שאלה:

מערך דו-מימדי מדורג הוא מערך שמספר האברים בשורה הראשונה קטן מזה של השורה השניה שקטן בהתאמה מזה של השורה השלישית וכך הלאה.

א. יש לאתחל מערך דו-מימדי כך שיהיה מדורג ולשלוח אותו לפונקציה אשר בודקת האם הוא מדורג:

```
public static boolean is2DStepArray(int[][] arr)
```

ב. מערך דו-מימדי מדורג לגמרי הוא מערך שבשורה הראשונה יש ערך אחד, הבשורה השני שני ערכים וכך הלאה.
יש לכתוב פונקציה המקבלת כפרמטר מערך דו-מימדי ומחזירה האם הוא מדורג לגמרי.

ג. יש לכתוב פעולה המקבלת כפרמטר מערך דו-מימדי, הופכת אותו למערך דו-מימדי מדורג ואז מחזירה האם הוא מערך דו-מימדי מדורג לגמרי.

פתרון:

```
// א
public static boolean is2DStepArray(int[][] arr) {
     for (int i = 0; i < arr.length - 1; i++){
          if (arr[i].length >= arr[i + 1].length){
               return false;
          }
     }
     return true;
}
```

```
// ב
public static boolean is2DStepArrayComplete(int[][] arr) {
     for (int i = 0; i < arr.length; i++){
          if (arr[i].length != i + 1){
               return false;
          }
     }

     return true;
}
```

```
// ג
public static boolean generate2DStepAndCheck(int[][] arr) {
     int[] temp;
     boolean isSwapped;

     do {
          isSwapped = false;

          for (int i = 0; i < arr.length - 1; i++){
               if (arr[i].length > arr[i + 1].length) {
                    temp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = temp;

                    isSwapped = true;
               }

          }
     } while (isSwapped);

     return is2DStepArrayComplete(arr);
}
```

שאלה:

פיקסל -**Pixel** היא יחידת מידע גרפית בסיסית במחשב המתארת נקודה בתמונה דיגיטלית. כל פיקסל מורכב משלושה צבעים: אדום ירוק וכחול.
כל אחד משלושת הצבעים מורכב מערך מספרי בין 0 ל-255.

לדוגמה:

- אדום – 0, ירוק – 0, כחול – 0: מתקבל צבע שחור.
- אדום – 255, ירוק – 255, כחול – 255: מתקבל צבע לבן.
- אדום – 255, ירוק – 0, כחול – 0: מתקבל צבע אדום.

למחלקה **Pixel** שלוש תכונות:

```
private int red ; // מייצגת את הצבע האדום
private int green ; // מייצגת את הצבע הירוק
private int blue ; // מייצגת את הצבע הכחול
```

במחלקה הוגדרה פעולה בונה (בנאי) המקבלת ערכים לשלוש תכונות, פעולות **get/set** לכל תכונה.

א. כתבו במחלקה **Pixel** את הפעולה **IsBlack()** הבודקת האם מהערכים של שלושת הצבעים מתקבל צבע שחור. אם כן – הפעולה תחזיר ערך **true**, ואם לא – הפעולה תחזיר ערך **false**.

נתונה המחלקה **Picture** המייצגת תמונה. למחלקה תכונה אחת בלבד, מערך דו-ממדי מטיפוס **Pixel** בגודל 256x256 בשם: **pixels**

```
private Pixel[] pixels = new Pixels[256][256];
```

ב. כתבו במחלקה **Picture** את הפעולה **IsBalancedPicture()** הפעולה תבדוק אם מספר הפיקסלים הלבנים שווה למספר הפיקסלים השחורים. אם כן – הפעולה תחזיר ערך **true**, ואם לא – הפעולה תחזיר ערך **false**.

ג. "תמונת שחור לבן" היא תמונה שבה כל הפיקסלים הם בצבעים שחור או לבן בלבד. כתבו במחלקה **Picture** את הפעולה **IsBlackWhite()** הפעולה תבדוק אם התמונה היא "תמונת שחור לבן". אם כן – הפעולה תחזיר ערך **true**, ואם לא – הפעולה תחזיר ערך **false**.

ד. "מסגרת התמונה" היא השורה הראשונה, השורה האחרונה, העמודה הראשונה והעמודה האחרונה. כתבו במחלקה **Picture** את הפעולה **SetWhiteBorder()** הפעולה תצבע את המסגרת של התמונה בלבן.

פתרון:

```
public class Pixel {
     private int red;
     private int green;
     private int blue;


     public Pixel(int blue, int green, int red) {
          this.blue = blue;
          this.green = green;
          this.red = red;
     }


     public int getRed() {
          return red;
     }


     public void setRed(int red) {
          this.red = red;
     }


     public int getGreen() {
          return green;
     }


     public void setGreen(int green) {
          this.green = green;
     }



     public int getBlue() {
          return blue;
     }


     public void setBlue(int blue) {
          this.blue = blue;
     }
```

```
    @Override
    public String toString() {
        return "Pixel{"
            + "red=" + red
            + ", green=" + green
            + ", blue=" + blue
            + '}';
    }



    // א
    public boolean isBlack() {
        return this.blue == 0
            && this.red == 0
            && this.green == 0;
    }



    // ב
    public boolean isWhite() {
        return this.blue == 255
            && this.red == 255
            && this.green == 255;
    }
}
```

```
public class Picture {
     private Pixel[][] pixels = new Pixel[256][256];


     // ב
     public boolean isBalancedPicture() {
          int blackPix = 0;
          int whitePix = 0;

          for (Pixel[] row : this.pixels) {
               for (Pixel pix : row) {
                    if (pix.isBlack()) {
                         blackPix++;
                    } else if (pix.isWhite()) {
                         whitePix++;
                    }
               }
          }

          return blackPix == whitePix;
     }



     // ג
     public boolean isBlackWhite() {
          for (Pixel[] row : this.pixels) {
               for (Pixel pix : row) {
                    if (!(pix.isBlack() || pix.isWhite())) {
                         return false;
                    }
               }
          }
          return true;
     }
```

```
    // ד
    public void setWhiteBorder() {
        for (int i = 0; i < this.pixels.length; i++) {
            for (int j = 0; j < this.pixels[i].length; j++) {
                if (j == 0
                    || i == 0
                    || j == this.pixels[i].length - 1
                    || i == this.pixels.length - 1) {
                    this.pixels[i][j].setBlue(255);
                    this.pixels[i][j].setGreen(255);
                    this.pixels[i][j].setRed(255);
                }
            }
        }
    }
}
```

תרגיל:

יש לכתוב פונקציה המקבלת כפרמטר מחרוזת. הפונקציה תגריל מקטע אקראי (רצף של מספר אותיות אקראי - לא בהכרח מתחילת המחרוזת).

פתרון:

```
import java.util.Random;

public static String getSubStr(String mainStr) {
     if (mainStr == null || mainStr.isEmpty()) {
          return "";
     }

     Random random = new Random();

     int idx1 = random.nextInt(mainStr.length());

     do {
          int idx2 = random.nextInt(mainStr.length());
     } while (idx1 == idx2);

     int start = Math.min(idx1, idx2);
     int end = Math.max(idx1, idx2) + 1;

     return mainStr.substring(start, end);
}
```

מיון בחירה - Selection Sort

מתחילים את סריקת המערך מאינדקס 0. מוצאים את האבר הקטן ביותר ושומרים על המיקום שלו. בסיום הסריקה הראשונה מחליפים את האבר הראשון עם האבר הקטן ביותר שמצאנו. חוזרים על פעולה זו רק שבכל פעם האינדקס שממנו מתחילים את החיפוש גדל ב- 1.

הקוד:

```
public static void selectionSort(int[] arr){
     int n = arr.length;

     for (int i = 0; i < n - 1; i++) {
          int min_idx = i;

          for (int j = i + 1; j < n; j++) {
               if (arr[j] < arr[min_idx]) {
                    min_idx = j;
               }
          }

          int temp = arr[i];
          arr[i] = arr[min_idx];
          arr[min_idx] = temp;
     }
}
```

מיון הכנסה - Insertion Sort

יוצרים בתוך המערך המקורי תת מערך ממויין בתאים מאינדקס 0. עבור כל ערך שנבדק מתבצעת השוואה האם הוא אמור להיות בתת המערך הממויין (כלומר הוא גדול מזה שצמוד לו משמאלו). במידה והוא אכן גדול - שומרים את הערך שלו ומתחילים להזיז את האברים בתת המערך הממויין ימינה עד למקום הריק שבו הערך יכנס.

הקוד:

```
public static void insertion_sort(int arr[]) {
     int n = arr.length;

     for (int i = 1; i < n; ++i) {
          int key = arr[i];
          int j = i - 1;

          while (j >= 0 && arr[j] > key) {
               arr[j + 1] = arr[j];
               j = j - 1;
          }

          arr[j + 1] = key;
     }
}
```

```
public static int[] bubble_sort(int[] arr) {
     boolean sorted;

     for (int i = 0; i < arr.length - 1; i++) {
          sorted = true;

          for (int j = 0; j < arr.length - 1 - i; j++) {
               if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    [j] = arr[j + 1];
                    [j + 1] = temp;
                    sorted = false;
               }
          }

          if (sorted) {
               return arr;
          }
     }

     return arr;
}
```

תרגיל:

מספר חסר הוא מספר שיש בו את כל הספרות מהספרה הקטנה ביותר עד לספרה הגבוהה ביותר שלו מלבד אחת.

לדוגמא:

- המספר 3514 הוא מספר חסר כי בין 1 ל- 5 חסרה בו רק ספרה אחת (הספרה 2).
- המספר 73264 הוא מספר חסר כי בין 2 ל- 7 חסרה בו רק ספרה אחת (הספרה 5).

יש לכתוב פעולה המקבלת כפרמטר מספר שלם ומחזירה האם הוא מספר חסר או לא.
מה הסיבוכיות של הפעולה שכתבת?

פתרון:

```
public static boolean isMissing(int num) {
     int cnt = 0;
     int dig;
     int maxDig = num % 10;
     int minDig = num % 10;
     int[] counter = new int[10];

     while (num > 0) {
          dig = num % 10;
          counter[dig]++;
          cnt++;
          num /= 10;

          if (dig < minDig) {
               minDig = dig;
          } else if (dig > maxDig) {
               maxDig = dig;
          }
     }

     if (cnt != maxDig - minDig + 1) {
       return false;
     }

     for (int i = minDig + 1; i < maxDig; i++) {
          if (counter[i] == 0) {
               return false;
          }
     }

     return true;
}
// @complexity - O(log(n))
```

תרגיל:

נתונה הפעולה **what** המקבלת שני מערכים של מספרים שלמים:

```
public static boolean what(int[] arr1, int[] arr2) {
     for (int i = 0; i < arr1.length; i++) {
          for (int k = 0; k < arr2.length; k++) {
               if (arr1[i] >= arr2[k]) {
                    return true; // (*)
               }
          }
     }
     return false; // (**)
}
```

א. עקבו בעזרת טבלת מעקב אחר ביצוע הפעולה **what** עבור הפרמטרים הבאים:

```
int[] arr1 = {4,6,2,7}
int[] arr2 = {8,7,9}
```

ב. מה מבצעת הפעולה **what** באופן כללי?

ג.

ג. מהי הסיבוכיות של הפעולה **what**? הסבירו את תשובתכם.

ד. כתבו פעולה חדשה, יעילה יותר, אשר מבצעת אותה מטלה כמו הפעולה **what**.

ה. מהי הסיבוכיות של הפעולה החדשה? הסבירו את תשובתכם.

ו. בפעולה **what** נעשו שני שינוים:

- הפקודה (*) הוחלפה ל- **return false**
- הפקודה (**) הוחלפה ל- **return true**

מה מבצעת הפעולה **what** באופן כללי אחרי השינוי?

פתרון:

// א

i	k	arr1[i]	arr2[k]	arr[i]>=arr2[k]	return
0	0	4	8	false	
0	1	4	7	false	
0	2	4	9	false	
1	0	6	8	false	
1	1	6	7	false	
1	2	6	9	false	
2	0	2	8	false	
2	1	2	7	false	
2	2	2	9	false	
3	0	7	8	false	
3	1	7	7	true	true
3	2	7	9		

ב //

הפעולה מחזירה אמת כאשר במערך **arr1** קיים ערך שהוא גדול או שווה לאחד הערכים במערך השני.

ג //

הסיבוכיות היא - **n * m** יש לנו לולאה בתוך לולאה וכל לולאה רצה מספר שונה של פעמים (גודל כל מערך לא ידוע).

```
// ד
public static boolean what(int[] arr1, int[] arr2) {
        int biggestInarr1 = Integer.MIN_VALUE;
        int smallestInarr2 = Integer.MAX_VALUE;

        for (int i = 0; i < arr1.length; i++) {
             if (biggestInarr1 < arr1[i]) {
                  biggestInarr1 = arr1[i];
             }
        }

        for (int i = 0; i < arr2.length; i++) {
             if (smallestInarr2 > arr2[i]) {
                  smallestInarr2 = arr2[i];
             }
        }

        return biggestInarr1 >= smallestInarr2;
}
```

ה //

הסיבוכיות של הפתרון החדש היא:

// @complexity - O(n)

כי יש לנו 2 לולאות אחת אחרי השניה.

ו //

הפעולה תחזיר אמת כאשר כל הערכים ב- **arr2** גדולים מאלו של **arr1**.

שאלה:

יש להגדיר את המחלקות הבאות:

- מחלקת מנוע בעלת התכונות: מספר מנוע, נפח מנוע.
- מחלקת מטוס בעלת התכונות: שם יצרן, שם דגם, מערך עם 4 מנועים.

יש לבנות עבור כל מחלקה בנאי ריק/מלא, פעולות **set/get** ו- **toString**.

* יש לבנות במחלקת מטוס פעולה המחזירה האם המטוס כשיר לטיסה. מטוס כשיר לטיסה כאשר לכל המנועים של המטוס יש נפח מנוע זהה.

יש ליצור מערך של 5 מטוסים (עם המנועים שלהם).

יש להגדיר במחלקה הראשית את הפונקציות הבאות:

1. פונקציה המקבלת כפרמטר מערך של מטוסים ומציגה את פרטיהם לפלט.
2. פונקציה המקבלת כפרמטר מערך של מטוסים ומחזירה כמה מטוסים כשירים יש במערך.
3. פונקציה המקבלת כפרמטר מערך של מטוסים ומחזירה האם קיימים במערך 2 מנועים בעלי מספר מנוע זהה.

פתרון:

```
public class Engine {
        private int number;
        private int volume;

        public Engine() {
             this.number = 0;
             this.volume = 0;
        }


        public Engine(int number, int volume) {
             this.number = number;
             this.volume = volume;
        }

        public int getNumber() {
             return number;
        }

        public void setNumber(int number) {
             this.number = number;
        }

        public int getVolume() {
             return volume;
        }


        public void setVolume(int volume) {
             this.volume = volume;
        }

        @Override
        public String toString() {
             return "Engine [number=" + number +
                  ", volume=" + volume + "]";
        }
}
```

```java
import java.util.Arrays;

public class Plane {
        private String manufacturer;
        private String model;
        private Engine[] engines;
        private int numOfEngines;

        public Plane() {
             this.manufacturer = "NONE";
             this.model = "NONE";
             this.engines = new Engine[4];
             this.numOfEngines = 0;
        }

        public Plane(String manufacturer, String model) {
             this.manufacturer = manufacturer;
             this.model = model;
             this.engines = new Engine[4];
             this.numOfEngines = 0;
        }

        public String getManufacturer() {
             return manufacturer;
        }

        public void setManufacturer(String manufacturer) {
             this.manufacturer = manufacturer;
        }

        public String getModel() {
             return model;
        }

        public void setModel(String model) {
             this.model = model;
        }

        public Engine[] getEngines() {
             return engines;
        }

        public int getNumOfEngines() {
             return this.numOfEngines;
        }
```

```
        @Override
        public String toString() {
             return "Plane [manufacturer=" + manufacturer +
                  ", model=" + model +
                  ", engines=" + Arrays.toString(engines) + "]";
        }

        public boolean addEngine(Engine e) {
             if (this.numOfEngines == this.engines.length) {
                  return false;
             } else if (this.numOfEngines == 0) {
                  this.engines[numOfEngines++] = e;
                  return true;
             } else if (this.engines[0].getVolume() == …
                                                 …e.getVolume()) {
                  this.engines[numOfEngines++] = e;
                  return true;
             }

             return false;
        }
}


public class Main {
        public static void gen4RandEngines(Plane p) {
             while (p.getNumOfEngines() < 4) {
                  p.addEngine(new Engine((int) (Math.random() *…
                    …1000) + 1, (int) (Math.random() * 2) + 2000));
             }
        }

        public static void addEnginesToAllPlanes(Plane[] arr) {
             for (Plane p : arr) {
                  gen4RandEngines(p);
             }
        }


        public static void displayInfo(Plane[] arr) {
             for (Plane p : arr) {
                  System.out.println(p.toString() + "\n");
             }
        }
```

```
        public static int[] getAllEngNum(Plane[] arr) {
            int[] engNumArr = new int[arr.length * 4];
            int index = 0;

            for (Plane p : arr) {
                for (Engine e : p.getEngines()) {
                    engNumArr[index++] = e.getNumber();
                }
            }

            return engNumArr;
        }

        public static boolean isSameEngineNum(Plane[] arr) {
            int[] engNumArr = getAllEngNum(arr);

            for (int i = 0; i < engNumArr.length; i++) {
                for (int j = i + 1; j < engNumArr.length; j++) {
                    if (engNumArr[i] == engNumArr[j]) {
                        return true;
                    }
                }
            }

            return false;
        }
    }
```

סוגי סיבוכיות כללית:

סיבוכיות	שם	דוגמא
O(1)	קבוע	כאשר אין תלות בגודל הקלט
O(log(n))	לוגריתמי	הקלט קטן בחצי בכל איטרציה, לדוגמא - חיפוש בינארי / כמות ספרות במספר
O(n)	לינארי	סריקה יחידה בגודל הקלט או ערך מקורב לו
O(n*log(n))	לינארי לוגריתמי	סריקה לינארית בסדר גודל של log(n) בכל איטרציה
$O(n^2)$	ריבועי	עבור כל אחד מהאברים נסרקים n אברים, לדוגמא: לולאה עד n ובתוכה לולאה עד n
O(n*m)	מכפלה	עבור כל אחד מ- n אברים מתבצעות m סריקות למשל: מעבר על מערך דו-מימדי בגודל m על n, לולאה בתוך לולאה כאשר אחת היא n פעמים והשנייה m, לולאה שרצה n פעמים וקוראת בכל פעם לפונקציה עם סיבוכיות m

חריגות - Exceptions

חריגה היא מחלקה המייצגת בעיה מסוימת בזמן ריצה. לתפיסה של חריגות בזמן ריצה יש לכתוב בלוק של try ובסיומו לבדוק באמצעות בלוקים **של catch** את החריגות האפשריות שיכולות להתרחש כתוצאה מהפעלה של הפקודות בבלוק של **try**.

דוגמא:

```
import java.util.Scanner;
import javax.swing.JOptionPane;

public class Main {
        public static void main(String[] args) {
             Scanner input = new Scanner(System.in);

             System.out.println("Enter a number: ");
             int index = input.nextInt();

             int arr[] = { 0, 1, 2, 0, 2 };

             try {
                  int result = arr[index];
                  int divide = result / arr[arr.length - 1];

                  System.out.println("Enter a number:");
                  String str = input.next();

                  int myInteger = Integer.parseInt(str);
                  System.out.println(result + ", " + divide +…
                                                  … ", " + str);
             } catch (ArrayIndexOutOfBoundsException e) {
                  JOptionPane.showMessageDialog(null, "number is…
                                                …out of index");
                  System.out.println("number is out of index");
             } catch (ArithmeticException e) {
                  JOptionPane.showMessageDialog(null, "cannot…
                                                …divide by zero");
                  System.out.println("cannot divide by zero");
             } catch (NumberFormatException e) {
                  JOptionPane.showMessageDialog(null, "make sure…
                                        …the string is an integer");
                  System.out.println("make sure the string is an…
                                                  …integer");
```

```
            } finally {
                System.out.println("done");
            }
        }
    }
```

שאלה:

א. נתונה הפעולה **one** המקבלת כפרמטר מספר שלם חיובי **num** וספרה **dig**.

```
public static int one(int num, int dig) {
    int res = 0;

    while (num > 0) {
        if (num & 10 == dig) {
            res++;
        }

        num /= 10;
    }

    return res;
}
```

1. תנו דוגמא של מספר **num > 1000** כך שתוצאת הזימון **one(num, 7)** תהיה 3.
2. מה מבצעת הפעולה **one** באופן כללי?

ב. נתונה הפעולה **two** המקבלת מערך **arr** של מספרים שלמים וחיוביים וספרה **dig**.

```
public static int two(int[] arr, int dig) {
    int res = 0;

    for (int i = 0; i < arr.length; i++) {
        res += one(arr[i], dig);
    }

    return res;
}
```

נתון מערך של מספרים שלמים וחיוביים **arr** הבא:

```
int[] arr = {24, 34783, 1245, 68, 468, 9445};
```

1. עקבו באמצעות טבלת מעקב אחר ביצוע זימון הפעולה **two(arr, 4)** ורשמו מה יהיה הפלט.
2. הסבירו מה מבצעת הפעולה **two** באופן כללי.

ג. נתונה הפעולה **three** המקבלת מערך של מספרים שלמים וחיוביים **arr**.

```
public static int three(int[] arr) {
    int res = 0;

    for (int i = 1; i < 10; i++) {
        if (two(arr, i) > two(arr, res)) {
            res = i;
        }
    }

    return res;
}
```

1. תנו דוגמה של מערך **arr** בגודל שישה תאים שעבורו תוצאת זימון הפעולה **three(arr)** תהיה 6.

2. הסבירו מה מבצעת הפעולה **three** באופן כללי.

פתרון:

א.

1.

one:						
iteration	**num**	**dig**	**res**	**num > 0**	**num % 10 == dig**	**return**
1	1777	7	0	TRUE	TRUE	
2	177	7	1	TRUE	TRUE	
3	17	7	2	TRUE	TRUE	
4	1	7	3	TRUE	FALSE	
5	0	7	3	FALSE		**3**

2. באופן כללי הפעולה בודקת כמה פעמים מופיעה הספרה **dig** במספר **num**.

ב.

1.

two:							
iteration	**arr**	**dig**	**res**	**i**	**i < arr.length**	**one(arr[i], dig)**	**return**
1	{24, 34783, 1245, 68, 468, 9445}	4	0	0	TRUE	1	
2	{24, 34783, 1245, 68, 468, 9445}	4	1	1	TRUE	1	
3	{24, 34783, 1245, 68, 468, 9445}	4	2	2	TRUE	1	
4	{24, 34783, 1245, 68, 468, 9445}	4	3	3	TRUE	0	
5	{24, 34783, 1245, 68, 468, 9445}	4	3	4	TRUE	1	
6	{24, 34783, 1245, 68, 468, 9445}	4	4	5	TRUE	2	
7	{24, 34783, 1245, 68, 468, 9445}	4	6	6	FALSE		6

2. באופן כללי הפעולה בודקת כמה פעמים מופיעה הספרה **dig** בכל המערך.

ג.

1.

three						
iteration	**arr**	**res**	**i**	**two(arr, i)**	**two(arr, res)**	**return**
1	{26, 34783, 666, 68, 468, 9445}	0	1	0	0	
2	{26, 34783, 666, 68, 468, 9445}	0	2	1	0	
3	{26, 34783, 666, 68, 468, 9445}	2	3	2	1	
4	{26, 34783, 666, 68, 468, 9445}	3	4	4	2	
5	{26, 34783, 666, 68, 468, 9445}	4	5	1	4	
6	{26, 34783, 666, 68, 468, 9445}	4	6	6	4	
7	{26, 34783, 666, 68, 468, 9445}	6	7	1	6	
8	{26, 34783, 666, 68, 468, 9445}	6	8	3	6	
9	{26, 34783, 666, 68, 468, 9445}	6	9	1	6	
10	{26, 34783, 666, 68, 468, 9445}	6				6

2. באופן כללי הפעולה בודקת איזו ספרה מ1-9 מופיעה הכי הרבה פעמים במערך.