# Assignment 1

EECS 2011 – Fundamentals of data structures.

Rajkumar Balakrishnan Lakshmi - 213141197

9/29/17

## Problem 1. [32%] - Finding a duplicate in a preconditioned array

For this question, I used the process of negating the elements absolute position. By this way I can easily find if an integer is being repeated. I was abled to attain the output for this question due to two main reasons, one is the input must always satisfy the pre-condition that the elements of the array should be in the range 1...n. This makes it possible for the code work properly as I tend to negate the values in the absolute position of the array elements, so there will always be an absolute position of the element I am trying to negate. The second reason is that, I can change the input array which happens in my algorithm.

Example:

input: {6, 2, 1, 6, 4, 5} output: 6

Code Assertion:

```
Round 0: i=0, ints[0]=6, result=-1,

ints[(Math.abs(ints[6]))-1 = 5 check if the value in it is negative,
if not negate it, so the above becomes ints[(Math.abs(ints[6]))-1 = -5

Round 1: i=1, ints[1]=2, result=-1,

ints[(Math.abs(ints[2]))-1 = 2 check if the value in it is negative,
if not negate it, so the above becomes ints[(Math.abs(ints[6]))-1 = -2

Round 2: i=2, ints[2]=1, result=-1,

ints[(Math.abs(ints[1]))-1 = 6 check if the value in it is negative,
if not negate it, so the above becomes ints[(Math.abs(ints[6]))-1 = -6

Round 3: i=3, ints[3]=6, result= 6,

ints[(Math.abs(ints[6]))-1 = -5 check if the value in it is negative,
if not negate it, so the above becomes ints[(Math.abs(ints[6]))-1 = 5
in this case we found a value that is already neagtive, which means
that the original element of it is being repeated, thus we break the
loop and return the result.
```

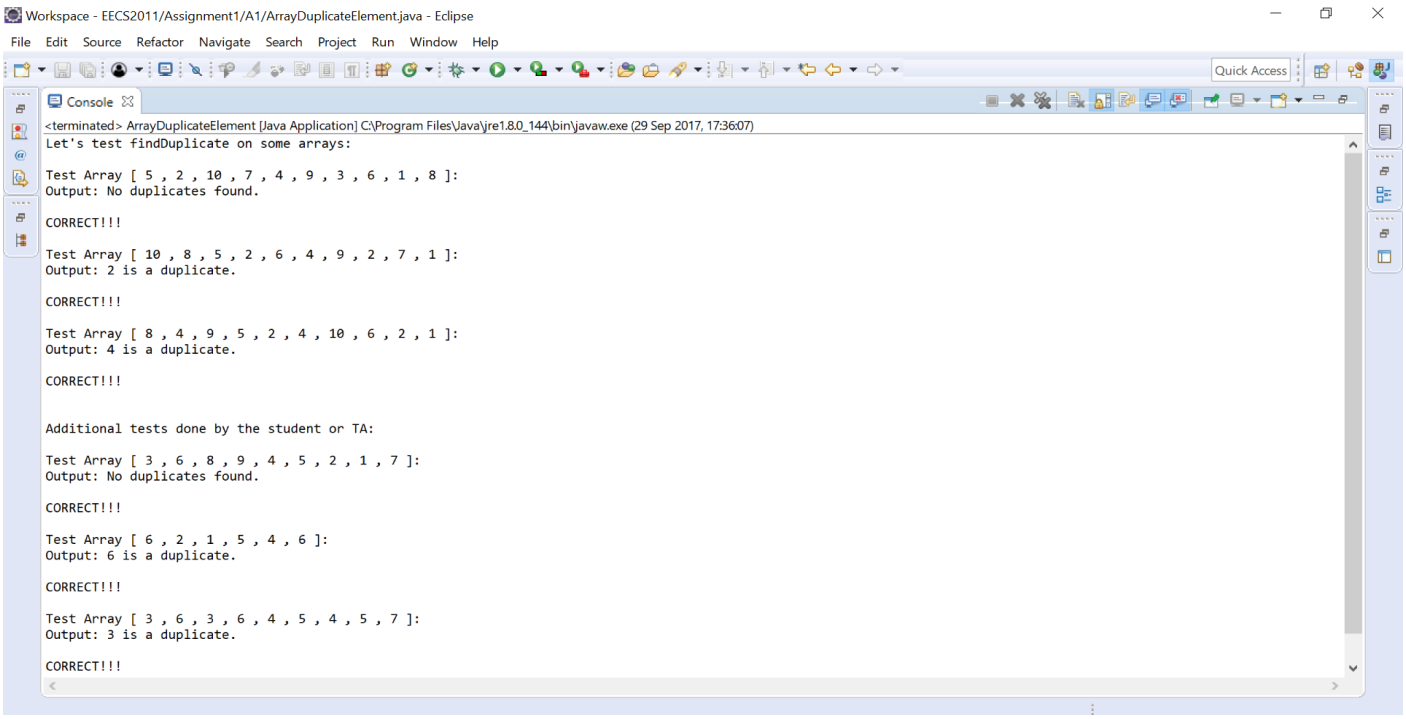
By this algorithm at any given input that satisfies the pre-condition the desired output will be achieved. Time complexity = o(n) & Space = 0(1).

## Test Cases:

Input:

```
    public static void main(String[] args)
     {

       System.out.println("Let's test findDuplicate on some arrays: \n ");

       // TEST 1:
       testDrive(new int[] { 5, 2, 10, 7, 4, 9, 3, 6, 1, 8 }, " -1 ");

       // TEST 2:
       testDrive(new int[] { 10, 8, 5, 2, 6, 4, 9, 2, 7, 1 }, " 2 ");

       // TEST 3:
       testDrive(new int[] { 8, 4, 9, 5, 2, 4, 10, 6, 2, 1 }, " 4 , 2 ");

       System.out.println("\nAdditional tests done by the student or TA:\n");

       // TEST 4:
       testDrive(new int[] { 3, 6, 8, 9, 4, 5, 2, 1, 7}, " -1 ");

       // TEST 5:
       testDrive(new int[] { 6, 2, 1, 5, 4, 6}, " 6 ");

       // TEST 6:
       testDrive(new int[] { 3, 6, 3, 6, 4, 5, 4, 5, 7}, " 3 , 6 , 4 , 5 ");
   }
```

Output:



## Problem 2. [36%] - Longest Almost Flat Subarray

The input here is array of integers ranging between 0 and n-1. I have used a while loop that scans through the array of integers with the condition "`while (i < ints.length && (i+1) < ints.length)`" i.e., it iterates through the array until it reaches the boundary of the array only once. There are temporary variables initial and length initialized to zero and one, that keeps track of the initial and length of flat arrays if any. This is achieved through if-else statement that checks if two continuous elements are either equal or has a difference of 1 and less than or equal to a max value. For the first time the max value is initialized to the length of the array, and it remains same if two continuous elements are equal (e.g. 1,1). The max value gets updated if two continuous elements have difference 1(e.g. 1, 2, max=2). Now, if a flat array is found the values of initial and length gets updated to the longest values, else remains same. By this manner 80% problem gets resolved and provides the right output. In case of an array like the below,

{7, 7, 2, **8, 7, 7, 8, 8, 8, 9, 9, 9, 8, 9, 9**, 6, 8} the underlined green 8 can be considered for both **red** and **purple** numbers. The output should be the flat array {**8, 8, 8, 9, 9, 8, 9, 9**} but with the above algorithm returns {**8, 7, 7, 8, 8, 8}** as output, since the iteration occurs element by element and only once. Thus, to resolve this I introduced another variable sublength initialized to one and it keeps track of the repetition of the max value only at the end of a flat array. The sublength value makes changes to the initial and length values only if the first element next to the end of a flat array is max+1 and sublength is greater than 1. For e.g. in the above case in first flat array max=8 and the first element next to the end of flat array is max+1(9), hence the sublength will be added respectively. This way I could solve the above issue and obtain the right output.

Example: Input Assertion -

Input: {7, 2, 8, 7, 7, 8, 8, 8, 9, 9, 8, 9, 9, 2} output: **[initial, length]**

Initial values: `initial=0, length=1, max=0, sublength=1`

```
Round 0: Input: [7!=2]
initial=0, length=1, max=14, sublength=1
```

Round 1: Input: [2!=8]

initial=0, length=1, max=0, sublength=1

Round 2: Input: [8-7=1]

initial=2, length=2, max=8, sublength=2

Round 3: Input: [7=7]

initial=2, length=3, max=8, sublength=1

Round 4: Input: [7-8=1]

initial=2, length=4, max=8, sublength=1

Round 5: Input: [8=8]

initial=2, length=5, max=8, sublength=2

Round 6: Input: [8=8] output: [2,6]

initial=2, length=6, max=8, sublength=3

End of flat array

New round values: initial=current position- sublength, length=sublength+1, max=14, sublength=1

Round 7: Input: [8-9=1]

initial=5, length=4, max=9, sublength=1

Round 8: Input: [9=9]

initial=5, length=5, max=9, sublength=2

Round 9: Input: [9-8=1]

initial=5, length=6, max=9, sublength=1

Round 10: Input: [8-9=1]

initial=5, length=7, max=9, sublength=1

Round 11: Input: [9=9] **output: [5,8]**

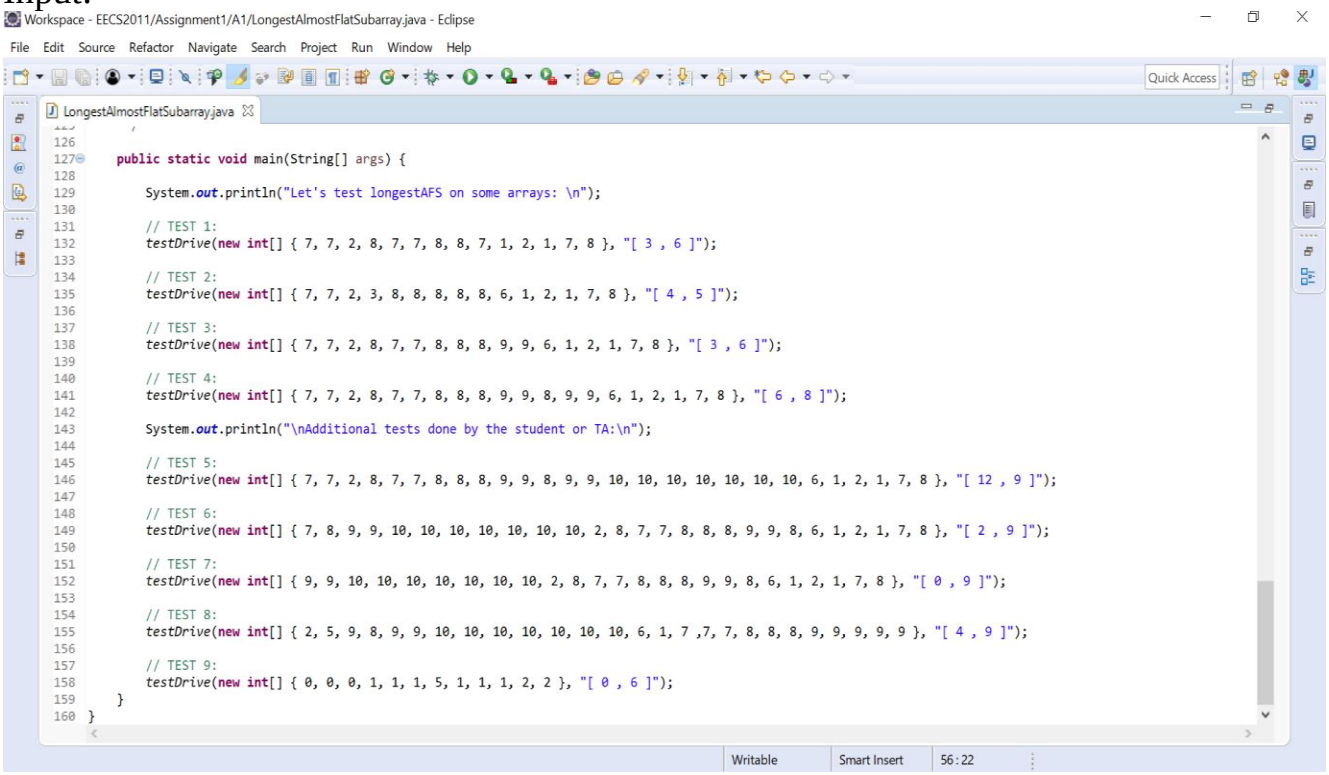
initial=5, length=8, max=9, sublength=2

Round 12: Input: [9!=2]

initial=13, length=1, max=9, sublength=1

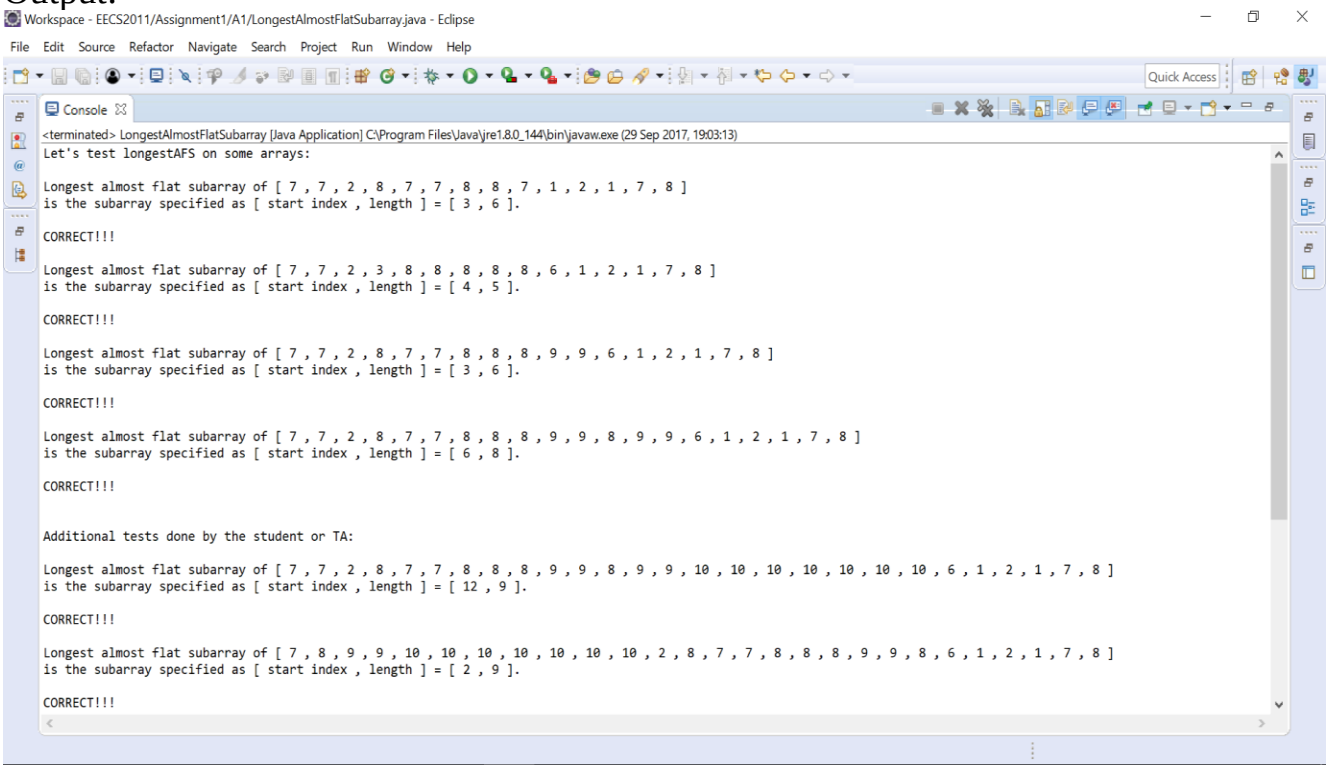
# Test Cases:

## Input:



```
Workspace - EECS2011/Assignment1/A1/LongestAlmostFlatSubarray.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
LongestAlmostFlatSubarray.java
126
127    public static void main(String[] args) {
128
129        System.out.println("Let's test longestAFS on some arrays: \n");
130
131        // TEST 1:
132        testDrive(new int[] { 7, 7, 2, 8, 7, 7, 8, 8, 7, 1, 2, 1, 7, 8 }, "[ 3 , 6 ]");
133
134        // TEST 2:
135        testDrive(new int[] { 7, 7, 2, 3, 8, 8, 8, 8, 8, 6, 1, 2, 1, 7, 8 }, "[ 4 , 5 ]");
136
137        // TEST 3:
138        testDrive(new int[] { 7, 7, 2, 8, 7, 7, 8, 8, 8, 9, 9, 6, 1, 2, 1, 7, 8 }, "[ 3 , 6 ]");
139
140        // TEST 4:
141        testDrive(new int[] { 7, 7, 2, 8, 7, 7, 8, 8, 8, 9, 9, 8, 9, 9, 6, 1, 2, 1, 7, 8 }, "[ 6 , 8 ]");
142
143        System.out.println("\nAdditional tests done by the student or TA:\n");
144
145        // TEST 5:
146        testDrive(new int[] { 7, 7, 2, 8, 7, 7, 8, 8, 8, 9, 9, 8, 9, 9, 10, 10, 10, 10, 10, 10, 10, 6, 1, 2, 1, 7, 8 }, "[ 12 , 9 ]");
147
148        // TEST 6:
149        testDrive(new int[] { 7, 8, 9, 9, 10, 10, 10, 10, 10, 10, 10, 2, 8, 7, 7, 8, 8, 8, 9, 9, 8, 6, 1, 2, 1, 7, 8 }, "[ 2 , 9 ]");
150
151        // TEST 7:
152        testDrive(new int[] { 9, 9, 10, 10, 10, 10, 10, 10, 10, 2, 8, 7, 7, 8, 8, 8, 9, 9, 8, 6, 1, 2, 1, 7, 8 }, "[ 0 , 9 ]");
153
154        // TEST 8:
155        testDrive(new int[] { 2, 5, 9, 8, 9, 9, 10, 10, 10, 10, 10, 10, 10, 6, 1, 7 ,7, 7, 8, 8, 8, 9, 9, 9, 9, 9 }, "[ 4 , 9 ]");
156
157        // TEST 9:
158        testDrive(new int[] { 0, 0, 0, 1, 1, 1, 5, 1, 1, 1, 2, 2 }, "[ 0 , 6 ]");
159    }
160 }
Writable Smart Insert 56 : 22
```

## Output:



```
Workspace - EECS2011/Assignment1/A1/LongestAlmostFlatSubarray.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Console
<terminated> LongestAlmostFlatSubarray [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (29 Sep 2017, 19:03:13)
Let's test longestAFS on some arrays:

Longest almost flat subarray of [ 7 , 7 , 2 , 8 , 7 , 7 , 8 , 8 , 7 , 1 , 2 , 1 , 7 , 8 ]
is the subarray specified as [ start index , length ] = [ 3 , 6 ].

CORRECT!!!

Longest almost flat subarray of [ 7 , 7 , 2 , 3 , 8 , 8 , 8 , 8 , 8 , 6 , 1 , 2 , 1 , 7 , 8 ]
is the subarray specified as [ start index , length ] = [ 4 , 5 ].

CORRECT!!!

Longest almost flat subarray of [ 7 , 7 , 2 , 8 , 7 , 7 , 8 , 8 , 8 , 9 , 9 , 6 , 1 , 2 , 1 , 7 , 8 ]
is the subarray specified as [ start index , length ] = [ 3 , 6 ].

CORRECT!!!

Longest almost flat subarray of [ 7 , 7 , 2 , 8 , 7 , 7 , 8 , 8 , 8 , 9 , 9 , 8 , 9 , 9 , 6 , 1 , 2 , 1 , 7 , 8 ]
is the subarray specified as [ start index , length ] = [ 6 , 8 ].

CORRECT!!!


Additional tests done by the student or TA:

Longest almost flat subarray of [ 7 , 7 , 2 , 8 , 7 , 7 , 8 , 8 , 8 , 9 , 9 , 8 , 9 , 9 , 10 , 10 , 10 , 10 , 10 , 10 , 10 , 6 , 1 , 2 , 1 , 7 , 8 ]
is the subarray specified as [ start index , length ] = [ 12 , 9 ].

CORRECT!!!

Longest almost flat subarray of [ 7 , 8 , 9 , 9 , 10 , 10 , 10 , 10 , 10 , 10 , 10 , 2 , 8 , 7 , 7 , 8 , 8 , 8 , 9 , 9 , 8 , 6 , 1 , 2 , 1 , 7 , 8 ]
is the subarray specified as [ start index , length ] = [ 2 , 9 ].

CORRECT!!!
```

## Problem 3 . [32%] A Hierarchy of Planar Shapes

For this question I just created the classes and their related methods as shown in the UML diagram.

### Class Ellipse

Area () - This is an overridden method implemented from the PlanerShape interface, that calculates and returns the area of an ellipse. I calculated the area using the formula $A = \pi * \text{Horizontal. axis} * Vertical. axis$

Contains (Point) - This is an overridden method implemented from the PlanerShape interface, that finds if a given point lies within the ellipse. I calculated this using the formula provided in the question.

### Test Cases:

Input1:

```
public static void main(String[] args) throws InvalidShapeException
 {

      Point2D.Double centre = new Point2D.Double(0,0);
      Ellipse e = new Ellipse(15.00,30.00, centre);

      Point2D.Double point = new Point2D.Double(4,2);

      System.out.printf("%.2f\n",e.area());
      System.out.println(e.contains(point));

      Point2D.Double point2 = new Point2D.Double(17,15);
      System.out.println(e.contains(point2));

      System.out.println(e.toString());
}
```

Output1:

```
1413.72
true
false
Ellipse [Centre = (0.0,0.0), Horizontal_Axis = 15.0, Vertical_Axis = 30.0]
```

Input2:

```
public static void main(String[] args) throws InvalidShapeException
 {
      Point2D.Double centre = new Point2D.Double(0,0);
      Ellipse e = new Ellipse(-15.00,30.00, centre);
 }
```

Output2:

```
Exception in thread "main" A1.InvalidShapeException: Invalid entry! Negative
ellipse axis has been entered
	at A1.Ellipse.sethAxis(Ellipse.java:71)
	at A1.Ellipse.<init>(Ellipse.java:48)
	at A1.Ellipse.main(Ellipse.java:164)
```

## Class Circle

Area () - This is an overridden method implemented and inherited from the parent class, that calculates and returns the area of a circle. I calculated the area using the formula $A = \pi r^2$.

Contains (Point) - This is an overridden method implemented and inherited from the parent class, that finds if a given point lies within the circle. I calculated this in the similar way as that of ellipse, since circle can be considered an ellipse.

Contains (circle) - This method I used to check if a circle exists inside another circle. I verified this using the formula,

```
Math.sqrt((Math.pow((this.center.x-c.center.x), 2)) + (Math.pow((this.center.y -
c.center.y), 2))) <= (Math.abs(this.radius-c.radius)));
```

Input1:

```
public static void main(String[] args) throws InvalidShapeException
 {
	Point2D.Double mainCentre = new Point2D.Double(0,0);
	Circle mainCircle = new Circle(5,mainCentre);

	Point2D.Double subCentre = new Point2D.Double(0,0);
	Circle subCircle = new Circle(5,subCentre);

	Point2D.Double subCentre2 = new Point2D.Double(8,5);
	Circle subCircle2 = new Circle(5,subCentre2);

	Point2D.Double point = new Point2D.Double(4,2);
	Point2D.Double point2 = new Point2D.Double(4,4);
	System.out.printf("%.2f\n",mainCircle.area());
	System.out.println(mainCircle.contains(point));
	System.out.println(mainCircle.contains(point2));
	System.out.println(mainCircle.contains(subCircle));
	System.out.println(mainCircle.contains(subCircle2));
	System.out.println(mainCircle.toString());

}
```

Output1:

```
78.54
true
false
true
false
Circle [ Centre = (0.0,0.0), Radius=5.0 ]
```

Input2:

```
public static void main(String[] args) throws InvalidShapeException
 {
       Point2D.Double mainCentre = new Point2D.Double(0,0);
       Circle mainCircle = new Circle(-5,mainCentre);
 }
```

Output2:

```
Exception in thread "main" A1.InvalidShapeException: Invalid entry! Negative
circle radius has been entered
       at A1.Circle.setRadius(Circle.java:69)
       at A1.Circle.<init>(Circle.java:47)
       at A1.Circle.main(Circle.java:146)
```