6.1)

לטובת מימוש המשימה השתמשנו במבנה נתונים של 2 רשימות מקושרות דו כיווניות ממויינות.

רשימה מקושרת אחת מכילה ערכי Container שמהווים את הקישורים שהמידע שבתוכם הוא הנקודות על גבי הצירים, ממויינים לפי ציר ה-X.

רשימה מקושרת נוספת מכילה ערכי Container שמהווים את הקישורים שהמידע שבתוכם הוא הנקודות על גבי הצירים, ממויינים לפי ציר ה-Y.

במחלקה DataStructure השדות הבאים:

```
//heads
private Container headX;
private Container headY;

//tails
private Container tailX;
private Container tailY;

//size of points at linkedlist
private int size;
```

השדה size הוא גודל הרשימות המקושרות (אשר גודלן זהה) ומהווה את כמות הנקודות שנמצאות במבנה הנתונים.

לציר הy 2 שדות: headY, tailY (הראש מכיל את הנקודה בעלת ערך הy הקטן ביותר, והזנב עם ערך הy הגדול ביותר).

לציר הx 2 שדות: headX,tailX (הראש מכיל את הנקודה בעלת ערך x הקטן ביותר, והזנב עם ערך ה x הגדול ביותר).

במחלקה Container השדות הבאים:

```
private Point data;//Don't delete or change this field;
private Container next;
private Container prev;
private Container connectToOtherList;
private Container newConnectToOtherList;
```

data זוהי הנקודה אשר נמצאת בContainer

next מצביע לקישור הבא ברשימה.

prev מצביע לקישור הקודם ברשימה.

connectToOtherList מצביע לקישור ברשימה השנייה אשר מכיל את ערכי אותה הנקודה (same data).

newConnectToOtherList מצביע עזר למימוש הפונקציה nearestPair והוא מצביע זמני לקישור ברשימה השנייה אשר מכיל את ערכי אותה הנקודה.

<u>הסבר במילים למימוש האלגוריתמים בהם השתמשנו למימוש שיטות הממשק DT</u>

addPoint

במימוש פונקציה זו, אשר מוסיפה נקודה חדשה למבנה הנתונים, הצבנו 3 תנאים.

במידה ואין נקודות כלל במבנה הנתונים, נוסיף את הנקודה לכל ציר כך שהראשים והזנבות של הרשימות המקושרות מצביעות על אותה הנקודה, כלומר, 2 מקומות שונים בזכרון, אשר מכילים את אותם ערכי נקודה, אחד לציר ה- Xואחד לציר ה-Y.

במידה וקיימת נקודה אחת במבנה הנתונים, נוסיף את הנקודה למקומה בצורה ממויינת, לראש או לזנב (תלוי בערך המתקבל כקלט ובנקודה שכבר נמצאת במערך, כך שhead.value>tail.value בשני הצירים (value מסמן כאן את ה- X או ה-Y).

במידה וקיים במבנה הנתונים יותר מנקודה אחת נוסיף את הנקודה למקומה בצורה ממויינת, לציר ה- Xולאחר מכן לציר ה- Y(גם כאן, 2 מקומות שונים בזכרון).

לאחר ההוספה נקדם את ערך size ונוסיף לו 1.

בפעולות בהן קיימת נקודה אחת או יותר, נעדכן את המצביעים של הרשימה המקושרת קדימה ואחורה כך שנוכל לרוץ על מבנה הנתונים בפונקציות הנוספות בצורה יעילה יותר, יתרה מכך, בכל הוספה נעדכן את המצביע בין הרשימות לנקודה אותה הוספנו בציר השני.

getPointsInRangeRegAxis

למימוש פונקציה זו, אשר מחזירה מערך של נקודות, נקבל 3 קלטים (ציר,מינימום, מקסימום), יצרנו משתנה i אשר מאותחל ל-0 ומהווה אינדקס וסופר את ערכי הנקודות במהלך הפונקציה.
בפונקציה זו יצרנו מערך שישמור את כל ערכי הנקודות (וגודלו size כך שיוכל לשמור בצורה מקסימלית את כל הנקודות) אשר ערכם לפי הציר המתקבל כקלט נמצא בין מינימום למקסימום כולל.

נרוץ מתחילת המבנה מהראש ועד הנקודה הראשונה שגדולה או שווה למינימום.

כעת, נתחיל להוסיף למערך את ערכי הנקודות עד שנגיע לסוף המבנה נתונים או עד אשר נגיע לנקודה שגודלה גדול ממקסימום ונוסיף 1 למשנה i בכל פעם שנפגוש בכזאת.

אם i שווה לאפס, כלומר, אין נקודות שמהוות את התנאים הדרושים, נחזיר מערך ריק בגודל 0.

אחרת, ניצור מערך חדש, נקרא לו ans, אשר גודלו הוא i(כלומר, כגודל הנקודות שעונות על התנאים).

כעת נעתיק את ערכי הנקודות מהמערך הראשוני שיצרנו אל המערך החדש בכדי לא ליצור ערכי null במערך, ונחזיר כפלט את ans.

getPointsInRangeOppAxis

בפונקציה זו, בדומה לפונקציה הקודמת, ניצור מערך שמכיל את הנקודות וגודלו מקסימלי והוא כגודל הנקודות שכרגע נמצאות במערך.

כמו כן, שוב בדומה לפונקציה הקודמת ניצור משתנה i שמהווה אינדקס ואת מספר הנקודות שהכנסנו למערך המכיל.

נרוץ מתחילת הפונקציה ועד סופה, בציר השונה ממה שהוכנס כקלט, וכאשר נפגוש בנקודות אשר עונות על ערכי הקלט, נכניס אותן למערך המכיל (לפי axis שהוכנס כקלט).

אם אין כלל נקודות במערך המכיל, נחזיר מערך ריק.

אחרת, ניצור מערך חדש וגודלו i בכדי שלא ניצור ערכי null מיותרים במערך, ונעתיק לתוכו את כל הנקודות שנמצאו.

נחזיר את המערך החדש כפלט.

getDensity

כדי לממש פונקציה זו, נבחין כי במידה וקיימת נקודה אחת במבנה הנתונים, הצפיפות היא 0, לכן נבדוק תחילה אם קיימת נקודה אחת במבנה הנתונים.

במידה ולא, נחזיר לפי נוסחת הצפיפות שניתנה בדף העבודה.

narrowRange

בפונקציה זו, נקבל כקלט את הערכים (ציר,מינימום, מקסימום) ונרצה למחוק ממבנה הנתונים את כל הנקודות שערך שלהם בציר גדול ממקסימום או קטן ממינימום.

תחילה, נרוץ מראש הרשימה המקושרת לפי הציר שהתקבל ונבדוק האם ערך הנקודה קטן ממינימום. אם הגודל הוא 1, נמחק נקודה זו ונסיים עם הפונקציה. אחרת, נמחק את הנקודה, לאחר מכן נמחק גם את הנקודה שברשימה המקושרת בציר השני, נשנה את ראש הרשימה בהתאם, ונוריד את שדה size באחד (כי מחקנו נקודה).

שנית, נעשה את הפעולה הדומה לערכים הגדולים מmax בכך שנרוץ מזנב הרשימה עד הנקודה שנמצאת בטווח המתאים, ונוריד אותה מהרשימה.

getLargestAxis

בפונקציה זו, נחזיר האם Ymin-Ymax > Xmin-Xmax, בכך שנחשב:

ערך זנב X– ערך ראש X> ערך זנב Y– ערך ראש Y.

נזכיר כי הרשימות המקושרות ממויינות ולכן מדובר בערכים מקסימליים ומינימליים.

במידה והנוסחא נכונה, נחזיר true ובמידה ולא, נחזיר false.

getMedian

בפונקציה זו, נחזיר את הערך שנמצא במקום האמצעי ברשימה המקושרת לפי קלט של ציר.

נרוץ מתחילת הרשימה המקושרת עד שנגיע לערך שנמצא במקום size/2 והוא יהיה אמצע הרשימה ולכן נחזיר אותו.

הסבר ל nearestPointsInStrip נמצא ב6.2 יחד עם חישוב זמן הריצה.

.6.2

addPoint

זמן הריצה של פונקציה זו הוא O(n) כאשר n הוא מספר הנקודות הקיימות במבנה הנתונים. זמן הריצה מקסימלי ב-n מכיוון שאנו עוברים על הרשימות המקושרות מההתחלה ועד הסוף בצורה מקסימלית פעמיים (אחת לציר ה- Xואחת לציר ה-Y) ולכן,

2n = O(n)

getPointsInRangeRegAxis

זמן הריצה של פונקציה זו הוא O(n) כשn זהו מספר הנקודות הקיימות במבנה הנתונים. זמן הריצה מקסימלי ב-n מכיוון שבמידה וערך המינימום גדול מהערך המקסימלי לפי הציר שהתקבל כקלט למשל, אנו עוברים על כל המערך פעם אחת. ולכן,

n=O(n)

getPointsInRangeOppAxis

זמן הריצה של פונקציה זו הוא O(n) כשn זהו מספר הנקודות הקיימות במבנה הנתונים. זמן הריצה מקסימלי ב-n מכיוון שבמידה וערך המינימום גדול מהערך המקסימלי לפי הציר שהתקבל כקלט למשל, אנו עוברים על כל המערך פעם אחת. ולכן,

n=O(n)

getDensity

זמן הריצה של פונקציה זו הוא O(1) מכיוון שאנו מחזירים לפי ערכים קבועים ולכן, בגלל שמתבצעות 2 פעולות בזמן קבוע של 1,

$$2 = O(1)$$

narrowRange

מכיוון שבפונקציה זו אנו רצים מתחילת המבנה נתונים ועד לנקודה שמממשת את ערכי הקלט, ולאחר מכן רצים מסוף המבנה עד לנקודה שמממשת את ערכי הקלט, מחקנו |A| נקודות ממבנה הנתונים, לכן,

O(|A|) כאשר |A| הינו מספר הנקודות שיש למחוק ממבנה הנתונים.

getLargestAxis

בפונקציה זו אנו מבצעים פעולה אחת בזמן קבוע של 1, לכן,

$$1 = O(1)$$

getMedian

בפונקציה זו אנו רצים מתחילת מבנה הנתונים ועד לאמצע מבנה הנתונים, כך שבעצם אנו מבצעים פעולה של $\frac{n}{2}$ כשn הינו מס' הנקודות במבנה הנתונים. לכן,

$$\frac{n}{2} = O(n)$$

nearestPairInStrip

בפונקציה זו אנו רצים על כל האיברים (B) בטווח הנתון לנו, ובודקים (**) האם BLogB קטן מn.

אם כן- נבצע הכנסה למערך של הנק' בטווח, ונמיין עפ"י הציר הנגדי בעזרת Arrays.sort סדר גודל של BlogB פעולות.

אחרת נשתמש בפונקציה המחזירה מערך של נק' ממויינות עפ"י הציר הנגדי סדר גודל של n פעולות.

נשווה כל נקודה במערך שהתקבל עם 7 הנקודות שאחריה (מס' השוואות מינימלי – כאשר לכל היותר 16 נקודות הן רלוונטיות בכל טווח במקסימום). כשיש לכל היותר B כאלו, סדר גודל של B פעולות.

מכיוון שאנו בוחרים בזמן המינימלי בהתאם לבדיקה (**), נקבל שזמן הריצה הוא O(min(BlogB,n)).

ישנם 4 פונקציות עזר:

- nearest4Points

מוצא את 2 הנקודות הקרובות במבנה נתונים בעל size<=4, זמן הריצה שלו הוא O(n) והוא עובר על מבנה הנתונים עד אשר הוא מוצא את 2 הנקודות הקרובות ביותר.

changeConnect
משנה את הקישוריות בין הרשימות המקושרות כדי לעזור לפונקציה
Narrowrange.
זמן הריצה הוא
בגלל שהוא מתבצע ע״י פעולות קבועות בלבד O(1)

changeHead
משנה את ראש הרשימה ומסייע לפונקציה

Narrowrange

זמן הריצה הוא O(1) כי הוא מתבצע ע״י פעולות קבועות בלבד.

changeTail
משנה את זנב הרשימה ומסייע לפונקציה

Narrowrange

זמן הריצה הוא O(1) כי הוא מתבצע ע״י פעולות קבועות בלבד.

6.3. **<u>SPLIT:</u>**

a. הערך המוחזר מהפונקציה SPLIT: מערך בעל 4 תאים המכיל (עפ"י הסדר הבא) מצביע ראש המצביע לאיבר הראשון באוסף שבו כל הערכים קטנים מvalue (תא 0), מצביע זנב המצביע לאיבר האחרון באותו האוסף (תא 1), מצביע ראש המצביע לאיבר הראשון באוסף שכל איבריו גדולים שווים לvalue (תא 2) ומצביע זנב המצביע לאיבר האחרון באוסף זה (תא 3).

***נשים לב- מההנחה בעבודה שאין שתי נק' בעלות אותו ערך X\Y לא יכול להיות יותר מאיבר אחד השווה בערכו ל value.**

המצביעים ראש וזנב:

מצביעים לאיברים מסוג container. לכל container ישנם השדות- data,next,prev.

Data- מכיל איבר מסוג point (בעצם הנקודות של האוסף)

Next – מכיל את הcontainer הבא ברשימה (ממויינת)

Prev- מכיל את הcontainer הקודם ברשימה (ממויינת)

מצביע הראש – מצביע לאיבר הראשון באוסף – ממנו יש ללכת רק קדימה כדי להגיע אל שאר האיברים.

מצביע הזנב- מצביע לאיבר האחרון באוסף- ממנו אין להמשיך להתקדם בעזרת מצביע הnext.

כדי לגשת לנקודה עצמה – יש לבצע get data()

- **לכל נקודה (איבר מסוג point) - יש שדה המכיל את הערך X ושדה המכיל את הערך Y על פיהם אנו ממיינים את האיברים ומבצעים את החלוקה לשני אוספים.**

כדי לגשת לcontainer הבא ברשימה יש לבצע getnext()

כדי לגשת לcontainer הקודם ברשימה יש לבצע getprev()

.b

```
Split (int value, Boolean axis){
        Boolean foundlow,foundhigh<- false
        Container head,tail,low,high
        Container[] pointers= new Container[4]
        If(axis){                           axis=true means X
                Low<- headX
                High<-tailX
                Head<-headX
                Tail<-tailX
                While (!foundlow AND !foundhigh){           runs for 2C containers
                        If (value<= high.getdata().getX())
                                High<-high.getprev()
                        Else
                                Foundhigh<-true
                        If(value>low.getdata().getX())
                                Low<-low.getnext()
                        Else
                                Foundlow<-true
                }
                If(foundlow=true AND foundhigh=false){          correcting pointers
                        High<-low       low points on the first container which =>value
                        Low<-low.getprev()
                }
                Else If(foundhlow=false AND foundhigh=true){
                        Low<-high       high points on the first container which<value
                        High<-high.getnext()
                }
                Else{           value is median => equal distance => C=n/2
                        Low<-low.getprev()
                        High<-high.getnext()
                }
        }
        Else{           axis=false means Y
                Low<- headY
                High<-tailY
                Head<-headY
                Tail<-tailY
                While (!foundlow AND !foundhigh){           runs for 2C containers
                        If (value<= high.getdata().getY())
                                High<-high.getprev()
                        Else
                                Foundhigh<-true
                        If(value>low.getdata().getY())
                                Low<-low.getnext()
                        Else
                                Foundlow<-true
```

```
            }
            If(foundlow=true AND foundhigh=false){          correcting pointers
                High<-low        low points on the first container which =>value
                Low<-low.getprev()
            }
            If(foundhlow=false AND foundhigh=true){
                Low<-high        high points on the first container which<value
                High<-high.getnext()
            }
            Else{            value is median => equal distance => C=n/2
                Low<-low.getprev()
                High<-high.getnext()
            }
        }
        Pointers[0]<-head
        Pointers[1]<-low                    low is the tail of the lowers list
        Pointers[2]<-high                   high is the head of the higher list
        Pointers[3]<-tail
        Return pointers
}
```

c. הפונקציה split מבצעת ריצה, בעזרת לולאה, במקביל מסוף הרשימה, ומתחילת הרשימה. הלולאה עוצרת כאשר מזהה את האיבר הראשון ברשימה שערכו גדול\קטן מvalue (בהתאם לכיוון הריצה- אם רצנו מהסוף האיבר שנעצור בו יהיה קטן מערכו של value, ואם רצנו מתחילת הרשימה האיבר שנעצור בו יהיה שווה לvalue, אם קיים ברשימה, או גדול אם לא קיים).
O(2C)=O(C).

כאשר נזהה איבר זה, נבצע תיקוני מצביעים בהתאם- מכיוון שרצנו עד כה רק על 2c איברים, והרשימה מכילה n איברים, ישנם n-2c איברים הצריכים להשתייך לאוסף כלשהו. בהתאם למיקום בו עצרנו נדע לשייך אותם אל האוסף המתאים (לדוגמה: אם הערך value היה קרוב יותר לסוף הרשימה (מצביע high יצביע על האיבר הראשון שערכו נמוך מvalue) – 2c האיברים שנותרו ישתייכו לתחילת האוסף ע"י השמת המצביע של זנב הרשימה התחתונה על האיבר שבו עצרנו high, ותיקון המצביע high שיצביע על האיבר הבא אחריו (בעל הערך value לכל הפחות)).
O(1).

הכנסת מצביעי ראש וזנב של שני האוספים (שתי הרשימות) לתאים במערך O(1).

החזרת המערך O(1).

סה"כ זמן הריצה של split הוא:

O(C)+O(1)+O(1)+O(1)=O(C) כנדרש.

6.4. <u>nearestPair:</u>

תנאי עצירת הפונקציה : גודל מבנה הנתונים קטן מ2 או קטן שווה מ-4.

תנאי עצירה גודל קטן מ2- החזרת מערך ריק O(1)
תנאי עצירה גודל קטן שווה מ4- קריאה לפונקציה nearest4Points- המבצעת בדיקה ידנית על המרחק המינימלי מבין 4 נקודות לכל היותר – O(1)
בכל כניסה לפונקציה- נבדוק מהו הציר בעל הערכים היותר גדולים – O(1)

נפצל את מבנה הנתונים לשני תתי מבנים, לפי ציר זה. (ראו הסבר על פיצול המבנה ב6.5) – O(n)

בדיקה האם המרחק בין הנק' קטן מmin- O(1)

קריאה לפונקציה nearestPairInStrip ע"מ למצוא את הנק' הכי קרובות בטווח min*2 מהחציון של הציר בעל הערכים הגדולים יותר – O(min(n,BlogB))- כלומר במקרה הגרוע O(n)

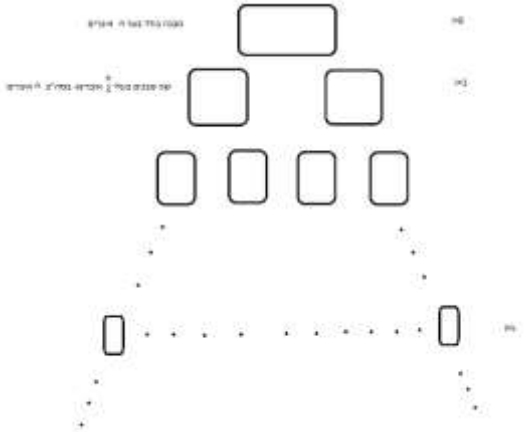
בדיקה של המרחק המינימלי על הנק' שחזרו מהפונקציה nearestPairInStrip לעומת min ועדכון ערכים – O(1)

סה"כ בקריאה בודדת לפוקציה נקבל: O(1)+ O(1)+ O(1)+O(n)+ O(1)+O(n)+
<u>O(1)+O(n)+O(1)=3O(n)=O(n)</u>

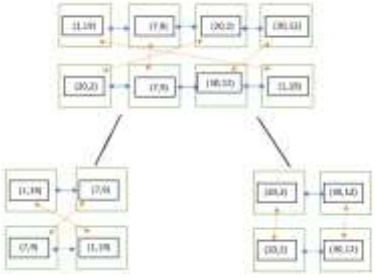
כעת, נניח כי בשכבת הפיצול הK, יש לנו K מבני נתונים מפוצלים, לכל קבוצה $\frac{n}{k}$ איברים – סה"כ מספר האיברים בכל K המבנים יחד יהיו n.

כיוון שמפצלים בכל הנתונים ל2 מבנים, הרקורסיה נקבל:

פעם את מבנה
אם נסתכל על עץ



לדוגמה:



נעצור כשנגיע למבנים בגודל קטן מ2, או קטן שווה ל4 לכן מס' השכבות בעץ הרקורסיה הן O(logn).

לכן נקבל כי זמן הריצה הכולל הוא O(nlogn)

6.5. פיצול של מבנה נתונים קיים לשני מבנים נפרדים המכילים את כל הנקודות עם ערך X קטן מהחציון (לצורך נוחות נקרא לה הרשימה התחתונה), וערך X גדול מהחציון (כולל) (הרשימה העליונה) ב(O(n:

תחילה נסביר את מבנה הנתונים שלנו:

שתי רשימות מקושרות ממויינות (המורכבות מContainers), אחת ממויינת עפ"י ערכי הX בPOINT והשניה עפ"י ערכי הY.

כאשר כל container מכיל את השדות:

Data מכיל POINT המורכב מערך X, ערך Y ושם טיסה

Prev מצביע לcontainer הקודם ברשימה

Next מצביע לcontainer הבא ברשימה

ConnectToOtherList מצביע לcontainer הזהה ברשימה השניה

newconnectToOtherList מצביע "זמני" שיכיל את הערך null, למעט במקרה בו יש צורך ב"פיצול" של מבנה הנתונים כמו במקרה המבוקש.

***הקוד של פיצול המבנה מופיע בפונקציה nearestPair**

נסביר על שלביו:

תחילה נשתמש בפונקציה getMedian על מנת למצוא את החציון של ציר הX – O(n)

לאחר מכן נגדיר משתנה זמני curr=headX שיצביע על ראש הרשימה הממויינת עפ"י ערכי X.

ובנוסף נגדיר עוד 4 משתנים newContainerLow1,newContainerHigh1, newContainerLow2 , newContainerHigh2 שיעזרו לנו בריצה על איברי הרשימה ויצירת Containerים חדשים. O(1)

כעת:

בשלב הראשון נרצה ליצור את ראש הרשימה התחתונה O(1):

ניצור container חדש newContainerhigh1, , שמכיל את אותם ערכי הdata כמו של החוליה הראשונה בציר הX (ערכי X קטנים מהחציון). נעדכן את מצביע הnewconnectToOtherList שלו להיות המצביע של curr.getConnectToOtherList() – כלומר החוליה החדשה שלנו תצביע באופן זמני על מיקום ראש הרשימה בציר הY, נעדכן גם את מצביע הnewconnectToOtherList של curr.getConnectToOtherList() להיות האיבר החדש שיצרנו. ונקדם את מצביע ה Curr שלנו שיצביע על האיבר הבא ברשימה. (בסיום השלב newContainerlow1, newContainerhigh1 יצביעו לאיבר האחרון שנוצר ברשימה התחתונה).

-לאחר שלב זה נעדכן את headX של המבנה התחתון להיות האיבר שיצרנו. O(1)

בשלב השני נרצה ליצור את המשך הרשימה התחתונה O(n) – נרוץ על כל האיברים הקטנים מהחציון ונבצע את אותן פעולות יצירה ועדכון מצביעים כמו בשלב הראשון + עדכון מצביעי prev וnext.

-לאחר שלב זה נעדכן את tailX של המבנה התחתון להיות האיבר שיצרנו.

כעת, כשסיימנו לבנות את ציר הX של הרשימה התחתונה curr יצביע על החציון.

בשלב השלישי ניצור את ראש הרשימה העליונה – בדומה לשלב הראשון+ עדכון headX של המבנה העליון: O(1)

בשלב הרביעי ניצור את המשך ציר הX של הרשימה העליונה, בדומה לשלב השני (בריצה עד שנגיע לאיבר האחרון ברשימה -tailX)+ עדכון tailX של המבנה העליון: O(n)

<u>עד כה בנינו את צירי הX של שתי הרשימות, כך שכל איבר בצירים אלו יצביע בעזרת ה newconnectToOtherList על מיקום האיבר הזהה לו בציר הY במבנה המקורי (ולהיפך).</u>

כל שנותר לנו הוא לבנות את ציר הY של כל אחת מהרשימות ולדאוג שבסיום בניית מבני הנתונים כל מצביעי הnewconnectToOtherList יהיו Null.

נגדיר מחדש את מצביעי העזר שלנו: curr=headY,

null= newContainerHigh2, newContainerLow2, newContainerHigh1,newContainerLow1

בשלב זה (השלב החמישי)- נרצה למצוא את ראש ציר הY של הרשימה התחתונה והעליונה O(n)

נבצע זאת ע"י ריצה על איברי ציר הY במבנה המקורי, (כל עוד curr!=null וגם כל עוד מצביע הראש של הרשימה התחתונה שונה מnull)- נבדוק בכל פעם האם ערך הX של curr קטן מהחציון – במידה וכן זה איבר הראש של רשימת הY שלנו (שכן הוא בעל ערך הY המינימלי, כך שערך הX שלו קטן מהחציון). כעת, בדומה לשלבים הקודמים, ניצור איבר חדש בעל אותו הDATA, ונעדכן את מצביע הקישור לרשימה הנגדית ConnectToOtherList להיות מצביע ה newconnectToOtherList של האיבר עליו אני עומדים ברשימה המקורית (Curr) ונעדכן את headY של הרשימה התחתונה להיות האיבר שיצרנו.

את אותן הפעולות נבצע ע"מ למצוא את מצביע הראש של ציר הY של הרשימה העליונה.

בשלב האחרון נבנה את המשך צירי הY של כל אחת משתי הרשימות-התחתונה והעליונה O(n)

***עד כה יש לנו רק את מצביעי הראש של צירי הY המצביעים בעזרת ConnectToOtherList על מיקומם בציר הX המתאים (בכל רשימה).**

***בשלב זה מצביעי הnewconnectToOtherList של האיברים בציר הY במבנה המקורי עדיין מצביעים על האיברים הזהים להם בצירי הX של כל אחת מהרשימות החדשות (ולהיפך).**

אז נרוץ פעם אחת בלבד על כל איברי ציר הY במבנה המקורי ונחלק אותם בהתאם לשני מבני הנתונים עפ"י בדיקת ערך הX שלהם (אם קטן <u>או</u> גדול שווה מהחציון) -נתחיל מראש הרשימה curr=headY.

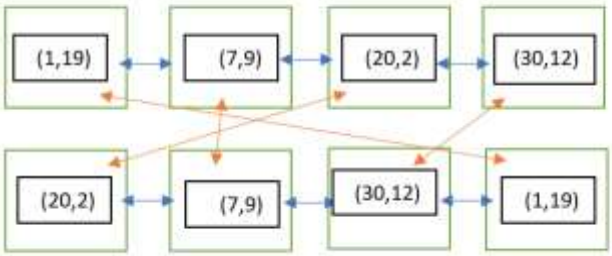
בכל פעם, ניצור איבר חדש (למשל newContainerHigh1) כמו בשלבים הקודמים ונעדכן את מצביעיו – כמו שעשינו בשלב החמישי. רק שהפעם נבצע בנוסף curr.setNewConnectToOtherList(null) ,
newContainerHigh1.getConnectToOtherList().setNewConnectToOtherList(null) – כלומר נחזיר את המצביעים להיות null כנדרש.

נמחיש גם בעזרת דוגמה:

המבנה המקורי-

חץ כחול – next,prev

ציר X:

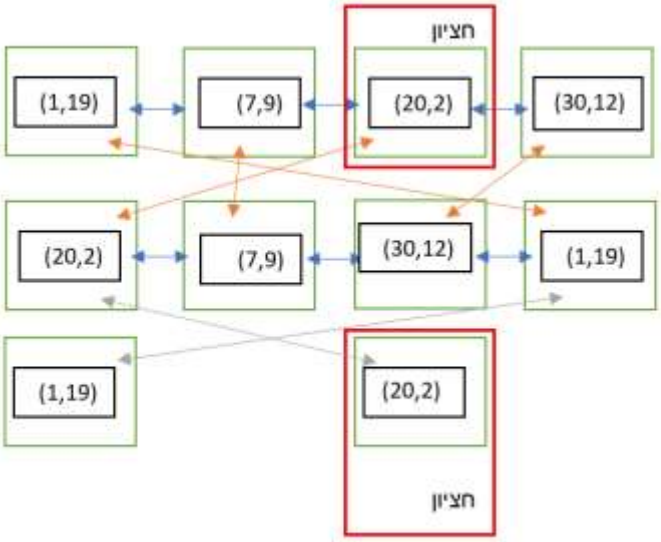


חץ כתום- ConnectToOtherList

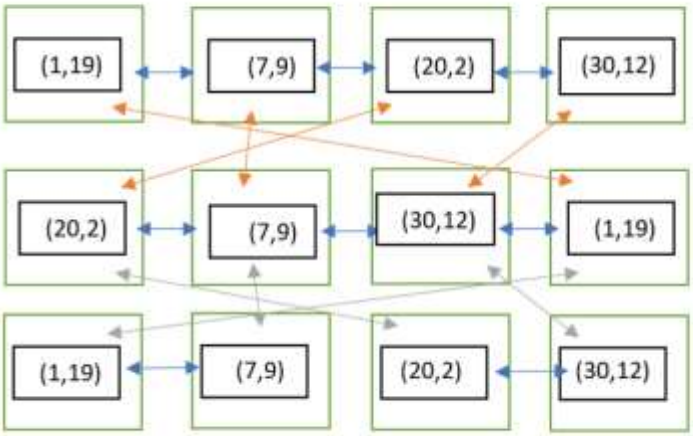
ציר Y:

חץ אפור- newConnectToOtherList

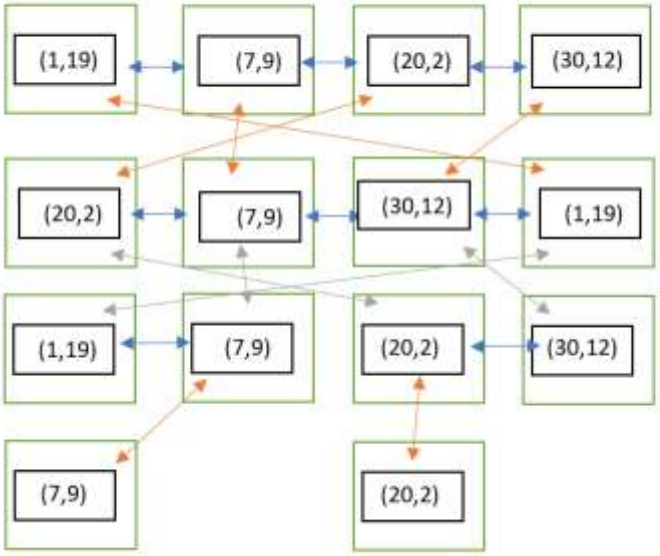
מציאת ראשים צירי X:



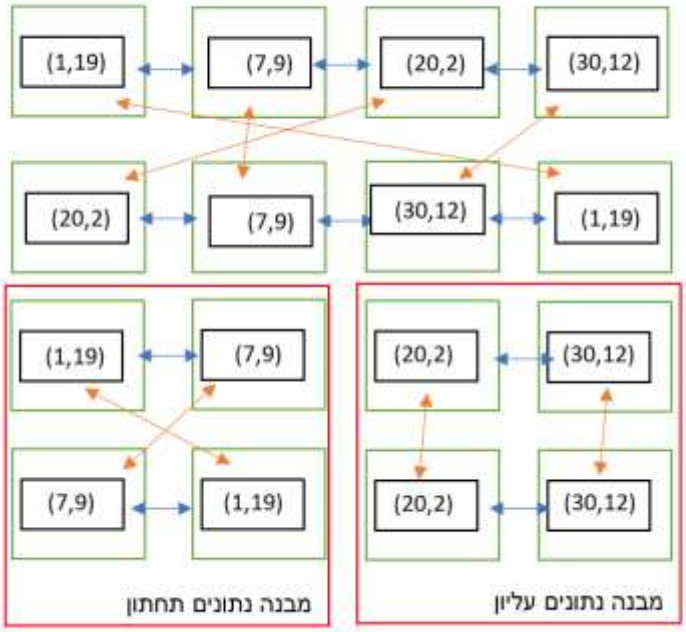
מציאת המשך צירי X:



מציאת ראשים ציר Y:



המשך מציאת צירי Y:



אז בסה"כ 4O(n)+4O(1)=O(n)