

מבוא לחישוב 2-7015710 סמסטר א' – מועד ג' 30/4/2023

- מרצים: אלינה בנסון אופלינסקי, ארקדי גורודישקר, בעז בן משה
- משך המבחן: שתיים וחצי (2.5 שעות).
- מחברת שורות. אין שימוש בחומר עזר.
- יש להחזיר את דף המבחן בסוף המבחן.
- במבחן ארבע שאלות, כולן חובה. לבחינה זו מצורפים 4 דפי דוגמאות קוד.
- בכל שאלה ניתן לכתוב פונקציות ואו מחלקות עזר כרצונכם.
- כל עוד לא נאמר אחרת ניתן בהחלט לפתור סעיף אחד בעזרת סעיף אחר.

שאלה 1:

מספר ראשוני הוא מספר טבעי (2 ומעלה) שמתחלק בעצמו וב 1 בלבד. נגדיר שני מספרים ראשוניים (p_1, p_2) להיות "סמוכים" אם ורק אם כל המספרים בין p_1 ל p_2 הם פריקים (לא ראשוניים).

(25 נקודות) כתבו פונקציה שמקבל שני מספרים טבעיים n_1, n_2 ומחזירה את המרחק הממוצע של ראשוניים סמוכים בתחום $[n_1, n_2]$ – ניתן להניח שקיים לפחות זוג אחד של ראשוניים סמוכים בתחום. לדוגמא עבור $[10, 19]$ המרחק הממוצע בין הראשוניים הסמוכים הוא (הממוצע בין 2,2,4) ~ 2.6667 :
 $\{10, \underline{11}, 12, \underline{13}, 14, 15, 16, \underline{17}, 18, \underline{19}\}$

```
public static double aveDistPrime(int n1, int n2) {
```

שאלה 2:

בשאלה זו נתייחס לממשק של צורות (GeoShape),
כזכור, הפעלת השיטה `getClass().getSimpleName()` מחזירה מחרוזת עם שם המחלקה ממנה נוצר האובייקט.

2.1 (8 נקודות) כתבו פונקציה סטטית שמקבלת מערך של צורות `arr`, וצורה `s`, ומחזירה אמת אם ורק אם המערך מכיל לפחות צורה אחת שהיא מאותה מחלקה כמו `s`.
הדרכה: ניתן להניח שצורה אינה שווה ל `null`, אך המערך יכול להיות שווה ל `null` (ואז צריך להחזיר `false`).

```
public static boolean sameClass(GeoShape[] arr, GeoShape s) {
```

2.2 (17 נקודות) כתבו פונקציה סטטית שמקבלת מערך של צורות, ומפצלת את כלל הצורות במערך לתוך אוסף של מערכים של צורות כאשר כל מערך באוסף מכיל צורות מאותה מחלקה.
לדוגמא נניח שבמערך `s` יש שתי נקודות, שלושה משולשים ועיגול, הפונקציה תחזיר `ArrayList` שמכיל 3 מערכים (של צורות) אחד שמכיל את שתי הנקודות, שני שמכיל את שלושת המשולשים, ושלישי שמכיל את העיגול.

```
public static ArrayList<GeoShape[]> splitToClasses(GeoShape[] s) {...}
```

שאלה 3:

בשאלה זו נתייחס לעצים בינאריים כפי שמופיעים בחומר המצורף.
3.1 (5 נקודות) שרטטו דוגמא לעץ בינארי שיש לו בדיוק שלושה עלים, ששניים מהעלים הם ברמה הכי רחוקה

מהשורש.

3.2 (20 נקודות) כתבו פונקציה סטטית שמקבלת עץ בינארי ומחזירה כמה עלים בעץ הם ברמה הכי רחוקה מהשורש. משמע, בעץ בעל שורש יחיד היא תחזיר 1, בעץ ריק היא תחזיר 0, בעץ כפי שתכנתת ב 3.1 היא תחזיר 2.

```
public static int numOffFarLeafs(BinaryTree bt) {...}
```

שאלה 4:

בשאלה זו נניח שקיימת לכם המחלקה Q שמייצגת קוביית משחק **הוגנת** (בכל "זריקה" הקובייה יכולה לקבל ערך 1,2,3,4,5,6 בלבד, בהסתברות אחידה). למחלקה יש בנאי ריק והיא מממשת את הממשק Game בעל השיטות הבאות:

```
public interface Game() {  
    void roll(); // roll the dice  
    int getVal(); // return the value of the last roll.  
}
```

1. שיטה roll() שלא מקבלת פרמטרים, ולא מחזירה ערכים, למעשה "זורקת" את הקובייה (או הקוביות).
2. שיטה getVal() מחזירה את ערך של הקובייה (או הקוביות) מהזריקה האחרונה (אם היא לא נזרקה עדיין מחזירה 0). שימו לב שכל עוד לא בוצע זריקה נוספת, הערך המוחזר לא ישתנה.

4.1 (7 נקודות) השלימו את המחלקה Q2 שמייצגת זוג קוביות משחק (הוגנות) בעלת בנאי ריק ושממשת את הממשק Game

```
public class Q2 implements Game { //...  
}
```

4.2 (10 נקודות) השלימו את המחלקה Q2_Err שמייצגת זוג קוביות "מזויפות" – יש לה "עדיפות" עבור מספרים גבהים. משמע: הסיכוי לקבל סכום של 12 (6,6) יהיה גבוה יותר מהסיכוי לקבל 2 (1,1).

```
public class Q2_Err //...  
{  
}
```

4.3 (8 נקודות) הסבירו (במילים) כיצד ניתן לבדוק שהמחלקה Q2 מייצגת זוג קוביות הוגנות, והמחלקה Q2_Err אינה מייצגת זוג קוביות הוגנות.

בהצלחה!!!

חומר עזר שמצורף לבחינה

- אם לא נאמר אחרת, ניתן לעשות שימוש בחומר העזר בפתרון השאלות.
- דוגמאות הקוד: String, Sort, Junit, Point2D, GeoShape, MyListInterface, BinaryTree

```
// String: char charAt(int i), String substring(int start, int end), String[] split(String d)
// Math.random(); // returns a random double in [0,1)
/** This is a simple interface for Binary Trees as published in: */
public interface BinaryTree<T> {
    /** @return the left (sub) tree - might be null. */
    public BinaryTree<T> getLeft();
    /** @return the right (sub) tree - might be null. */
    public BinaryTree<T> getRight();
    public T getRoot(); // The root data (type T).
    public boolean isEmpty();
    /** @return the number of nodes in this tree. */
    public int size();
    /** Adds the data "a" to this tree, in a regular BT can be implemented using a random (left/right).
    In Binary Search Tree-is done using the InOrder (natural) Order. */
    public void add(T a);
    /** @return the i'th node using inorder "indexind"*/
    public T get(int i);
    /** search the binary tree for the first node that equals to t. If none returns null */
    public T find(T t);
    /** returns an in_order iterator */
    public Iterator<T> iterator();
    /** removes the first node that equals to t. If exists - returns it, else returns null */
    public T remove(T element);
}
/** Basic String Comparator – as defined in java.util*/
class StringComparator implements Comparator<String> {
    public StringComparator(){};
    public int compare(String obj1, String obj2) {
        if (obj1 == obj2) {return 0;}
        if (obj1 == null) {return -1;}
        if (obj2 == null) {return 1;}
        return obj1.compareTo(obj2);
    }
}
/** This interface represents a set of operations on list of T's. */
public interface MyListInterface<T> {
    /** Adds a String to the i'th link of the List. */
    public void addAt(T a, int i);
    /** Removes the i'th element (link) of this List. */
    public void removeElementAt(int i);
    /** Tests if 'data' is a member of this List. */
    public boolean contains(T data);
    /** Returns the i'th element in this List. */
    public T get(int i);
}
```

```

        /** Returns the number of Links in this List. */
        public int size();
    }

    /** This class represents a simple 2D Point in the plane */
    public class Point2D {
        public static final double EPS = 0.001;
        public static final Point2D ORIGIN = new Point2D(0,0);
        private double _x, _y;
        public Point2D(double a,double b) { _x=a; _y=b; } // Standard Constructor.
        public Point2D(Point2D p) { this(p.x(), p.y()); }. // Copy Constructor
        /** String Constructor: following this String structure: "-1.2,5.3" --> (-1.2,5.3) ; */
        public Point2D(String s) {
            String[] a = s.split(",");
            _x = Double.parseDouble(a[0]);
            _y = Double.parseDouble(a[1]); }
        public double x() {return _x;}
        public double y() {return _y;}
        public Point2D add(Point2D p) {
            return new Point2D(p.x()+x(),p.y()+this.y()); }
        /** Translates this point by a vector like representation of p. */
        public void move(Point2D p) { _x += p.x(); _y += p.y();}
        public String toString() {return _x+", "+_y; }
        public double distance() {return this.distance(ORIGIN); }
        /** distance(this,p2) = Math.sqrt(dx^2 + dy^2) */
        public double distance(Point2D p2) {
            double dx = this.x() - p2.x(), dy = this.y() - p2.y();
            return Math.sqrt(dx*dx+dy*dy); }
        /**return true iff: this point equals to p. */
        public boolean equals(Object p) {
            if(p==null || !(p instanceof Point2D)) {return false;}
            Point2D p2 = (Point2D)p;
            return (_x==p2._x) && (_y==p2.y()); }
        public boolean equals(Point2D p) {
            if(p==null) {return false;}
            return ((_x==p._x) && (_y==p._y)); }
        public boolean close2equals(Point2D p2, double eps) {
            return ( this.distance(p2) < eps ); }
    }

    public interface GeoShape {
        /** Computes if the point (ot) falls inside this (closed) shape. */
        public boolean contains(Point2D ot);
        /** Computes the area of this shape */
        public double area();
        /** Computes the perimeter of this shape. */
        public double perimeter();
        /** Move this shape by the vector 0,0-->vec
         * Note: this method changes the inner state of the object. */
        public void move(Point2D vec);
        public GeoShape copy(); /** computes a new (deep) copy of this GeoShape. */
    }

```

```

        public String toString(); /** This method returns an String representing this shape. */
        /** This method returns an inner point – within this GeoShape. */
        public Point2D innerPoint();
    }
    //////////// MERGE SORT ////////////
    public static void mergeSort(int[] a) {
        int len = a.length;
        double[] tmp = new double[len];
        for(int i=0;i<len;i=i+1) {tmp[i]=a[i];}
        mergeSort(tmp);
        for(int i=0;i<len;i=i+1) {a[i] = (int)tmp[i];}
    }
    public static void mergeSort(double[] a) {
        int size = a.length;
        if(size>=2) {
            int mid = size/2;
            double[] left = getSubArray(a,0,mid);
            double[] right = getSubArray(a,mid,size);
            mergeSort(left); // recursive call
            mergeSort(right); // recursive call
            double[] merge = mergeArrays(left,right);
            for(int i=0;i<merge.length;i=i+1) {a[i] = merge[i];}
        }
    }
    public static double[] getSubArray(double[] a, int min, int max) {
        double[] ans = new double[max-min];
        for(int i=min;i<max;i=i+1) {ans[i-min] = a[i];}
        return ans;
    }
    /** This function merges two sorted arrays into a single sorted array. */
    public static double[] mergeArrays(double arr1[], double arr2[]) {
        double[] res = new double[arr1.length + arr2.length];
        int i=0, j=0;
        while ( i < arr1.length && j < arr2.length ) {
            if (arr1[i] <= arr2[j]) { res[i+j] = arr1[i]; i=i+1;}
            else {res[i+j] = arr2[j]; j=j+1;}
        }
        while ( i < arr1.length) {res[i+j] = arr1[i++];}
        while ( j < arr2.length) {res[i+j] = arr2[j++];}
        return res;
    }
    public static int[] randomIntArray(int size, int range){
        int[]arr = new int[size];
        ++range;
        for(int i=0; i<size; i=i+1) {arr[i] = (int)(Math.random()*range);}
        return arr;
    }
    public static boolean isSortedAscending(int[] arr){
        for (int i = 1; i < arr.length; i++) {
            if (arr[i-1] > arr[i]) {return false; }
        }
    }

```

```

        return true;
    }
}

```

```

class SortTest {
    public static final int K = 1000, M = K*K;
    public static int[] arrK = null, arrM = null;
    @BeforeEach
    void setUp() {
        arrK = randomIntArray(K, K);
        arrM = randomIntArray(M, M);
    }
    @Test
    void testMergeSort() {
        int[] a1 = {3,1,2,1,42};
        mergeSort(a1);
        boolean isSorted = isSortedAscending(a1);
        assertTrue(isSorted);
    }
    @Test
    void testInsertionSort() {
        int[] arr = {5,1,2,0,9};
        insertionSort(arr);
        if(MyArrayLibrary.isSortedAscending(arr)!=true) {
            fail("arr should be sorted");
        }
    }
}
////////// Performance Testing //////////
@Test
void testMergeSort1() {
    long start = System.currentTimeMillis();
    mergeSort(arrM);
    long end = System.currentTimeMillis();
    double dt_sec = (end-start)/1000.0;
    boolean isSorted = isSortedAscending(arrM);
    System.out.println("Recursive Merge sort time = "+dt_sec+" secs, is sorted? "+
isSorted);
    assertTrue(isSorted);
    assertTrue(dt_sec<1.0);
}
@Test
@Timeout(value = 1000, unit = TimeUnit.MILLISECONDS)
void testMergeSort2() {
    mergeSort(arrDoubleM);
    boolean isSorted = isSortedAscending(arrDoubleM);
    assertTrue(isSorted);
}
}

```