חלק ב'- הסבר תיאורתי:

בחירת מבנה נתונים:

מבנה הנתונים אותו בחרנו הינו השדה היחידי במחלקה DataStructure, המבנה הינו מערך בגודל 2 המכיל בכל אחד מהתאים רשימה מקושרת, ממיינת, דו כיוונית, המכילה מצביעים לחוליות ה-first, וה-last. בנוסף במחלקה DLink (double link) הוספנו שדה שנקרא otherLink המשמש מצביע לחוליה המכילה את אותה ה-data ברשימה המקבילה, דבר המאפשר לנו מעבר מהיר וזול בין שתי הרשימות.

1. DataStructure()

בנאי- מאתחל לנו את ה- comparators הדרושים בונה שתי רשימות מקושרות ממויינות דו-כיווניות, בונה מערך בגודל 2 ומציב את הרשימות בתאי המערך כאשר index 0 ממויין לי ציר ה-x, index 1 ממיין לפי ציר ה-y.

זמן ריצה: 5 שורות בזמן ריצה קבוע לכן O(1).

2. Void addPoint(Point point)

פונקציה זו קוראת לפונקציית add של SLL אשר עוברת חוליה חוליה ברשימה המקושרת ומוסיפה חוליה חדשה עם ה data החדש במקום אשר לא יפגע בסדר הממויין של הרשימה . לאחר ההוספה הפונקציה add של SLL מחזירה את החוליה החדשה שנוצרה, בכדי לאפשר לנו לעדכן את השדה otherLink ב DLink הנדרש.

זמן ריצה: במקרה הגרוע ביותר פונקציית add של SLL תעבור על כל החוליות בשביל למצוא את המקום הדרוש לחולייה החדשה, ולכן פעולה זאת תיקח לכל היותר O(n), בנוסף עדכון הotherLink הינו צירוף של שני פעולות קבועות ולכן דורש בנוסף O(1), מכאן סה"כ זמן הריצה של הפונקציהה 2n+2 שזהו O(n) .

3. Point[] getPointInRangeRegAxis(int min,int max,Boolean axis)

ראשית הפונקציה ניגשת לרשימה המתאימה לפי ציר ה-axis לאחר מכן מדלגים חוליה חוליה עד שמגיעים לחוליה אשר עונה על הדרישות (הערך הנדרש גדול מ-min), ממשיכים לדלג חוליה חוליה תוך כדי ספירה של כמה חוליות עברנו עד שמגיעים לחוליה שכבר לא עונה על הדרישות (הערך הנדרש גדול מ-max), כעת נקרא לפונקצית עזר פרטית שבונה מערך בגודל הנדרש, רצה מהחוליה הראשונה עד לרגע שהמערך מלא ומוסיפה את ה- data שלהן ומחזריה את המערך.

זמן ריצה: במקרה הגרוע ביותר ערך ה- max, גדול מהערך הנדרש בחוליה- last, ולכן נעבור חוליה חוליה על כל ה-SLL (O(n)), לאחר מכן פונקצית העזר אשר בונה את המערך צריכה להוסיף את כל ה- data שמוכל בחוליות שבתחום, וגם כאן במקרה הגרוע ביותר O(n) .
סה"כ 2n שזה O(n).

4. Point[]bgetPointInRangeOppAxis(int min,int max,Boolean axis)

על מנת לייעל את האלגוריתם שלנו, העדפנו לחזור על חלקים מפונקציה מס' 3.
בשביל לרדת לזמן ריצה ליניארי. נעבור על הציר הoppAxis חוליה חוליה ונבדוק האם הערכים שלה בציר ה-axis נמצאים בין min ל- max אם כן נוסיף אותה ל LinkList רגילה כאשר אנו יודעים כי היא ממויינת (בשביל להשתמש ב-add של LinkList).
נמיר למערך

זמן ריצה: לסרוק את ה SLL המתאימה במקרה הגרוע ביותר זהו O(n) , להעביר למערך במקרה הגרוע ביותר לוקח גם O(n), לכן סה"כ הפונקציה רצה בO(n).

5. Double getDensity()

שיטה זו בודקת כי במבנה הנתונים לפחות 2 נק, לוקחת את המידע הדרוש מהמבנה בפעולות שקוראות בזמן O(1), מציבה בנוסחה הנתונה בדרישות השאלה ומחזירה את הצפיפות.

זמן ריצה: מורכבת רק מתשעה פעולות קבועות לכן O(1).

6. Void narrowRange(int min, int max, Boolean axis)

שיטה זה עוברת חוליה חוליה מההתחלה עד אשר מגיעים לחוליה הראשונה שערך הX או Y שלה גדול מmin לפי ציר הaxis. עבור כל חוליה אשר נמצאת מחוץ לתחום נשתמש בפונקציית remove(otherLink) אשר זמן הריצה שלה הוא O(1) וכך נוכל להוריד את כל החוליות גם מהSLL לפי ציר הaxis!.

באותה שיטה נעבור גם חוליה חוליה מהסוף עד החוליה הראשונה שערך הXorY שלה קטן מmax.

לבסוף נחתוך את הקצוות בציר הaxis ונאתחל את שדות הfirst&last של הsll.

זמן הריצה הוא מעבר רק על הנקודות הנמצאות מחוץ לתחום כאשר כל פעולות remove היא בזמן ריצה של O(1) ולכן זמן הריצה הסופי יהיה O(|A|) כאשר |A| הינו מספר הנקודות שיש למחוק.

7. Boolean getLargestAxis()

הפונקציה לוקחת את המידע ממבנה הנתונים (ערך X מקסימלי ומינימלי, ערך Y מקסימלי ומינימלי) ובודקת איזה ציר גדול יותר (ההפרש הגדול יותר הינו הציר הגדול יותר).

זמן ריצה: הפונקציה מורכבת מחמישה פעולות בזמן ריצה קבוע ולכן סה"כ O(1).

8. Container getMedian(Boolean axis)

משום שאנו עובדים עם SLL הנקודה החציונית בציר הנבחר היא בעצם הנקודה במקום [n/2](הערך השלם). ולכן ניגשנו לרשימה הנכונה לפי ציר ה-axis , נעדכן את שדות ה-

Container במצביע על חוליה, ה- data שלה וה- otherLink של החוליה (להחליף את ה- get).

זמן ריצה: מישום שאנו עובדים עם SLL על מנת להגיע למיקום ה [n/2] יש לעבור חוליה חוליה המשמעות הי סה"כ O(n).

9. Point[] nearestPairInStrip(Container container, double width, Boolean axis)

על מנת לא לעבור בצורה מיותר על נקודות שמחוץ לרצועה, השתמשנו במידע הנוסף שאנו מקבלים בפונקציה הזאת והוא ה- container, אשר אנו יודעים כי הוא ממוקם בידיוק באמצע רוחב הרצועה, בעזרתו ייצרנו רשימה מקושרת (כאשר אנו יודעים כי הינה ממויינת לפי הציר ההפוך מ-axis), כעת נותר לבדוק מי זוג הנקודות הכי קרובות בתוך רשימה זו, לשם כך נשתמש בהוכחה מתמטית שמראה כי מספיק לבדוק רק את ה7 נק' הקרובות - http://people.csail.mit.edu/indyk/6.838-old/handouts/lec17.pdf בעמוד 7-8.
את המרחק נבדוק באמצעות פונקיית עזר double findDistance(Point a,Point b) אשר עובדת ב(O(1, נשווה בין המרחקים הזוג הכי קרוב הינו הזוג הקרוב ביותר ברצועה ואתו נחזיר במערך.

זמן ריצה : יצירת רשימה את לקחה מאיתנו O(|c|) כאשר c הינו מספר הנקודות ברצועה. מציאת הזוג הקרוב יקח לנו 7 השוואות עבור כל חוליה מתוך c החוליות, סה"כ 8c שזה O(|c|).

10. Point[] nearestPair()

פונקציה זו הינה פונקציה רקורסיבית אשר בחלק קריאה מחלקת את מבנה הנתונים לשנים בזמן ריצה של O(n) ומחזירה את זוג הנקודות הקרובות ביותר במבנה הנתונים. תנאי העצירה שלה הוא כשאר מבנה הנתונים מכיל פחות מ3 נק. כאשר היא מקבל זוג נק מהחלק הראשון,וזוג מהחלק השני, היא משווה בין המרחקים בודקת איזה זוג קרוב יותר ובודקת בעזרת nearsetPairInStrip אם אין שתי נק אחת מחלק אחד ואחת מחלק שני שקרובות יותר מהזוג שמצאנו עד כה. לבסוף מחזירה את הנקודות הקרובות ביותר במבנה.

זמן ריצה: מפני שהפונקציה בכל קריאה רקורסיבית מפצלת ל2 קיימים log(n) פיצולים כשאר כל פיצול לוקח O(n) ומפעלים את nearsetPairInStrip אשר במקרה הכי גרוע c=n אז כל קראיה דורשת O(n) סה"כ O(nlog(n)).

מפני שפיצלנו את ה- DataStructuer הרווחנו את כל הפעולות שמוגדרות עליו ובעיקר את nearsetPairInStrip ולא היינו צריכים לבצע התאמה שהייתה דרושה במידה והיינו מפצלים את הרשימה בתוך הDataStructuer .

כיצד פיצלנו את מבנה הנתונים ב(O(n :

בנינו בנאי חדש ב-SLL אשר מקבל חוליה ראשונה,אחרונה,compartor,וגודל הרשימה. בנאי זה מאפשר לנו לפצל רשימות מקושרות ממויינות ב (O(n, לאחר מכן יצרנו בנאי חדש ב DataStructuer אשר מקבל מערך של SLL ומאתחל אותו מיד בשדה היחיד של המחלקה. כעת אנו יכולים לפצל את השדה של מבנה הנתונים ולייצר ממנו 2 מבני נתונים חדשים כל זה ב(O(n.

<u>הפונקציה public SortedLinkedList[] split(int value , Boolean axis) :</u>

הפונקציה ספליט מחזירה מערך בגודל 2 המכיל SortedLinkedList.

בתא הראשון יתקבל הSLL שערך הaxis של כל האיברים בו קטנים ממש מvalue .

ובתא השני יתקבל SLL שערך הaxis של כל האיברים בו גדולים מvalue.

הגישה לכל נקודה באוסף תתבצע בעזרת מעבר רגיל על הSLL כאשר ניתן להתחיל מהחוליה הראשונה או מהחוליה האחרונה בכיוון הרצוי.

זמן הריצה של הפונקציה ספליט הוא (O(|C| משום שלולאת הwhile תרוץ במקביל מהסוף לכיוון הvalue ומההתחלה לכיוון הvalue עד אשר תמצא החוליה המדויקת בה תתבצע ההפרדה.

במקרה כזה לולאת הwhile תעצר לאחר שתעבור על |C|+1 חוליות במקרה הגרוע ביותר.

```
SortedLinkedList[] output = new SortedLinkedLists[2];

SortedLinkedList myList  ,  beginToValue , endToValue;

If(axis)    myList = sortedLists[0];       else   myList = sortedLists[1];

DLink fromBegin = myList.getFirst();     DLink fromEnd = myList.getLast();

Int counter;

While(    ( getXorY( fromBegin.getData() , axis ) < value ) &&
.             ( getXorY( fromEnd.getData() , axis ) >= value )           )       {

            fromBegin = fromBegin.getNext();

            fromEnd = fromEnd.getPrev();

            counter++                                                                  }

DLink toChange;     int  firstSize  ,   secondSize;

if(! getXorY( fromBegin.getData() , axis ) < value )

{

            toChange = fromBegin . getPrev() ;

            firstSize = counter   ;   secondSize = myList.size – counter

}

else

{

             toChange = fromEnd;

             secondSize = counter   ;   firstSize = myList.size – counter

}

beginToValue.first = myList.first;     endToValue.last = myList.last;

beginToValue.last = toChange;        endToValue.first = toChange.getNext();

beginToValue.size = firstSize   ;      endToValue.size = secondSize;

beginToValue . last . setNext (null );

endToValue . first . setPrev ( null );

output[0] = beginToValue;     output[1] = endToValue;

return output;
```