<https://docs.google.com/document/d/1cYyDE4zAVVzS-PHjYySU5_XJJvP5Q_4gIC0bYrwmj6A/edit?usp=sharing>

**מבוא לחישוב 2-7015710 סמסטר א' – מועד א׳ 1.2.2023**

**הצעת פתרון**

·   מרצים: אלינה בננסון אופלינסקי, ארקדי גורודישקר, בעז בן משה

·   משך המבחן: שעתיים 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:**

נתון מערך של מספרים ממשיים (double), ניתן להניח שמערך תקין (לא null, ולא ריק).

1.1 (13 נקודות) כתבו פונקציה אשר מערבבת את המערך באופן אחיד, משמע שההסתברות של כל איבר במערך להיות בכל אינדקס במערך היא אחידה. **הדרכה**: בשאלה זו עליכם לכתוב את הפונקציה בעצמכם, ללא שימוש פונקציות ערבוב מובנות ב java. ניתן להניח שכל האיברים במערך שונים זה מזה.

public static void shuffle(double[] arr) //{…}

**תשובה1.1** :

**// As in https://github.com/benmoshe/Intro2CS/blob/main/src/week4/ShuffleAndSort.java**

**// take as input an array of doubles and rearrange them in a random order**

**public static void shuffle (double[] arr) {**

**int n = arr.length;**

**for (int i = 0; i < n; i++) {**

**int r = i+ (int) (Math.random() \* (n-i));   // between i and n-1**

**exch(arr, i, r); // double t=arr[i];arr[i]=arr[j];arr[j]=t;**

**}**

**}**

**public static void exch(double[] a, int i, int j) {double swap = a[i]; a[i] = a[j]; a[j] = swap; }**

1.2 (12 נקודות) כתבו פונקציית בדיקה (JUnit) אשר בודקת שהפעלת פונקציית הערבוב על מערך של 100 איברים אכן מעורבב באופן אקראי (בהסתברות אחידה).

public void testSuffle() // {…}

הדרכה: אם נשווה בין המערך המקורי והמערך לאחר ערבוב ונמצא שבאותם אינדקסים במקרים רבים (נניח מעל 50%) יש אותו הערך, נגדיר שהפונקציה אינה מערבבת באופן אקראי.

**תשובה 1.2:**

**@Test**

**void testShuffle() {**

**int times = 50, size = 100;**

**for(int i =0; i<times; i=i+1) {**

**double[] arr = init(size);**

**double[] cp = copy(arr);**

**shuffle(cp);**

**double same =  sameCount(arr,cp) / size;**

**assertTrue(ratio<0.5); }}**

**////////////////////////////////////////////**

**private static double[] init(int size) {**

**double[] s = new double[size];**

**for(int i=0;i<size;i=i+1) {s[i] = Math.random();}**

**return s;**

**}**

**private static double[] copy(double[] arr) {**

**int size = arr.length;**

**double[] ans = new double[size];**

**for(int i=0;i<size;i=i+1) {ans[i] = arr[i];}**

**return ans;**

**}**

**private static double sameCounter(double[] a, double[] b) {**

**double ans = 0.0;**

**for(int i=0;i<size;i=i+1) {**

**if(a[i] == b[i]) {ans++;}**

**return ans;**

**}**

**// for a better tester (not required in the exam) see:**

**// https://github.com/benmoshe/Intro2CS/blob/main/src/week4/ShuffleAndSortTest.java**

**שאלה 2:**

נגדיר ששתי צורות (GeoShape) **חותכות** זו את זו אם הן אינן **זרות** ואינן **מכילות** אחת את השנייה. באופן יותר פורמאלי שתי צורות  s1,s2**נחתכות** אם ורק אם קיימת נקודה p שמוכלת בשתיהן, נקודה p1 שמוכלת **רק** ב s1 ונקודה p2 שמוכלת **רק** ב s2. הניחו שקיימת לכם פונקציה שבודקת אם שתי צורות חותכות אחת את השנייה:

public static boolean intersect(GeoShape s1, GeoShape s2);

**הערה חשובה**: בשאלה זו ניתן להניח ״קלט תקין״ – משמע הפרמטרים שמועברים לפונקציות שונים מ null.

2.1 (5 נקודות) כתבו פונקציה סטטית שמחזירה אמת אם ורק אם הצורה (s1) **מכילה** את הצורה שניה (s2):

public static boolean contains(GeoShape s1, GeoShape s2);

**תשובה 2.1:**

public static boolean contains(GeoShape s1, GeoShape s2) {

**boolean inter = intersect(s1, s2);**

**boolean cont = s1.contains(s2.innerPoint());**

**return cont && !inter && s1.area() > s2.area();**

**}**

2.2 (5 נקודות) כתבו פונקציה סטטית שמחזירה אמת אם ורק אם הצורות (s1) ו (s2) **זרות** זו לזו:

public static boolean disjoint(GeoShape s1, GeoShape s2);

**תשובה 2.2:**

public static boolean disjoint(GeoShape s1, GeoShape s2) {

**boolean inter = intersect(s1, s2);**

**boolean cont = contains(s1, s2);**

**return !cont && !inter;**

**}**

**הניחו שקיימת** לכם מחלקה בשם ShapeWithHoles אשר **מממשת** את הממשק GeoShape ומייצגת צורה עם חורים. ראו דוגמא של שתי צורות עם חורים: פוליגון עם שני חורים ומעגל עם 4 חורים.

**הגדרה**: החורים הם צורות **זרות** ש**מוכלות** במלואן בצורה החיצונית.

2.3 (15 נקודות) השלימו את השיטה **addHole** אשר מקבלת צורה שמייצגת ״חור״ ומוסיפה אותה לאוסף החורים \_holes אם ורק אם הצורה מייצגת ״חור״ לפי **ההגדרה** מעלה. אם הצורה הוספה יוחזר ״אמת״ אחרת ״שקר״.

public class ShapeWithHoles implements **GeoShape**{

   private GeoShape \_outer;

   private ArrayList<GeoShape> \_holes;

   public ShapeWithHoles(GeoShape out) {\_outer = out.copy();}

   public boolean **addHole**(GeoShape hole) { // **Add you code here!!!** }

**תשובה 2.3:**

public boolean **addHole**(GeoShape hole) {

**boolean ans = false;**

**if(hole!=null && contains(this, hole)) {**

**boolean ok = true;**

**for(int i=0;ok && i<\_holes.size();i=i+1) {**

**ok = disjoint(hole, <\_holes.get(i));**

**}**

**ans = ok;**

**if(ok) {\_holes.add(hole.copy());**

**}**

**return ans;**

**}**

**שאלה 3:**

3.1 (12 נקודות) כתבו פונקציה סטטית שמקבלת עץ בינארי ומחזירה את העומק המינימאלי שבו קיים עלה בעץ (עבור עץ ריק תחזיר 1- ועבור עץ בעל שורש בלבד תחזיר 0).

public static int minLevelLeaf(BinaryTree bt) //{…}

**תשובה 3.1**

**public static int minLevelLeaf(BinaryTree bt) {return minLevelLeaf(bt, 0); {**

**public static int minLevelLeaf(BinaryTree bt, int d) {**

**int ans = -1;**

**if(bt!=null && !bt.isEmpty()) {**

**if(bt.size()==1) {ans = d;} // if isLeaf**

**else {**

**int l = minLevelLeaf(bt.getLeft(), d+1);**

**int r = minLevelLeaf(bt.getRight(), d+1);**

**if(l!=-1) {ans = l;}**

**if(r!=-1} {**

**if(l!=-1) {ans = Math.min(l,r);}**

**else{ ans = r;}**

**}**

**} // else**

**} // if**

**return ans;}**

 3.2 (13 נקודות) כתבו פונקציה סטטית שמקבלת עץ בינארי ומחזירה מערך של כמות העלים שיש בכל רמה בעץ (עבור עץ ריק תחזיר null ועבור עץ בעל שורש בלבד תחזיר את המערך {1}).

public static int[] numberOfLeafsByLevel(BinaryTree bt) //{…}

**תשובה 3.2**

**public static int[] numberOfLeafsByLevel(BinaryTree bt) {**

**int[] ans = null;**

**if(bt!=null && bt.size()>0) {**

**ans = new int[bt.height()];**

**numberOfLeafsByLevel(bt, ans, 0);**

**return ans;**

**}**

**private static void numberOfLeafsByLevel(BinaryTree bt, int[] ans, int d) {**

**if(bt!=null && bt.size()>0) {**

**if(bt.size() == 1) {ans[d]++;}**

**else {**

**numberOfLeafsByLevel(bt.getLeft(), ans, d+1);**

**numberOfLeafsByLevel(bt.getRight(), ans, d+1);**

**}**

**}**

**}**

**///////////////////////////////////////**

**private static int height(BinaryTree bt) {**

**int ans = -1;**

**if(bt!=null && !bt.isEmpty()) {**

**ans = 1 + Math.max(height(bt.getLeft()), height(bt.getRight()));**

**}**

**return ans;**

**}**

**שאלה 4:**

בשאלה זו נתייחס למערכים דו מימדיים מלבניים מעל השלמים, בשביל הפשטות נניח שהערך 1 מוגדר  כ״לבן״ והערך 0 מוגדר כ״שחור״, והערך 2 מגודר כ״אפור״. עבור מערך דו מימדי שכולל צבעים של “שחור” ו”לבן” בלבד, נגדיר שקיים מסלול בין תא x1,y1 לתא x2,y2 במערך אם קיים מסלול רציף של תאים (בין ההתחלה לסוף) שאינו עובר באף תא שחור. אורך מסלול מוגדר להיות כמות התאים הרציפים (למעלה, למטה, ימינה ושמאלה). כאשר אם לא קיים מסלול כזה נגדיר את אורכו להיות (ערך מקסימלי - MAX).

״סטודנט״ מימש אלגוריתם לחישוב אורך המסלול הקצר ביותר בין שתי נקודות (ראו מטה), ״הסטודנט בדק את הקוד״ על דוגמאות קטנות והקוד עבד היטב.

public static final int BLACK=0, WHITE=1, GRAY=2, MAX = 1000\*1000\*1000;

public static int shortestPath(int[][] mat, int x1, int y1, int x2, int y2) {

         if(!isValid(mat, x1,y1) | ! isValid(mat, x2,y2)) {return MAX}

         if(x1==x2 & y1==y2) {return 0;}

         // recursive calls

         int l = shortestPath(mat, x1-1, y1, x2, y2);

         int r = shortestPath(mat, x1+1, y1, x2, y2);

int u = shortestPath(mat, x1, y1+1, x2, y2);

int d = shortestPath(mat, x1, y1-1, x2, y2);

int min = Math.min(Math.min(l, r), Math.min(u,d));

if(min<MAX) {min=min+1;}

return min;

}

public static boolean isValid(int[][] mat, int x, int y) {

         boolean  ans = x1>=0 && y1>=0 && x1<mat.length && y1<mat[0].length;

         if(ans) {ans = mat[x][y]==WHITE;}

return ans;

}

4.1 (5 נקודות) הסבירו (בעברית) מדוע הקוד לא יעיל (3-5 שורות).

**תשובה 4.1:**

**בקוד המצורף, כל משבצת לבנה למעשה קוראת ל 4 משבצות סמוכות (למעלה, ימינה, שמאלה, ולמטה). נשים לב שהמשבצות ״אינן מסומנות״ לפיכך יהיו קריאות ״חזרה״ למשבצות שנקראו כבר. לדוגמא, נניח שהתחלנו במשבצת ״לבנה״ שיש לה 4 שכנים לבנים, לאחר שתי איטרציות נקבל 4 קריאות רקורסיביות מהמשבצת המקורית. משמע כמות הקריאות הרקורסיביות תלויה אקספוננציאלית באורך המסלול הקצר ביותר.**

4.2 (5 נקודות) הוסיפו שורת קוד (מעל השורה // recursive calls) שתייעל את הקוד.

**תשובה 4.2**: **נוסיף את השורה:**

**mat[x1][y1] = GRAY;**

4.3 (5 נקודות) הסבירו כיצד השינוי מייעל את הקוד, וכיצד ניתן לבדוק השיפור בקוד (3-5 שורות):

**תשובה 4.3:**

**ברגע שנסמן משבצת ״לבנה״ בצבע ״אפור״, המשבצת הזו לא תהיה ״חוקית״ ולפיכך לא תיקרא בפעם הבאה, אלא רק פעם אחת במהלך החישוב. בצורה הזו כמות הקריאות הרקורסיביות תהיה מוגבלת בגודל המטריצה (לכל היותר).**

4.4 (10 נקודות) ממשו שיטה שמקבלת שתי נקודות (p1, p2 התחלה וסוף) ונקודה נוספת (q) ומחזירה אמת אם ורק אם קיים מסלול קצר ביותר (מינימלי) בין p1 ל p2 שעובר דרך q.  אפשר להניח שקיים מסלול בין p1 ל p2.

public static boolean isOnTheShortestPath(int[][] mat, int x1, int y1, int x2, int y2, int xq, int yq)

**תשובה 4.4:**

public static boolean isOnTheShortestPath(int[][] mat, int x1, int y1, int x2, int y2, int xq, int yq){

**int dist12 = shortestPath(mat, x1, y1, x2, y2);**

**if(dist12==MAX) {return false;}  // not required!**

**int dist1q = shortestPath(mat, x1, y1, xq, yq);**

**int dist2q = shortestPath(mat, x2, y2, xq, yq);**

**return dist12 == dist1q + dist2q;**

**}**

**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];

         or(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);

         }

<https://docs.google.com/document/d/1JuN7W_sGlde2wCLTQkRIdF2k4l0v8JTToJUhWmGq9dY/edit>

**מבוא לחישוב 2-7015710 סמסטר א' – מועד ב׳ 1.3.2023**

**הצעת פתרון**

·   מרצים: אלינה בננסון אופלינסקי, ארקדי גורודישקר, בעז בן משה

·   משך המבחן: שעתיים 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:**

בשאלה זו נתייחס לממשק שמייצג פונקציה (מתמטית) רציפה מהממשיים לממשיים.

Public interface ContinenceFunction {

         public double f(double x); // f(x);

}

1.1 (10 נקודות)  הניחו שקיימת לכם פונקציה רציפה (ContinenceFunction), ערך מינימום x1 וערך מקסימום x2. ניתן לשערך את אורך הפונקציה בין f(x1) ל f(x2) ע״י כך שנחלק את התחום [x1,x2] ל n קטעים שווים ונחשב את אורך המסלול (כפי שמוצג בשרטוט מעלה, שכולל פוליגון שאורכו משוערך ע״י מסלול של 7 נקודות). השלימו את הפונקציה לשיערוך אורך הפונקציה בתחום  [x1,x2].

**public static double length(ContinenceFunction f, double x1, double x2, int n)** {

**תשובה:**

**double ans = 0;**

**if(f!=null && n>0 && x1<x2) {**

**double delta = (x2-x1)/n, x=x1;**

**for(int i=0;i<n;++) {**

**double y = f.f(x), xx = x+delta, yy = f.f(xx);**

**Point2D p = new Point2D(x,y), pp = = new Point2D(xx,yy);**

**ans += p.distance(pp);**

**x += delta;**

**}**

**}**

**return ans**

**}**

1.2 (5 נקודות) השלימו את המחלקה MyFunc, שמייצגת את הפונקציה הרציפה: **f(x) = a\*sin(x\*b + c),** עבור המשתנים הממשיים a, b, c. כזכור במחלקה Math, קיימת הפונקציה sin(double angRad) double:

public class MyFunc implements ContinenceFunction {

         private double \_a, \_b, \_c;

         public MyFunc (double a, double b, double c) {\_a=a; \_b=b; \_c=c;}

         // **add your code here**

**תשובה:**

**@Override**

**public double f(double x) {return \_a\*Math.sin(\_b\*x+\_c);}**

}

1.3 (10 נקודות) הניחו שיש לכם פונקציה רציפה (ContinenceFunction), שני ערכים x1, x2 (כך ש, x1<x2, f(x1)\*f(x2)<=0), וערך אפסילון גדול מאפס. השלימו את הפונקציה שמחשבת ערך x (x1<=x<=x2) כך ש |f(x)|<=eps

**public static double root(ContinenceFunction f, double x1, double x2, double eps)** {

**תשובה:**

**double x0 = (x1+x2)/2;**

**double fx0 = f.f(x0);**

**while(Math.abs(fx0)>=eps) {**

**if(f.f(x1)\*f.f(x0)>0) {x1=x0;}**

**else {x2=x0;}**

**x0 = (x1+x2)/2;**

**fx0 = f.f(x0);**

**}**

**return x0;**

**}**

**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 שתי צורות s1,s2, ומחזירה אמת אם ורק אם אף אחת מהן אינה שווה ל null, וגם הן מאותה מחלקה.

**public static boolean sameClass(GeoShape s1, GeoShape s2)**  {

**תשובה**:

**boolean ans = false;**

**if(s1!=null & s2!=null) {**

**String st1 = s1.getClass().getSimpleName();**

**String st2 = s2.getClass().getSimpleName();**

**ans = st1.euals(st2);**

**}**

**return ans;**

**}**

2.2 (17 נקודות) כתבו פונקציה סטטית שמקבלת מערך של צורות ומחזירה את **מספר** המחלקות **השונות** אליהן שייכות הצרות במערך.

**public static int numOfClasses(GeoShape[] s)** {

**תשובה:**

**ArrayList<String> set = new ArrayList<String>();**

**for(int i=0;s!=null && i<s.length;i=i+1) {**

**String st = s[i].getClass().getSimpleName();**

**If(!set.contains(st)) {**

**set.add(st);**

**}**

**}**

**return set.size();**

**}**

**שאלה 3:**

בשאלה זו נתייחס לעצים בינאריים כפי שמופיעים בחומר המצורף. נגדיר קודקוד בגרף להיות ״ניתן להרחבה״  אם לפחות אחד מבניו הוא null.

3.1 (12 נקודות) כתבו פונקציה סטטית שמקבלת עץ בינארי ומחזירה את העומק המינימאלי שבו קיים קודקוד  ״ניתן להרחבה״ (עבור עץ ריק תחזיר 1- ועבור עץ בעל שורש בלבד תחזיר 0, כך גם אם לשורש יש רק בן יחיד).

**public static int minLevel(BinaryTree bt) {**

**תשובה:**

**if(bt==null || bt.isEmpty()) {return -1;}**

**int l = minLevel(bt.getLeft());**

**int r = minLevel(bt.getRight());**

**return 1+ Math.min(l,r);**

**}**

3.2 (13 נקודות) כתבו פונקציה סטטית שמקבלת עץ בינארי ואובייקט c מסוג T, ומוסיפה את  c למקום הפנוי הקרוב ביותר לשורש.

הדרכה: ניתן להניח שהעץ אינו null. אם קיימים מספר מקומות באותו מרחק מהשורש – על הפונקציה להוסיף במקום השמאלי ביותר.

הדרכה: בשאלה זו ניתן להניח שלעץ בינארי קיימות גם השיטות של setLeft(BinaryTree<T> b), setRight(BinaryTree<T> b)

**public static void addClosest(BinaryTree<T> bt, T c) {**

**תשובה:**

**if(bt.isEmpty) {bt.add(c);}**

**else {**

**if(bt.getLeft() == null || bt.getRight()==null) {**

**if(bt.getLeft()==null) {bt.addLeft(bt, c);}**

**else  {bt.addRight(bt, c);}**

**}**

**else {**

**int l = minLevel(bt.getLeft());**

**int r = minLevel(bt.getRight());**

**if(l<=r) { addClosest(bt.getLeft(), c);}**

**else { addClosest(bt.getRight(), c);}**

**}**

**}**

**}**

**private static void addLeft(BinaryTree<T> bt, T c, ) {**

**BinaryTree<T>  n = new BinaryTree<T>();**

**n.add(c);**

**bt.setLeft(n);**

**}**

**private static void addRightBinaryTree<T> bt, T c) {**

**BinaryTree<T>  n = new BinaryTree<T>();**

**n.add(c);**

**bt.setRight(n);**

**}**

**////////////////////////// Alternative solution: /////////////////////////**

**public static void addClosest(BinaryTree<T> bt, T c) {**

**if(bt.isEmpty) {bt.add(c);}**

**boolean added = addHere(bt, c);**

**if(!added) {**

**int l = minLevel(bt.getLeft());**

**int r = minLevel(bt.getRight());**

**if(l<=r) { addClosest(bt.getLeft(), c);}**

**else { addClosest(bt.getRight(), c);}**

**}**

**}**

**}**

**private static boolean addhere(BinaryTree<T> bt, T c){**

**boolean ans = false;**

**BinaryTree<T>  n = new BinaryTree<T>();**

**n.add(c);**

**if(bt.getLeft() == null) {**

**bt.setLeft(n);**

**ans = true;**

**}**

**if(!ans && bt.getLeft() == null) {**

**bt.setLeft(n);**

**ans = true;}**

**}**

**return ans;**

**}**

**שאלה 4:**

הניחו שקיימת לכם מחלקה של מצולע (Polygon) שמממשת את GeoShape, בעלת בנאי ריק, שיטה  add(Point2D p) שמוסיפה קודקוד לפוליגון. ושיטה getPoints שמחזירה מערך עם כל הקודקודים (בסדר שהוספתם).

נגדיר **מצולע משוכלל** (Regular Polygon) להיות מצולע פשוט (פוליגון שלא חותך את עצמו) שכל צלעותיו זהות, כל קודקודיו יושבים על מעגל חוסם יחיד ומרכזו הוא ממוצע הקודקודים, ראו שרטוט מטה.

דוגמאות למצולעים משוכללים, בעלי 3-10 צלעות, מקור:<https://en.wikipedia.org/wiki/Regular_polygon>

4.1  (10 נקודות) הסבירו בקצרה (בעברית או באנגלית) כיצד ניתן לבדוק האם מצולע ((Polygon הוא למעשה גם מצולע משוכלל. (בסעיף זה בלבד אין צורך לכתוב קוד).

תשובה:

**נכתוב פונקציה שמקבלת פוליגון ומחזירה אמת אם הוא פוליגון משוכלל, באופן הבא:**

**1.** **אם הפוליגון הוא null, נחזיר אמת  או שקר – שכן לא הוגדר מקרה בסיס (שתי האופציות תקינות)**

**2.** **אחרת נקבל מערך )נניח s) של כל קודקודי הפוליגון (בעזרת השיטה getPoints).**

**3.** **נחשב את נקודת המרכז p\_cen של s (הממוצע על ה x, והממוצע על ה  y).**

**4.** **נבדוק האם המרחק מכל קודקוד ב s לשכן (הבא שלו) הוא זהה על כדי eps, אם לא נחזיר שקר.**

**5.** **נבדוק האם המרחק מכל קודקוד ב s לנקודת המרכז הוא זהה עד כדי eps, אם לא נחזיר שקר.**

**6.** **נחזיר אמת.**

4.2  (15 נקודות) ממשו את המחלקה **RegularPolygon** שמייצגת מצולע משוכלל, מממשת את הממשק GeoShape, בעלת בנאי שמקבל נקודת מרכז, רדיוס וכמות קודקודים, ומייצרת מצולע משוכלל.

**תשובה:**

**public class RegularPolygon extends Polygon implements GeoShape{**

**public RegularPolygon(Point2D cen, double rad, int nodes) {**

**super();**

**double da = Math.PI\*2/nodes;**

**for(int i=0; i<nodes; i++) {**

**double dx = rad\*Math.cos(i\*da);**

**double dy = rad\*Math.sin(i\*da);**

**Point2D p = new Point2D(dx+cen.x(), dy+cen.y());**

**super.add(p);**

**}**

**}**

**@Override**

**public void add(Point2D p) {;} // do nothing in order to avoid adding**

**//arbitrary points - not required for the exam.**

**}**

**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];

         or(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);

         }