Assignment 3

קורס: מבני נתונים

שם : אמיר אבו אלהיגא ת"ז: 213034655

שם : מוחמד סואלחה ת"ז : 207525510

שאלה 1 – עצי AVL:

סעיף א: $x\ leaf\ and\ d(x)\ is\ the\ debth\ of\ x\ leaf\ , h\ is\ the\ hight\ of\ the\ tree$

נוכיח את האי שוויון באינדוקציה על גובה העץ h נתחיל במקרה בסיס שזה גובה העץ h=0 n=1 אז הטענה מתקיימת.

עכשיו צעד האינדוקציה נגיד שיש לנו עץ עם שורש בשם A אז אנחנו מניחים שהבנים של השורש הזה הבן הימני והבן השמאלי מקיימים את האי שוויון אז קודם נתחיל בזה שעץ

AVL יש לו תכונה: $-1 \le h(A.left) - h(A.right) \le 1$ זה אומר שההפרש בין גובה שני העצים המשורשרים לבנים של A הו לכל היותר 1

אז נתבונן $h(A.left) > h(A.right)$ אז אמרנו קודם ש h הוא גובה העץ עם שורש A

אז במקרה הזה $h(A.left) \le h-2, h(A.right) \ge h-1$

עכשיו יש לנו שני מקרים או ש העלה שייך לתת העץ משורשר בבן הימני או בבן השמאלי

עכשיו נוכיח $d(x) \ge \left\lceil \frac{h}{2} \right\rceil$ בעזרת הצעד

לפי ההנחה אנחנו הגדרנו $h(A.left) > h(A.right)$

מקרה 1: העלה שייך לתת העץ משורשר בבן הימני

אז יוצא $d(x) - 1 \le \left\lceil \frac{h-1}{2} \right\rceil$, מכוון שעכשיו אנחנו נדברים על תת העץ שמשורשר בבן הימני של A אז האי שוויון נובע מההנחה

מכאן יוצא: $d(x) \le \left\lceil \frac{h}{2} \right\rceil - \left\lceil \frac{1}{2} \right\rceil + 1 \longrightarrow d(x) \le \left\lceil \frac{h}{2} \right\rceil - 1 + 1 \longrightarrow d(x) \le \left\lceil \frac{h}{2} \right\rceil$ **כנדרש**

מקרה 2: העלה שייך לתת העץ משורשר בבן השמאלי

אז יוצא $d(x) - 1 \le \left\lceil \frac{h-2}{2} \right\rceil$, מכוון שעכשיו אנחנו נדברים על תת העץ שמשורשר בבן השמאלי של A אז האי שוויון נובע מההנחה

מכאן יוצא: $d(x) \le \left\lceil \frac{h}{2} \right\rceil - \left\lceil \frac{2}{2} \right\rceil + 1 \longrightarrow d(x) \le \left\lceil \frac{h}{2} \right\rceil - 1 + 1 \longrightarrow d(x) \le \left\lceil \frac{h}{2} \right\rceil$ **כנדרש**

החסם הדוק במקרה שבו העץ הוא עץ AVL מינימאלי , נסמן את הקודקוד עם הגובה המינימאלי בmin(h). מכוון שהעץ הוא מינימאלי אז כל פעם שאנחנו יורדים לבנים של הקודקוד יהיה הפרש

בין הגבהים הוא 1 או -1 בדיוק(תלוי אם העץ משקלו מצד ימין או שמאל), נתבונן בעץ שהבן הימיני הוא שיש לו הגובה המינימלי ואז נתחיל לרדת כל פעם h-2 מהגובה של האב אז $min(h) = 1 + min(h-2) + \ldots\ldots = \left\lceil \frac{h}{2} \right\rceil$ כי כל פעם יורדים 2 מגובה העץ שלפניו עד שנגיע לאמצע גובה העץ +1 כדי שזה יהיה מאוזן.

סעיף ב:

נראה את זה במשפחת העצים המינימאליים בעלי גובה h מסוים (נתבונן במקרה ש גובה תת עץ הימני קטן מהשמאלי)נתחיל בזה שבעץ AVL שמשורשר ההפרש בין גובה של כל תת עץ לא יהיה יותר מ 1 או פחות מ -1. נסמן ב c_i בקודקוד מסוים את האיבר הימיני ביותר שאנחנו רוצים למחוק מהעץ ונסמן את כל האיברים שעוברים דרכן מהקודקוד הזה עד השורש לכל $c_k, 0 < k < i$ כאשר c_0, ו השורש של העץ הוא c_{k-1}, האב של האיבר שאנחנו רוצים למחוק נתחיל לסדר את כל קודקוד מעל האיבר הזה c_i אז במקרה של עץ מינימאלי אם מוחקים את c_0 כלומר נעלה כל פעם לאב הקודקוד שהגענו אליו ועשינו סיבוב בו עד השורש שזה מסעיף הקודם ראינו שהעומק של קודקוד $d(c_i) \geq \left\lceil \frac{h}{2} \right\rceil$ במקרה שלנו העץ מינימאלי אז $i = \left\lceil \frac{h}{2} \right\rceil$ גם ו כאשר n הוא מספר הקודקודים $i = \left\lceil \frac{\log(n)}{2} \right\rceil$ אז יוצא $h \geq \lfloor \log(n) \rfloor$ AVL כמובן בעץ מכאן $\left\lceil \frac{\log(n)}{2} \right\rceil$ פעמים שזה שווה ל i בעץ. אז הגענה למסקנה שזה שאנחנו עושים את הסיבובים יוצא

$$i = \left\lceil \frac{\log(n)}{2} \right\rceil \in \Omega(\log(n))$$

כי לכל $0 \leq c \leq \frac{1}{2}$ מתקיים

$$c \log(n) \leq \left\lceil \frac{\log(n)}{2} \right\rceil$$

סעיף ג: אפשר לתמוך בזמן דרך הוספת מצביעים ל $successor, predecessor$. הוספת השדות לכל קודקוד לא משפיעה על זמן הריצה . תוך כדי הכנסה כל איבר מסוים נחפש את ה $successor, predecessor$ שלו וכפי שלמדנו זה לוקח לכל היותר $O(logn)$ וגם ההכנסה

עצמה לוקחת $O(logn)$, מבחינת ההכנסה זה לא משפיע בכלל על זמן הריצה של ההכנסה בעץ AVL . וגם במחיקת כל איבר כלשהו אנו יודעים שזה לוקח $O(logn)$ ואם מוחקים איבר x כלשהו משנים את המצביעים של $successor(x), predecessor(x)$ כלומר
$successor(x).predecessor = predecessor(x)$ וגם
$predecessor(x).successor = successor(x)$ וכמובן זה לא משביע על זמן הריצה המחיקה בעץ AVL . עכשיו אם נרצה להדפיס את k האיברים הגדולים x $find(x,k)$ נחפש קודם את האיבר ואז נתחיל להדפיס מ ה successor(x) ואז נתחיל לעבור על successor של כל קודקוד מתחילת ה successor(x).successor ועד k-1 . (כי הדפסנו (successor(x) פעמים או עד שנגיע ל null ובזה יש סיבוכיות חיפוש של successor(x) שזה לוקח כפי שלמדנו $O(logn)$ וגם הדפסת ה k איברים שאנו נרצה זה לוקח $O(k)$ שזה בסוף $O(logn+k)$.

שאלה 2 – קבוצות:

מבני הנתונים יורכב מ:

1. AVL יחיד
2. בנוסף כל קודקוד יהיה שדה נוסף שמכיל עץ AVL אחר (חוץ מהשדות של הבן הימני והשמאלי ו האב)

פעולה	תיאור
Init()	נאתחל עץ AVL ריק וכפי שלמדנו אתחול למבני נתונים ריק לוקח O(1)
insert(x,y,hight)	קודם נחפש את המקום שאנו נרצה לחבר את הקודקוד החדש לפי ה x .נתחיל מראש העץ הראשי שלנו ואז נתקדם בחיפוש המקום לפי ה x ים אם הקודקוד שאנחנו עומדים עליו עכשיו $current.x < x$ נלך ימינה בעץ אם $current.x > x$ נלך שמאלה עכשיו הדבר השונה ממה שלמדנו זה שאם מכניסים key שיש לו x שווה למה שהכנסנו קודם המקרה הזה $x = current.x$ נכנס לעץ שמשורשר בקודקוד הזה ונבצע הכנסה רגילה בעץ הזה לפי הע y ים (מההנחה שלא מכניסים שני איברים עם אותם כאורדינטות אז בפעולת ההכנסה הרגילה לא תהיה לנו בעיה אם מכניסים לפי השוואת הע y ים) נמשיך בתהליך חיפוש על מקום פנוי המתאים בעץ עד שנגיע למקרה שפו מצאנו מקום ואז נבנה key עם אותם ערכים שקבלנו בקלט ואז נחבר אותו למקום שמצאנו .וגם אם נתקלנו במקרה שפו ההכנסה תפגע

	באיזון של העצים אפשר לתקן אותם על ידי סיבובים (העץ הראשי נתייחס אליו כאילו המפתחות שלו הם לפי ה x ים והעץ המשורשר בכל קודקוד נתייחס אליו כאילו המפתחות הם ה y ים) כמובן גם נתייחס שבהם העץ ריק אז נכניס את הקודקוד שיהיה השורש של העץ. ניתוח זמן ריצה :נסמן ב a את מספר האיברים בעץ הראשי וb מספרים האיברים המקסימלי שיש באחד מהעצים שמשורשרים בקודקודים אז פעולת ההכנסה לוקחת כפי שלמדנו $\mathrm{O}(\log(n))$ כאשר n מספר האיברים בעץ , אז במצב שלנו יש שני עצים עץ ראשון שלוקח $\log(a)$, וגם עץ שמשורשר בקודקוד שלוקח $\log(b)$ יחד זה יעלה $\log(a) + \log(b)$ וגם כמות האיברים במבני כולו גדול מ a, b אז בסופו של דבר ההכנסה לא תדרוש יותר מ $\log(n)$ כאשר n היא כמות האיברים במבני כולו.
delete(x,y)	במחיקת איבר נשתמש בפונקציות $\mathrm{AVL}.successor, rotations$. קודם נחפש את האיבר שאנחנו רוצים למחוק לפי מפתח x שלו בעץ הראשי ומתחילים עם השורש ונתחיל כל פעם להשוואת את מפתח x של כל קודקוד אנו מגיעים אליו אם $x < current.x$ נלך ימינה בעץ, ואם $x > current.x$ נלך שמולה בעץ , ואם הוא שווה נכנס לעץ שמשורשר בקודקוד הזה ונעשה חיפוש דומה לחיפוש הראשון אבל לפי המפתח y וגם לא תהיה לנו בעיה עם מפתח y שווה כי אפשר להניח שאין שני קודקודים שווים ב key שלהם ואז נמשיך בחיפוש עד שנגיע לקודקוד שאנחנו רוצים ואז נבצע עליו אותו מקרים של delete שלמדנו בכיתה(case1/2/3). במקרה של case1 נמחק את הקודקוד מהעץ ונבדוק עם האב שלו אם הוא היה הבן הימני שלו או השמאלי ואז נעדכן שיהיה null בהתאם למקום שלו . במקרה 2 נמחק את הקודקוד ונשים את הבן היחיד שיש לו במקומו ואז נעדכן את המצביעים של האב ושל הבן של הקודקוד שרוצים למחוק. במקרה case3 נמחק את הקודקוד ונשים במקומו את ה successor שלו כדי שנשמור על תכונת העץ של חיפוש בינארי וגם נעדכן את המצביעים של כל השכנים שהיו לקודקוד שמחקנו וגם את המצביעים של ה successor. נבצע את אותם המקרים בעץ שמשורשר בקודקוד אם האיבר שרוצים למחוק נמצא באחד העצים שמשורשרים עם הקודקודים וגם נעדכן בהתאם אבל במקרה הזה נתייחס למפתחות כאילו הם ה y ים ונבצע את אותם מצבי מחיקה כפי שהסברנו בעץ הראשי. יש מקרה גם שבו האיבר שרוצים למחוק נמצא בעץ הראשי ויש עץ שיש בו איברים שמשורשר בו(חוץ מהשכנים שיש לקודקוד)במקרה כזה נמחק את האיבר נשים במקומו את שורש העץ שמשורשר ונעדכן את המצביעים של כל הקודקודים ולגבי השורש החדש בעץ זה יהיה ה successor של השורש הקודם. בסוף נעלה למעלה בעץ (מנקודת המחיקה) ונבדוק אם העץ מאוזן אם, לא היה

	מאוזן נבצע סיבובים בעץ נמשיך לבצע סיבובים עד שכל העץ יהיה מאוזן. ניתוח זמן ריצה: קודם אנו עושים חיפוש בתוך העץ על האיבר שאנחנו רוצים למחוק נסמן ב a את מספר האיברים בעץ הראשי וb מספרים האיברים המקסימלי שיש באחד מהעצים שמשורשרים בקודקודים אז פעולת חיפוש לוקחת כפי שלמדנו $O(\log(n))$ כאשר n היא כמות האיברים בכל המבנה, אז אם יש לנו חיפוש שני עצים זה לוקח $\log(a) + \log(b)$ שזה עדיין לוקח בסופו של דבר $O(\log(n))$ מכיוון שa,b קטנים מn ובנוסף למחוק את האיבר במקרה הכי גרוע שזה ב case 3 לוקח גם $O(\log(n))$ בשני העצים זה אותו דבר כי או שמוחקים איבר בעץ הראשי או שנמחק איבר באחד העצים המשורשרים באחד הקודקודים כאילו זה לוקח $O(\log(b))$ או $O(\log(a))$ שזה כמובן עדיין $O(\log(n))$ וגם אם נרצה לתקן את האיזון של העצים כפי שלמדנו שזה גם לוקח $O(\log(n))$ וגם יש לנו שני מקרים פו אם מתקנים בעץ הראשי או שמשורשר באחד הקודקודים $O(\log(b))$ או $O(\log(a))$ שזה כמובן עדיין $O(\log(n))$ אז הסופו של דבר כל פעולה שעושים באלגוריתם לא תדרוש יותר מ $O(\log(n))$ לכן הסיבוכיות הכוללת היא $O(\log(n))$
find(x,y)	בפעולת החיפוש נבצע חיפוש רגיל כפי שלמדנו בכיתה. החיפוש הראשון זה חיפוש בעץ הראשי שלנו שזה יהיה לפי מפתח x של כל קודקוד, נתחיל בשורש ואחר כך נתקדם לפי השוואת המפתחות לפי ה x ים אם הקודקוד שאנחנו עומדים עליו עכשיו $current.x < x$ נלך ימינה בעץ, אם $current.x > x$ נלך ימינה, עכשיו הדבר השונה ממה שלמדנו זה שאם מחפשים key שיש לו x שווה לקודקוד שאנחנו עומדים עליו ברגע מסוים אך לא שווה מבחינת ה y ים במקרה הזה $current.x = x$ נכנס לעץ שמשורשר בקודקוד הזה ונבצע חיפוש רגיל לפי ה y ים של הקודקודים שנמצאים בעץ הזה ובסוף נחזיר את הגובה של הקודקוד שמצאנו. ניתוח זמן ריצה: כפי שלמדנו בעולת חיפוש בעצי AVL לוקחת $O(\log(n))$ כאשר n היא כמות האיברים בכל המבנה. במקרה הכי גרוע במבני שלנו אם נרצה לחפש איבר שנמצא בעץ שמשורשר ב אחד הקודקודים שנמצאים בעץ הראשי זה ידרוש חיפוש בשני עצים אז נסמן ב a את מספר האיברים בעץ הראשי וb מספרים האיברים המקסימלי שיש באחד מהעצים שמשורשרים בקודקודים אז פעולת החיפוש תדרוש $\log(a) + \log(b)$ מכיוון שa,b קטנים ממספר האיברים הכולל במבני שלנו לכן בסופו של דבר פעולת החיפוש לא תדרוש יותר מ $O(\log(n))$
printAll(x)	בפעולה הזאת קודם נעשה חיפוש רגיל בעץ הראשי שלנו , נתחיל בשורש ואחר כך נתקדם לפי השוואת המפתחות לפי ה x ים אם

הקודקוד שאנחנו עומדים עליו עכשיו $current.x < x$ נלך ימינה בעץ, אם $current.x > x$ נלך ימינה. אם $current.x = x$ מהנקודה הזאת נתחיל להדפיס את הגבהים נתחיל עם הקודקוד שעומדים עליו ואחר כך נכנס לעץ שמשורשר בו ונבצע הדפסה לכל העץ כי כולם האיקסים שלהם שווים. ניתוח זמן ריצה: כפי שלמדנו חיפוש זה דורש לכל היותר $O(\log(n))$ כאשר n היא כמות האיברים בכל המבנה. וגם פעולת ההדפסה של עץ מסוים לוקחת $O(k)$ כאשר k הוא מספר האיברים בעץ כלשהו אז והאלגוריתם שלנו ידרוש לכל היותר $O(\log(n)) +$ $O(k)$ שזה בדיוק $O(\log(n) + k)$	

שאלה3- מיזוג/פיצול:

סעיף א: (שני עצי AVL , T_1 בגובה h_1 ו- , T_2 בגובה h_2 וגם כל מפתח ב T_2 גדול מכל מפתח ב T_1 ואיבר x המקיים $maxT_1 < x < minT_2$)

קודם נשים לב שיש שני מקרים שאנחנו נתקל בהם ←

מקרה 1: $h_1 > h_2$

במקרה כזה אנו נלך כל צעד ימינה בעץ T_1 עד שנמצא קודקוד שגובה העץ שמשורש בו יהיה שווה לגובה העץ T_2

או גדול ממנו ב אחד נסמן קודקוד כזה בשם n_y , ואז נשרשר את תת העץ של הקודקוד n_y שתהיה הבן השמאלי של x ולשרשר את T_2 שתהיה את הבן הימני של x כלומר($x.left =$ n_y , $x.right = T_2$)

למה זה יעבוד?

קודם נסביר למה יהיה קודקוד כזה , יש לנו עצי AVL ויש להם תכונה שאומרת שלכל קודקוד מסוים נסמן ב $r_1, r_2, \ldots, r_i$ את כל הקודקודים שעוברים בהם בצד הכי ימיני בעץ אז הפרש הגבהים של הבנים של כל קודקוד לא יהיה יותר מ 1 או קטן מ -1

$$(-1 \le h(r_i.left) - h(r_i.rught) \le 1)$$

אז אם נתחיל לרדת מהשורש נקבל תת עץ שהגובה שלה $- \; h(r_i) = h(r_{i-1})$

$h(r_{i-1}) - 2$ $1 \; or$ אז אם מתחילים לרדת בעץ בטח נמצא קודקוד שיהיה הגובה שלו גדול ב 1 אם היה ההפרש הוא 2 מהגובה של תת העץ היינו בו קודם,

שווה אם היה ההפרש 1 עכשיו נראה למה זה יעבוד אם מחברים עם x . קודם הקודקוד x גדול מכל מפתח ב T_1 אז אם משרשרים את x עם האב של הקודקוד שמצאנו לא תהיה לנו בעיה בהרכב העץ מבחינה עץ חיפוש בינארי

וגם אם מחברים את הבן השמאלי שלו ל תת העץ שכוללת את קודקוד שמצאנו זה יעמוד לנו בדרישות של

$(key.r_i < key.x)$ וגם אם מחברים את העץ T_2 להבן הימני של x גם ישמור לנו על הרכב של עץ חיפוש בינארי מכוון$(x < minT_2)$ עכשיו נראה איך נסדר את העץ שקבלנו שיהיה עץ AVL אם מוסיפים x לעץ

אולי יהיה לנו מקרה שבו העץ של האב של x לא יהיה מאוזן אז במקרה הכי גרוע אנו נצטרך לעבור על כל קודקוד בעליה למעלה עד שנגיע לשורש ובכל קודקוד נבצע סיבובים לעץ שאנחנו מגיעים אליו אז זה אומר שנבצע סיבובים בכמות העומק שנמצא בו הקודקוד כלומר $- \; c = h_1$ $h(r_i)$ כאשר c הוא עומק הקודקוד שמצאנו וגם

$h_2 \approx h(r_i)$ $(0 \leq h(r_i) - h_2 \leq 1)$ לכן נצטרך $O(c)$ סיבובים שזה בעצם $O(|h_1 - h_2|)$

<u>**עכשיו נטפל במקרה השני:**</u>$(h_1 \leq h_2)$

אפשר לעשות את אותם הצעדים אבל הפעם נמצא מקום בשרשרת הכי שמאלית שבתוך T_2 ונחבר את x ו T_1

וקודקוד שמספק לנו את התנאים שאנו רוצים וגם במצב הזה יהיה לנו אתה סיבוכיות $O(|h_1 - h_2|)$

סעיף ב:

באלגוריתם שלנו נצטרך שתי רשימות לשמור בהן תתי שאנחנו עוברים בהם בעץ הראשי כדי למזג אותם אחר כך דרך השיטה שפרטנו בסעיף הקודם.

איך האלגוריתם יעבוד?

נסמן את הרשימות ב $list_1,\ list_2$ ($list_1$ לכל העצים שמפתח השורש בהם קטן או שווה ל k, $list_2$ לכל העצים שמפתח השורש בהם גדול מ k).נעשה חיפוש בתוך העץ כאילו נרצה לחפש את k ואז נעבור בעץ על כל הקודקודים שקטנים ממנו או גדולים , אם נעבור בקודקוד שהמפתח שלו קטן או שווה מ k נשים את הקודקוד וכל תת העץ שמשורשר בו לתוך הרשימה שיצרנו $list_1$, ואם המפתח היה גדול מ k נשים את הקודקוד ואת העץ שמשורשר בו לתוך הרשימה $list_2$, נעשה אותו דבר על כל קודקוד שאנחנו עוברים בו עד שנגיע לעלה בעץ . אחר כך ניצר שני עצים $T_{\le k}, T_{>k}$ ואז נעבור על הרשימות שיש לנו(כל רשימה תבנה לנו עץ שאנחנו רוצים) נתחיל עם אחת מהרשימות אז כל שני עצים וגם ראש של העץ השני שבא ברשימת העצים נשלח אותו עם העץ הראשון ואז נעשה להם חיבור דרך הפקודה שהסברנו בסעיף הקודם נעבור ברשימות בצורה שיהיה לנו העץ הראשון הוא העץ האחרון שעברנו בו בחיפוש ואז נעלה עד הסוף (עד סיום הרשימה) לדוגמה:

$\mathrm{T}_{\le k} = merge(list[0].left, lest[0], list[1])$

i=2

$while(i < list.length):$

$\mathrm{T}_{\le k} = merge(list[i].left, lest[i], \mathrm{T}_{\le k})$

בדוגמה עשינו את אחת העצים אז נעשה אותו דבר לעץ השני $T_{>k}$

עכשיו נראה איך עומדים בסיבוכיות הנדרשת $\mathrm{O}(\log(n))$

נתחיל בפעולה הראשונה האלגוריתם שהיא לפזר את כל תתי העץ האפשריים הסברנו שזה דומה לפקודת חיפוש אז כפי שלמדנו בכיתה חיפוש ידרוש $\mathrm{O}(\log(n))$ אחר כך נבצע מיזוג של כל תתי העץ שנמצאים ברשימות שיצרנו

נשים לב שכל פעולת מיזוג תדרוש $O(|h_i - h_j|)$ כאשר h_i, hj הם הגבהים של שני העצים שנרצה למזג

בשת הרשימות יהיה לנו בערך($\log(n)$ במקרה הכי גרוע) עצים שנרצה למזג כי עברנו מסלול חיפוש כפי שפרטתי לעל אם ניקח את המקרה הכי גרוע שיש אז נבצע את המיזוג ב $(h_i - h_j) *$ $\log(n)$ לכל i, j של העצים שנרצה למזג . נבהיר גם שהעצים שנרצה למזג יש להם גבהים שונים אבל עדיין נגיע למצב שבו אפשר לאלץ את הביטוי $(h_i - h_j)$ שיהיה מספר שלם ואז בכך נגיע שפעולת המיזוג לא תדרוש יותר מ $O(\log(n))$ אז הראינו שכל שלב באלגוריתם לא ידרוש יותר מ $O(\log(n))$ לכן הסיבוכיות של הקוד היא $O(\log(n))$.

<u>סעיף ג:</u>

למבני הנתונים שלנו נצטרך שני עצים של AVL:

T_{black} ששומר את הקודקודים שעם צבע **<u>שחור</u>**

T_{white}ששומר את הקודקודים שעם צבע **<u>לבן</u>**

פעולה	תיאור
$Add(x)$	קודם נבדוק אם הצבע של x אם הוא שחור נכניס את x ל T_{black} ואם היה לבן נכניס אותו ל T_{white} את ההכנסה בתוך כל עץ נבצע ופעולת ההכנסה הרגילה של AVL וכפי שלמדנו היא תבצע את ההכנסה ב $O(\log(n))$ והיא גם תסדר את העץ אם מבחינת האיזון.
$Color(k)$	בפעולה הזאת נבצע חיפוש רגילה של עצי AVL בשני עצים ונחזיר את הצבע של הקודקוד בעל מפתח k כפי שלמדנו חיפוש בעצי AVLש לוקח $O(\log(n))$ ובגלל שבצענו את החיפוש בשני העצים את זה ייקח$2 * \log(n)$ אבל זה עדיין עומד בדרישות שזה $O(\log(n))$ (כלומר לאותה משפחת פונקציות)

$FlipColors(k)$	במקרה הזה נשתמש בשני הפונקציות של סעיפים א+ב בהתחלה נחלק את שני העצים לארבעה דרך הפונקציה שעשינו בסעיף ב כלומר יהיה לנו $T_{black,\leq k}, T_{black,>k}, T_{white,\leq k}, T_{white,>k}$ ואז נחבר אותם בעזרת הפונקציה שהגדרנו שבסעיף א **מה לחבר?** נחבר את $T_{black,>k}$ וגם $T_{white,\leq k}$ איך?(נשלח אותם יחד עם קודקוד שאין לו חשיבות ואז נמחק אותו)ויחד ונשים אותם לתוך העץ השחור וגם נחבר את $T_{white,>k}$ וגם $T_{black,\leq k}$ איך?(נשלח אותם יחד עם קודקוד שאין לו חשיבות ואז נמחק אותו)ו לתוך העץ הלבן ואז בכך הפכנו את הצבעים של כל הקודקודים בעלי מפתח קטן מ k . **למה זה עומד בדרישות הסיבוכיות הנדרשת? כפי שראיונו פעולת חילוק לעצים לוקחת** $O(\log(n))$ וגם בעולת המיזוג של העצים ראינו שהיא לוקחת $O(\lvert h_i - h_j \rvert)$ אז הסיבוכיות הכוללת תהיה לכל היותר $O(\log(n))$

שאלה 4 – B-trees:

קודם כל נוסיף שדה יחיד לכל קודקוד משתנה בשם size ששומר את גודל תת העץ המושרש כלומר :

node.size = 1 + node.c[0].size + … + node.c[node.n].size

כאשר c הוא מערך של מצביעים לבנים של הקודקוד ו n הוא מספר המפתחות בקודקוד.

כדי למצוא את האיבר בעל הדרגה k נממש את הפונקציה הבאה שמטפלת בזה :

```
Select(node,k)
   i ← 0                                      Θ (1)
   while ( i <= node.n )                      Θ (2t - 1)
       if (node.c[i] + 1) = k                 Θ (1)
            return (node,i)                   Θ (1)
       else if (node.c[i] + 1) < k            Θ (1)
            k ← k - node.c[i] – 1             Θ (1)
       else                                   Θ (1)
            return Select(c[i], k)
       i ← i + 1                              Θ (1)
   return Select(c[i], k)
```

כשרוצים להשתמש בפונקציה הזאת נשלח לה את ה root ואת ה k, ואז מתחילים לבדוק אם מספר האיברים בתת העץ של הבן השמאלי ביותר עם המפתח הראשון שווה ל k , אם כן אז האיבר שמחפשים אותו הוא המפתח הראשון, אחרת אם המספר היה קטן מ k אז נחסיר אותו מ k ונעבור לבדוק אם מספר האיברים בתת העץ של הבן השני עם המפתח השני ונחזור על אותם שלבים, אחרת אם היה מספר האיברים בתת העץ של הבן ה i עם המפתח ה i גדול מ k אז נכנס להבן ה i ונחזור עוד פעם על אותם שלבים.

ניתוח זמן ריצה :

נתבונן ב while (i <= node.n) משום שיש מקסימום 2t-1 מפתחות בכל קודקוד אז הלולאה תתבצע במצב הגרוע ביותר Θ (2t - 1) כלומר $\Theta(t)$ ומשום שהגובה המקסימלי של העץ הוא $log_t \frac{n+1}{2}$

אז נקבל בסך הכל שהזמן הריצה הוא $\Theta(t \cdot \log{(n)})$