מבוא לחישוב 2-7015710 סמסטר א' – הנחיות לבחינה + חומר עזר

- מרצים: אסף חוגי, נתן דילברי , בעז בן משה.
- משך הבחינה: שעתיים וחצי (2.5 שעות).
- שאלון הבחינה כולל 4 שאלות חובה (עמ׳ 1-2) ונספח עם דוגמאות קוד (עמ׳ 3-7) לשימושכם.
- חומר מותר: שאלון הבחינה ומחברת שורות בלבד. חל איסור על שימוש בכל חומר אחר.
- בסיום הבחינה יש למסור את השאלון ומחברת הבחינה.
- במסגרת פתרון שאלה/סעיף ניתן לכתוב פונקציות ומחלקות עזר כרצונכם.
- תשובות מסורבלות או ארוכות שלא לצורך לא יזכו בניקוד מלא.
- כל עוד לא נאמר אחרת, ניתן בהחלט לפתור סעיף אחד בעזרת סעיף אחר.
- הקפידו על כתב-יד ברור ככל הניתן.
- יש לסמן טיוטה (מלל שלא לבדיקה) במחברת הבחינה במפורש באמצעות ״טיוטה״ מעל החלק הרלוונטי (או ע״י סימן X על המלל שאינו נדרש לבדיקה).

בהצלחה!!!

נספח קוד ודוגמאות

- אם לא נאמר אחרת, ניתן לעשות שימוש בקוד שמופיע בנספח בפתרון השאלות.
- דוגמאות הקוד: String, Sort, Junit, Point2D, GeoShape, MyCollectionInterface,

```
// Math.random(); // returns a random double in [0,1)

public class StringFunctions {  // This is a very simple “main” that uses String and ArrayList
	public static void main(String[] a) {
		ArrayList<String> arr = new ArrayList<String>();
		String s = "12345", s2 = “12321”;
		arr.add(s);
		arr.add(s2);
		arr.add(s2.subString(1,4)); // “232”
		if(!arr.contains("232")) {arr.add(s);}
		for(int i=0;i<arr.size();i++) {
			boolean isSim = isSimetric(arr.get(i));
			System.out.println("arr[" +i+ "] "+arr.get(i)+" isSimetric: " +isSim);
		}
		while(!arr.isEmpty()) {
			s = arr.remove(0);
			System.out.println("rev("+s+")=" +reverse(s));
		}
		String words = "these are few words ...";
		String[] ww = words.split(" ");
		for(String w:ww) {System.out.println(w);}
	}

	public static boolean isSimetric(String s) {
		boolean ans = false;
		String t = reverse(s);
		ans = t.equals(s);
		return ans;
	}

	public static String reverse(String s) {
		String ans = "";
		for(int i=s.length()-1; i>=0;i=i-1) {
			ans=ans+s.charAt(i);
		}
		return ans;
	}
}


/** 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 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) {
            if(s==null | s.split(“,”).length <2) {
                        throw new Runtime exception(“ERR: wrong format should be “1.1, -2.2”}
            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 );
    }
}
```

```
/** This interface represents a set of operations on an ordered collection of T’s. */
public interface MyCollectionInterface<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();
}

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. */
    @ Override
    public String toString();   /** This method returns a String representing this shape. */
    @ Override
    /** Returns true IFF t is not null and is logically the same as this object.
    public boolean equals(Object t);
}


/////////// 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("Merge sort dt: "+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);
    }
```