מבוא לחישוב 2-7015710 סמסטר א' – מועד א׳ + הצעת פתרון

23-2-2025

- מרצים: אסף חוגי, בעז בן משה.
- משך הבחינה: שעתיים וחצי (2.5 שעות).
- שאלון הבחינה כולל 4 שאלות חובה (עמ׳ 2-3) ונספח עם דוגמאות קוד (עמ׳ 4-8) לשימושכם. 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!!!

שאלה 1:

1.1 (20 נקודות) כתבו פונקציה שמקבלת מערך של מספרים שלמים ומחזירה מערך חדש שמייצג את **הסדרה החשבונית הארוכה ביותר** מכל תת הרצפים במערך arr.

public static int[] maxS(int[] arr) {…}

הדרכה:

- סדרה חשבונית היא רצף מספרים ש**ההפרש** בין כן זוג איברים **סמוכים** הוא **קבוע**!
- ניתן להניח שהמערך arr מכיל לפחות שני מספרים.
- אם יש יותר מסדרה חשבונית מקסימאלית אחת - יש להחזיר את הראשונה.

1.2 (5 נקודות) כתבו פונקציית בדיקה (JUnit) שבודקת את הנכונות של הפונקציה maxS (שכתבתם בסעיף א') עבור המערכים {0,0,0,0}, {5,1,3,5,8}

public maxSTest() {…}

```
    public static int[] maxS(int[] arr) {
        if(arr==null || arr.length<2) {throw new RuntimeException("Wrong input: should
be an array of length >= 2");}
        int min=0, max =2, len = arr.length;
        for(int s=0;s<len-2;s++) {
            for(int e=s+2;e<=len;e++) {
                if(isSidra(arr,s,e)) {
                    if(e-s>max-min) {
                        min = s;max=e;
                    }
                }
            }
        }
        int[] ans = new int[max-min];
        for(int i=min;i<max;i++) {ans[i-min]=arr[i];}
        return ans;
    }
    private static boolean  isSidra(int[] arr, int s, int e) {
        boolean ans = true;
        int d = arr[s+1] - arr[s];
        for(int i=s+1;i<arr.length-1 & i<e-1 & ans;i++) {
            int d1 = arr[i+1]-arr[i];
            if(d1!=d) {ans = false;}
         }
        return ans;
    }

@Test
public void maxSTest() {
    int[] a1 = {5,1,3,5,8};
    int[] a2 = {0,0,0,0};
    int[] r1 = MA.maxS(a1);
    int[] r2 = MA.maxS(a2);
    assertEquals(3, r1.length);
```

```
    assertEquals(4, r2.length);
}
```

שאלה 2:

בשאלה זו נעסוק במחלקות (בפרט בשמות המחלקות) של אובייקטים. כזכור לכל מחלקה יש שיטה getClass().getSimpleName() שמחזירה מחרוזת עם שם המחלקה אליו שייך האובייקט (הטיפוס המוצבע בזמן אמת)

2.1 (10 נקודות) כתבו פונקציה סטטית שמקבלת מערך של צורות (GeoShape) ומחרוזת (cls) שמייצגת שם מחלקה. הפונקציה מחזירה אוסף של צרות (ArrayList<GeoShape>) שכוללת **עותק עמוק** של כל הצורות במערך שהן מהסוג של cls.

```
public static ArrayList<GeoShape> filter(GeoShape[] arr, String cls);
```

2.2 (15 נקודות) כתבו פונקציה שמקבלת מערך של צורות (GeoShape) ומחזירה את **קבוצת** שמות המחלקות אליהן שייכות הצורות במערך.

```
public static ArrayList<String> setOfClasses(GeoShape[] arr);
```

```
public static ArrayList<GeoShape> filter(GeoShape[] arr, String cls) {
   ArrayList<GeoShape> ans = new ArrayList<GeoShape>();
   for(int i=0;arr!=null && i<arr.length;i++) {
      if(arr[i]!=null && arr[i].getClass().getSimpleName().equals(cls)) {
         ans.add(arr[i].copy());
      }
   }
   return ans;
}
private static ArrayList<String> setOfClasses(GeoShape[] arr) {
   ArrayList<String> ans = new ArrayList<String>();
   for(int i=0;arr!=null && i<arr.length;i++) {
      if(arr[i]!=null) {
         String cls = arr[i].getClass().getSimpleName();
         if(!ans.contains(cls)) {
            ans.add(cls);
         }
      }
   }
   return ans;
}
```

שאלה 3:

בשאלה זו נתייחס לממשק של עצים בינאריים כפי שמופיע בדוגמאות הקוד שמצורפות לבחינה.

3.1 (5 נקודות) כתבו פונקציה סטטית שמקבל עץ בינארי ומחשבת את הגובה שלו. כזכור גובה של עץ בינארי הוא 1- אם העץ הוא null או שהוא העץ הריק, הוא 0 אם העץ כולל שורש בלבד, אחרת גובה העץ הוא 1+ המקסימום בין הגובה של תת העץ השמאלי לבין הגובה של תת העץ הימני.

```
public static <T> int height(BinaryTree<T> bt) {…}
```

3.2 (15 נקודות) הניחו שקיימת לכם מחלקה BTD שמייצגת עץ בינארי מעל הממשיים.
למחלקה קיים בנאי ריק שיוצר עץ בינארי ריק.

```
public class BTD implements BinaryTree<Double> { // …}
```

כתבו פונקציה סטטית שמקבלת מערך של מספרים ממשיים ומחזירה עץ בינארי בעל גובה מינימאלי שמכיל את כל המספרים במערך (אין חשיבות לסדר).
הדרכה: שימו לב שאתם יכולים לעשות שימוש בשיטות:

```
public void setLeft(BinaryTree<T> l); // sets the left (subtree) to l
public void setRight(BinaryTree<T> r); // sets the right (subtree) to r
```

```
public static BTD create(double[] arr) {…}
```

3.3 (5 נקודות) כתבו פונקציית בדיקה שמייצרת מערך של 100 איברים ממשיים אקראיים (בין [0,1)), מייצרת ממנו עץ בינארי בעל גובה מינימאלי ובודקת שאכן הוא בעל גובה מינימאלי, הקפידו לציין בפירוש מהו הגובה המינימאלי של עץ בינארי בעל 100 קודקודים.

```
public static <T> int height(BinaryTree<T> bt) {
    int ans = -1;
    if(bt!=null && !bt.isEmpty()) {
        int hl = height(bt.getLeft());
        int hr = height(bt.getRight());
        return 1 + Math.max(hl, hr);
    }
    return ans;
}
public static BTD create(double[] arr) {
    BTD ans = new BTD();
    if(arr!=null && arr.length>0) {
        int mid = arr.length/2;
        double root = arr[mid];
        ans.add(root);
        double[] l_arr = subArr(arr,0,mid);
        double[] r_arr = subArr(arr,mid+1,arr.length);
        ans.setLeft(create(l_arr));
        ans.setRight(create(r_arr));
    }
    return ans;
}

private static double[] subArr(double[] arr, int start, int end) {
    int len = end-start;
    double[] ans = new double[len];
    for (int i=start;i<end;i++) {
        ans[i-start] = arr[i];
    }
    return ans;
}
```

```
@Test
public void createTest() {
    int size = 100; // {0,1,2,3,100,2^10}
    double[] arr = new double[size];
    for (int i=0;i<size;i++) {
        arr[i] = Math.random();
    }
    BTD btd = MA.create(arr);
    int h = MA.height(btd);

    int eh = -1;
    while(size>0) {size=size/2;eh++;}
    assertEquals(eh,6);
    assertEquals(eh,h);
}
```

שאלה 4:

בשאלה זו נניח שקיימת לכם המחלקה Range1 שמייצגת תחום חצי פתוח [min,max) ובעלת בנאי Range1(double min, double max), המלקה מממשת את הממשק Range1D

```
public interface Range1D {
    public boolean isEmpty(); // returns true iff min>=max.
    public boolean isInRange(double d); // returns true iff d is in [min,max).
    public double randomInRange(); //returns a random num in this range
    public Range1D intersect(Range1D r); // returns a new range (this&r)
}
```

(25 נקודות) כתבו את המחלקה Range2 שמייצגת תחום דו מימדי (מלבן דו מימדי), למחלקה בנאי שמקבל שני תחומים חד מימדיים והיא מממשת את הממשק Range2D

```
public interface Range2D {
    public boolean isEmpty(); //returns false iff both 1D ranges aren't empty
    public Range1D getXRange(); // returns the x 1D range
    public Range1D getYRange(); // returns the y 1D range
    public boolean isInRange(Point2D p); ); // returns true iff p is in both
    public Point2D randomInRange(); //returns a random point in this range
    public Range2D intersect(Range2D r); ); // returns a new range (this&r)
}
class Range2 implements Range2D {
    private Range1D _xRange, _yRange;
    public Range2(Range1D x, Range1D y) {
        _xRange = x;
        _yRange = y;
    }
    @Override
    public boolean isEmpty() {
        return _xRange.isEmpty() || _yRange.isEmpty();
    }
```

```
    @Override
    public Range1D getXRange() {return _xRange;}
    @Override
    public Range1D getYRange() {return _yRange;}

    @Override
    public boolean isInRange(Point2D p) {
        return _xRange.isInRange(p.x()) &
                _yRange.isInRange(p.y());
    }

    @Override
    public Point2D randomInRange() {
        if(isEmpty()) {throw new RuntimeException("ERR: empty 2D
range!");}
        double x = _xRange.randomInRange();;
        double y = _yRange.randomInRange();
        return new Point2D(x,y);
    }

    @Override
    public Range2D intersect(Range2D r) {
        Range1D x = r.getXRange().intersect(_xRange);
        Range1D y = r.getYRange().intersect(_yRange);
        return new Range2(x,y);
    }
}

@Test
public void range2DTest() {
   MA.Range1D x1 = new Range1(1, 3);
   MA.Range1D y1 = new Range1(0, 5);
   MA.Range2 x1y1 = new MA.Range2(x1,y1);
   MA.Range1D x2 = new Range1(1.5, 3.7);
   MA.Range1D y2 = new Range1(3, 5.3);
   MA.Range2 x2y2 = new MA.Range2(x2,y2);
   MA.Range2D x12y12 = x2y2.intersect(x1y1);
   Point2D p1 = new Point2D(2,2);
   Point2D p2 = new Point2D(2,4);
   assertTrue(x1y1.isInRange(p1));
   assertTrue(x1y1.isInRange(p2));
   assertFalse(x2y2.isInRange(p1));
   assertTrue(x2y2.isInRange(p2));
   assertTrue(x12y12.isInRange(p2));
   assertFalse(x12y12.isInRange(p1));
}
```

חומר עזר שיצורף לבחינה

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

```
// String: char charAt(int i), String substring(int start, int end), String[] split(String d)
// Math.random(); // returns a random double in [0,1)

public class StringFunctions {  // a 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 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(); // gets the root data (type T).
        public void setRoot(T v); // sets the root data (type T) into v.
        public void setLeft(BinaryTree<T> l); // sets the left (subtree) to l
        public void setRight(BinaryTree<T> r); // sets the right (subtree) to r

        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);
}
/** This interface represents a set of operations on a 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);
        /** This method computes a new (deep) copy of this GeoShape. */
        public GeoShape copy();
        /** This method returns an String representing this shape. */
        public String toString();
        /** 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);
	}
```