולות שונות בעץ בכך שהוא מבטיח שלצמתים אמיתיים תמיד יהיו בנים כלשהם.

מתודות

$\_\_init\_\_(self, key: int = None, height: int = 0, value = None)$

מטרה : בנאי של המחלקה

שיטת פעולה : אתחול המחלקה לפי הקלטים הנ"ל. כל צומת מאותחלת ללא בנים או הורה וירטואליים, אלון יאותחלו במידת הצורך בהמשך.

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$\_\_repr\_\_(self)$

מטרה : הצגה ברורה של המחלקה

שיטת פעולה : -

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$\_\_eq\_\_(self, other: AVLNode) -> bool$

$\_\_lt\_\_(self, other: AVLNode) -> bool$

$\_\_le\_\_(self, other: AVLNode) -> bool$

$\_\_gt\_\_(self, other: AVLNode) -> bool$

$\_\_ge\_\_(self, other: AVLNode) -> bool$

מטרה : בדיקת יחס בין צמתים

שיטת פעולה : השוואת המפתחות

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$get\_key(self) -> int$

מטרה : קבלת המפתח של הצומת

שיטת פעולה : גישה למאפיין

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$get\_value(self) -> Any$

מטרה : קבלת ערך הצומת

שיטת פעולה : גישה למאפיין

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$get\_left(self) -> AVLNode$

מטרה : קבלת הבן השמאלי של הצומת

שיטת פעולה : גישה למאפיין

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$get\_right(self) -> AVLNode$

מטרה : קבלת הבן הימני של הצומת

שיטת פעולה : גישה למאפיין

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

get_parent(self) −> AVLNode

מטרה : קבלת ההורה של הצומת

שיטת פעולה : גישה למאפיין

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

get_height(self) −> int

מטרה : קבלת ערך הגובה של הצומת

שיטת פעולה : גישה למאפיין

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

get_size(self) −> int

מטרה : קבלת הגודל של הצומת

שיטת פעולה : גישה למאפיין

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

get_relative_direction(self) −> str

מטרה : קובעת את כיוון הצומת ביחס להורה שלו, משמע האם הוא בן ימני, שמאלי או שורש

שיטת פעולה : השוואה של הצומת עם ילדיו של ההורה שלו

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: השוואות קבועות

$get\_bf(self) -> int$

מטרה : מחשבת את ערך ה Balance Factor-של הצומת כפי שנלמד בכיתה

שיטת פעולה : אם קיימים לצומת בנים (אמיתיים או וירטואליים), נחשב לפי גובה הבן השמאלי בחיסור גובה הבן הימני

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$set\_key(self, key: int) -> None$

מטרה : הגדרת מפתח לצומת

שיטת פעולה : כתיבה למאפיין של המחלקה

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$set\_value(self, value: Any) -> None$

מטרה : הגדרת ערך לצומת

שיטת פעולה : כתיבה למאפיין של המחלקה

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$set\_left(self, node: AVLNode) -> None$

מטרה : הגדרת בן שמאלי לצומת

שיטת פעולה : כתיבה למאפיין של המחלקה

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$set\_right(self, node: AVLNode) -> None$

מטרה : הגדרת בן ימני לצומת

שיטת פעולה : כתיבה למאפיין של המחלקה

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$set\_parent(self, node: AVLNode) -> None$

מטרה : הגדרת הורה לצומת

שיטת פעולה : כתיבה למאפיין של המחלקה

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$set\_height(self, h: int) -> None$

מטרה : הגדרת גובה לצומת

שיטת פעולה : כתיבה למאפיין של המחלקה

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$set\_size(self, s: int) -> None$

מטרה : הגדרת גודל לצומת

שיטת פעולה : כתיבה למאפיין של המחלקה

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$height\_manager(self) -> bool$

מטרה : מעדכנת את גובה הצומת ומחזירה ערך בוליאני לגבי ביצוע עדכון לגובה

שיטת פעולה : חישוב הגובה בעת הקריאה והשוואתו עם הערך השמור בצומת. ביצוע עדכון במידת הצורך והחזרת True אם בוצע שינוי

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: עומד בתנאים שהוצגו בכיתה (תלוי בערכי הבנים וניתן לחישוב בזמן קבוע) ולכן מתבצע בזמן קבוע. לאחר מכן השוואה (זמן קבוע).

$update\_size(self) -> None$

מטרה : עדכון גודל הצומת

שיטת פעולה : חישוב הגודל לפי הגודל של הבנים בתוספת 1

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$is\_real\_node(self) -> bool$

מטרה : בדיקה האם הצומת אמיתי (לא דמה)

שיטת פעולה : בדיקה האם קיים מפתח לצומת

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$has\_dummy\_child(self) -> bool$

מטרה : בדיקה האם לצומת קיים לפחות בן דמה אחד

שיטת פעולה : תנאי לוגי שקורא למתודה is_real_node על שני בניו של הצומת

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

is_leaf(self) −> bool

מטרה : בדיקה האם הצומת הוא עלה

שיטת פעולה : בדיקה האם שני בניו של הצומת הם צמתי דמה

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

add_left_dummy(self) −> None

add_right_dummy(self) −> None

מטרה : הוספת בן דמה כבן שמאלי או ימני

שיטת פעולה : יצירת צומת חדש בגובה 1- ובגודל 0, ללא מפתח

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

add_dummy_nodes(self) −> None

מטרה : הוספת 2 ילדי דמה לצומת

שיטת פעולה : קריאה ל-add_left_dummy ול-add_right_dummy

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

set_as_dummy(self) −> None

מטרה : הגדרת הצומת כצומת דמה

שיטת פעולה : שינוי כלל המאפיינים כהגדרת צומת דמה (ללא מפתח, גובה 1-)

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$$\text{set\_as\_other\_node(self, other: AVLNode, with\_parent: bool} = \text{True}) \text{->} \text{None}$$

מטרה : הגדרת צומת מסוים כצומת אחר ללא יצירת צומת חדש

שיטת פעולה : אם הצומת האחר הוא דמה, קריאה ל-set_as_dummy. אחרת, השמה של מאפייני הצומת האחר בצומת הנוכחי

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

עץ - AVLTree

מאפייני המחלקה

- root (AVLNode) – שורש העץ. מצביע לצומת. ערך דיפולטי ()AVLNode.
- min (AVLNode) – הצומת המינימלית בעץ. מצביע לצומת. ערך דיפולטי ()AVLNode.
- max (AVLNode) - הצומת המקסימלית בעץ. מצביע לצומת. ערך דיפולטי ()AVLNode.

בכתיבת המתודות של מחלקת העץ השתדלנו לאזן בין קריאות לשכפול קוד, לכן מצד אחד ישנן מתודות סימטריות, כמו למשל גלגול ימין וגלגול לשמאל ב-2 מתודות נפרדות, או חיפוש מינימום ומקסימום, שמבצעות פעולות כמעט זהות שניתן היה לאחד למתודה אחת, מנגד, איחוד שכזה היה פוגע בקריאות הקוד בדיוק בגלל אותם שינויים עדינים בין כל צד בסימטריה. בכדי לפתור קונפליקט זה עטפנו כפילויות שכאלו במתודה אחת, בכך מובטח שהן תקראנה רק בקריאות עמוקות יותר בכל מתודה "ראשית" ולא תבלבלנה את הקורא. נוסף על כך השימוש ב-relative_direction מאפשר לצמצם כפילויות ולקונן את כלל האפשרויות למספר מצומצם של מתודות.

מתודות

$\_\_init\_\_(self, root\text{: } AVLNode\ =\ AVLNode())$

מטרה : בנאי של המחלקה

שיטת פעולה : אתחול המחלקה לפי הגדרת שורש. שורש יהיה צומת דמה כערך ברירת מחדל

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$\_\_repr\_\_(self)$

מטרה : הצגה ברורה של המחלקה

שיטת פעולה : -

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$\text{is\_empty(self)} -> \text{bool}$

מטרה : בדיקה האם העץ ריק

שיטת פעולה : קריאה של is_real_node לשורש

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: קריאה של מתודה של $O(1)$

$\text{get\_root(self)} -> \text{AVLNode}$

מטרה : קבלת שורש העץ

שיטת פעולה : קריאה למאפיין root של העץ הנוכחי

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$\text{should\_update\_min(self, node: AVLNode)} -> \text{bool}$

$\text{should\_update\_max(self, node: AVLNode)} -> \text{bool}$

מטרה : בדיקה האם יש צורך לעדכן את המצביעים לצומת המינימלי/מקסימלי בעץ

שיטת פעולה : אם קיים מינימום/מקסימום, השוואת מפתחות של הצומת הנוכחי עם הצומת המינימלי/מקסימלי

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: -

$\text{get\_subtree\_min(self)} -> \text{AVLNode}$

$\text{get\_subtree\_max(self)} -> \text{AVLNode}$

מטרה : החזרת הצומת בעל המפתח המינימלי/מקסימלי

שיטת פעולה : החל מהשורש, נרד בעץ עד לצומת השמאלי/ימני ביותר

סיבוכיות זמן : $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: ירידה בכל העץ היא לינארית בגובה העץ, אשר מובטח להיות $O(\log n)$ תמיד.

$\text{init\_min\_max(self)} \text{ —> None}$

מטרה : אתחול מינימום ומקסימום לעץ

שיטת פעולה : אם העץ לא ריק (קריאה ל-is_empty), נבצע השמה למאפייני min/max של העץ ע"י הפלטים של get_subtree_min/max

סיבוכיות זמן : $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: קריאה לשתי מתודות בסיבוכיות $O(\log n)$ אחת אחרי השנייה, לכן סה"כ $O(\log n)$.

$\text{search(self, key: int)} \text{ —> AVLNode | None}$

מטרה : מציאת צומת בעל מפתח נתון

שיטת פעולה : נאתחל מצביע לשורש, ובלולאת while נבצע השוואות בין המפתח של המצביע למפתח מהקלט. אם הקלט קטן מהמפתח הנוכחי נקדם את המצביע לבן השמאלי, ואם גדול מהמפתח הנוכחי נקדם לבן הימני. אם מתקיים שוויון נחזיר את הצומת עליה נצביע. הלולאה תעצור כשנגיע לצומת דמה. אם לא נמצא צומת בעל מפתח זהה נחזיר None.

סיבוכיות זמן : $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: לכל היותר נבצע הליכה מהשורש לעלה העמוק ביותר בעץ. מכיוון שמובטח לנו עץ AVL אנו יודעים שההליכה לינארית בגובה שמובטח להיות $\log n$ לכל היותר, בכל צעד נבצע פעולות אריתמטיות של $O(1)$, לכן סה"כ $O(\log n)$.

$\text{set\_as\_child\_after\_rotation(self, node: AVLNode, relative\_direction: str)} \text{ —> None}$

מטרה : קביעת הורה לצומת לאחר ביצוע גלגול

שיטת פעולה : אם הכיוון היחסי הוא "שורש", הגדרת הצומת כשורש העץ. אחרת, קביעת השורש כבן של ההורה שלו בכיוון היחסי. לצומת יש הורה בקריאה לפונקציה מכיוון שהיא נקראת אך ורק לאחר ביצוע גלגול. היא למעשה "מסדרת" את המצביעים לקראת גלגול

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: החלפת מצביעים.

$right\_rotation(self, node: AVLNode, relative\_direction: str \mid None) -> None$

$left\_rotation(self, node: AVLNode, relative\_direction: str \mid None) -> None$

מטרה : ביצוע גלגול ימינה/שמאלה

שיטת פעולה : תחילה נבצע שינוי מצבעים לגלגול מלא בהתאם לנלמד בכיתה ללא השמת ההורה האחרונה. כעת, אם התקבל relative_direction נבצע "חצי גלגול" בכיוון המתאים ונעדכן את הגובה של הצמתים הרלוונטיים. אחרת, נקרא ל- set_as_child_after_rotation עבור הצומת המתאים והכיוון הנתון. לבסוף נבצע את השמת ההורה האחרונה ונעדכן את הגדלים של הצמתים.

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: בכל מקרה נבצע שינוי מצביעים שמתבצע בזמן קבוע, לאחר מכן נקרא ל-height_manager או ל-set_as_child_after_rotation, שניהם פועלים בזמן קבוע. לבסוף נבצע עוד שינוי משתנה יחיד ונחשב size עבור כל צומת - זמן קבוע. נבצע כמות קבועה (משתנה ללא תלות בכמות האיברים) של פעולות לכל סוג קריאה, לכן סה"כ $O(1)$.

$right\_then\_left\_rotation(self, node: AVLNode, relative\_direction: str) -> None$

$left\_then\_right\_rotation(self, node: AVLNode, relative\_direction: str) -> None$

מטרה : ביצוע גלגול ימינה/שמאלה ואז שמאלה/ימינה

שיטת פעולה : נבצע קריאה ל-right_rotation ולאחריה ל-left_rotation (או להיפך עבור שמאלה ואז ימינה) כאשר הקריאה הראשונה תתבצע על הבן הימני/שמאלי של הצומת הנתונה בכדי "לסדר" את העץ לקראת חצי הגלגול השני

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: נבצע 2 קריאות עוקבות למתודות שפועלות בזמן קבוע בכל מקרה, לכן גם מתודות אלה יפעלו בזמן קבוע בעצמן.

$\text{rotate}(\text{self}, \text{node: AVLNode}) -> \text{ int}$

מטרה: מבצעת את מנגנון הגלגול של עצי AVL כפי שנלמד בכיתה

שיטת פעולה: ראשית המתודה בודקת האם הצומת הנתונה היא בן שמאלי/ימני/שורש ע״י קריאה ל-get_relative_direction של AVLNode ומחשבת את ה-BF שלו ע״י get_bf. בהתאם ל-BF המתקבל, המתודה מחשבת את ה-BF של הבן המתאים (BF שווה ל-2 משמע נבצע סיבוב עם הבן השמאלי ונחשב לפיו, אחרת נחשב עבור הבן הימני). במידה וה-BF של הצומת שווה ל-2 או 2- יתבצע סיבוב (או חצי סיבוב) בהתאם ל-BF של הילד ע״י קריאות למתודות הסיבוב המתאימות כמתואר קודם לכן. בנוסף המתודה מונה את מספר פעולות האיזון (קבוע מראש כתלות בסוג הסיבוב). לבסוף נעדכן את הגודל והגובה החדשים ע״י קריאות ל-height_manager ו-update_height של AVLNode ונחזיר את מניית פעולות האיזון

סיבוכיות זמן: $O(1)$

ניתוח סיבוכיות במקרה הגרוע: חישוב BF של הצומת וילדיו מתבצע בזמן קבוע וכך גם כל פעולות הסיבוב כפי שתואר. גם עדכון הגובה והגודל מתבצע בזמן קבוע לכן סה״כ המתודה פועלת כולה בזמן קבוע ללא תלות בכמות האיברים בעץ

$\text{rebalance\_up}(\text{self}, \text{start\_node: AVLNode}) -> \text{ int}$

מטרה: ביצוע ומניית פעולות איזון בעץ לשמירה על מאפייני עץ AVL

שיטת פעולה: שיטת פעולה Bottom-Up: החל מצומת הנתון, נבדוק האם יש צורך בעדכון גובה העץ ע״י קריאה למתודת height_manager של AVLNode ונחשב את הערך המוחלט של ה-BF של הצומת ע״י קריאה ל-get_bf. כעת, כפי שנלמד בכיתה אנחנו יודעים שה-BF הוא בין 2- ל-2, לכן נקבל מספר בין 0 ל-2. בהצלבה עם בדיקת שינוי הגובה, נפצל ל-3 מקרים:

1. <u>BF קטן ממש מ-2 ובוצע שינוי גובה</u>: נעדכן גודל, נקדם את המצביע להורה ונוסיף פעולת איזון.
2. <u>BF קטן ממש מ-2 ולא בוצע שינוי גובה</u>: נעדכן גודל ונחזיר את מספר פעולות האיזון שנצבר.
3. <u>BF שווה 2</u>: נבצע גלגול ע״י קריאה למתודת rotate על המצביע הנוכחי ונצבור את פעולות האיזון שהמתודה מחזירה. נחזיר את פעולות האיזון שהצטברו

סיבוכיות זמן: $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: נשים לב שלכל היותר בכל מקרה נבצע פעולות של $O(1)$: שינוי מצביע, חישוב מאפיין ע״י הבנים בזמן קבוע (size) או ביצוע rotate שכאמור היא $O(1)$. במקרה

הגרוע נתחיל בעלה העמוק ביותר ונקדם את המצביע עד שנגיע לשורש, כאשר בכל קידום נבצע לכל היותר $O(1)$ פעולות, לכן הסיבוכיות לינארית בגובה העץ, משמע $O(\log n)$.

$\text{BST\_insert(self, node: AVLNode)} -> \text{None}$

מטרה: הכנסה לעץ לפי אלגוריתם הכנסה לעץ חיפוש בינארי קלאסי לפי הנלמד בכיתה

שיטת פעולה: המתודה מייצרת לצומת צמתי דמה כבנים עם קריאה ל-add_dummy_nodes של AVLNode ומוודאה האם העץ ריק (במקרה כזה תכניס את הצומת כשורש). אחרת, נרד בעץ בלולאה בהתאם להשוואות בין המפתח של הצומת הנתון לצמתים בעץ ע"י 2 מצביעים עד שנגיע למיקום המתאים להכנסת הצומת ע"י זיהוי צומת דמה (קריאה ל-is_real_node) לאחר שמצאנו, נכניס את הצומת למקום המיועד ונסדר את המצביעים בהתאם

סיבוכיות זמן: $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: הכנסה לעץ חיפוש בינארי לינארית עם הגובה של העץ, אך מכיוון שכתנאי קדם אנו יודעים שאנו מכניסים לעץ AVL בלבד, גובה העץ חסום ע"י $\log n$, לכן במקרה הגרוע נקבל $O(\log n)$.

$\text{insert(self, key: int, val)} -> \text{int}$

מטרה: הכנסת צומת לעץ בהינתן מפתח וערך, וספירת כמות פעולות האיזון שבוצעו בהכנסה

שיטת פעולה: ראשית, נייצר צומת חדש מהנתונים שהוכנסו. כעת, נבצע הכנסה לעץ לפי אלגוריתם הכנסה של עץ חיפוש בינארי קלאסי ע"י קריאה ל-BST_insert. לאחר מכן נבדוק האם יש צורך בעדכון בצומת המינימלי/מקסימלי בעץ ע"י קריאה ל-should_update_min/max.
לבסוף, נבצע פעולות איזון לעץ בכדי לשמר את המבנה של עץ AVL ע"י קריאה למתודה rebalance_up החל מההורה של הצומת שהוכנס. את הפלט של rebalance_up אשר מונה את פעולות האיזון שבוצעו כפי שהוגדרו במטלה נשמור במשתנה ונחזיר אותו כפלט

סיבוכיות זמן: $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: כלל מתודות העזר נקראות אחת אחרי השנייה לכן נסכום את סיבוכיות כלל הפעולות:

1. $\text{should\_update\_min/max} - O(1)$
2. $\text{BST\_insert} - O(\log n)$
3. $\text{rebalance\_up} - O(\log n)$

לכן סה״כ נקבל $O(\log n)$.

successor(self, node: AVLNode) −> AVLNode | None

predecessor(self, node: AVLNode) −> AVLNode | None

מטרה : מוצאת את הצומת עם המפתח הבא/הקודם בגודלו בעץ, אם קיים

שיטת פעולה : ראשית, המתודה בודקת האם הצומת הנתון הוא המקסימלי/מינימלי או שהעץ ריק (קריאה ל-is_empty), במקרה כזה אין לו צומת עוקב/קודם והמתודה תחזיר None. אחרת, אם לצומת קיים בן ימני/שמאלי (קריאה ל-is_real_node של AVLNode), נקדם את המצביע אליו. כעת, נרד בכיוון הנגדי עד כמה שניתן ונחזיר את הצומת האחרון שהצבענו עליו. אם לצומת אין בן ימני/שמאלי, נעלה למעלה בעץ בכיוון המתאים עד כמה שניתן ונחזיר את הצומת הבאה בכיוון ההפוך, הכל כפי שנלמד בכיתה

סיבוכיות זמן : $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: במקרה הגרוע נעלה או נרד בכל העץ ונבצע פעולות בזמן קבוע בכל איטרציה, לכן הסיבוכיות לינארית לגובה העץ. מכיוון שמובטח לנו עץ AVL, גובה העץ חסום ע״י $\log n$ ובסך הכל נקבל $O(\log n)$.

replace_nodes(self, original_node: AVLNode, new_node: AVLNode) −> None

מטרה : הכנסת צומת חדש למיקום של צומת נתון

שיטת פעולה : תחילה נבדוק את המיקום היחסי של הצומת המקורי ע״י קריאה ל-get_relative_direction של AVLNode, ולאחר מכן נבצע החלפת מצביעים עם עבור הבנים של הצומת המקורי, וחיבור במיקום המתאים של הצומת החדש עם ההורה של הצומת המקורי

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: החלפת משתנים וקריאה למתודה של $O(1)$.

$BST\_delete(self, node: AVLNode) -> AVLNode \mid None$

מטרה: מחיקת צומת מהעץ לפי אלגוריתם מחיקה של עץ חיפוש בינארי קלאסי וסימון הצומת ממנו נבצע איזון מעלה

שיטת פעולה: המתודה מפרידה בין שלושת המקרים האפשריים למחיקה:

1. הצומת הוא עלה
2. לצומת בן אחד
3. לצומת 2 בנים

נבדיל בין מקרים 1 ו-2 ל-3 ע"י בדיקה האם לצומת יש בן דמה (קריאה ל-has_dummy_node של AVLNode) מכיוון שאם יש לו, אז אנחנו לא במקרה 3. כעת, נבדוק האם אנחנו במקרה 1 או 2, נמחק את הצומת, נחליפו בדמה ונקשר את הבנים הרלוונטיים להורה של הצומת המחוק. נחזיר את ההורה. במקרה 3, כפי שלמדנו בכיתה, נרצה להחליף את הצומת המיועד למחיקה עם העוקב שלו. לשם כך, ראשית נבצע קריאת successor למציאת העוקב, לאחר מכן נבצע קריאה רקורסיבית ל-BST_insert עם הצומת העוקב בכדי לנתקו מן העץ. לבסוף נבצע החלפה של הצומת המקורי עם העוקב שלו ע"י קריאה למתודה replace_nodes. נחזיר את הצומת העוקב במקומו החדש

סיבוכיות זמן: $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: נשים לב שמקרים 1 ו-2 פועלים בזמן קבוע שכן מתבצעות שם כמות קבועה של שינוי מצביעים, משמע $O(1)$. לגבי מקרה 3: מובטח לנו שבאופן בו מימשנו את מתודת successor לאחר קריאה אחת נקבל צומת שאין לה בן שמאלי. עובדה זו נטועה בכך שהאלגוריתם למציאת עוקב עוצר כאשר לא ניתן להמשיך לרדת בעץ עם היצמדות שמאלה. כפי שראינו, סיבוכיות successor היא $O(\log n)$. כעת, בקריאה הרקורסיבית בהכרח נעמוד בתנאי מקרה 1 או 2, לכן הקריאה תיעצר בעומק רקורסיה 1 ותבצע פעולות בסיבוכיות של $O(1)$. מתודת החלפת הצמתים replace_nodes פועלת ב-$O(1)$ כפי שראינו. לכן, מקרה 3 במקרה הגרוע הוא $O(\log n)$. לסיכום, במקרה הגרוע בעת מחיקה ניכנס למקרה 3 ולכן סיבוכיות המתודה הכללית היא $O(\log n)$.

$\mathrm{delete(self, node: AVLNode)} -> \mathrm{int}$

מטרה : מחיקת צומת מן העץ והחזרת מספר פעולות האיזון שבוצעו

שיטת פעולה : ראשית נבדוק האם הצומת הוא המינימום/מקסימום ונעדכן את המאפיין בעץ לפי העוקב/קודם של הצומת. לאחר מכן נבצע מחיקת עץ חיפוש בינארי קלאסית ע"י קריאה ל-BST_delete ונשמור את הצומת המוחזר. אם אכן התבצעה מחיקה, החל מהצומת המוחזר נתחיל לבצע תיקונים Bottom-Up לשמירת מאפייני עץ AVL ע"י קריאה ל-rebalance_up שבתורה תחזיר את מספר פעולות האיזון שהתבצעו, אותן נחזיר. אם לא התבצעה מחיקה נחזיר 0.

סיבוכיות זמן : $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: פעולת successor או predecessor אחת : $O(\log n)$. קריאה אחת ל-BST_delete: $O(\log n)$. קריאה ל-rebalance_up: $O(\log n)$. הפעולות נקראות אחת אחרי השנייה לכן סה"כ : $O(\log n)$.

$\mathrm{size(self)} -> \mathrm{int}$

מטרה : קבלת גודל העץ

שיטת פעולה : קריאה ל-get_size של AVLNode על שורש העץ

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: שינוי מצביעים

$\mathrm{set\_as\_other\_tree(self, other: AVLTree)} -> \mathrm{None}$

מטרה : הגדרת עץ אחר כעץ המקורי

שיטת פעולה : העתקת מצביעי השורש, המינימום והמקסימום של העץ המקורי לעץ החדש

סיבוכיות זמן : $O(1)$

ניתוח סיבוכיות במקרה הגרוע: שינוי מצביעים

$\text{get\_subtree\_root(self, direction: str, subtree\_height: int)} -> \text{ AVLNode}$

מטרה: קבלת שורש של תת עץ בגובה רצוי בכיוון יחיד מהשורש

שיטת פעולה: החל מהשורש, נרד על הדופן השמאלית או הימנית של העץ בהתאם לקלט direction, עד שנגיע לצומת בגובה קטן או שווה לקלט subtree_height. נחזיר את הצומת שנצביע עליה

סיבוכיות זמן: $\mathrm{O}(\log n)$

ניתוח סיבוכיות במקרה הגרוע: במקרה הגרוע נרצה תת עץ בגובה 0, משמע עלה, לכן נבצע הליכה על כל גובה העץ, מכאן שהסיבוכיות לינארית לגובה העץ שהיא $O(\log n)$.

$\text{join\_with\_dummy(self, other: AVLTree, pivot\_node: AVLNode)} -> \text{ None}$

מטרה: ביצוע איחוד של עץ ריק ועץ לא ריק

שיטת פעולה: נבדוק מי מהעצים לא ריק ע"י קריאה ל-is_empty, ונבצע פעולת הכנסה של צומת הציר לעץ זה. נקרא למתודה set_as_other_tree במידת הצורך כדי לוודא שהעץ עליו נקראת המתודה יהיה העץ שאינו ריק

סיבוכיות זמן: $\mathrm{O}(\log n)$

ניתוח סיבוכיות במקרה הגרוע: בדיקת עץ ריק עולה $\mathrm{O}(1)$, הכנסת צומת במקרה הגרוע היא $\mathrm{O}(\log n)$ כפי שראינו, והחלפת עצים ע"י set_as_other_tree עולה גם היא $\mathrm{O}(1)$ לכן בסה"כ נקבל סיבוכיות של $\mathrm{O}(\log n)$.

$\text{join\_from\_left(self, other: AVLTree, pivot\_node: AVLNode)} -> \text{ int}$

מטרה: איחוד מצידו השמאלי של העץ הנוכחי

שיטת פעולה: ראשית נחשב את הבדל הגבהים בין העץ הנוכחי לעץ האחר. כעת, בהתאם להבדל הגבהים, המתודה תבצע אחת מ-3 אפשרויות:

1. <u>איחוד פשוט</u>: הבדל הגבהים קטן או שווה 1 לכן נוכל לבצע חיבור של שורשי העצים כבנים של צומת הציר והגדרת צומת הציר כשורש החדש.
2. <u>הנוכחי גבוה מהאחר</u>: נקרא למתודה get_subtree_root עם כיוון שמאל ובגובה של העץ האחר. כעת נחבר את צומת הציר עם השורש של העץ האחר והשורש של תת העץ שמצאנו ונחליף את שורש תת העץ עם צומת הציר.

3. <u>האחר גבוה מהנוכחי</u>: נבצע פעולה סימטרית למקרה (2) כאשר נחליף בין העץ הנוכחי לאחר ובין כיוון שמאל לימין. נגדיר את השורש של העץ הנוכחי להיות השורש של העץ האחר כדי לוודא שהעץ המאוחד הוא זה שנקרא עליו המתודה.

המתודה תחזיר ערך מוחלט של הפרש הגבהים ועוד 1 כנדרש.

סיבוכיות זמן: $O(|\text{other. root. height} - \text{self. root. height}| + 1)$

ניתוח סיבוכיות במקרה הגרוע: במקרה של איחוד פשוט נבצע שינוי קבוע של משתנים ולכן הסיבוכיות תהיה $O(1)$. במקרים 2 ו-3 נבצע קריאה למתודה get_subtree_root אך נשים לב שכמות הצעדים מהשורש שהמתודה תבצע חסומה ע״י הבדל הגבהים בין העצים, שכן אנו רוצים לייצר תת עץ השווב בגובהו לעץ אחר. לכן, במקרה זה סיבוכיות המתודה תהיה $O(|\text{other. root. height} - \text{self. root. height}|)$. נוסף על הקריאה למתודה נבצע כמו קבועה של שינוי משתנים. לכן, ניתן לכתוב בהכללה את הסיבוכיות של 3 הפעולות יחד כ- $O(|\text{other. root. height} - \text{self. root. height}| + 1)$.

join_from_right(self, other: AVLTree, pivot_node: AVLNode) −> int

מטרה: איחוד מצידו הימני של העץ הנוכחי

שיטת פעולה: סימטרית למתודה join_from_left, כאשר כאן העץ הנוכחי הוא השמאלי ביותר

סיבוכיות זמן: $O(|\text{other. root. height} - \text{self. root. height}| + 1)$

ניתוח סיבוכיות במקרה הגרוע: זהה לניתוח ב-join_from_left.

join(self, tree: AVLTree, key: int, val: Any) −> int

מטרה: איחוד שני עצי AVL

שיטת פעולה: ראשית נבנה צומת ציר מ-key ו-val הנתונים. שנית, נבדוק האם אחד העצים ריק ע״י קריאה ל-is_empty. אם כן, נחשב את הפרש הגבהים, נבצע איחוד ע״י join_with_dummy ונחזיר את ההפרש שחישבנו. אחרת, נמקם את העץ הנוכחי ביחס לעץ האחר ע״י השוואת המפתח של שורש העץ הנוכחי עם המפתח של צומת הציר (מובטח לנו שמפתח צומת הציר נמצא בין כל מפתחות עץ אחד לכל מפתחות העץ האחר). בהנתן המיקום, נקרא למתודת האיחוד המתאימה join_from_left/right ונשמור את הפרש הגבהים שהיא מחזירה. לאחר מכן נעדכן את המינימום/מקסימום של העץ הנוכחי בהתאם לכיוון האיחוד ונבצע איזון של העץ החל מצומת הציר מעלה כנדרש ע״י קריאה ל-rebalance_up. נחזיר את הפרש הגבהים בערך מוחלט ועוד 1 כנדרש

סיבוכיות זמן: $O(|\text{other. root. height} - \text{self. root. height}| + 1)$

ניתוח סיבוכיות במקרה הגרוע: נשים לב שבכל מקרה נקרא לאחת מתוך 3 מתודות העזר הנ״ל, לכן במקרה הגרוע נקבל סיבוכיות של $O(|\text{other. root. height} - \text{self. root. height}| + 1)$. לאחר מכן נקרא ל-rebalance_up אך בדומה לחסם העליון של get_subtree_root במתודות join_from_left/right, גם כאן הסיבוכיות חסומה ע״י הפרש הגבהים, לכן הסיבוכיות של rebalance_up במקרה זה היא $O(|\text{other. root. height} - \text{self. root. height}| + 1)$. נקבל כי בסה״כ סיבוכיות האיחוד תהיה $O(|\text{other. root. height} - \text{self. root. height}| + 1)$.

split(self, node: AVLNode) −> list[AVLTree]

מטרה: פיצול עץ AVL לשני עצים מצומת בעץ

שיטת פעולה: נבצע פיצול ע״י איחודים כפי שנלמד בשיעור. נאתחל 2 עצים, ימני ושמאלי, שיהיו העצים המפוצלים בסוף הריצה. נאתחל את העצים עם שורשים שהם בניו של הצומת המפצל. כעת, החל מההורה של צומת הפיצול, נתחיל לעלות בעץ עד שנגיע לשורש. בכל איטרציה נגדיר עץ וצומת זמניים חדשים ונבדוק את הכיוון היחסי של ההורה של הצומת להורה שלו (בדיקת היחס בין אבא של הצומת לסבא של הצומת) ע״י קריאה ל-get_relative_direction של AVLNode. כעת נגדיר את הצומת הזמני כצומת הנוכחית ע״י set_as_other_node של AVLNode, ונגדיר את השורש של העץ הזמני להיות הצומת הזמנית. נאתחל מינימום ומקסימום ע״י קריאה ל-init_min_max ולבסוף נבצע איחוד של העץ הזמני עם העץ המתאים לכיוון היחסי ע״י קריאה ל-join כאשר ההורה של הצומת הנוכחית משמש כצומת הציר. לאחר שנסיים לעלות בכל העץ, נתחזק את המינימום והמקסימום של העצים המפוצלים ע״י קריאות successor/predecessor על צומת הפיצול והחלפת מצביעים עם המינימום והמקסימום של העץ המתפצל. לבסוף נחזיר רשימה של שני העצים המפוצלים, הראשון יהיה השמאלי (מפתחות קטנים יותר ממפתח הפיצול) והשני יהיה הימני (מפתחות גדולים מצומת הפיצול)

סיבוכיות זמן: $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: נתמקד ראשית במתודות העזר. get_relative_direction עולה$O(1)$, set_as_other_node עולה גם היא $O(1)$ ובמקרה הגרוע קריאות ל-init_min_max ול-successor/predecessor עולות $O(\log n)$ כ״א. פרט לפעולות אלה, ראינו בכיתה שבמקרה הגרוע נבצע $n - 1$ קריאות ל-join, לכן הסיבוכיות הכוללת של הפעולות הללו תובע ע״י טור טלסקופי סופי שמסתכם ל-$\log n$, לכן הסיבוכיות הכוללת של המתודה כולה הוא $O(\log n)$.

$\text{rank(self, node: AVLNode)} -> \text{int}$

מטרה : קבלת הדרגה של צומת בעץ

שיטת פעולה : נאתחל משתנה לערך ה-size של הבן השמאלי של הצומת ועוד 1. כעת, החל מהצומת הנתונה, אם הצומת הוא בן ימני (נבדק ע״י קריאה ל-get_relative_direction) נוסיף למשתנה את ערך ה-size של אחיו השמאלי ועוד 1 ונמשיך לעלות בעץ עד שנגיע לשורש. נחזיר את המשתנה

סיבוכיות זמן : $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: מכיוון שאנו מתחזקים מאפיין size לצמתים, ומכיוון שמשתנה זה עומד ב-2 התנאים המספיקים שנלמדו בכיתה (חישובו תלוי רק בילדיו וזמן החישוב שלו קבוע) ראינו שניתן לקבל את ערכו במקרה הגרוע ב-$O(\log n)$, למעשה נבצע $\log n$ פעולות של $O(1)$.

$\text{select(self, i: int)} -> \text{AVLNode}$

מטרה : קבלת צומת בעלת דרגה נתונה

שיטת פעולה : ראשית נבדוק האם הדרגה שאנו מחפשים היא דרגת השורש, ונחזירו בהתאם. אחרת, המתודה בודקת האם הצומת בעל הדרגה הנדרשת נמצא מימין לשורש או משמאלו. לאחר מכן, החל מהמקסימום/מינימום בהתאמה למיקום הצומת, המתודה תבצע $i - 1$ או $n - i$ קריאות successor או predecessor בהתאמה עד שתגיע לצומת הנדרש ותחזיר אותו

סיבוכיות זמן : $O(\log n)$

ניתוח סיבוכיות במקרה הגרוע: במקרה הגרוע נבצע $\max\{i - 1, n - i\}$ קריאות successor/predecessor. כפי שלמדנו בתרגול, הסיבוכיות של כמות פעולות אלה היא $O(h + \max\{i - 1, n - i\}) = O(\log n)$.

$\text{avl\_to\_array(self)} -> \text{list[(int, Any)]}$

מטרה : קבלת רשימה ממוינת (עולה) של זוגות סדורים של ערכי הצמתים והמפתח שלהם, ממוינים לפי המפתחות

שיטת פעולה : נאתחל רשימה ריקה. במידה והעץ לא ריק (בדיקה ע״י קריאה ל-is_empty), החל מהצומת המינימלי, נבצע n פעולות successor ונבצע הוספה של הזוג הסדור (מפתח, ערך) לסוף הרשימה בכל איטרציה.

סיבוכיות זמן : $O(n)$

ניתוח סיבוכיות במקרה הגרוע: בדומה לניתוח במתודת select ובהתאם לנלמד בתרגול 3 (שאלה 4), ביצוע n פעולות successor היא בסיבוכיות של $O(h + n) = O(\log n + n) = O(n)$.

חלק ניסויי - תשובות

.1

א. ראשית, מתוך העובדה שאנחנו מכניסים את האיברים לעץ לפי סדרם במילון, נבחין כי עבור צומת חדשה- נסמנה v_i, מספר החילופים עבורה שקול למספר הצמתים הקיימים בעץ שערך המפתח שלהם גדול משל v_i. (כי הצמתים הקיימים בעץ היו במיקומים $i >$ במערך).

כלומר $num\ of\ swaps\ for\ v_i = size(tree) - rank(v_i)$.

מימשנו חישוב זה בדרך הבאה:

ראשית נבחין כי כל עוד $v_i.key \geq maximum$, אין החלפות.

אם $v_i.key < maximum$:

נעלה מהמקסימום במעלה העץ ונעצור בצומת v_j כך שמיקום ההכנסה של v_i הוא בתת עץ השמאלי של v_j.

מכאן, ממשיכים מv_j עם חיפוש בינארי רגיל כפי שנלמד בכיתה, בתוספת קטנה, חישוב ההחלפות תוך כדי (נעשה בדומה לחישוב $rank$). בכל איטרציה שבה נצטרך לרדת לבן השמאלי, נסכום את גודל תת העץ הימני של הצומת ממנה אנחנו יורדים$(y) + 1$

$swapsCount \leftarrow swapsCount + y.right.size + 1$

טבלת ניתוחי ניסויים:

i	מספר חילופים במערך ממוין-הפוך	עלות מיון AVL עבור מערך ממוין-הפוך	מספר חילופים במערך מסודר אקראית	עלות מיון AVL עבור מערך מסודר אקראי	מספר החילופים במערך כמעט ממוין	עלות מיון AVL עבור מערך כמעט ממוין
1	4498500	67805	2064996	60740	448201	52500
2	17997000	147635	8482333	130295	896701	112606
3	71994000	319297	32913292	301246	1793701	232823
4	287988000	686623	134184239	630265	3587701	473260
5	1151976000	1469277	523350064	1375904	7175701	954137

ב. מספר החילופים עבור מערך ממוין-הפוך בעל n איברים הוא:

$$\sum_{i=1}^{n} n - i = n^2 - \sum_{i=1}^{n} i = n^2 - \left(\frac{n(1+n)}{2}\right) = n^2 - \frac{n}{2} - \frac{n^2}{2} = \frac{n^2}{2} - \frac{n}{2} = \frac{n(n-1)}{2}.$$

ההסבר לכך הוא נובע מכך שלכל איבר שנמצא במיקום i במערך, כל האיברים שנמצאים במיקומים $i+1$ עד n עונים על הגדרת ההחלפה, משמע ישנן $n-i$ החלפות.

עלות החיפושים:

הסבר כללי

נבחין כי עבור מערך ממוין-הפוך, האיבר המקסימלי במערך הוא האיבר הראשון בעץ. בנוסף, כל איבר חדש עם מפתח i שנכנס לעץ הוא האיבר המינימלי החדש. נבחין כי ההכנסה נעשית לעץ עם $n+1-i$ איברים (כלומר גובה העץ $h = O(\log n + 1 - i)$). נשים לב- מכיוון שהמערך ממוין הפוך, ההכנסה מתחילה מאיבר שהמפתח שלו $i = n$. לכן מספר האיברים במערך הוא $n + 1 - n = 1$ ונגמרת עם האיבר שהמפתח שלו $i = 1$.) מכאן, שבכל הכנסה נצטרך לעלות מהמקסימום עד לשורש העץ, ומהשורש לרדת עד לעלה/לאבא של עלה כלומר סה״כ $\Omega(\log n + 1 - i)$ חיפושים.

* הוכחת חסם אסימפטוטי בעמוד הבא

נוכיח $T(n) = \Theta(n \cdot \log n)$

הערה: לפי ההסבר הכללי נבחין כי האיבר הראשון בסכימה מתייחס להכנסת האיבר האחרון ($key = 1$) במערך לעץ, בעוד שסכימת האיבר האחרון ($i = n$) מתייחס להכנסת האיבר הראשון במערך ($key = n$) לעץ.

<u>$T(n) = O(n \cdot \log n)$</u>:

$$\sum_{i=1}^{n} O(\log(n+1-i)) = O(\log(\prod_{i=1}^{n}(n+1-i))) = O(\log n!)$$
$$= O(n \log n)$$

<u>$T(n) = \Omega(n \cdot \log n)$</u>:

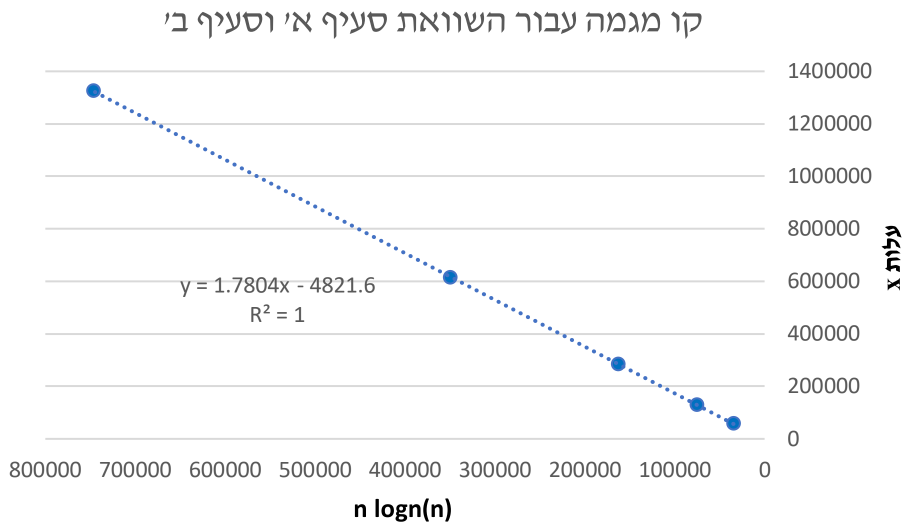
$$\sum_{i=1}^{n} \Omega(\log(n+1-i)) \geq \underbrace{\frac{n}{2}}_{\text{מחצית האיברים האחרונים}} \cdot \underbrace{\Omega(\log\frac{n}{2})}_{\text{עבודה באיבר האמצעי}} = \Omega(n \cdot \log n)$$

ג. <u>עבור מספר החילופים</u>:

בבירור ניתן לראות כי מספר החילופים שקיבלנו בסעיף א׳ עבור מערך ממוין הפוך זהה למספר החילופים שקיבלנו בסעיף ב׳.

<u>עבור מספר החיפושים</u>:

נבדוק קו מגמה באמצעות אקסל בצורה הבאה. לכל גודל מערך n, עבורו קיבלנו בסעיף א׳ עלות מיון מסויימת x, נייצר קואורדינטה חדשה המייצגת עבור אותו n את העלות מסעיף א׳ ואת העלות מסעיף ב׳ ($\equiv n \log n$) ונקבל קו מגמה בעל קואורדינטות $(n \log n\,, x)$.



טבלת עלויות חיפושים

i	סעיף א'	סעיף ב'
1	58836	$3000 \cdot \log 3000 \cong 34652$
2	129668	$6000 \cdot \log 6000 \cong 75304$
3	283332	$12000 \cdot \log 12000 \cong 162608$
4	614660	$24000 \cdot \log 24000 \cong 349217$
5	1325316	$3000 \cdot \log 3000 \cong 746435$

.1

א. טבלת ניתוחי ניסויים:

i	עלות join ממוצע עבור split אקראי	עלות join מקסימלי עבור split אקראי	עלות join ממוצע עבור split של איבר מקסימלי בתת העץ השמאלי	עלות join מקסימלי עבור split של איבר מקסימלי בתת העץ השמאלי
1	2.5	4	2.4545	13
2	2.4545	4	2.8333	15
3	2.5454	5	2.9166	16
4	2.75	10	2.5	17
5	3.0909	9	2.8235	19
6	2.6875	7	2.4375	20
7	2.2352	8	2.75	21
8	2.5263	7	2.9411	22
9	3.0555	5	2.5555	23
10	2.6111	9	2.65	25

ב. נסמן את עומק צומת x הפיצול בd. ראשית, נבחין כי נצטרך לבצע d פעולות $join$- לכל צומת במסלול מx לשורש.

בפרט, מכיוון שכל הפעולות ב$split$ הן מסיבוכיות אסימפטוטית $O(1)$ מלבד הרצות $join$, לצד העובדה שסיבוכיות פיצול אסימפטוטית היא $O(d)$, נקבע כי עלות ממוצעת של $join$ היא $\frac{O(d)}{d} = O(1)$.

ג. נסמן את צומת הפיצול, האיבר המקסימלי בתת העץ השמאלי של השורש ב x, ואת הבן השמאלי של שורש העץ בy.

נבחין כי תת העץ הימני של x הוא בוודאי ריק. מהגדרת איבר מקסימלי, נבחין כי במהלך פעולת הפיצול, בכל פעם שנעלה בעץ עד לצומת y נצטרך לבצע $\Theta(\log n)$ (כעומק x פחות 1) פעולות $join$ לתת העץ השמאלי של x. נבחין כי הפרשי הגבהים בין שני העצים אותם נאחד יהיה לכל היותר 1. מכאן, כל פעולה כזו, תיקח $O(1)$, סה״כ $\Theta(\log n)$

מכאן, הפעם הראשונה שבה נעלה שמאלה בעץ תהיה מy לשורש העץ המקורי וזו פעולת ה$join$ האחרונה שתתבצע. הפעולה תופעל על תת העץ הימני של x, שכבר ציינו שהוא ריק ועל תת העץ הימני של השורש המקורי. כלומר עלות $join$ זה נקבעת לפי גובה תת העץ הימני של השורש, ועל כן היא $\Theta(\log n)$.

כלומר $\Theta(\log n)$ היא עלות ה$join$ המקסימלי עבור פיצול באיבר המקסימלי של תת העץ השמאלי של השורש.

נראה שעלות זו עולה בקנה אחד עם התוצאות שקיבלנו בסעיף א׳.

לכל $1 \le i < 10$ נסמן $n_i = 1500 \cdot 2^i$, ואת גובה העץ המתאים לו $h_{n_i} \equiv \log n_i$ אזי מתקיים:

$$h_{n_{i+1}} = \log n_{i+1} = \log\left(1500 \cdot 2^{i+1}\right) = \log 1500 + \log 2^{i+1} = \underbrace{\log 1500}_{\cong 10.5} + (i+1) = 11.5 + i$$

$$h_{n_i} = \log n_i = \log\left(1500 \cdot 2^{i}\right) = \log 1500 + \log 2^{i} = \underbrace{\log 1500}_{\cong 10.5} + i = 10.5 + i$$

כלומר, ההפרש שנצפה בעלות $join$ המקסימלי בניסוי השני בין עץ בגודל n_{i+1} לבית עץ בגודל n_i הוא בערך 1, וזה תואם את התוצאות בטבלה בסעיף א.