האוניברסיטה הפתוחה

י"ד באלול תש"ף

3

מס' שאלון - 475

בספטמבר 2020

סמסטר 2020ב

מס' מועד 96

4 / 20441

שאלון בחינת גמר

20441 - מבוא למדעי המחשב ושפת Java

משך בחינה: 4 שעות

בשאלון זה **19** עמודים

מבנה הבחינה:

קראו בעיון את ההנחיות שלהלן:
* בבחינה יש חמש שאלות.
* כל התכניות צריכות להיות מתועדות היטב.
יש לכתוב תחילה בקצרה את האלגוריתם וכל הסבר נוסף הדרוש להבנת התכנית.
יש לבחור בשמות משמעותיים למשתנים, לפונקציות ולקבועים שבתכנית.
תכנית שלא תתועד כנדרש לעיל תקבל לכל היותר % 85 מהניקוד.
* יש להקפיד לכתוב את התכניות בצורה מבנית ויעילה.
תכנית לא יעילה לא תקבל את מלוא הנקודות.
* אם ברצונכם להשתמש בתשובתכם בשיטה או במחלקה הכתובה בחוברת השקפים,
אין צורך שתעתיקו את השיטה או את המחלקה למחברת הבחינה. מספיק להפנות
למקום הנכון, ובלבד שההפניה תהיה מדויקת (פרמטרים, מיקום וכו').
* אין להשתמש במחלקות קיימות ב- Java , חוץ מאלו המפורטות בשאלות הבחינה.
* יש לשמור על סדר; תכנית הכתובה בצורה בלתי מסודרת עלולה לגרוע מהציון.
* בכתיבת התכניות יש להשתמש אך ורק במרכיבי השפה שנלמדו בקורס זה
אין להשתמש במשתנים גלובליים!
* אפשר לתעד בעברית. אין צורך בתיעוד API .
כל התשובות צריכות להיכתב בתוך קובץ המבחן במקומות המתאימים בלבד.
תשובה שתיכתב שלא במקומה לא תיבדק.

בהצלחה !!!

ʃ21003535 17h~

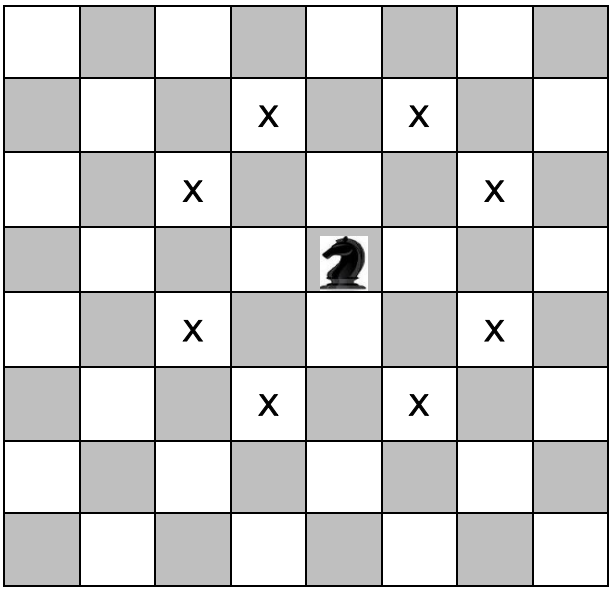
שאלה 1 (25 נקודות)

נתון מערך דו-ממדי mat, בגודל n×m (n שורות ו-m עמודות) שערכיו הם מספרים **שלמים וחיוביים ממש**.

- נגדיר **מסלול-פרש** (knight-path) במערך אם הוא מקיים את התנאים הבאים:

1. המסלול מתחיל בתא הראשון במערך – שורה ראשונה ועמודה ראשונה.
2. המסלול מסתיים בתא האחרון במערך – שורה אחרונה ועמודה אחרונה.
3. אפשר לעבור מתא אחד לשני רק בצעדים של **פרש** (knight) בלוח שחמט. כלומר, שילוב של צעד אחד ישר (למעלה, למטה, ימינה ושמאלה) וצעד אחד באלכסון (ארבעה כיוונים). להלן איור המתאר את הצעדים של הפרש בשחמט.

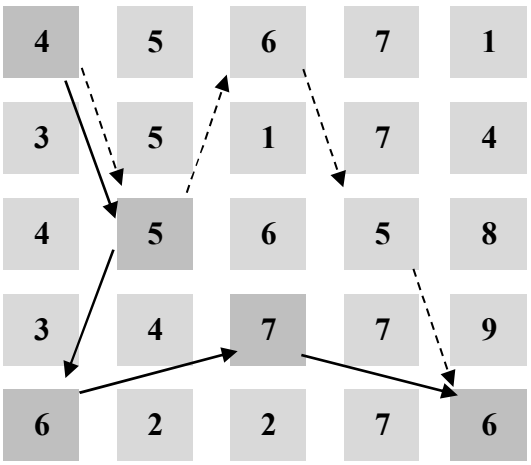
התאים אליהם יכול לנוע הפרש המסומן על לוח השחמט שלהלן מסומנים בתו X.



4. אפשר ללכת מתא לתא רק אם ההפרש בין הערכים שנמצאים בשני התאים הוא לכל היותר 1. למשל, מתא שערכו הוא 4 אפשר ללכת לתא שערכו הוא 3 או 4 או 5.
5. **המסלול לא יכול לחזור לתא בו הוא היה כבר!**

עליכם לכתוב שיטה סטטית רקורסיבית המקבלת מערך דו-ממדי, מלא במספרים שלמים. השיטה צריכה להחזיר את ערכו של הסכום הגדול ביותר של ערכי התאים במסלול-פרש, מבין כל מסלולי הפרש הקיימים במערך.
אם אין מסלול כזה השיטה תחזיר 1-.

לדוגמא, במערך להלן מסומן מסלול-פרש עם חיצים מלאים שסכומו מקסימלי – **28**. שימו לב שיש עוד מסלול-פרש אחד במערך, והוא מסומן עם חיצים מקווקווים והוא: 6-5-6-5-4 אבל סכומו הוא 26 (לא הסכום המירבי).
שימו לב, קיימים עוד מסלולים במערך המורכבים מצעדי פרש אך הם אינם מסלולי-פרש כי ההפרשים בין התאים בהם אינם מתאימים להגדרה.



חתימת השיטה היא:

```
public static int maxSumKnight(int[][] mat)
```

השיטה צריכה להיות רקורסיבית ללא שימוש בלולאות כלל. כך גם כל שיטות העזר שתכתבו (אם תכתבו) לא יכולות להכיל לולאות.

אפשר להשתמש בהעמסת-יתר (overloading).

מותר לשנות את המערך במהלך השיטה, אבל בסופה הוא צריך לחזור למצבו המקורי.

אין צורך לדאוג ליעילות השיטה, אבל כמובן שצריך לשים לב לא לעשות קריאות רקורסיביות מיותרות! אל תשכחו לתעד את מה שכתבתם!

התשובה היא:

שאלה 2 (25 נקודות)

a הוא מערך המכיל מספרים שלמים ממוין בסדר עולה **ממש** (כלומר המספרים שונים זה מזה).

נגדיר **מערך טווחים** של a כמערך שנבנה באופן הזה: עבור כל רצף של **מספרים עוקבים ב-a**, יהיה במערך הטווחים אובייקט שמכיל שני שדות של מספרים שלמים. שדה אחד הוא המספר הקטן ביותר ברצף (small) ושדה שני הוא המספר הגדול ביותר ברצף (big). רצף יכול להיות באורך 1 או יותר. אם הרצף הוא באורך 1, הוא מיוצג במערך הטווחים על ידי אובייקט ששני השדות שלו שווים.

לדוגמא, עבור המערך **a**:

0	1	2	3	4	5	6	7	8	9	10
3	4	5	12	19	20	100	101	102	103	104

מערך הטווחים **rangeA** יהיה:

0	1	2	3
<3 , 5>	<12 , 12>	<19 , 20>	<100 , 104>

(כל אובייקט מסומן כזוג סדור עם סוגריים משולשים).

לשם כך הגדרנו מחלקה בשם Range בה כל אובייקט מייצג טווח של מספרים לפי המתואר לעיל.

להלן המחלקה Range:

```
public class Range
{
    private int _small, _big;

    public Range(int s, int b)
    {
        _small = s;
        _big = b;
    }

    public int getSmall()    {
        return _small;
    }

    public int getBig()      {
        return _big;
    }
}
```

אפשר להניח שבכל אובייקט מהמחלקה Range, הערך _small תמיד קטן או שווה לערך _big, ואין צורך לבדוק זאת.

אפשר גם להניח שבמערך המקורי a כל המספרים שונים זה מזה.

כתבו שיטה בוליאנית יעילה המקבלת כפרמטר, מערך טווחים rangeA (המייצג מערך של מספרים שלמים a ממוינים בסדר עולה ממש) השיטה צריכה להחזיר את הערך של המספר החיובי (ממש) הקטן ביותר במערך a. אם אין מספר חיובי ממש במערך, השיטה תחזיר 1-.

לדוגמא,

עבור המערך rangeA הבא:

0	1	2	3
<-7, -3>	<-1, -1>	<4, 6>	<20, 22>

שמייצג את המערך a:

0	1	2	3	4	5	6	7	8	9	10	11
-7	-6	-5	-4	-3	-1	4	5	6	20	21	22

השיטה תחזיר את הערך **4**.

אם בתא באינדקס 1 של המערך rangeA יהיה הטווח <-1, 2>, השיטה תחזיר את הערך 1.

חתימת השיטה היא:

```
public static int minimalPositive (Range[] rangeA)
```

חשוב מאד –

1. אסור להמיר את מערך הטווחים rangeA למערך המספרים השלמים a ולעבוד על המערך a. השיטה אמורה לקבל את מערך הטווחים rangeA בלבד ולעבוד עליו.

2. אורך הקלט n הוא גודל מערך הטווחים rangeA. כלומר, בדוגמא לעיל n = 4.

שימו לב:

השיטה שתכתבו צריכה להיות יעילה ככל הניתן, גם מבחינת סיבוכיות הזמן וגם מבחינת סיבוכיות המקום. תשובה שאינה יעילה מספיק כלומר, שתהיה בסיבוכיות גדולה יותר מזו הנדרשת לפתרון הבעיה תקבל מעט נקודות בלבד.

מה סיבוכיות זמן הריצה וסיבוכיות המקום של השיטה שכתבתם? הסבירו תשובתכם.

אל תשכחו לתעד את מה שכתבתם!

התשובה היא:

שאלה 3 (17 נקודות)

נניח שהמחלקה Node שלהלן מממשת צומת עץ בינרי שערכיו הם מספרים שלמים (int).

Constructor Summary	
Node(int data) Constructs a Node object.	
Method Summary	
Node	**getLeftSon**() Returns the left son of the node.
int	**getData**() Returns the value of the node.
Node	**getRightSon**() Returns the right son of the node.

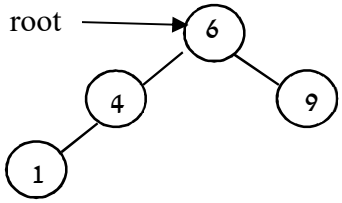
במחלקה Node קיימת בנוסף שיטה שחתימתה

```
public Node getGrandPaNode(Node root)
```

השיטה מקבלת כפרמטר שורש של עץ בינרי root ומחזירה את הצומת שהוא **הסבא** (האבא של האבא) של הצומת (this) שעליו מופעלת השיטה בעץ ששורשו הוא root.

אם לצומת שעליו מופעלת השיטה אין סבא בעץ ששורשו root, כלומר הוא השורש של העץ או בנו של השורש, או שהוא כלל לא שייך לעץ ששורשו root - השיטה מחזירה null.

לדוגמא, בעץ להלן:



הסבא של הצומת 1 הוא הצומת 6 (שורש העץ). לצמתים 4, 9, 6 אין סבא בעץ.

בנוסף קיימת שיטה סטטית שחתימתה

```
private static int divNum(int num)
```

שמקבלת מספר שלם חיובי ממש ומחזירה את מספר המחלקים שלו (כולל 1 והוא עצמו.)

לדוגמא,

אם num=12 אז מספר המחלקים הוא 6 (המחלקים: 1, 2, 3, 4, 6, 12).

אם num=25, אז מספר המחלקים שלו הוא 3 (המחלקים: 1, 5, 25).

אם num=7, אז מספר המחלקים הוא 2 (המחלקים: 1, 7).

לנוחותכם, בטבלה שלהלן כתבנו לצד כל מספר (מ-1 עד 100) את מספר המחלקים שלו.

4	**91**	5	**81**	2	**71**	2	**61**	4	**51**	2	**41**	2	**31**	4	**21**	2	**11**	1	**1**
6	**92**	4	**82**	12	**72**	4	**62**	6	**52**	8	**42**	6	**32**	4	**22**	6	**12**	2	**2**
4	**93**	2	**83**	2	**73**	6	**63**	2	**53**	2	**43**	4	**33**	2	**23**	2	**13**	2	**3**
4	**94**	12	**84**	4	**74**	7	**64**	8	**54**	6	**44**	4	**34**	8	**24**	4	**14**	3	**4**
4	**95**	4	**85**	6	**75**	4	**65**	4	**55**	6	**45**	4	**35**	3	**25**	4	**15**	2	**5**
12	**96**	4	**86**	6	**76**	8	**66**	8	**56**	4	**46**	9	**36**	4	**26**	5	**16**	4	**6**
2	**97**	4	**87**	4	**77**	2	**67**	4	**57**	2	**47**	2	**37**	4	**27**	2	**17**	2	**7**
6	**98**	8	**88**	8	**78**	6	**68**	4	**58**	10	**48**	4	**38**	6	**28**	6	**18**	4	**8**
6	**99**	2	**89**	2	**79**	4	**69**	2	**59**	3	**49**	4	**39**	2	**29**	2	**19**	3	**9**
9	**100**	12	**90**	10	**80**	8	**70**	12	**60**	6	**50**	8	**40**	8	**30**	6	**20**	4	**10**

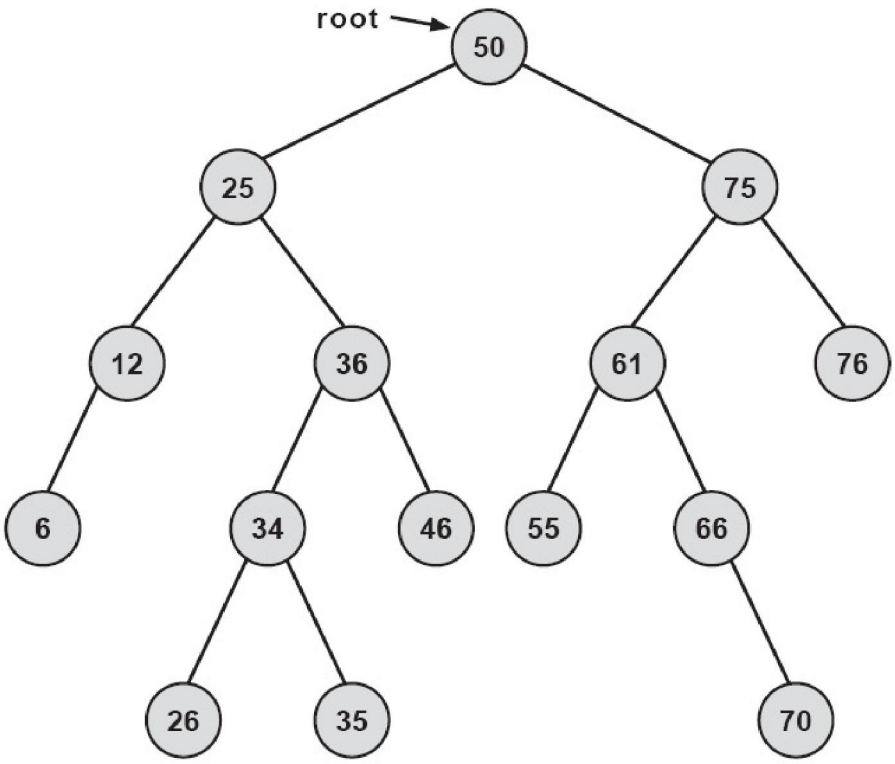
המחלקה BinaryTree מאגדת בתוכה שיטות סטטיות לטיפול **בעץ בינרי**.

בין השיטות קיימת השיטה `what` הבאה **שפועלת על עץ חיפוש בינרי.**

```
public static int what(Node root, Node curr)  {
    if (curr == null)
        return 0;
    Node grPa = curr.getGrandPaNode(root);
    int result = 0;
    if (grPa != null)
    {
        int currNum = divNum(curr.getData());

        if (grPa.getData() > curr.getData())
        {
            if (grPa.getRightSon() != null &&
                grPa.getRightSon().getLeftSon() != null &&
  divNum(grPa.getRightSon().getLeftSon().getData()) ==
currNum)
            {
                    result++;
            }
            if (grPa.getRightSon() != null &&
                grPa.getRightSon().getRightSon() != null &&
 divNum(grPa.getRightSon().getRightSon().getData()) ==
currNum)
            {
                    result++;
            }
        }
    }
    return result + what(root, curr.getLeftSon()) +
                    what(root, curr.getRightSon());
}
```

נתון עץ **החיפוש הבינרי** הבא:



סעיף א (4 נקודות)

אם נקרא לשיטה what כאשר שני הפרמטרים הניתנים לה הם שורש העץ הנתון לעיל, כלומר:

```
System.out.println(what(root, root));
```

מה יודפס על הפלט?

התשובה היא:

סעיף ב (5 נקודות)

איזה שינוי **מינימלי** אפשר לעשות **על העץ** לעיל, כדי שהקריאה לשיטה what (root, root) תחזיר ערך שגדול ב- 1 מהערך שהוחזר בסעיף א? **שימו לב, השינוי צריך להיות אך ורק בערכים השוכנים בצומתי העץ. לא במבנה העץ ולא בשיטה עצמה.**

התשובה היא:

סעיף ג (8 נקודות)

מה מבצעת השיטה **what** באופן כללי כשהיא מקבלת כפרמטרים **פעמיים** שורש של **עץ חיפוש בינרי** root? שימו לב, עליכם לתת תיאור ממצה של **מה עושה השיטה באופן כללי**, ולא תיאור של מה עושה כל שורה בשיטה, או איך היא מבצעת זאת. **כלומר, מה המשמעות של הערך שהשיטה מחזירה? התייחסו למקרי קצה.**

התשובה היא:

שאלה 4 (15 נקודות)

נתונות המחלקות Veg, Tomato, Pepper, CherryTomato, ChillyPepper הבאות, כל אחת בקובץ נפרד, כמובן, וכולן באותן חבילה:

```
public class Veg
{
    private int _weight;
    final public int AMOUNT_PER_BOX=10;

    public Veg(int w){
        _weight=w;
    }

    public int getWeight(){
        return _weight;
    }

    public int amountInBox(){
        return AMOUNT_PER_BOX;
    }
}

//-------------------------------------------------------

public class Tomato extends Veg
{
    public Tomato(int w){
        super(w);
    }

    public boolean equals(Object t) {
        if(t instanceof Tomato)
            if(getWeight()==((Tomato)t).getWeight())
                return true;
        return false;
    }
}

//-------------------------------------------------------

public class Pepper extends Veg
{
    public Pepper(int w){
       super(w);
    }

    public boolean equals(Object o){
        if(o instanceof Pepper)
           if(getWeight()==((Pepper)o).getWeight())
              return true;
        return false;
    }
}
//-------------------------------------------------------
```

```
public class CherryTomato extends Tomato
{
    public CherryTomato(int w) {
        super(w);
    }

    public int amountInBox(){
        return AMOUNT_PER_BOX*10;
    }
}

//-------------------------------------------------------
----

public class ChillyPepper extends Pepper
{
    public ChillyPepper(int w){
        super(w);
    }

    public int amountInBox(){
        return AMOUNT_PER_BOX*10;
    }
}
```

כתבנו שיטה המקבלת אובייקט של אחת המחלקות Veg, Tomato, Pepper, CherryTomato, ChillyPepper לעיל, ומדפיסה מהו סוג האובייקט. כלומר, מאיזו מחלקה הוא נוצר. עליכם להשלים את החסר בשיטה.

הניחו שהשיטה מוגדת במחלקה Driver כלשהי שנמצאת באותה חבילה עם המחלקות שלעיל.

לדוגמא, אם היינו כותבים ב- main את הפקודות:

```
Pepper p = new Pepper(3);
Veg vt = new Tomato(5);
whichClass(p);
whichClass(vt);
```

אז היה מודפס על הפלט:

```
Pepper
Tomato
```

חתימת השיטה היא:

```
public static void whichClass(Veg obj)
```

שימו לב, עליכם להשלים את השיטה שאנחנו כתבנו. אם לא תעשו כך, ותכתבו שיטה משלכם, עליכם לעמוד בהגבלות הכתובות להלן, ובכל מקרה, התשובה שלכם תקבל לכל היותר 10 נקודות, גם אם היא תהיה נכונה.

כדי לבדוק מהו סוג האובייקט, יש להיעזר בשיטות `amountInBox, equals` שהוגדרו במחלקות לעיל, וגם בשיטה `equals` של המחלקה `Object`. מותר ליצור אובייקטים חדשים.

אסור להשתמש בשיטה `instanceOf` או בשיטות אחרות של המחלקה `Object`.

אסור לשנות את המחלקות `Veg, Tomato, Pepper, CherryTomato, ChillyPepper`.

אפשר להניח שהאובייקט `obj` שייך לאחת המחלקות הנ"ל, והוא אינו `null`.

התשובה היא:

```
public ____________________________________________
{
   int w=obj.getWeight();
   Tomato t= ______________________________
   Pepper p=new Pepper(w);
   if_____________________________________________________
       System.out.println("Veg");
   else
   {
       if_____________________________________________
           if_____________________________________
               System.out.println("Pepper");
           else
               System.out.println("ChillyPepper");
       else
           if_____________________________________
               ____________________________________
           else
               System.out.println("CherryTomato");
   }
}
```

שאלה 5 (18 נקודות)

נגדיר את הטיפוס **תור-כפול DoubleQueue** כמבנה נתונים המורכב משני תורים של מספרים שלמים.

לפניכם ממשק חלקי של המחלקה **DoubleQueue**:

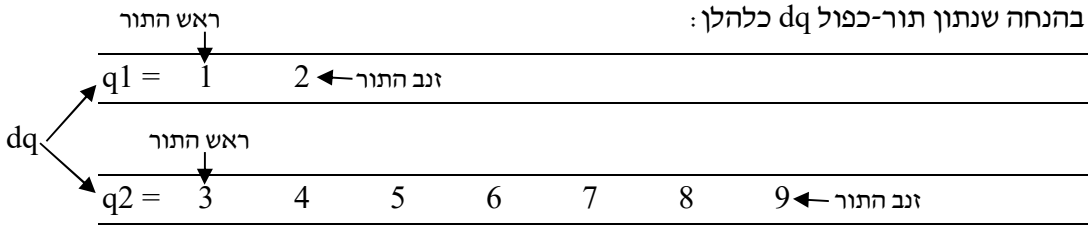
תיאור השיטה	**חתימת השיטה**
השיטה מחזירה את מספר האיברים בתור מספר i. ערכו של הפרמטר i הוא 1 או 2 בלבד. אם ערכו של i הוא 1 השיטה תחזיר את מספר האיברים בתור הראשון, ואם ערכו של i הוא 2 השיטה תחזיר את מספר האיברים בתור השני.	public int getQSize(int i)
השיטה מעבירה איבר **אחד** מראש תור i לסוף תור j. ערכי הפרמטרים i , j הם 1 או 2 בלבד, כאשר 1 מייצג את התור הראשון ו-2 את התור השני. אם ערכי i, j שווים, השיטה תעביר את האיבר שבראש תור i לסופו. **אפשר להניח** כי תור i אינו ריק.	public void moveElement(int i, int j)

במחלקה DoubleQueue נתונה השיטה what הבאה:

אפשר להניח ש- i ו-j הם מספרים שלמים שאינם שונים מ- 1 ו- 2, n שלם חיובי ממש.

```
public void what(int i, int j, int n)
{
    int sizeI = getQSize(i);
    for (int k=0; k<sizeI-n; k++)
        moveElement(i,i);
    for (int k=0; k<n; k++)
        moveElement(i,j);
}
```

בהנחה שנתון תור-כפול dq כלהלן:



סעיף א: (2 נקודות)

איך ייראה התור הכפול dq לאחר שנפעיל עליו את השיטה what

```
dq.what(2, 1, 4);
```

התשובה היא:

התור q1 יהיה זה (יש לכתוב את המספרים משמאל לימין, כאשר בשמאל ראש התור ובימין זנבו).

התור q2 יהיה זה (יש לכתוב את המספרים משמאל לימין, כאשר בשמאל ראש התור ובימין זנבו).

סעיף ב: (5 נקודות)

מה מבצעת השיטה **what** באופן כללי? הסבירו בקצרה **מה** השיטה עושה ולא **כיצד** היא מבצעת זאת.

שימו לב, עליכם לתת תיאור ממצה של **מה עושה השיטה באופן כללי** כשהיא מופעלת על תור-כפול כלשהו עם פרמטרים i, j ו- n, ולא תיאור של מה עושה כל שורה בשיטה, או איך היא מבצעת זאת. **התייחסו למקרי קצה.**

התשובה היא:

במחלקה DoubleQueue נתונה השיטה secret הבאה:

```
public  void secret ()
{
    int size1 = getQSize(1);
    int size2 = getQSize(2);
    int diff = size1-size2;
    if (Math.abs(diff)>1)
    {
        if (diff > 0)
            what(1, 2, diff/2);
        else
            what(2, 1, -1*diff/2);
    }
}
```

סעיף ג: (4 נקודות)

איך ייראה התור הכפול dq **המקורי** (זה שמצויר לעיל, לפני הפעלת השיטה what בסעיף א עליו) לאחר שנפעיל עליו את השיטה secret

```
dq.secret();
```

התשובה היא:

התור q1 יהיה זה (יש לכתוב את המספרים משמאל לימין, כאשר בשמאל ראש התור ובימין זנבו).

התור q2 יהיה זה (יש לכתוב את המספרים משמאל לימין, כאשר בשמאל ראש התור ובימין זנבו).

סעיף ד: (7 נקודות)

מה מבצעת השיטה **secret** באופן כללי? הסבירו בקצרה **מה** השיטה עושה ולא **כיצד** היא מבצעת זאת.

שימו לב, עליכם לתת תיאור ממצה של **מה עושה השיטה באופן כללי** כשהיא מופעלת על תור-כפול כלשהו, ולא תיאור של מה עושה כל שורה בשיטה, או איך היא מבצעת זאת. **התייחסו למקרי קצה.**

התשובה היא: