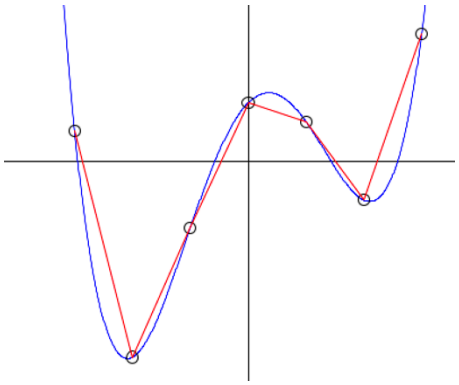
מבוא לחישוב 2-7015710 סמסטר א' – מועד ב׳ 1.3.2023



- 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 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) {…}

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 double sin(double angRad):

```
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…
}
```

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) {…}

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) {…}

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

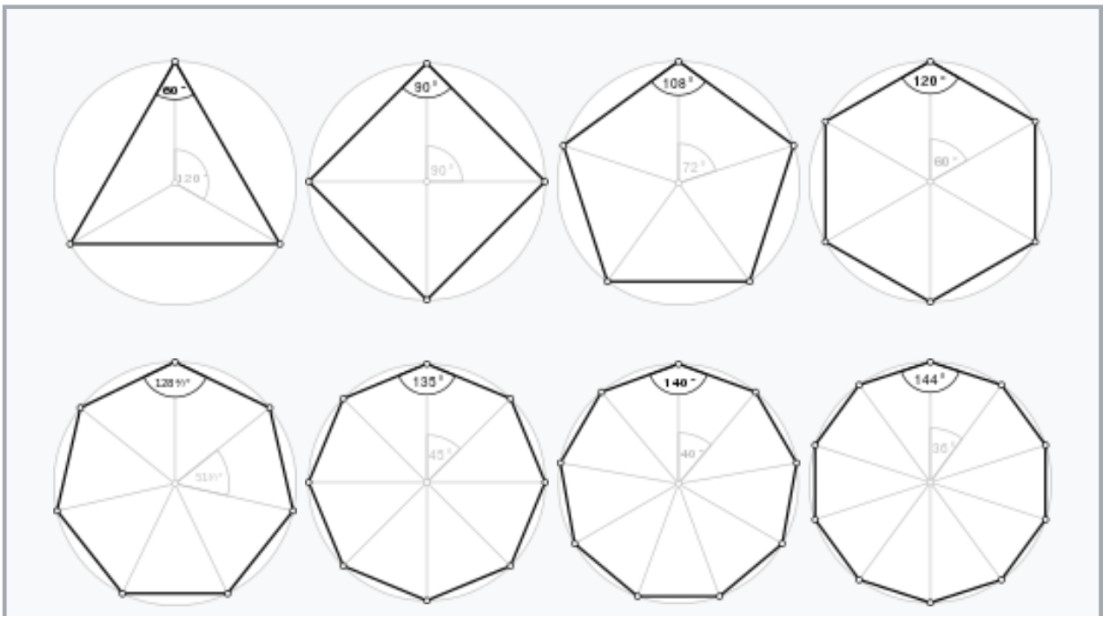
public static int numOfClasses(GeoShape[] s) {…}

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) {...}

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

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).
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.

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);
	}
```