# Data structures and algorithms project

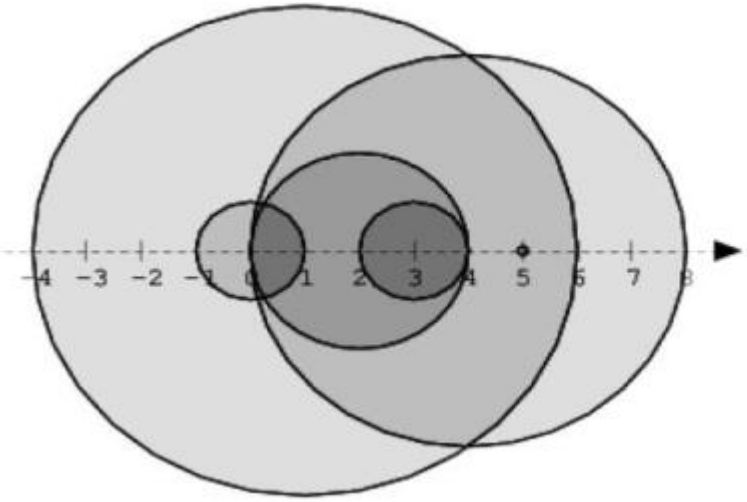
## Project Number: 1

# Contents

# Problem Statement

## Project 1: Intersecting Discs **(two solutions minimum)**

We draw N discs on a plane. The discs are numbered from 0 to N - 1. A zero-indexed array A of N non-negative integers, specifying the radiuses of the discs, is given. The J-th disc is drawn with its center at (J, 0) and radius A[J]. We say that the J-th disc and K-th disc intersect if $J \neq K$ and the J-th and K-th discs have at least one common point (assuming that the discs contain their borders).




Sample Input:
The figure below shows discs drawn for N = 6 and A as follows:
A[0] = 1
A[1] = 5
A[2] = 2
A[3] = 1
A[4] = 4
A[5] = 0
Sample Output:
There are eleven (11) (unordered) pairs of discs that intersect, namely:

- discs 1 and 4 intersect, and both intersect with all the other discs;
- disc 2 also intersects with discs 0 and 3.

# Solution 1

## Pseudocode

1. Program starts
2. Make class called intersection
3. Make variable N to store array length
4. Make array circleRadius to store radiuses or start points
5. Make array circleMidpoint to store midpoints or end points
6. For loop:
   To move through all elements in the array
   Calculate radiuses by values – it's index in the array
   Calculate midpoints by values + it's index in the array
   End for loop
7. Sort arrays of radius and mid points
8. Make number line and put sorted arrays on it
9. Make variable open circles to store number of opened circles
10. Make variable intersections to store number of intersections and increment it
11. While loop:
    to make sure we opened all the circles and counts all the intersections
    end while loop
12. Return intersections
13. Call method from class in main ()
14. End program

## Used Algorithms and Data Structures

- Sort
- Arrays

## Sample Output

```
run:
11
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Code

```
1. import java.util.Arrays;
2. public class circle {
3.   public int intersection(int[] array){
4.       int N = array.length;
5.       int [] circleRadius = new int [N];
6.       int [] circleMidpoint = new int [N];
7.       for (int i = 0; i < N; i++) {
8.           circleRadius[i] = i - array[i];
9.           if (Integer.MAX_VALUE - array[i] < i) {
10.                  circleMidpoint[i] = Integer.MAX_VALUE;
11.              } else {
12.                  circleMidpoint[i] = i + array[i];
13.              }
14.           }
15.           Arrays.sort(circleRadius);
16.           Arrays.sort(circleMidpoint);
17.           int RadiusIndex = 0;
18.           int MidPointIndex = 0;
19.           int openCircles = 0;
20.           int intersections = 0;
21.             while (RadiusIndex < N) {
22.                 if (circleRadius[RadiusIndex] <=
   circleMidpoint[MidPointIndex]) {
23.                     intersections = intersections + openCircles;
24.                     openCircles++;
25.                     RadiusIndex++;
26.                 } else {
27.                     openCircles--;
28.                     MidPointIndex++;
29.                 }
30.
31.             }
32.             return intersections;
33.
34.         }
35.     }
36.  //main function
37.     public static void main(String[] args) {
38.
39.
40.         int[] array = {1, 5, 2, 1, 4, 0};
41.         System.out.println("number of intersection circles is " + new
   circle().intersection(array));
42.  }
```

# Complexity: O(n)

# Solution 2

## Pseudocode

1. intialize arraylist to get input from user with dynamic size
2. user input the center points of discs and flag to start the operation
3. loop over array and arraylist to copy user input from arraylist to array of size arraylist
4. call intersectingDiscs method and send the input array as parameter
5. intialize two arrays of start and end points
6. do calculations over the array values to fill the two arrays with index-radius=startpoint & index+radius= endpoints
7. and store values in the arrays
8. sort the two arrays
9. intialize counter
10. intialize position variable
11. reverse loop over the two arrays with bubble search to find end point position in the start point array  with complixity O(log n)
12. increment position variable to loop over end points
13. add position to counter to calculate all startpoints less than the end
14. get the total amount of combinations;
15. count= unintersected- all
16. return all

## Used Algorithms and Data Structures

- Binary Search
- Sort
- Arrays
- ArrayList

# Code

```
1. import java.util.ArrayList;
2. import java.util.Arrays;
3. import java.util.Scanner;
4.
5. /**
6.  *
7.  * @author Administrator
8.  */
9. public class Pdsal {
10.
11.         /**
12.          * @param args the command line arguments
13.          */
14.         public static void main(String[] args) {
15.             ArrayList<Integer> list = new ArrayList<>();
16.             Scanner input = new Scanner(System.in);
17.             System.out.println("Enter number of Circles ");
18.             System.out.println("Enter any character to stop:");
19.             while (input.hasNextInt()) {
20.                 list.add(input.nextInt());
21.             }
22.             int[] arr = new int[list.size()];
23.             for (int i = 0; i < arr.length; i++) {
24.                 arr[i] = list.get(i);
25.             }
26.             System.out.println(intersectingDiscs(arr));
27.         }
28.
29.         public static int intersectingDiscs(int[] A) {
30.             long[] EndPoint = new long[A.length];
31.             long[] startPoint = new long[A.length];
32.
33.             for (int i = 0; i < A.length; i++) {
34.                 EndPoint[i] = (long) A[i] + i;
35.                 startPoint[i] = -((long) A[i] - i);
36.             }
37.             Arrays.sort(EndPoint);
38.             Arrays.sort(startPoint);
39.
40.             long count = 0;
41.
42.             for (int i = A.length - 1; i >= 0; i--) {
43.                 int position = Arrays.binarySearch(startPoint,
   EndPoint[i]);
44.                 if (position >= 0) { //end loop
45.                     while (position < A.length && startPoint[position]
   == EndPoint[i]) {
46.                         position++;
47.                     }
48.                     count += position;
```

```
                } else { // element not there
                    int intersectionPoint = -(position + 1);
                    count += intersectionPoint;
                }

            }

            long unIntersectedDiscs = (long) A.length * ((long) A.length + 1) / 2;
            count -= unIntersectedDiscs;

            return (int) count;
        }

    }
```

## Complexity: O(nlogn)

## Sample Output

```
run:
Enter number of Circles
Enter any character to stop:
1
5
2
1
4
0
j
number of intersections =11
BUILD SUCCESSFUL (total time: 13 seconds)
```

# Solution 3

## Pseudocode

1. Declare an array (6) of an integers
2. For i as integer = 0 to less than N
3. Start point = i – A
4. End point = i+A
5. End for
6. Sort array
7. Intersection = 0
8. J = 0
9. For i as integer = 0 to less than N
10. While j is smaller than N and end point is greater than start point
11. Intersection = intersection + j
12. Intersection =intersection – j
13. End for
14. End while

## Used Algorithms and Data Structures

- **Sort**
- **Arrays**

# Code

```
1. package project.ds;
2.
3. import java.util.*;
4.
5. class problem {
6.
7.     public static void main(String[] args) {
8.         int[] array = {1, 5, 2, 1, 4, 0};
9.         System.out.println("pairs of discs that intersect = " +
   number_of_intersections(array));
10.         }
11.
12.         private static int number_of_intersections(int[] A) {
13.             int N = A.length;
14.             long[] start_point = new long[N];
15.             long[] end_point = new long[N];
16.             for (int i = 0; i < N; i++) {
17.                 start_point[i] = i - A[i];
18.                 end_point[i] = i + A[i];
19.             }
20.
21.             Arrays.sort(start_point);
22.             Arrays.sort(end_point);
23.
24.             int intersection = 0;
25.             int j = 0;
26.             for (int i = 0; i < N; i++) {
27.                 while (j < N && end_point[i] >= start_point[j]) {
28.                     intersection += j;
29.                     intersection -= i;
30.                     j++;
31.                 }
32.             }
33.             return intersection;
34.         }
35.     }
36.
```

# Complexity: O(n^2)

# Sample Output

```
run:
pairs of discs that intersect = 11
BUILD SUCCESSFUL (total time: 0 seconds)
```