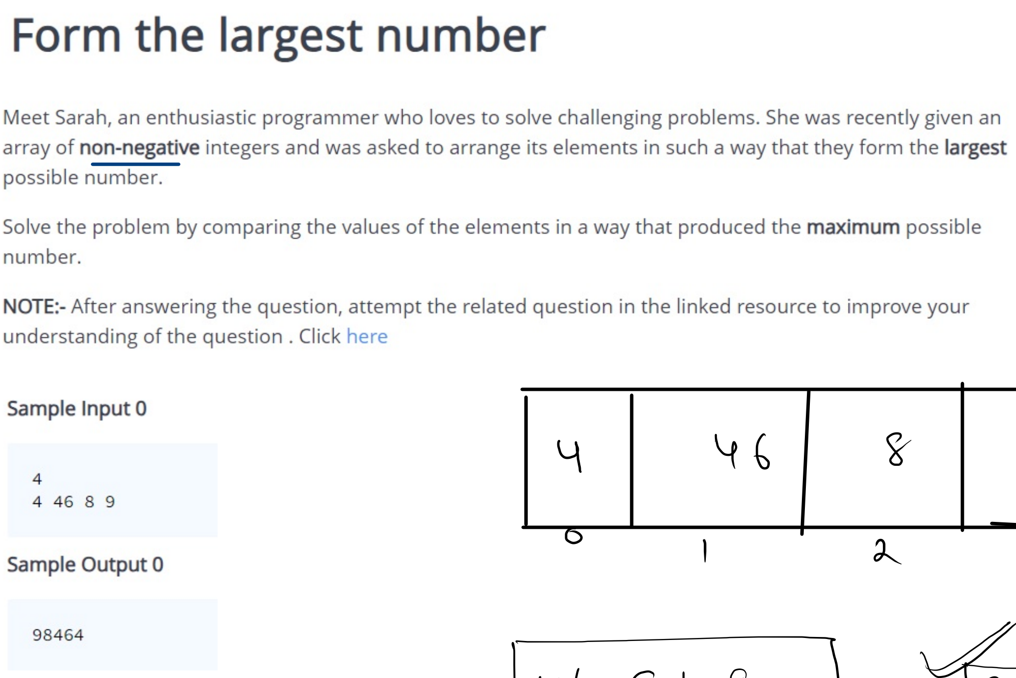


Form the largest number

Meet Sarah, an enthusiastic programmer who loves to solve challenging problems. She was recently given an array of **non-negative** integers and was asked to arrange its elements in such a way that they form the **largest** possible number.

Solve the problem by comparing the values of the elements in a way that produced the **maximum** possible number.

NOTE:- After answering the question, attempt the related question in the linked resource to improve your understanding of the question . Click here

Sample Input 0

```
4
4 46 8 9
```

Sample Output 0

```
98464
```

4	46	8	9
0	1	2	3

46,8 4 9 < ✓ 98,46 4

Case 1:

8	4	9

→ decreasing. ✓

948.

Case 2:

3	30

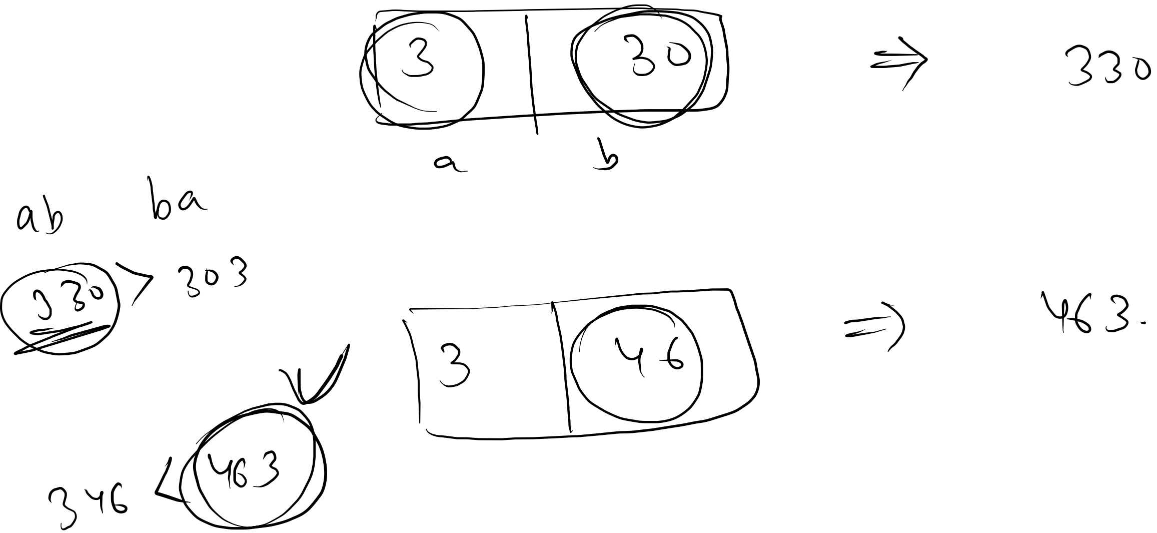
→ ✗ decreasing order

303. < 330

Case 3:

3	46

→ decreasing. ✓


3
30
330
a
b
ab
ba
330
303
3
46
463.
463
346

Custom Comparator

3	30
a	b

str [$x = a+b = 330$

$y = b+a = 303$

$y - x$] int

$303 - 330 = -ve$

46 3.

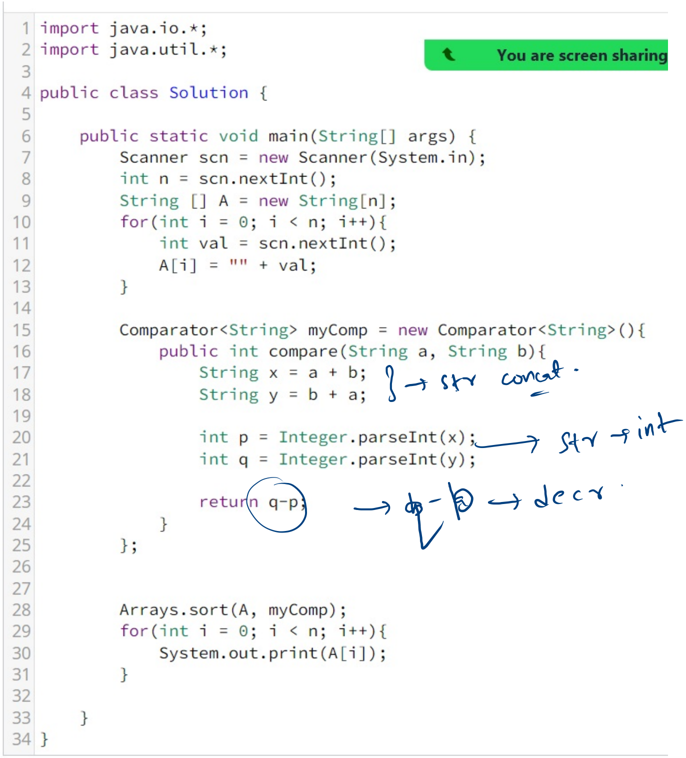
~~3~~	~~46~~
a	b

$x = a+b = 346$

$y = b+a = 463$

$y - x$] int

$463 - 346$] $= +ve$.

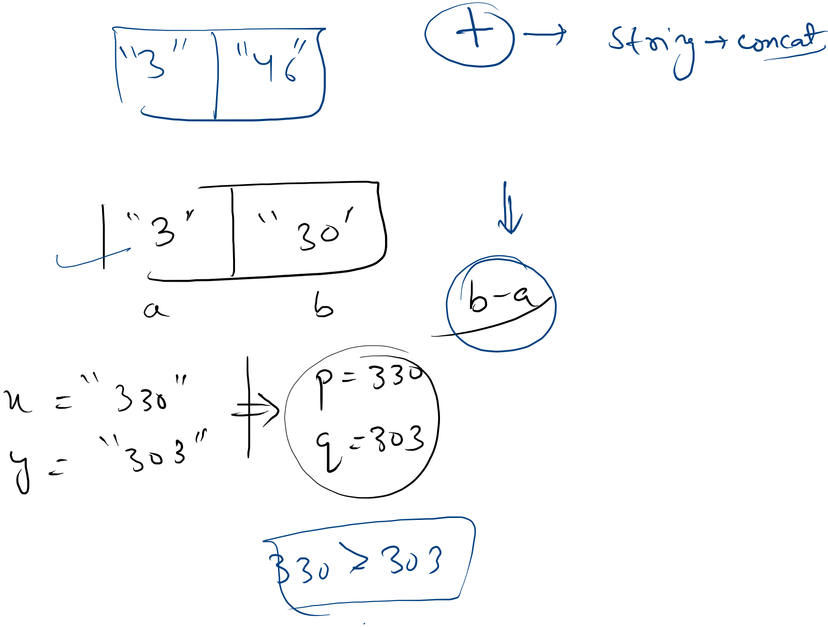


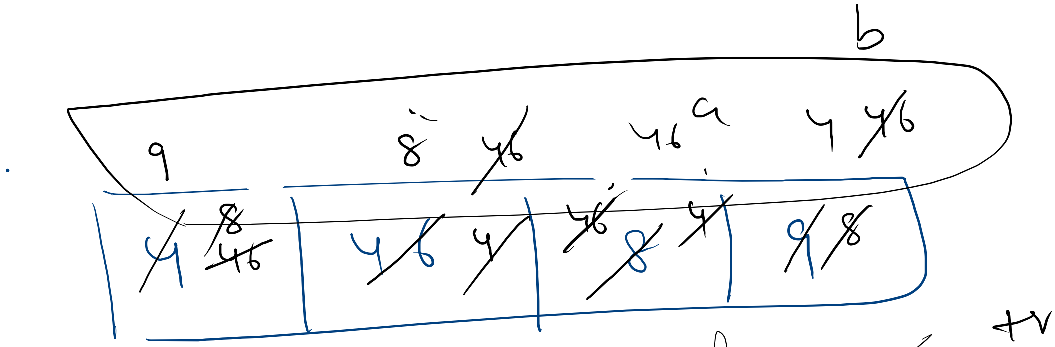
```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        String [] A = new String[n];
        for(int i = 0; i < n; i++){
            int val = scn.nextInt();
            A[i] = "" + val;
        }

        Comparator<String> myComp = new Comparator<String>(){
            public int compare(String a, String b){
                String x = a + b;
                String y = b + a;

                int p = Integer.parseInt(x);
                int q = Integer.parseInt(y);

                return q-p;
            }
        };


        Arrays.sort(A, myComp);
        for(int i = 0; i < n; i++){
            System.out.print(A[i]);
        }

    }
}
```

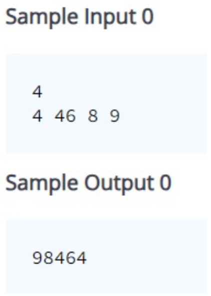
} → str concat.

→ str → int

→ b - a → decr.







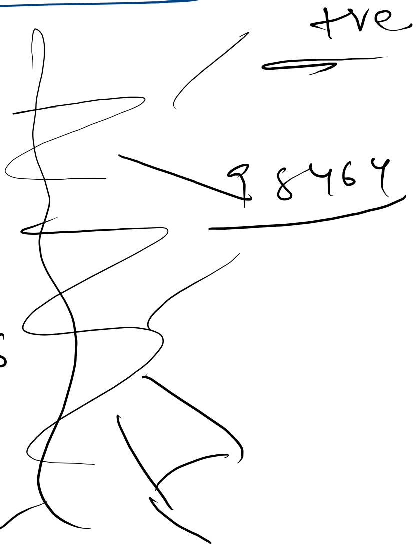
Sample Input 0

```
4
4  46  8  9
```

Sample Output 0

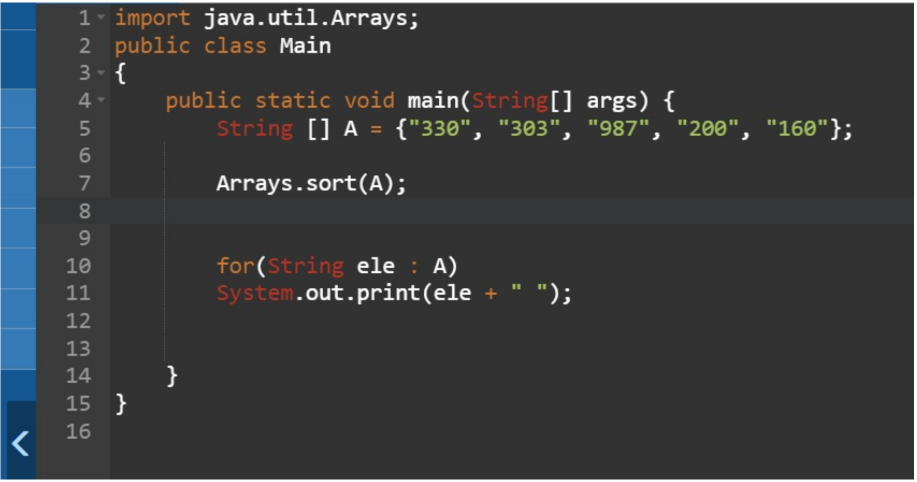
```
98464
```

446 < 464



468   846

Sorting with Strings



```java
import java.util.Arrays;
public class Main
{
    public static void main(String[] args) {
        String [] A = {"330", "303", "987", "200", "160"};

        Arrays.sort(A);


        for(String ele : A)
        System.out.print(ele + " ");


    }
}

```

Peak Index in a Mountain Array 2

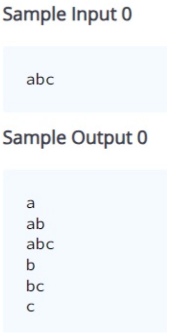
Try as HW

Sub-string → inbuilt

10	20	30

a b c

0 1 2



Sample Input 0

```
abc
```

Sample Output 0

```
a
ab
abc
b
bc
c
```

${}_0a_0$

${}_0ab_1$ ${}_1b_1$ ${}_2c_2$

${}_0abc_2$ ${}_1bc_2$

${}_0 10_0$

${}_0 10\ 20_1$ ${}_1 20_1$ ${}_2 30_2$

${}_0 10\ 20\ 30_2$ ${}_1 20\ 30_2$

st (i)	end (j)	
0	0 1 2	< n
1	1 2	< n
2	2	< n

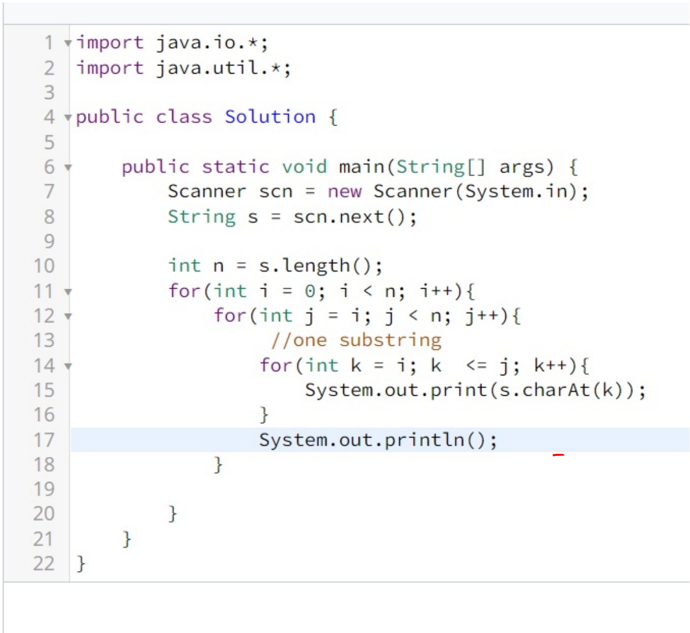
$j = i$

n = 3

"abc"

0 1 2

i j



```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String s = scn.next();

        int n = s.length();
        for(int i = 0; i < n; i++){
            for(int j = i; j < n; j++){
                 //one substring
                for(int k = i; k  <= j; k++){
                    System.out.print(s.charAt(k));
                }
                System.out.println();
            }

        }
    }
}
```

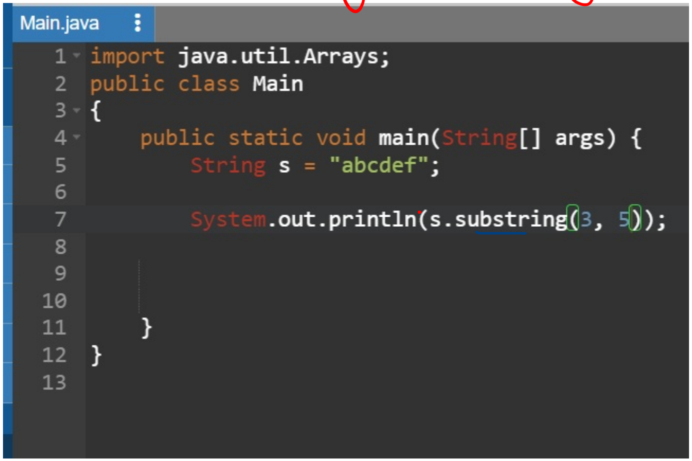
ab

0 1

a

ab

Substring → using inbuilt

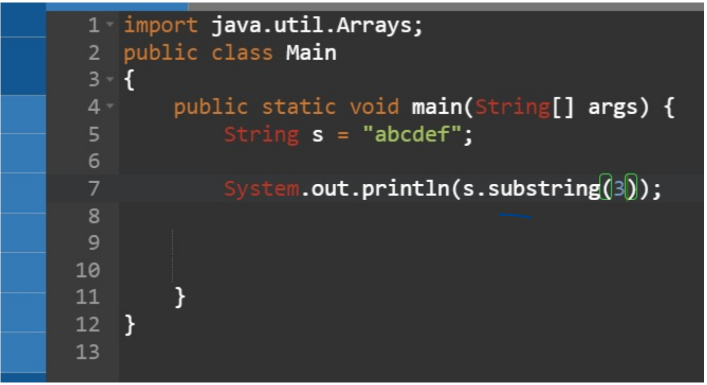


```java
import java.util.Arrays;
public class Main
{
    public static void main(String[] args) {
        String s = "abcdef";

        System.out.println(s.substring(3, 5));



    }
}

```

⇒ de

a b c d e f

0 1 2 3 4 5



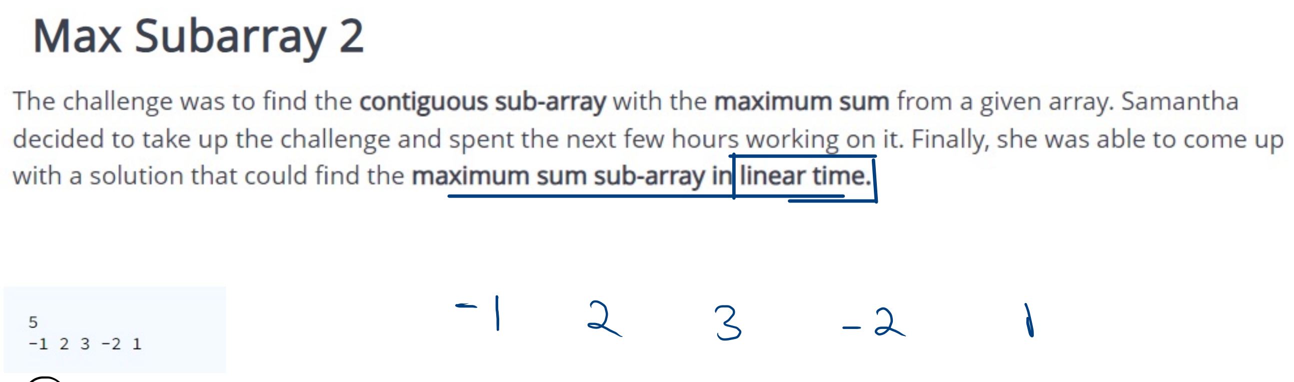
```java
import java.util.Arrays;
public class Main
{
    public static void main(String[] args) {
        String s = "abcdef";

        System.out.println(s.substring(3));



    }
}

```

⇒ def

★ (OOPs).

polymorphism

★ function (method) overloading



Max Subarray 2

The challenge was to find the **contiguous sub-array** with the **maximum sum** from a given array. Samantha decided to take up the challenge and spent the next few hours working on it. Finally, she was able to come up with a solution that could find the **maximum sum sub-array in linear time.**

```
5
-1 2 3 -2 1
```

5

-1 2 3 -2 1

-1 → -1
-1 2 → 1
-1 2 3 → 4
-1 2 3 -2 → 2
-1 2 3 -2 1 → 3

2 → 2
2 3 → 5
2 3 -2 → 3
2 3 -2 1 → 4

3 → 3
3 -2 → 1
3 -2 1 → 2

-2 → -2
-2 1 → -1

1

(story)

Kadane's Algo

-1 2 3 -2 1

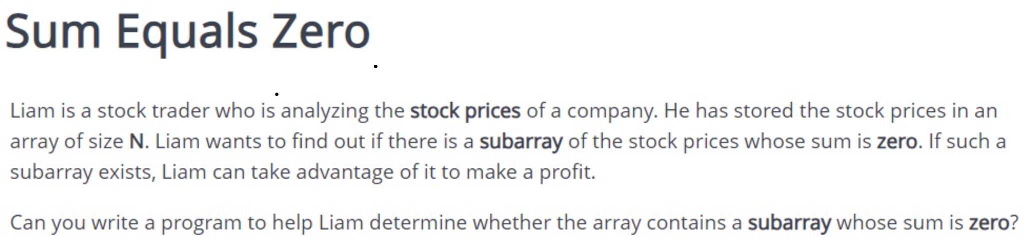
Method → find all subarray → $\underline{O(n^3)}$. → $o(n)$ → Kadane's Algo

↳ sum

↳ max

★ ★

Revision. → all. the subarray,



Sum Equals Zero

Liam is a stock trader who is analyzing the **stock prices** of a company. He has stored the stock prices in an array of size **N**. Liam wants to find out if there is a **subarray** of the stock prices whose sum is **zero**. If such a subarray exists, Liam can take advantage of it to make a profit.

Can you write a program to help Liam determine whether the array contains a **subarray** whose sum is **zero**?

10 20 30

false

4

-1 1 2 3

true.

-1

-1 1 → 0

-1 1 2

-1 1 2 3

1

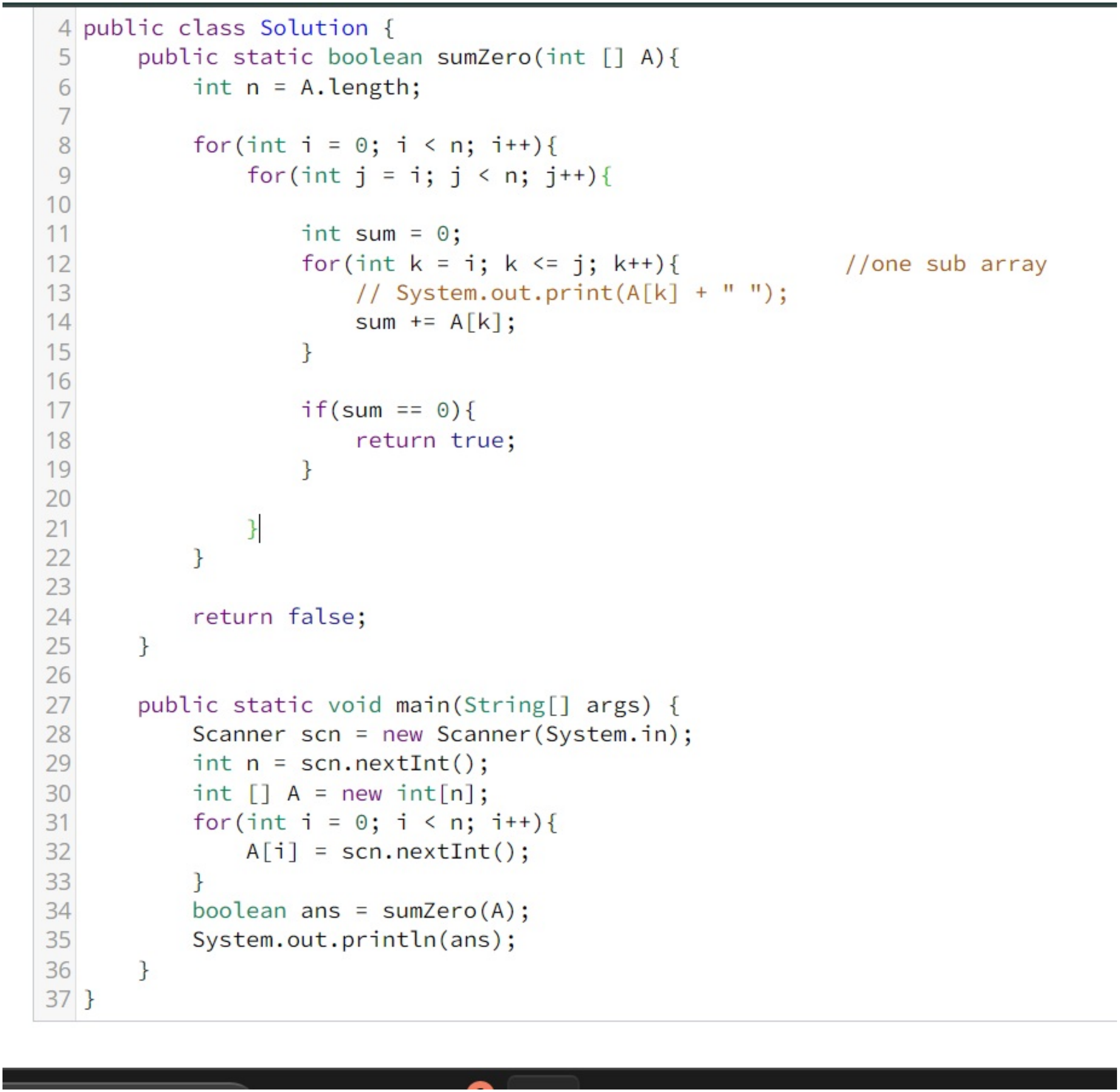
1 2

1 2 3

2

2 3

3



```java
public class Solution {
    public static boolean sumZero(int [] A){
        int n = A.length;

        for(int i = 0; i < n; i++){
            for(int j = i; j < n; j++){

                int sum = 0;
                for(int k = i; k <= j; k++){            //one sub array
                    // System.out.print(A[k] + " ");
                    sum += A[k];
                }

                if(sum == 0){
                    return true;
                }

            }
        }

        return false;
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] A = new int[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }
        boolean ans = sumZero(A);
        System.out.println(ans);
    }
}
```