

ACCENTURE CODING ROUND QUESTIONS

A1 Sum to Target Number (code_1)

Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Input Format

- The First Line contains an Integer `N`
- The Second Line contains `N` integers representing the array elements.
- The Third Line contains an Integer denoting the value of `target`.

Constraints

- $2 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$
- Only one valid answer exists.

Output Format

Print the Indexes of two integers whose sum is equal to `target`

Sample Input 0

```
3
3 2 4
6
```

Sample Output 0

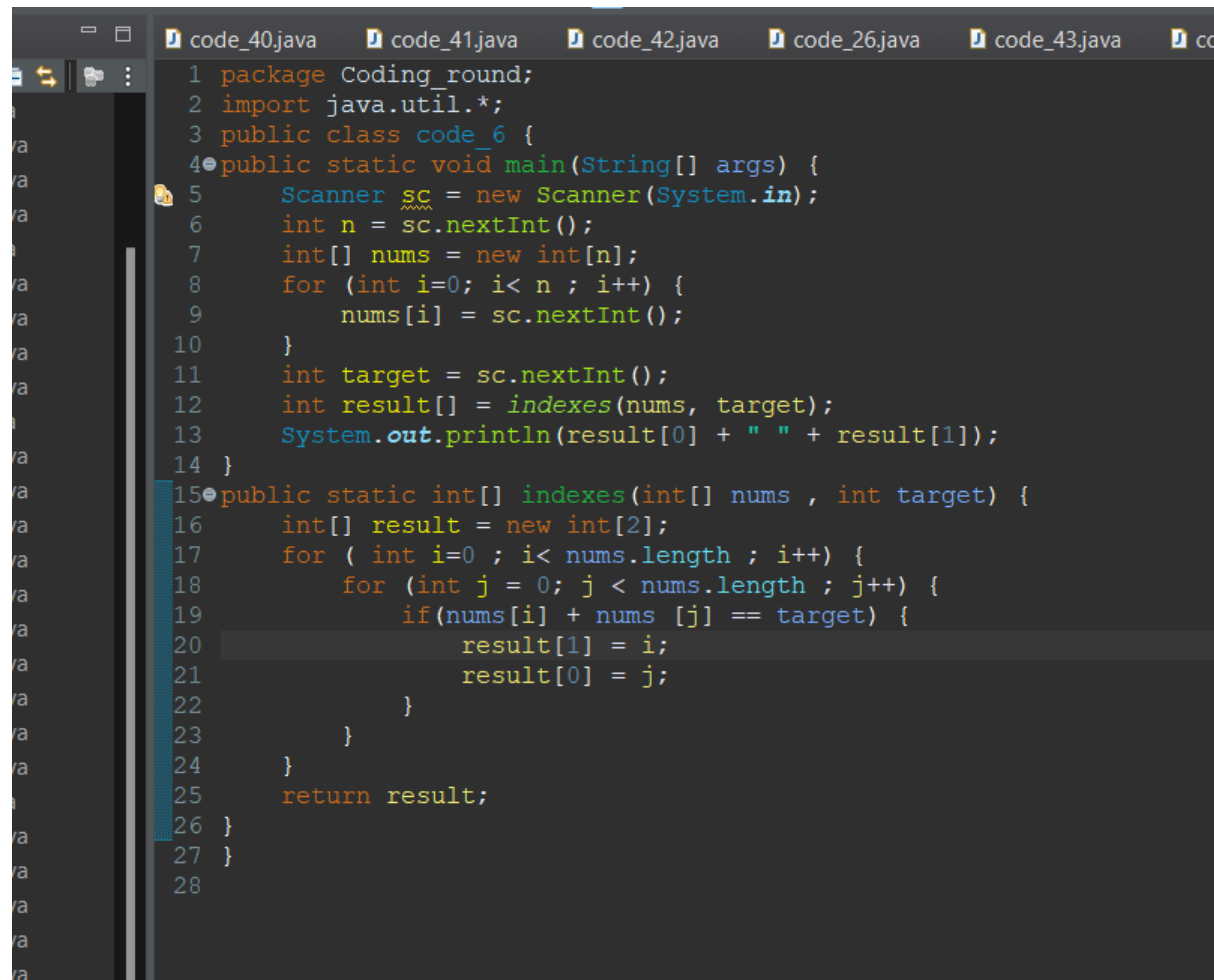
1 2

Sample Input 1

7
4272383 -943113 -466257 -458745 -474695 42855686 -280096
47128069

Sample Output 1

0 5

A screenshot of a Java IDE with a dark theme. The file name 'code_6.java' is visible in the tab bar. The code is as follows:

```
1 package Coding_round;
2 import java.util.*;
3 public class code_6 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int n = sc.nextInt();
7         int[] nums = new int[n];
8         for (int i=0; i< n ; i++) {
9             nums[i] = sc.nextInt();
10        }
11        int target = sc.nextInt();
12        int result[] = indexes(nums, target);
13        System.out.println(result[0] + " " + result[1]);
14    }
15    public static int[] indexes(int[] nums , int target) {
16        int[] result = new int[2];
17        for ( int i=0 ; i< nums.length ; i++) {
18            for (int j = 0; j < nums.length ; j++) {
19                if(nums[i] + nums [j] == target) {
20                    result[1] = i;
21                    result[0] = j;
22                }
23            }
24        }
25        return result;
26    }
27 }
28
```

A3 No Repeating (code_2)

Given a string s, find the length of the longest substring without repeating characters.

Input Format

- The First line will have a integer N denoting the length of string.
- The Second line will have a string S.

Constraints

- $0 \leq s.length \leq 5 * 10^4$
- s consists of English letters, digits, symbols and spaces.

Output Format

Print the integer which is the answer to the question

Sample Input 0

18
fgsurhydrjrkxjlvhvo

Sample Output 0

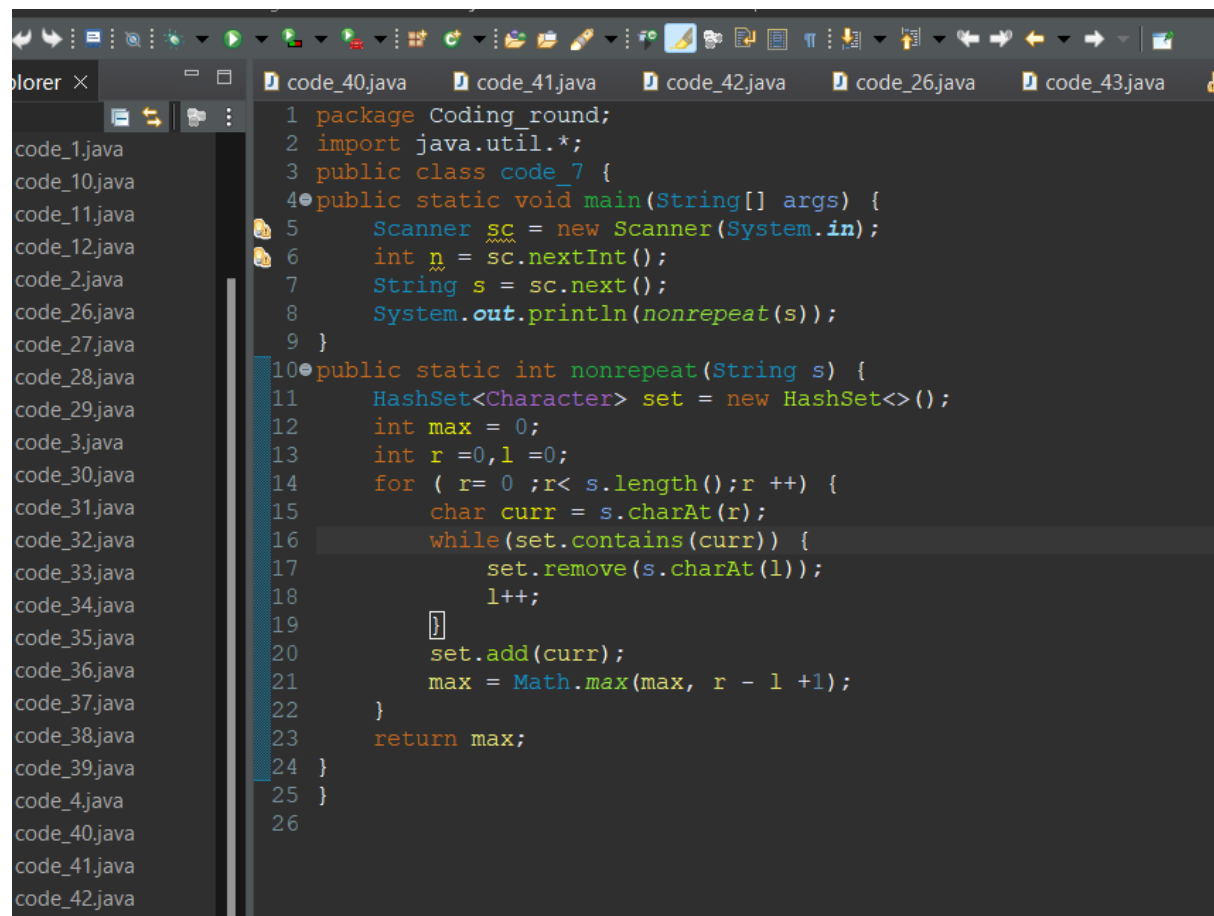
8

Sample Input 1

9
mjjlcldn

Sample Output 1

3



```
1 package Coding_round;
2 import java.util.*;
3 public class code_7 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int n = sc.nextInt();
7         String s = sc.next();
8         System.out.println(nonrepeat(s));
9     }
10    public static int nonrepeat(String s) {
11        HashSet<Character> set = new HashSet<>();
12        int max = 0;
13        int r = 0, l = 0;
14        for ( r = 0 ; r < s.length(); r++) {
15            char curr = s.charAt(r);
16            while(set.contains(curr)) {
17                set.remove(s.charAt(l));
18                l++;
19            }
20            set.add(curr);
21            max = Math.max(max, r - l + 1);
22        }
23        return max;
24    }
25 }
26
```

A4 Median of the Sorted Arrays (code_3)

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return the median of the two sorted arrays.

The overall run time complexity should be $O(\log(m+n))$.

Input Format

- The First Line contains two integers `N` and `M` denoting the size of two arrays respectively.
- The Second Line contains `N` integers denoting the numbers of elements in the array 1.
- The Third Line contains `M` integers denoting the numbers of elements in the array 2.

Constraints

- `nums1.length == n`
- `nums2.length == m`
- $0 \leq n \leq 100000$
- $0 \leq m \leq 100000$
- $1 \leq m + n \leq 200000$
- $-10^6 \leq \text{nums1}[i], \text{nums2}[i] \leq 10^6$

Output Format

Print the answer to the question.

Sample Input 0

```
9 7
-962 -811 -799 -436 -151 287 344 453 770
257 775 1592 3072 5093 6278 6502
```

Sample Output 0

398.5

Sample Input 1

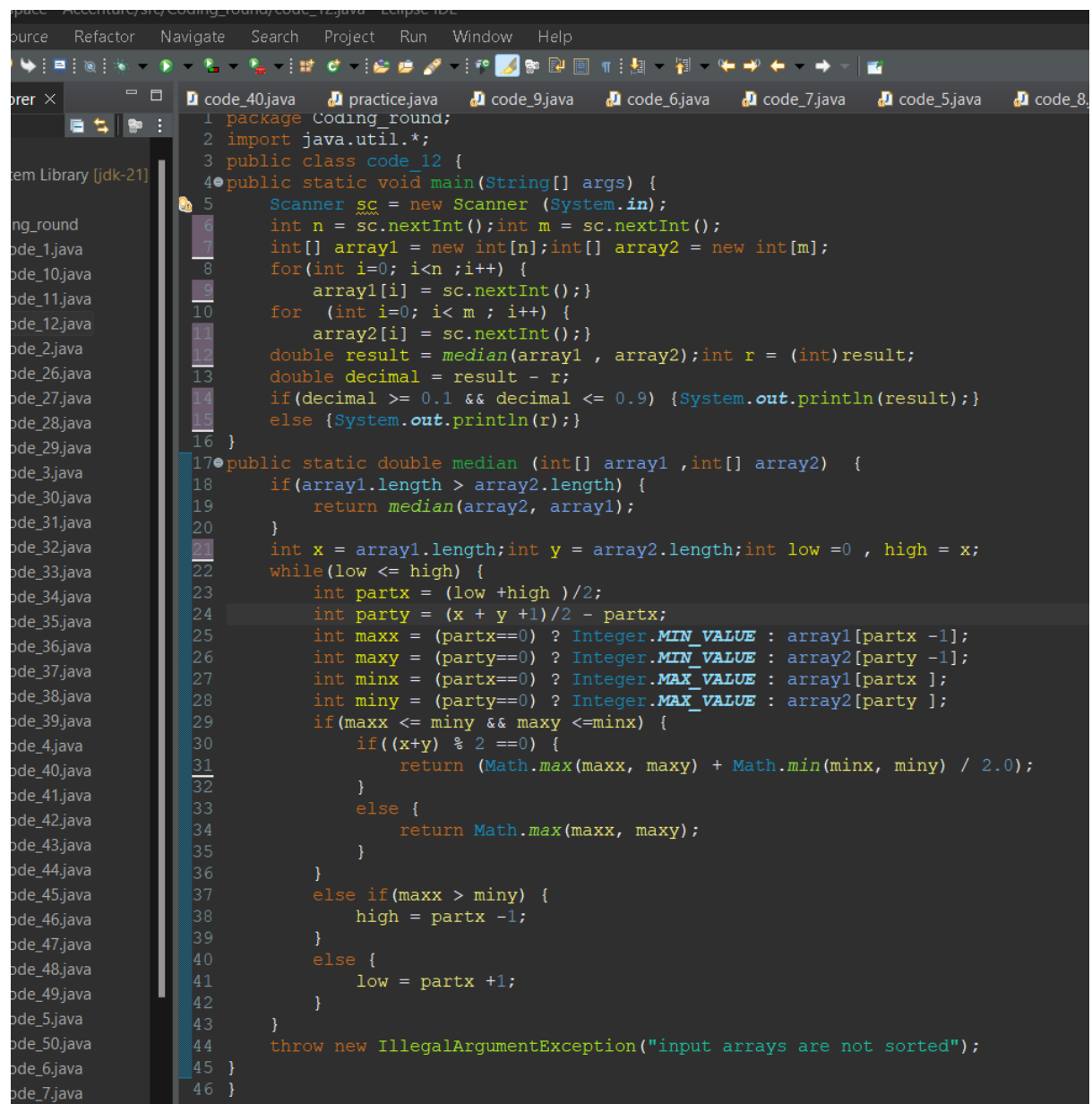
6 9

-931 -802 -648 -85 763 869

1684 1736 2634 3576 5301 7607 8221 9244 9479

Sample Output 1

1736



```
package Coding_round;
import java.util.*;
public class code_12 {
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        int n = sc.nextInt();int m = sc.nextInt();
        int[] array1 = new int[n];int[] array2 = new int[m];
        for(int i=0; i<n; i++) {
            array1[i] = sc.nextInt();
        }
        for (int i=0; i<m; i++) {
            array2[i] = sc.nextInt();
        }
        double result = median(array1 , array2);int r = (int)result;
        double decimal = result - r;
        if(decimal >= 0.1 && decimal <= 0.9) {System.out.println(result);}
        else {System.out.println(r);}
    }
    public static double median (int[] array1 ,int[] array2) {
        if(array1.length > array2.length) {
            return median(array2, array1);
        }
        int x = array1.length;int y = array2.length;int low =0 , high = x;
        while(low <= high) {
            int partx = (low +high )/2;
            int party = (x + y +1)/2 - partx;
            int maxx = (partx==0) ? Integer.MIN_VALUE : array1[partx -1];
            int maxy = (party==0) ? Integer.MIN_VALUE : array2[party -1];
            int minx = (partx==0) ? Integer.MAX_VALUE : array1[partx ];
            int miny = (party==0) ? Integer.MAX_VALUE : array2[party ];
            if(maxx <= miny && maxy <=minx) {
                if((x+y) % 2 ==0) {
                    return (Math.max(maxxx, maxy) + Math.min(minx, miny) / 2.0);
                }
                else {
                    return Math.max(maxxx, maxy);
                }
            }
            else if(maxxx > miny) {
                high = partx -1;
            }
            else {
                low = partx +1;
            }
        }
        throw new IllegalArgumentException("input arrays are not sorted");
    }
}
```

A6 Reverse Int (code_4)

Given a signed 32-bit integer x , return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range $[-2^{31}, 2^{31} - 1]$, then return 0.

Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

Input Format

Given an integer x

Constraints

$$-2^{31} \leq x \leq 2^{31} - 1$$

Output Format

Return an integer which is the answer to the question

Sample Input 0

404

Sample Output 0

404

Sample Input 1

123

Sample Output 1

321

```
code_9.java code_6.java code_7.java code_8.java
1 package Coding_round;
2 import java.util.*;
3 public class code_2 {
4
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7         int n = sc.nextInt();
8         System.out.println(rev(n));
9     }
10    public static int rev ( int n) {
11        int sign = n / Math.abs(n);
12        n = Math.abs(n);
13        int n2 =0;
14        while (n > 0) {
15            int rem = n%10;
16            n2 = n2 * 10 + rem;
17            n = n / 10;
18        }
19        return n2*sign;
20    }
21 }
22
```

A8 Palindrome (code_5)

Given an integer x, return true if x is a palindrome, and false otherwise.

Input Format

Contains a single integer

Constraints

$-2^{31} \leq x \leq 2^{31} - 1$

Output Format

Return either true or false.

Sample Input 0

```
12221
```

Sample Output 0

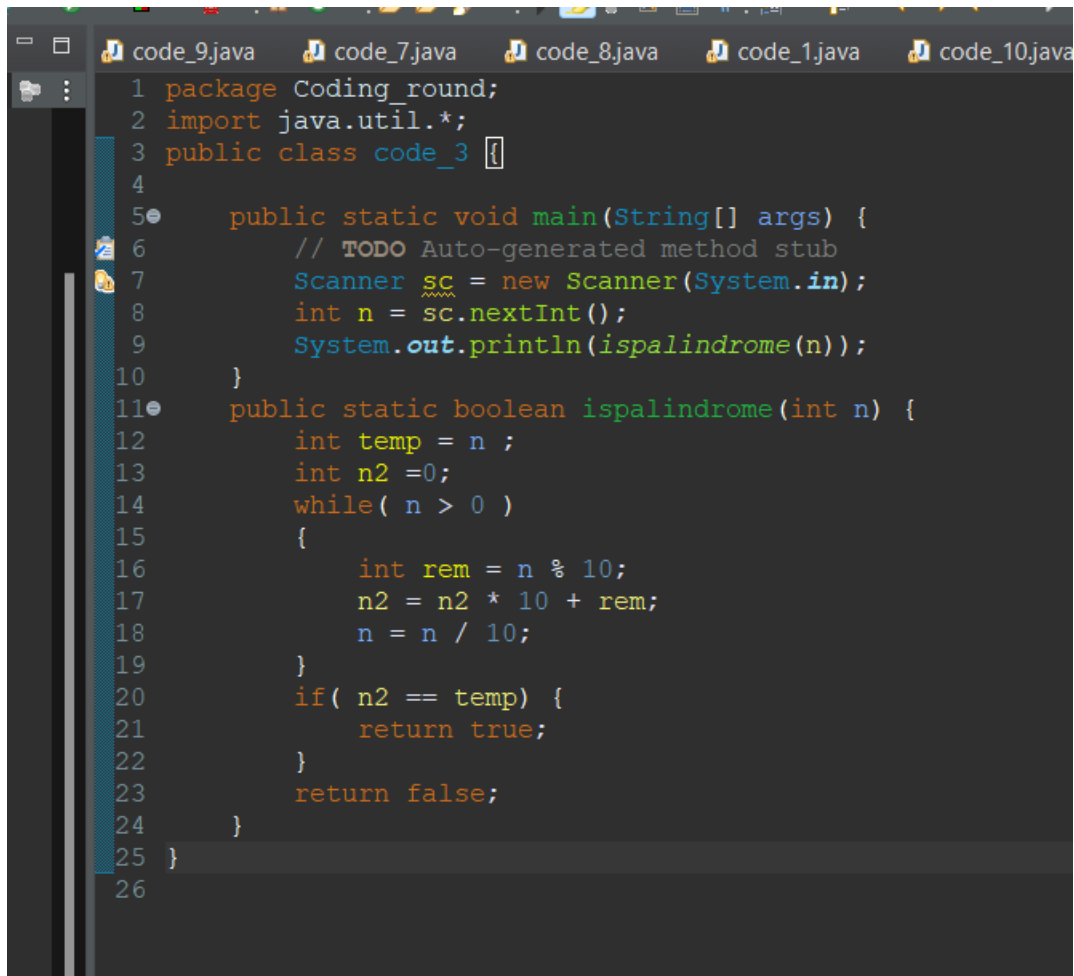
```
true
```

Sample Input 1

-23434

Sample Output 1

false

A screenshot of a Java IDE with a dark theme. The top of the window shows several tabs: 'code_9.java', 'code_7.java', 'code_8.java', 'code_1.java', and 'code_10.java'. The active tab is 'code_3.java'. The code is as follows:

```
1 package Coding_round;
2 import java.util.*;
3 public class code_3 {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Scanner sc = new Scanner(System.in);
8         int n = sc.nextInt();
9         System.out.println(ispalindrome(n));
10    }
11    public static boolean ispalindrome(int n) {
12        int temp = n ;
13        int n2 =0;
14        while( n > 0 )
15        {
16            int rem = n % 10;
17            n2 = n2 * 10 + rem;
18            n = n / 10;
19        }
20        if( n2 == temp) {
21            return true;
22        }
23        return false;
24    }
25 }
26
```

A12 Roman to Integer (code_6)

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

1. Symbol -> Value
2. I -> 1
3. V -> 5
4. X -> 10
5. L -> 50
6. C -> 100

7. D -> 500

8. M -> 1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Input Format

Given a single string

Constraints

- $1 \leq s.length \leq 15$
- s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M').
- It is guaranteed that s is a valid roman numeral in the range [1, 3999].

Output Format

Print an Integer

Sample Input 0

X

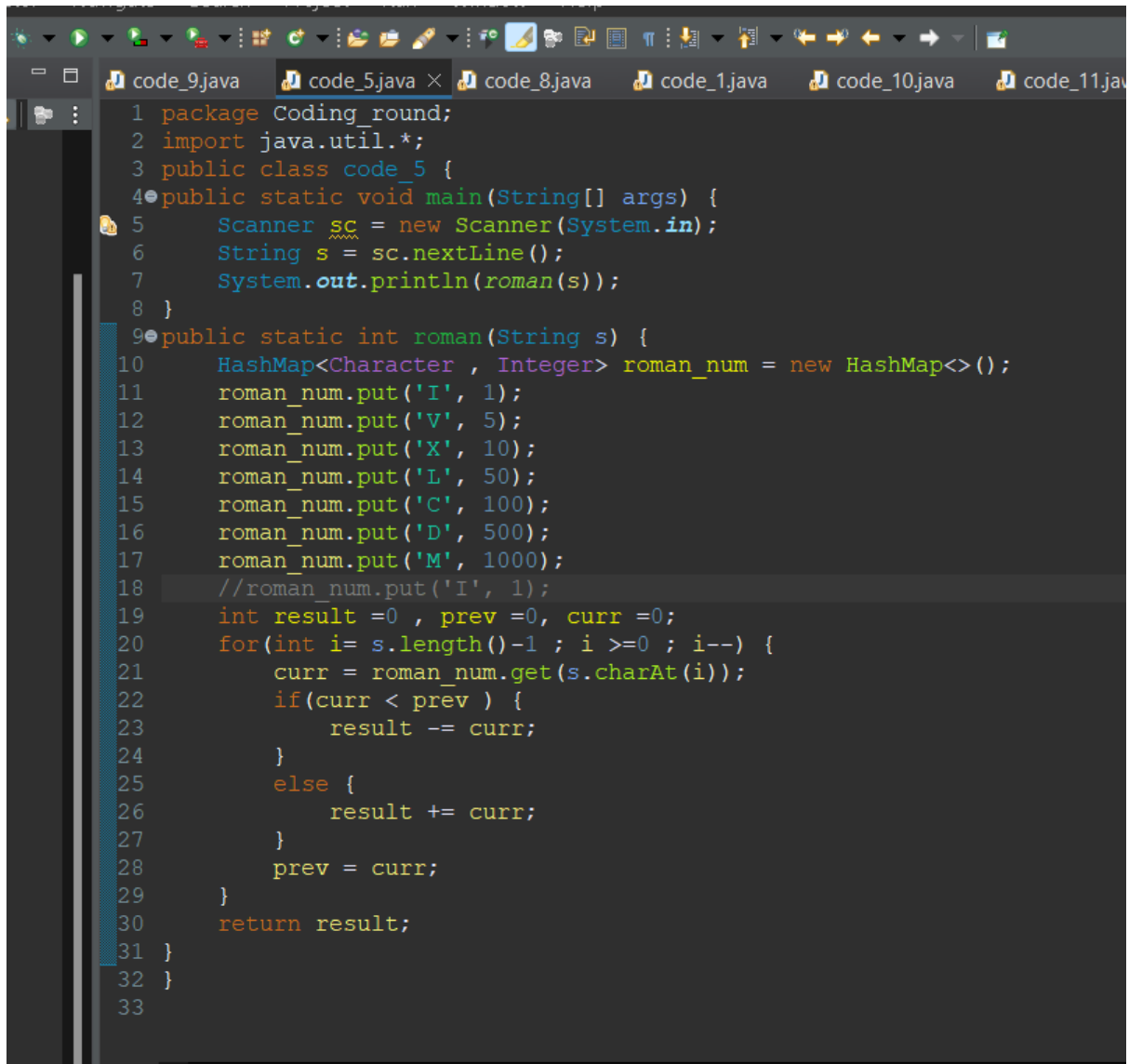
Sample Output 0

10

Sample Input 1

III

Sample Output 1



```

1 package Coding_round;
2 import java.util.*;
3 public class code_5 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         String s = sc.nextLine();
7         System.out.println(roman(s));
8     }
9     public static int roman(String s) {
10         HashMap<Character, Integer> roman_num = new HashMap<>();
11         roman_num.put('I', 1);
12         roman_num.put('V', 5);
13         roman_num.put('X', 10);
14         roman_num.put('L', 50);
15         roman_num.put('C', 100);
16         roman_num.put('D', 500);
17         roman_num.put('M', 1000);
18         //roman_num.put('I', 1);
19         int result = 0, prev = 0, curr = 0;
20         for(int i = s.length()-1 ; i >= 0 ; i--) {
21             curr = roman_num.get(s.charAt(i));
22             if(curr < prev ) {
23                 result -= curr;
24             }
25             else {
26                 result += curr;
27             }
28             prev = curr;
29         }
30         return result;
31     }
32 }
33

```

A13 Longest Common Prefix (code_7)

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

Input Format

- The first line contains the size of array

- The Second line contains an array of strings

Constraints

- $1 \leq \text{strs.length} \leq 200$
- $0 \leq \text{strs}[i].\text{length} \leq 200$
- $\text{strs}[i]$ consists of only lowercase English letters.

Output Format

Print a string which is the answer to the question

Sample Input 0

```
3
flow
flow
flower
```

Sample Output 0

```
flow
```

Sample Input 1

```
3
bwjepxu
beehgn
bevrwvov
```

Sample Output 1

```
b
```

```
code_9.java code_5.java code_8.java × code_1.java code_10.java
1 package Coding_round;
2 import java.util.*;
3 public class code_8 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int n = sc.nextInt();
7         String[] s = new String[n];
8         for (int i=0; i < n ; i++) {
9             s[i] = sc.next();
10        }
11        System.out.println(prefix(s));
12    }
13    }
14    public static String prefix(String[] s) {
15        if (s == null || s.length==0) {
16            return "";
17        }
18        String pre = s[0];
19        for(int i=1; i< s.length ; i++) {
20            while(s[i].indexOf(pre) !=0) {
21                pre = pre.substring(0 , pre.length()-1);
22                if(pre.isEmpty()) {
23                    return "";
24                }
25            }
26        }
27        return pre;
28    }
29 }
30
```

A27 Spiral Matrix (code_8)

Given an m x n matrix, return all elements of the matrix in spiral order.

Input Format

First line contains the integers N and M denoting the number of rows and columns in the matrix. The next N lines contains m integers in every line denoting the elements of the matrix

Constraints

- $m == \text{matrix.length}$
- $n == \text{matrix}[i].\text{length}$

- $1 \leq m, n \leq 1000$
- $-1000 \leq \text{matrix}[i][j] \leq 1000$

Output Format

Print the array of integers which is the spiral form for the given matrix.

Sample Input 0

```
4 3
-56 4 -73
-65 -38 97
-14 90 4
-90 -72 -56
```

Sample Output 0

```
-56 4 -73 97 4 -56 -72 -90 -14 -65 -38 90
```

Sample Input 1

```
6 6
85 -35 -70 43 -93 21
-65 41 -39 49 7 -68
-82 51 -46 -61 -63 96
-18 23 -60 -48 -17 -11
58 -41 47 -12 -21 -47
14 -5 -12 -11 27 48
```

Sample Output 1

```
85 -35 -70 43 -93 21 -68 96 -11 -47 48 27 -11 -12 -5 14 58 -18 -82 -65 41 -39 49 7 -63 -17 -21 -12 47
-41 23 51 -46 -61 -48 -60
```

```

code_40.java  practice.java  *code_9.java X  code_6.java  code_7.java  code_5.java  code_8.java
1 package Coding_round;
2 import java.util.*;
3 public class code_9 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner (System.in);
6         int n = 0, m =0;
7         n = sc.nextInt();
8         m = sc.nextInt();
9         int [][] matrix = new int[n][m];
10        for(int i =0; i< n ; i++) {
11            for(int j=0;j<m ; j++) {
12                matrix[i][j] = sc.nextInt();
13            }
14        }
15        List<Integer> result = spiral(matrix);
16        for(int num: result) {
17            System.out.print(num+ " ");
18        }
19    }
20    public static List<Integer> spiral(int[][] matrix) {
21        List<Integer> result = new ArrayList<>();
22        if(matrix == null || matrix.length == 0 || matrix[0].length==0) {
23            return result;
24        }
25        int top =0 , bottom = matrix.length -1, left=0 , right = matrix[0].length -1;
26        while (top <= bottom && left <= right) {
27            for(int i =left ; i<= right; i++) {
28                result.add(matrix[top][i]);
29            }top ++;
30            for(int i=top ; i<= bottom ;i++) {
31                result.add(matrix[i][right]);
32            }right --;
33            if(top <= bottom) {
34                for(int i = right; i>=left;i--) {
35                    result.add(matrix[bottom][i]);
36                }bottom --;
37            }
38            if(left <= right) {
39                for (int i= bottom ; i >=top ; i--) {
40                    result.add(matrix[i][left]);
41                }left ++;
42            }
43        }
44        return result;
45    }
46 }

```

A33 The staircase (code_9)

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Input Format

The first line contains an integer n

Constraints

$1 \leq n \leq 45$

Output Format

Print a integer which is the answer to the question

Sample Input 0

2

Sample Output 0

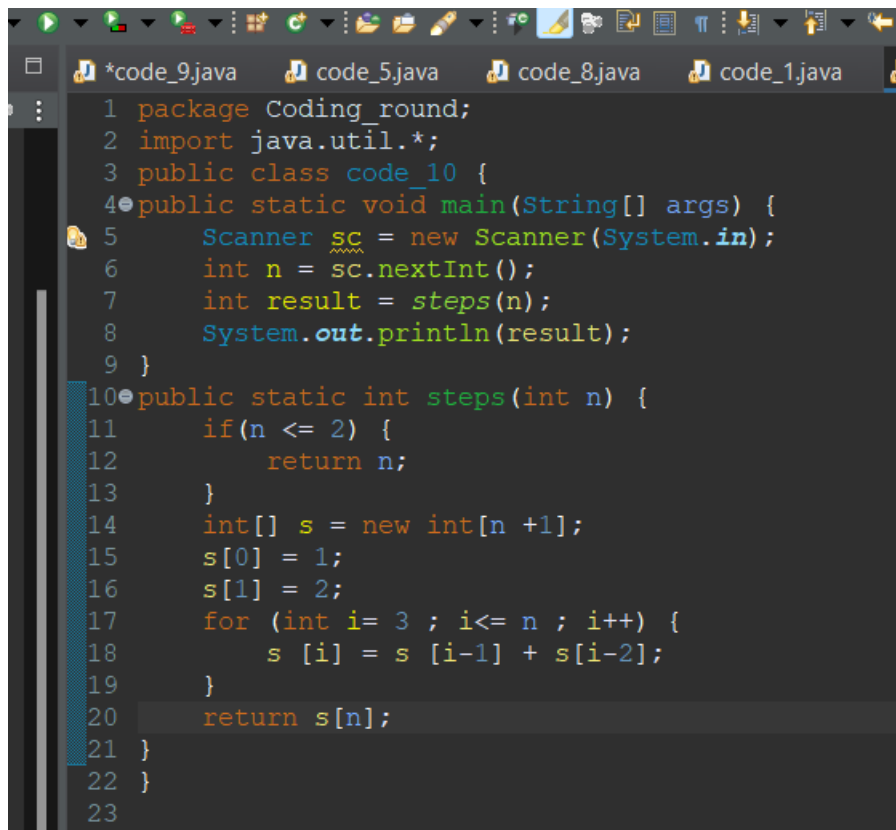
2

Sample Input 1

1

Sample Output 1

1

A screenshot of a Java IDE with a dark theme. The code is for a class named 'code_10' in the package 'Coding_round'. It imports 'java.util.*' and contains a 'main' method that uses a 'Scanner' to read an integer 'n' from 'System.in', calls a static method 'steps(n)', and prints the result. The 'steps' method is a recursive-like function that uses an array 's' of size 'n+1'. It has a base case where if 'n' is less than or equal to 2, it returns 'n'. Otherwise, it initializes 's[0] = 1' and 's[1] = 2', and then iterates from 'i = 3' to 'n', calculating 's[i] = s[i-1] + s[i-2]'. Finally, it returns 's[n]'.

```
1 package Coding_round;
2 import java.util.*;
3 public class code_10 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int n = sc.nextInt();
7         int result = steps(n);
8         System.out.println(result);
9     }
10    public static int steps(int n) {
11        if(n <= 2) {
12            return n;
13        }
14        int[] s = new int[n + 1];
15        s[0] = 1;
16        s[1] = 2;
17        for (int i = 3 ; i <= n ; i++) {
18            s[i] = s[i-1] + s[i-2];
19        }
20        return s[n];
21    }
22 }
23
```

A42 Max Profit in K Transactions (code_10)

You are given an integer array prices where prices[i] is the price of a given stock on the ith day, and an integer k.

Find the maximum profit you can achieve. You may complete at most k transactions.

Note: You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

Input Format

- The First Line contains two integers N and K denoting the size of Array nums and value of K.
- The Second Line contains 'N' integers denoting the elements of Array nums.

Constraints

- $1 \leq k \leq 100$
- $1 \leq \text{prices.length} \leq 1000$
- $0 \leq \text{prices}[i] \leq 1000$

Output Format

Print a integer which is the answer to the question

Sample Input 0

```
6 2
17 21 26 35 18 37
```

Sample Output 0

```
37
```

Sample Input 1

```
7 3
19 12 3 14 5 16 5
```

Sample Output 1

```
22
```



```
*code_9.java  code_5.java  code_8.java  code_1.java  code_10.java  code_11.java
1 package Coding_round;
2 import java.util.*;
3 public class code_11 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int n = sc.nextInt();
7         int k = sc.nextInt();
8         int[] prices = new int[n];
9         for (int i=0 ; i< n ; i++) {
10             prices[i] = sc.nextInt();
11         }
12         int[][] s = new int[k+1][n];
13         for (int i=0 ; i<= k ; i++) {
14             int max = -prices[0];
15             for(int j=0 ; j < n ; j++) {
16                 s[i][j] = Math.max(s[i][j-1], prices[j]+max);
17                 max = Math.max(max, s[i-1][j] - prices[j]);
18             }
19         }
20         System.out.println(s[k][n-1]);
21     }
22 }
23 }
```

A51 The Fibonacci (code_11)

The Fibonacci numbers, commonly denoted $F(n)$ form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

- $F(0) = 0, F(1) = 1$
- $F(n) = F(n - 1) + F(n - 2)$, for $n > 1$.

Given n , calculate $F(n)$.

Input Format

A single integer N .

Constraints

$0 \leq n \leq 30$

Output Format

Print a integer which is the answer to the question

Sample Input 0

3

Sample Output 0

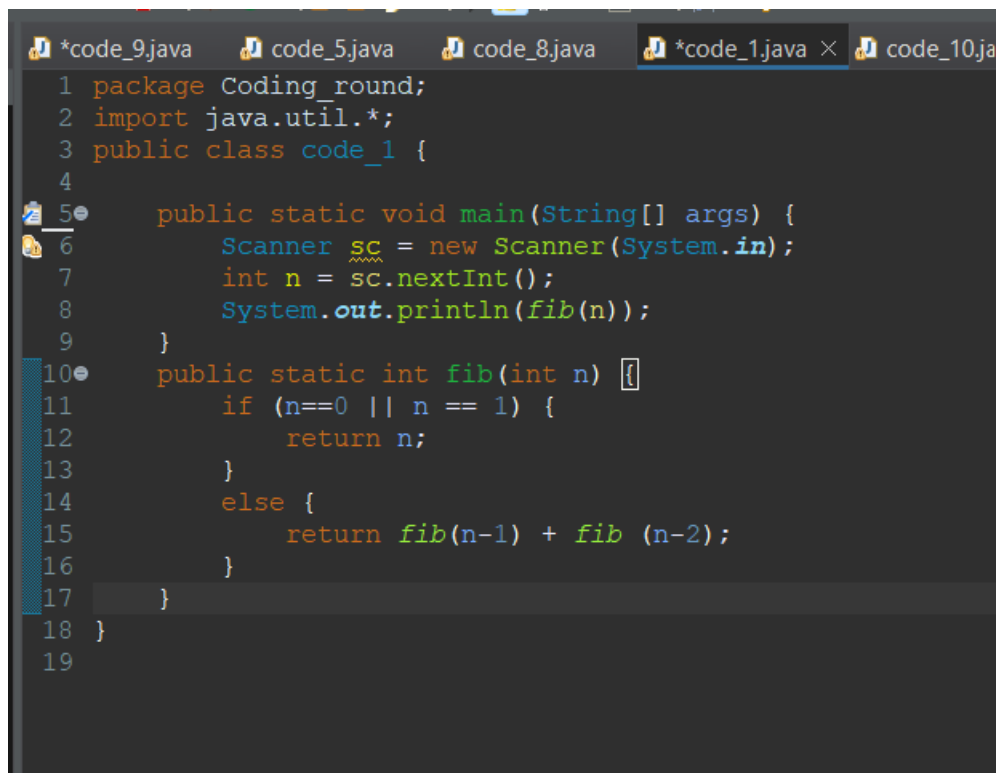
2

Sample Input 1

2

Sample Output 1

1



```
1 package Coding_round;
2 import java.util.*;
3 public class code_1 {
4
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7         int n = sc.nextInt();
8         System.out.println(fib(n));
9     }
10    public static int fib(int n) {
11        if (n==0 || n == 1) {
12            return n;
13        }
14        else {
15            return fib(n-1) + fib (n-2);
16        }
17    }
18 }
19
```

A59 Overlapping Rectangles (code_12)

An axis-aligned rectangle is represented as a list [x1, y1, x2, y2], where (x1, y1) is the coordinate of its bottom-left corner, and (x2, y2) is the coordinate of its top-right

corner. Its top and bottom edges are parallel to the X-axis, and its left and right edges are parallel to the Y-axis.

Two rectangles overlap if the area of their intersection is positive. To be clear, two rectangles that only touch at the corner or edges do not overlap.

Given two axis-aligned rectangles `rec1` and `rec2`, return `true` if they overlap, otherwise return `false`.

Input Format

The First line contains 4 integers representing the rectangle 1. The Second line contains 4 integers representing the rectangle 2.

Constraints

`rec1.length == 4` `rec2.length == 4` $-10^9 \leq \text{rec1}[i], \text{rec2}[i] \leq 10^9$ `rec 1` and `rec 2` represent a valid rectangle with a non-zero area.

Output Format

Print `true` if they overlap, otherwise print `false`.

Sample Input 0

```
0 0 1 1
1 0 2 1
```

Sample Output 0

```
false
```

Sample Input 1

```
1 1 3 3
1 1 2 2
```

Sample Output 1

```
true
```

```
*code_12.java ×
1 package Coding round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_12 {
6     public static boolean isRectangleOverlap(int[] rec1, int[] rec2) {
7         boolean left = rec1[2] <= rec2[0];
8         boolean right = rec1[0] >= rec2[2];
9         boolean below = rec1[3] <= rec2[1];
10        boolean above = rec1[1] >= rec2[3];
11        return !(left || right || below || above);
12    }
13
14    public static void main(String[] args) {
15        Scanner scanner = new Scanner(System.in);
16        int[] rec1 = new int[4];
17        int[] rec2 = new int[4];
18
19        for (int i = 0; i < 4; i++) {
20            rec1[i] = scanner.nextInt();
21        }
22
23        for (int i = 0; i < 4; i++) {
24            rec2[i] = scanner.nextInt();
25        }
26
27        boolean result = isRectangleOverlap(rec1, rec2);
28        System.out.println(result);
29    }
30 }
31
```

A61 Closest Person (code_13)

You are given an array representing a row of seats where `seats[i] = 1` represents a person sitting in the *i*th seat, and `seats[i] = 0` represents that the *i*th seat is empty (0-indexed).

There is at least one empty seat, and at least one person sitting.

Alex wants to sit in the seat such that the distance between him and the closest person to him is maximized.

Return that maximum distance to the closest person.

Input Format

- The First line contains a integer N denoting the size of array.
- The Second line contains N integers denoting the elements of array.

Constraints

- $2 \leq \text{seats.length} \leq 2 * 10^4$
- `seats[i]` is 0 or 1.
- At least one seat is empty.
- At least one seat is occupied.

Output Format

Print a integer which is the answer to the question.

Sample Input 0

```
3
0 1 0
```

Sample Output 0

```
1
```

Sample Input 1

```
4
1 0 0 0
```

Sample Output 1

```
3
```

```
*code_12.java  *code_13.java x
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_13 {
6     public static int maxDistToClosest(int[] seats) {
7         int maxDistance = 0;
8         int prevOccupiedSeat = -1;
9
10        for (int i = 0; i < seats.length; i++) {
11            if (seats[i] == 1) {
12                if (prevOccupiedSeat == -1) {
13                    maxDistance = i;
14                } else {
15                    maxDistance = Math.max(maxDistance, (i - prevOccupiedSeat) / 2);
16                }
17                prevOccupiedSeat = i;
18            }
19        }
20        maxDistance = Math.max(maxDistance, seats.length - 1 - prevOccupiedSeat);
21        return maxDistance;
22    }
23
24    public static void main(String[] args) {
25        Scanner scanner = new Scanner(System.in);
26        int n = scanner.nextInt();
27        int[] seats = new int[n];
28
29        for (int i = 0; i < n; i++) {
30            seats[i] = scanner.nextInt();
31        }
32
33        int result = maxDistToClosest(seats);
34        System.out.println(result);
35    }
36 }
37
```

A70 Daily Temperatures (code_14)

Given an array of integers temperatures represents the daily temperatures, return an array answer such that answer[i] is the number of days you have to wait after the ith day to get a warmer temperature. If there is no future day for which this is possible, keep answer[i] == 0 instead.

Input Format

- The first line contains a integer N denoting the size of array.
- The Second line contains N integers denoting the elements of array.

Constraints

- $1 \leq \text{temperatures.length} \leq 10^5$
- $30 \leq \text{temperatures}[i] \leq 100$

Output Format

Print the array of integers representing the array answer.

Sample Input 0

```
8
73 74 75 71 69 72 76 73
```

Sample Output 0

```
1 1 4 2 1 1 0 0
```

Sample Input 1

```
5
31 49 52 66 47
```

Sample Output 1

```
1 1 1 0 0
```

```
*code_14.java ×
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 //import java.util.*;
6
7 public class code_14 {
8     public static int[] dailyTemperatures(int[] temperatures) {
9         int n = temperatures.length;
10        int[] result = new int[n];
11        Stack<Integer> stack = new Stack<>();
12
13        for (int i = 0; i < n; i++) {
14            while (!stack.isEmpty() && temperatures[i] > temperatures[stack.peek()]) {
15                int prevIndex = stack.pop();
16                result[prevIndex] = i - prevIndex;
17            }
18            stack.push(i);
19        }
20
21        return result;
22    }
23
24    public static void main(String[] args) {
25        Scanner scanner = new Scanner(System.in);
26        int n = scanner.nextInt();
27        int[] temperatures = new int[n];
28
29        for (int i = 0; i < n; i++) {
30            temperatures[i] = scanner.nextInt();
31        }
32
33        int[] result = dailyTemperatures(temperatures);
34        for (int temp : result) {
35            System.out.print(temp + " ");
36        }
37    }
38 }
39
```

A72 Min Cost to Reach the Top (code_15)

You are given an integer array cost where cost[i] is the cost of ith step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1.

Return the minimum cost to reach the top of the floor.

Input Format

- The first line contains a integer N denoting the size of array.
- The Second line contains N integers denoting the elements of array.

Constraints

$2 \leq \text{cost.length} \leq 1000$ $0 \leq \text{cost}[i] \leq 999$

Output Format

Print a integer which is the answer to the question

Sample Input 0

```
5
6 7 8 9 12
```

Sample Output 0

```
16
```

Sample Input 1

```
3
10 15 20
```

Sample Output 1

```
15
```



```
*code_15.java x
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_15 {
6     public static int minCostClimbingStairs(int[] cost) {
7         int n = cost.length;
8         int[] s = new int[n];
9
10        s[0] = cost[0];
11        s[1] = cost[1];
12
13        for (int i = 2; i < n; i++) {
14            s[i] = cost[i] + Math.min(s[i - 1], s[i - 2]);
15        }
16
17        return Math.min(s[n - 1], s[n - 2]);
18    }
19
20    public static void main(String[] args) {
21        Scanner scanner = new Scanner(System.in);
22        int n = scanner.nextInt();
23        int[] cost = new int[n];
24
25        for (int i = 0; i < n; i++) {
26            cost[i] = scanner.nextInt();
27        }
28
29        int result = minCostClimbingStairs(cost);
30        System.out.println(result);
31    }
32 }
33
```

A82 Array Intersection (code_16)

Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must be unique and you may return the result in sorted order.

Input Format

- The first line contains two integers N and M.
- The Second line contains N integers representing the nums1 elements.
- The third line contains M integers representing the nums2 elements.

Constraints

- $1 \leq \text{nums1.length}, \text{nums2.length} \leq 100000$

- $0 \leq \text{nums1}[i], \text{nums2}[i] \leq 1000$

Output Format

Print the array of integers which is the answer to the question

Sample Input 0

```
4 2
1 2 2 1
2 3
```

Sample Output 0

```
2
```

Sample Input 1

```
2 3
1 2
2 3 1
```

Sample Output 1

```
1 2
```

```

1  *code_15.java  *code_16.java x
2  import java.io.*;
3  import java.util.*;
4  public class code_16 {
5      public static int[] intersection(int[] nums1, int[] nums2) {
6          Set<Integer> set1 = new HashSet<>();
7          for (int num : nums1) {
8              set1.add(num);
9          }
10
11          Set<Integer> resultSet = new HashSet<>();
12          for (int num : nums2) {
13              if (set1.contains(num)) {
14                  resultSet.add(num);
15              }
16          }
17          int[] result = new int[resultSet.size()];
18          int index = 0;
19          for (int num : resultSet) {
20              result[index] = num;
21              index++;
22          }
23          Arrays.sort(result);
24          return result;
25      }
26      public static void main(String[] args) {
27          Scanner scanner = new Scanner(System.in);
28          int N = scanner.nextInt();
29          int M = scanner.nextInt();
30          int[] nums1 = new int[N];
31          int[] nums2 = new int[M];
32
33          for (int i = 0; i < N; i++) {
34              nums1[i] = scanner.nextInt();
35          }
36
37          for (int i = 0; i < M; i++) {
38              nums2[i] = scanner.nextInt();
39          }
40
41          int[] result = intersection(nums1, nums2);
42          for (int num : result) {
43              System.out.print(num + " ");
44          }
45      }
46  }
47

```

A83 Sorted Intersection (code_17)

Given two integer arrays `nums1` and `nums2`, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in sorted order.

Input Format

- The first line contains two integers N and M.
- The Second line contains N integers representing the nums1 elements.
- The third line contains M integers representing the nums2 elements.

Constraints

$1 \leq \text{nums1.length}, \text{nums2.length} \leq 100000$ $0 \leq \text{nums1}[i], \text{nums2}[i] \leq 1000$

Output Format

Print the array of integers which is the answer to the question

Sample Input 0

```
3 5
4 9 5
9 4 9 8 4
```

Sample Output 0

```
4 9
```

Sample Input 1

```
2 3
1 2
2 3 1
```

Sample Output 1

```
1 2
```

```

1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4 public class code_17 {
5     public static int[] intersection(int[] nums1, int[] nums2) {
6         Map<Integer, Integer> countMap = new HashMap<>();
7
8         for (int num : nums1) {
9             countMap.put(num, countMap.getOrDefault(num, 0) + 1);
10        }
11        List<Integer> resultList = new ArrayList<>();
12        for (int num : nums2) {
13            if (countMap.containsKey(num) && countMap.get(num) > 0) {
14                resultList.add(num);
15                countMap.put(num, countMap.get(num) - 1);
16            }
17        }
18        int[] result = new int[resultList.size()];
19        int index = 0;
20        for (int num : resultList) {
21            result[index] = num;
22            index++;
23        }
24        Arrays.sort(result);
25        return result;
26    }
27    public static void main(String[] args) {
28        Scanner scanner = new Scanner(System.in);
29        int N = scanner.nextInt();
30        int M = scanner.nextInt();
31        int[] nums1 = new int[N];
32        int[] nums2 = new int[M];
33
34        for (int i = 0; i < N; i++) {
35            nums1[i] = scanner.nextInt();
36        }
37        for (int i = 0; i < M; i++) {
38            nums2[i] = scanner.nextInt();
39        }
40        int[] result = intersection(nums1, nums2);
41        for (int num : result) {
42            System.out.print(num + " ");
43        }
44    }
45 }

```

A84 Most Frequent Elements (code_18)

Given an integer array `nums` and an integer `k`, return the `k` most frequent elements. You must return the answer in sorted order.

Input Format

- The First Line contains 2 integers N and K.
- The Next line contains N integers denoting the array elements.

Constraints

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- k is in the range [1, the number of unique elements in the array].
- It is guaranteed that the answer is unique.

Output Format

Print the array of integers which is the answer to the question.

Sample Input 0

```
6 2
1 1 1 2 2 3
```

Sample Output 0

```
1 2
```

Sample Input 1

```
5 2
3 5 3 1 4
```

Sample Output 1

```
3 5
```

```

1 package Coding_round;
2 import java.util.*;
3 public class code_18 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int n = sc.nextInt();
7         int n1 = sc.nextInt();
8         HashSet<Integer> set = new HashSet<>();
9         int a[] = new int[n+2];
10        a[0] = n;
11        a[1] = n1;
12        for(int i=0 ; i< a.length;i++) {
13            a[i] =sc.nextInt();
14        }
15        Arrays.sort(a);
16        for(int i=0 ; i<a.length;i++) {
17            for(int j =i+1 ; i< a.length;j++) {
18                if(a[i]== a[j]) {
19                    set.add(a[i]);
20                }
21            }
22        }
23        for(int i:set) {
24            System.out.print(i+" ");
25        }
26    }
27 }
28

```

A87 Fizzy or Buzzy? (code_19)

Given an integer n, return a string array answer (1-indexed) where:

- `answer[i] == "FizzBuzz"` if i is divisible by 3 and 5.
- `answer[i] == "Fizz"` if i is divisible by 3.
- `answer[i] == "Buzz"` if i is divisible by 5.
- `answer[i] == i` (as a string) if none of the above conditions are true.

Input Format

The First line contains a integer N.

Constraints

$1 \leq n \leq 10^4$

Output Format

Print the array of strings as answer.

Sample Input 0

3

Sample Output 0

1 2 Fizz

Sample Input 1

2

Sample Output 1

1 2

```
*code_18.java  *code_19.java ×
1 package Coding_round;
2 import java.util.*;
3 public class code_19 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int n = sc.nextInt();
7         for(int i=0; i<= n ;i++) {
8             if(i%3 ==0 && i %5 ==0) {
9                 System.out.print("Fizz Buzz");
10            }
11            else if(i% 3 == 0){
12                System.out.print("Fizz");
13            }
14            else if(i % 5 ==0) {
15                System.out.print("Buzz");
16            }
17            else {
18                System.out.print(i+ " ");
19            }
20        }
21    }
22 }
23
```

A89 Sorted Anagram (code_20)

Given two strings s and p , return an array of all the start indices of p 's anagrams in s . You should return the answer in a sorted order.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Input Format

- The First line contains a string s .
- The Second line contains a string p .

Constraints

- $1 \leq s.length, p.length \leq 3 * 10^4$
- s and p consist of lowercase English letters.

Output Format

Print the array of integers which is the answer to the question

```

1 package Coding_round;
2 import java.util.ArrayList;
3
4 public class code_20 {
5     public static List<Integer> findAnagrams(String s, String p) {
6         List<Integer> result = new ArrayList<>();
7
8         if (s == null || s.length() == 0 || p == null || p.length() == 0) {
9             return result;
10        }
11        int[] pCount = new int[26];
12        for (char c : p.toCharArray()) {
13            pCount[c - 'a']++;
14        }
15        int left = 0; int right = 0;
16        int sLength = s.length(); int pLength = p.length();
17        int[] sCount = new int[26];
18        while (right < sLength) {
19            sCount[s.charAt(right) - 'a']++;
20            if (right - left + 1 > pLength) {
21                sCount[s.charAt(left) - 'a']--;
22                left++;
23            }
24            if (right - left + 1 == pLength && matches(sCount, pCount)) {
25                result.add(left);
26            }
27            right++;
28        }
29        return result;
30    }
31    private static boolean matches(int[] sCount, int[] pCount) {
32        for (int i = 0; i < 26; i++) {
33            if (sCount[i] != pCount[i]) {
34                return false;
35            }
36        }
37        return true;
38    }
39    public static void main(String[] args) {
40        Scanner sc = new Scanner(System.in);
41        String s1 = sc.next();
42        String p1 = sc.next();
43        List<Integer> result1 = findAnagrams(s1, p1);
44        for (int index : result1) {
45            System.out.print(index + " ");
46        }
47    }
48 }

```

A95 Not in the Range (code_21)

Given an array nums of n integers where nums[i] is in the range [1, n], return an array of all the integers in the range [1, n] that do not appear in nums.

Input Format

- The First line contains a integer N denoting the size of array.

- The Next line contains N integers denoting the elements of the array.

Constraints

- `n == nums.length`
- `1 <= n <= 10^5`
- `1 <= nums[i] <= n`

Output Format

Print the array of integers which is the answer to the question

Sample Input 0

```
8
4 3 2 7 8 2 3 1
```

Sample Output 0

```
5 6
```

Sample Input 1

```
3
1 2 2
```

Sample Output 1

```
3
```

```

1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4 public class code_21 {
5     public static List<Integer> findDisappearedNumbers(int[] nums) {
6         List<Integer> result = new ArrayList<>();
7
8         for (int num : nums) {
9             int index = Math.abs(num) - 1;
10            if (nums[index] > 0) {
11                nums[index] = -nums[index];
12            }
13        }
14
15        for (int i = 0; i < nums.length; i++) {
16            if (nums[i] > 0) {
17                result.add(i + 1);
18            }
19        }
20
21        return result;
22    }
23
24    public static void main(String[] args) {
25        Scanner scanner = new Scanner(System.in);
26        int n = scanner.nextInt();
27        int[] nums = new int[n];
28
29        for (int i = 0; i < n; i++) {
30            nums[i] = scanner.nextInt();
31        }
32
33        List<Integer> result = findDisappearedNumbers(nums);
34        for (int num : result) {
35            System.out.print(num + " ");
36        }
37    }
38 }
39

```

H11: Game of Abhishek and Badri (code_22)

Abhishek and Badri are playing a game in which they are having a array 'A' of size 'N' that contains positive integers in it.

Before the game begins, Abhishek chooses an integer $k \geq 0$. The game lasts for k stages, the stages are numbered from 1 to k . During the i -th stage, Abhishek must remove an element from the array that is less than or equal to $k-i+1$. After that, if the array is not empty, Badri must add $k-i+1$ to an arbitrary element of the array. Note that both Abhishek's move and Badri's move are two parts of the same stage of the game. If

Abhishek can't delete an element during some stage, he loses. If the k -th stage ends and Abhishek hasn't lost yet, he wins.

Your task is to determine the maximum value of k such that Abhishek can win if both players play optimally. Badri plays against Abhishek, so he tries to make him lose the game, if it's possible.

Input Format

The first line contains a single integer n — the size of the array A . The second line contains n integers a_1, a_2, \dots, a_n .

Constraints

- $1 \leq n \leq 100$
- $1 \leq a[i] \leq n$

Output Format

Print one integer — the maximum value of k such that Abhishek can win if both players play optimally.

Sample Input 0

```
4
4 1 4 1
```

Sample Output 0

```
1
```

Sample Input 1

```
8
5 1 6 7 3 3 5 4
```

Sample Output 1

```
1
```

```
*code_21.java  *code_22.java x
1 package Coding_round;
2 import java.util.*;
3 import java.util.Arrays;
4 public class code_22 {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7         int n = sc.nextInt();
8         int[] a = new int[n];
9         for(int i=0 ; i < n;i++) {
10             a[i] = sc.nextInt();
11         }
12         int maxk = findmaxk(a);
13         System.out.println(maxk);
14     }
15     public static int findmaxk(int[] a) {
16         Arrays.sort(a);
17         int maxk=0;
18         for(int i=0 ; i<a.length;i++) {
19             if(a[i] <= maxk+1) {
20                 maxk =a[i];
21             }
22         }
23         return maxk;
24     }
25 }
26
```

A18 The Permutation of an array (code_23)

A permutation of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].

The next permutation of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the next permutation of that array is the permutation that follows it in the sorted container. If such arrangement

is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

- For example, the next permutation of $arr = [1,2,3]$ is $[1,3,2]$.
- Similarly, the next permutation of $arr = [2,3,1]$ is $[3,1,2]$.
- While the next permutation of $arr = [3,2,1]$ is $[1,2,3]$ because $[3,2,1]$ does not have a lexicographical larger rearrangement.

Given an array of integers `nums`, find the next permutation of `nums`.

The replacement must be in place and use only constant extra memory.

Input Format

- The First line contains an integer `N` denoting the size of array `A`.
- The second line denotes `N` integers denoting the elements of array `A`.

Constraints

- $1 \leq \text{nums.length} \leq 10000$
- $0 \leq \text{nums}[i] \leq 1000$

Output Format

Print the array of integers which is the next permutation of the given array

Sample Input 0

```
6
352 520 9 356 692 171
```

Sample Output 0

```
352 520 9 692 171 356
```

Sample Input 1

```
6
218 30 183 660 68 503
```

Sample Output 1

```
218 30 183 660 503 68
```

```

1 package Coding_round;
2 import java.util.Scanner;
3 public class code_23 {
4     public static void nextPermutation(int[] nums) {
5         int n = nums.length;
6         int i = n - 2;
7         while (i >= 0 && nums[i] >= nums[i + 1]) {
8             i--;
9         }
10        if (i == -1) {
11            reverse(nums, 0, n - 1);
12            return;
13        }
14        int j = n - 1;
15        while (nums[j] <= nums[i]) {
16            j--;
17        }
18        swap(nums, i, j);
19        reverse(nums, i + 1, n - 1);
20    }
21    public static void swap(int[] nums, int i, int j) {
22        int temp = nums[i];
23        nums[i] = nums[j];
24        nums[j] = temp;
25    }
26    public static void reverse(int[] nums, int start, int end) {
27        while (start < end) {
28            swap(nums, start, end);
29            start++;
30            end--;
31        }
32    }
33    public static void main(String[] args) {
34        Scanner scanner = new Scanner(System.in);
35        int n = scanner.nextInt();
36        int[] nums = new int[n];
37
38        for (int i = 0; i < n; i++) {
39            nums[i] = scanner.nextInt();
40        }
41        nextPermutation(nums);
42        for (int i = 0; i < n; i++) {
43            System.out.print(nums[i] + " ");
44        }
45    }
46 }

```

A20 Smallest Missing Integer (code_24)

Gopi likes randomness and always wants to store things in a random order. He applies this method in datastructures too. He would always want to store elements in random

fashion(Unsorted). One day he was told that few elements of the array are missing, and he wants to find the smallest one in quick time. Since he is busy in a random walk, you being his friend should help him find the smallest missing element.

Given an unsorted integer array `nums`, return the smallest missing positive integer.

You must implement an algorithm that runs in $O(n)$ time and uses constant extra space.

Input Format

- The first line contains a integer `N`
- The second line contains `N` integers denoting the elements of array `nums`.

Constraints

- $1 \leq \text{nums.length} \leq 10^5$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$

Output Format

Print a integer which is the answer to the question.

Sample Input 0

```
9
6 7 8 8 3 2 7 5 8
```

Sample Output 0

```
1
```

Sample Input 1

```
8
1 2 2 3 4 3 2 4
```

Sample Output 1

```
5
```

```
*code_24.java ×
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_24 {
6     public int firstMissingPositive(int[] nums) {
7         int n = nums.length;
8         for (int i = 0; i < n; i++) {
9             if (nums[i] <= 0) {
10                 nums[i] = n + 1;
11             }
12         }
13         for (int i = 0; i < n; i++) {
14             int num = Math.abs(nums[i]);
15             if (num <= n) {
16                 nums[num - 1] = -Math.abs(nums[num - 1]);
17             }
18         }
19         for (int i = 0; i < n; i++) {
20             if (nums[i] > 0) {
21                 return i + 1;
22             }
23         }
24         return n + 1;
25     }
26     public static void main(String[] args) {
27         Scanner scanner = new Scanner(System.in);
28         int n = scanner.nextInt();
29         int[] nums = new int[n];
30         for (int i = 0; i < n; i++) {
31             nums[i] = scanner.nextInt();
32         }
33
34         code_24 solution = new code_24();
35         int result = solution.firstMissingPositive(nums);
36         System.out.println(result);
37     }
38 }
39
```

A26 Largest Sum (code_25)

Given an integer array nums, find the subarray with the largest sum, and return its sum.

Input Format

The first line contains an integer N The second line contains N integers denoting the value of array nums

Constraints

$1 \leq \text{nums.length} \leq 10^5$ $-10^4 \leq \text{nums}[i] \leq 10^4$

Output Format

Print the integer which is the answer to the question

Sample Input 0

7
4 -1 -5 -5 4 -9 -9

Sample Output 0

4

Sample Input 1

7
2 5 -9 -6 7 -2 3

Sample Output 1

8

```
*code_24.java  *code_25.java ×
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_25 {
6     public static int maxSubArray(int[] nums) {
7         int maxSum = nums[0];
8         int currentSum = nums[0];
9
10        for (int i = 1; i < nums.length; i++) {
11            currentSum = Math.max(nums[i], currentSum + nums[i]);
12            maxSum = Math.max(maxSum, currentSum);
13        }
14
15        return maxSum;
16    }
17
18    public static void main(String[] args) {
19        Scanner scanner = new Scanner(System.in);
20        int n = scanner.nextInt();
21        int[] nums = new int[n];
22
23        for (int i = 0; i < n; i++) {
24            nums[i] = scanner.nextInt();
25        }
26
27        int result = maxSubArray(nums);
28        System.out.println(result);
29    }
30 }
31
```

A47 Ratan Walks Home (code -26)

Ratan Kumar walks home and wants to remember all the house numbers on the way back to home.

He wants to arrange all the house numbers that fall on the way to his home, from his office, in sorted order.

He hates shortcuts and would like to arrange the numbers in the traditional way,(without using built-in functions).

Given an array of integers nums, sort the array in ascending order and return it.

You must solve the problem without using any built-in functions in $O(n\log(n))$ time complexity and with the smallest space complexity possible.

Input Format

- The First Line contains a value N denoting the size of array
- The Second Line contains N integers denoting the elements of array.

Constraints

- $1 \leq \text{nums.length} \leq 5 * 10^4$
- $-5 * 10^4 \leq \text{nums}[i] \leq 5 * 10^4$

Output Format

Print an array of integers in a sorted manner.

Sample Input 0

```
7
1 6 7 2 4 5 7
```

Sample Output 0

```
1 2 4 5 6 7 7
```

Sample Input 1

```
4
7 1 3 8
```

Sample Output 1

```
1 3 7 8
```

```
code_1.java  code_2.java  code_3.java  code_4.java  code_5.java
1 package Coding_round;
2 import java.util.*;
3 import java.util.Arrays;
4 public class code_25 {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7         int n = sc.nextInt();
8         int[] array = new int[n];
9         for(int i=0 ; i< n ;i++) {
10             array[i] = sc.nextInt();
11         }
12         Arrays.sort(array);
13         for(int i =0; i< n; i++) {
14             System.out.print(array[i] + " ");
15         }
16     }
17 }
18
```

A92 The Compression (code_27)

Given an array of characters chars, compress it using the following algorithm:

Begin with an empty string s. For each group of consecutive repeating characters in chars:

If the group's length is 1, append the character to s. Otherwise, append the character followed by the group's length. The compressed string s should not be returned separately, but instead, be stored in the input character array chars. Note that group lengths that are 10 or longer will be split into multiple characters in chars.

After you are done modifying the input array, return the new length of the array.

You must write an algorithm that uses only constant extra space.

Input Format

- The First Line contains a integer N denoting the size of array.
- The Next line contains N characters representing the array elements.

Constraints

- $1 \leq \text{chars.length} \leq 8000$
- `chars[i]` is a lowercase English letter, uppercase English letter, digit, or symbol.

Output Format

Print a integer which is the answer to the question

Sample Input 0

```
7
aabbccc
```

Sample Output 0

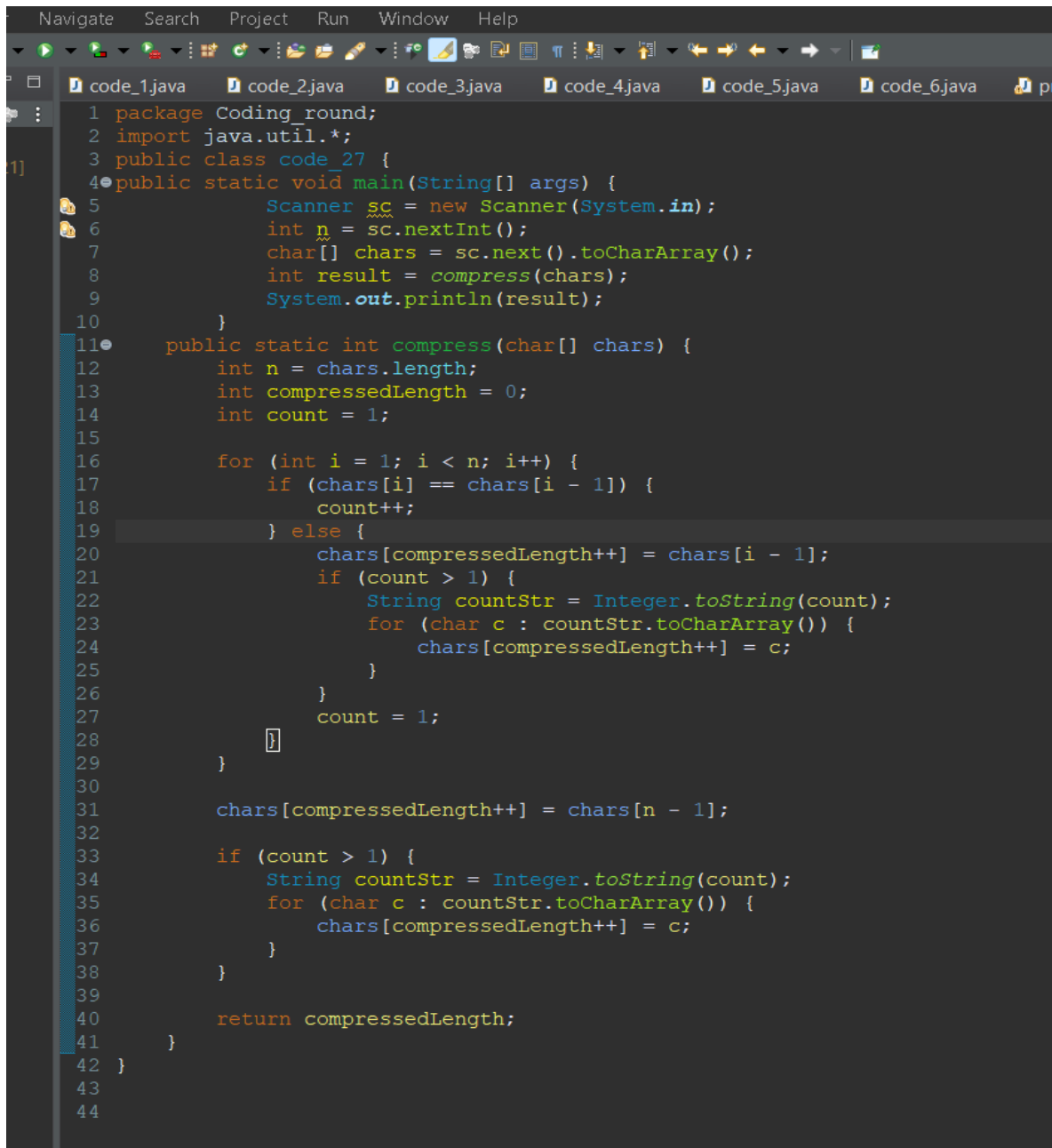
```
6
```

Sample Input 1

```
1
a
```

Sample Output 1

```
1
```

A screenshot of an IDE window showing a Java file named 'code_27.java'. The code is a public class 'code_27' with a 'main' method and a 'compress' method. The 'main' method uses a 'Scanner' to read an integer 'n' and a character array 'chars' from 'System.in'. It then calls the 'compress' method and prints the result. The 'compress' method takes a character array 'chars' and returns an integer 'compressedLength'. It iterates through the array, counting consecutive identical characters. When a change is detected, it appends the count and the character to the result array. Finally, it appends the last count and character and returns the total length of the compressed array.

```
1 package Coding_round;
2 import java.util.*;
3 public class code_27 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int n = sc.nextInt();
7         char[] chars = sc.next().toCharArray();
8         int result = compress(chars);
9         System.out.println(result);
10    }
11    public static int compress(char[] chars) {
12        int n = chars.length;
13        int compressedLength = 0;
14        int count = 1;
15
16        for (int i = 1; i < n; i++) {
17            if (chars[i] == chars[i - 1]) {
18                count++;
19            } else {
20                chars[compressedLength++] = chars[i - 1];
21                if (count > 1) {
22                    String countStr = Integer.toString(count);
23                    for (char c : countStr.toCharArray()) {
24                        chars[compressedLength++] = c;
25                    }
26                }
27                count = 1;
28            }
29        }
30
31        chars[compressedLength++] = chars[n - 1];
32
33        if (count > 1) {
34            String countStr = Integer.toString(count);
35            for (char c : countStr.toCharArray()) {
36                chars[compressedLength++] = c;
37            }
38        }
39
40        return compressedLength;
41    }
42 }
43
44
```

A118 Palindrome by char deletion (code_28)

Given a string *s*, return true if the *s* can be palindrome after deleting at most one character from it.

Input Format

The Input contains a single string *s*

Constraints

- $1 \leq s.length \leq 1e5$
- s consists of lowercase English letters.

Output Format

Print true if the s can be palindrome after deleting at most one character from it otherwise false.

Sample Input 0

aba

Sample Output 0

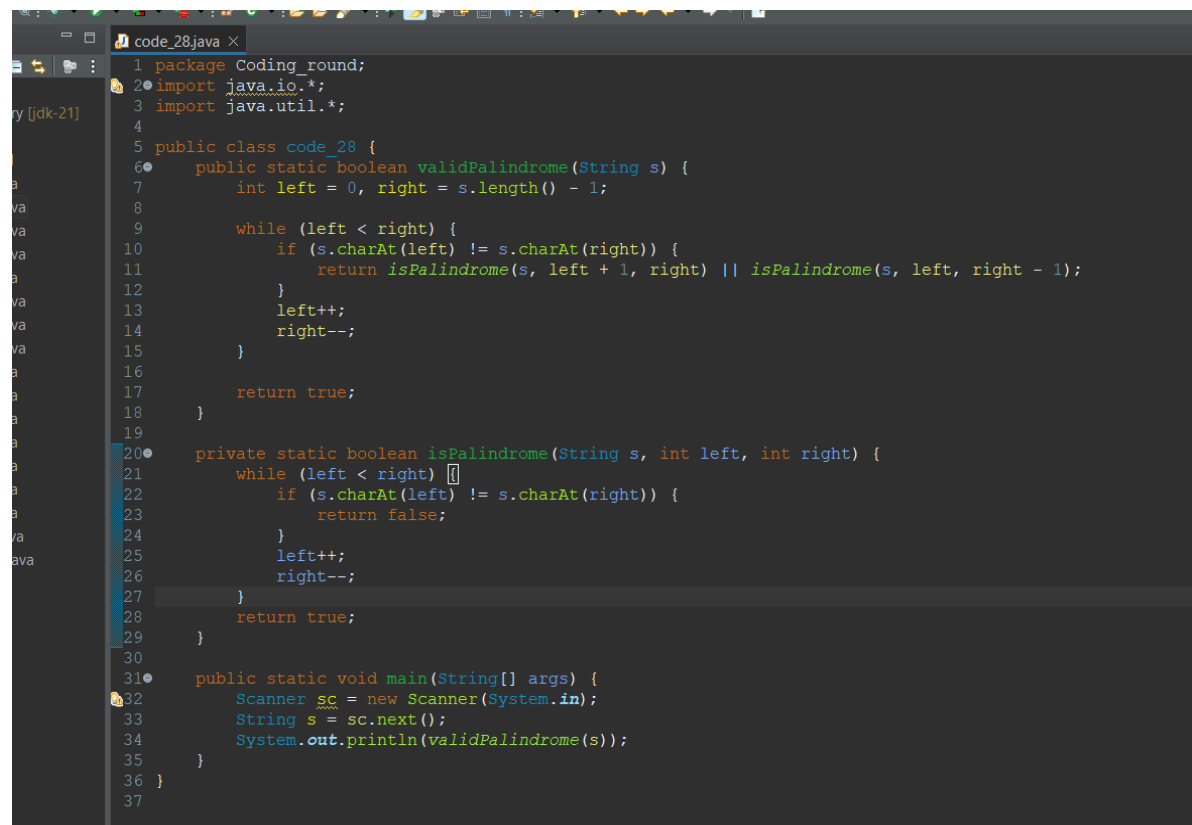
true

Sample Input 1

abc

Sample Output 1

false



```
code_28.java X
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_28 {
6     public static boolean validPalindrome(String s) {
7         int left = 0, right = s.length() - 1;
8
9         while (left < right) {
10             if (s.charAt(left) != s.charAt(right)) {
11                 return isPalindrome(s, left + 1, right) || isPalindrome(s, left, right - 1);
12             }
13             left++;
14             right--;
15         }
16
17         return true;
18     }
19
20     private static boolean isPalindrome(String s, int left, int right) {
21         while (left < right) {
22             if (s.charAt(left) != s.charAt(right)) {
23                 return false;
24             }
25             left++;
26             right--;
27         }
28         return true;
29     }
30
31     public static void main(String[] args) {
32         Scanner sc = new Scanner(System.in);
33         String s = sc.next();
34         System.out.println(validPalindrome(s));
35     }
36 }
37
```


A120 Kth Positive Integer (code_29)

Given an array `arr` of positive integers sorted in a strictly increasing order, and an integer `k`.

Return the `k`th positive integer that is missing from this array.

Input Format

- The First Line contains two integers `N` and `K`
- The Next Line contains `N` integers separated by a space

Constraints

- $1 \leq \text{arr.length} \leq 1000$
- $1 \leq \text{arr}[i] \leq 1000$
- $1 \leq k \leq 1000$
- $\text{arr}[i] < \text{arr}[j]$ for $1 \leq i < j \leq \text{arr.length}$

Output Format

Print a integer as answer to the question

Sample Input 0

```
4 2
1 2 3 4
```

Sample Output 0

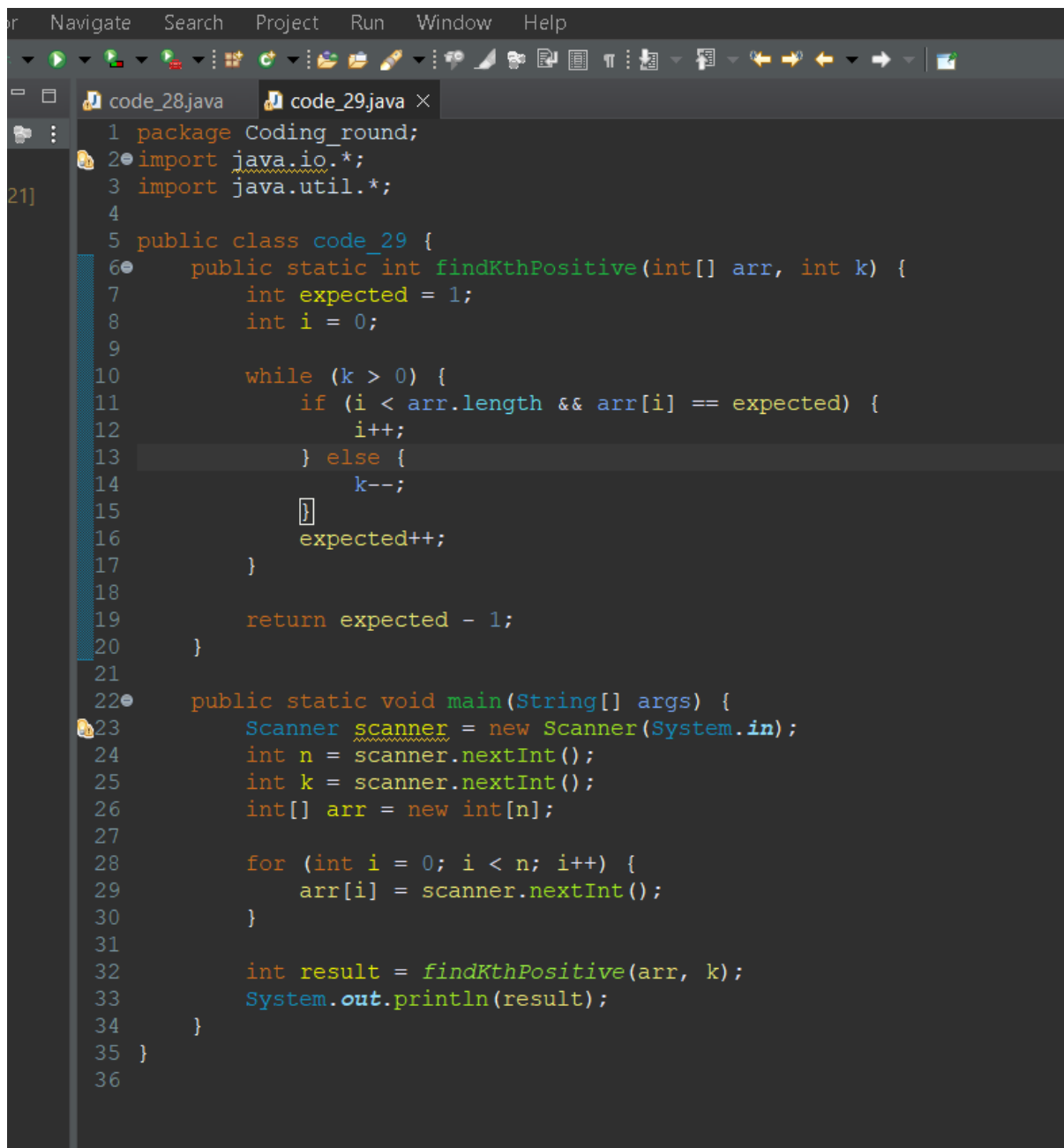
```
6
```

Sample Input 1

```
5 5
2 3 4 7 11
```

Sample Output 1

```
9
```

A screenshot of an IDE window showing two tabs: 'code_28.java' and 'code_29.java'. The 'code_29.java' tab is active, displaying a Java program. The code defines a package 'Coding_round', imports 'java.io.*' and 'java.util.*', and defines a public class 'code_29'. Inside the class, there is a static method 'findKthPositive' that takes an integer array 'arr' and an integer 'k' as input. It uses a while loop to find the kth positive integer in the array. The 'main' method uses a 'Scanner' to read input 'n' and 'k', creates an array 'arr' of size 'n', and then calls 'findKthPositive' to find the result, which is printed using 'System.out.println'.

```
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_29 {
6     public static int findKthPositive(int[] arr, int k) {
7         int expected = 1;
8         int i = 0;
9
10        while (k > 0) {
11            if (i < arr.length && arr[i] == expected) {
12                i++;
13            } else {
14                k--;
15            }
16            expected++;
17        }
18
19        return expected - 1;
20    }
21
22    public static void main(String[] args) {
23        Scanner scanner = new Scanner(System.in);
24        int n = scanner.nextInt();
25        int k = scanner.nextInt();
26        int[] arr = new int[n];
27
28        for (int i = 0; i < n; i++) {
29            arr[i] = scanner.nextInt();
30        }
31
32        int result = findKthPositive(arr, k);
33        System.out.println(result);
34    }
35 }
36
```

A119 Walk on the Path (code_30)

Given a string path, where path[i] = 'N', 'S', 'E' or 'W', each representing moving one unit north, south, east, or west, respectively. You start at the origin (0, 0) on a 2D plane and walk on the path specified by path.

Return true if the path crosses itself at any point, that is, if at any time you are on a location you have previously visited. Return false otherwise.

Input Format

The Input contains a single string

Constraints

- $1 \leq \text{path.length} \leq 14$
- `path[i]` is either 'N', 'S', 'E', or 'W'.

Output Format

Print true if the path crosses itself at any point, that is, if at any time you are on a location you have previously visited. Return false otherwise.

```
Navigate Search Project Run Window Help
code_28.java code_29.java code_30.java x
1 package Coding_round;
2 import java.util.HashSet;
3 import java.util.Scanner;
4
5 public class code_30 {
6     public static boolean isPathCrossing(String path) {
7         int x = 0, y = 0;
8         HashSet<String> visited = new HashSet<>();
9         visited.add("0,0");
10        for (char move : path.toCharArray()) {
11            if (move == 'N') {
12                y++;
13            } else if (move == 'S') {
14                y--;
15            } else if (move == 'E') {
16                x++;
17            } else if (move == 'W') {
18                x--;
19            }
20
21            String currentPosition = x + "," + y;
22            if (visited.contains(currentPosition)) {
23                return true;
24            }
25
26            visited.add(currentPosition);
27        }
28
29        return false; // The path does not cross itself
30    }
31
32    public static void main(String[] args) {
33        Scanner sc = new Scanner(System.in);
34        String path = sc.nextLine();
35
36        boolean result = isPathCrossing(path);
37        System.out.println(result);
38    }
39 }
40
```

A122 Operations on Array (code_31)

You are given two 0-indexed integer arrays `nums` and `multipliers` of size `n` and `m` respectively, where `n >= m`.

You begin with a score of 0. You want to perform exactly m operations. On the i th operation (0-indexed) you will:

- Choose one integer x from either the start or the end of the array `nums`.
- Add `multipliers[i] * x` to your score.
- Note that `multipliers[0]` corresponds to the first operation, `multipliers[1]` to the second operation, and so on.
- Remove x from `nums`.

Return the maximum score after performing m operations.

Input Format

- The First line contains two integers N and M .
- The Next Line contains N integers representing the elements of the array `nums`
- The Next Line contains M integers representing the elements of the array `multipliers`

Constraints

- $n == \text{nums.length}$
- $m == \text{multipliers.length}$
- $1 \leq m \leq 300$
- $m \leq n \leq 1e5$
- $-1000 \leq \text{nums}[i], \text{multipliers}[i] \leq 1000$

Output Format

Print a integer as a answer to the question

```
code_28.java code_29.java code_30.java *code_31.java X
1 package Coding_round;
2 import java.util.Scanner;
3
4 public class code_31 {
5     public static int maxScore(int[] nums, int[] multipliers) {
6         int n = nums.length;
7         int m = multipliers.length;
8         int[][] dp = new int[m + 1][m + 1];
9         for (int k = 1; k <= m; k++) {
10             for (int i = 0; i <= k; i++) {
11                 int j = k - i;
12                 int pickStart = nums[i] * multipliers[k - 1];
13                 int pickEnd = nums[n - j - 1] * multipliers[k - 1];
14                 if (i > 0) {
15                     pickStart += dp[i - 1][j];
16                 }
17                 if (j > 0) {
18                     pickEnd += dp[i][j - 1];
19                 }
20                 dp[i][j] = Math.max(pickStart, pickEnd);
21             }
22         }
23
24         return dp[m][m];
25     }
26
27     public static void main(String[] args) {
28         Scanner sc = new Scanner(System.in);
29         int n = sc.nextInt();
30         int m = sc.nextInt();
31
32         int[] nums = new int[n];
33         int[] multipliers = new int[m];
34
35         for (int i = 0; i < n; i++) {
36             nums[i] = sc.nextInt();
37         }
38
39         for (int i = 0; i < m; i++) {
40             multipliers[i] = sc.nextInt();
41         }
42
43         int result = maxScore(nums, multipliers);
44         System.out.println(result);
45     }
46 }
```

A125 : Min steps to reach the last (code_32)

Given an array of integers arr, you are initially positioned at the first index of the array.

In one step you can jump from index i to index:

- i + 1 where: i + 1 < arr.length.

- $i - 1$ where: $i - 1 \geq 0$.
- j where: $\text{arr}[i] == \text{arr}[j]$ and $i \neq j$.

Return the minimum number of steps to reach the last index of the array.

Notice that you can not jump outside of the array at any time.

Input Format

- The First Line contains a integer N
- The Next Line contains N integers seperated by a space.

Constraints

- $1 \leq \text{arr.length} \leq 5 * 10^4$
- $-10^8 \leq \text{arr}[i] \leq 10^8$

Output Format

Print a integer as the answer to the question

Sample Input 0

```
8
7 6 9 6 9 6 9 7
```

Sample Output 0

```
1
```

Sample Input 1

```
1
7
```

Sample Output 1

```
0
```

```

code_28.java code_29.java code_30.java *code_31.java code_11.java *code_32.java
1 package Coding_round;
2 import java.util.Scanner;
3
4 public class code_32 {
5     public static int minSteps(int[] arr) {
6         int n = arr.length;
7         int[] dp = new int[n];
8
9         for (int i = 1; i < n; i++) {
10             dp[i] = Integer.MAX_VALUE;
11             for (int j = 0; j < i; j++) {
12                 if (arr[i] == arr[j] && i != j) {
13                     dp[i] = Math.min(dp[i], dp[j] + 1);
14                 }
15             }
16             dp[i] = Math.min(dp[i], dp[i - 1] + 1);
17         }
18
19         return dp[n - 1];
20     }
21
22     public static void main(String[] args) {
23         Scanner scanner = new Scanner(System.in);
24         int n = scanner.nextInt();
25         int[] arr = new int[n];
26
27         for (int i = 0; i < n; i++) {
28             arr[i] = scanner.nextInt();
29         }
30
31         int result = minSteps(arr);
32         System.out.println(result);
33     }
34 }
35

```

A129 Sum up to a Target (code_33)

Given a 1-indexed array of integers numbers that is already sorted in non-decreasing order, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where $1 \leq \text{index1} < \text{index2} \leq \text{numbers.length}$.

Return the indices of the two numbers, index1 and index2, added by one as an integer array [index1, index2] of length 2.

The tests are generated such that there is exactly one solution. You may not use the same element twice.

Your solution must use only constant extra space.

Input Format

- The First line contains a integer N and T denoting the size of array and value of target respectively.
- The Next line contains N integers denoting the elements of the array.

Constraints

- $2 \leq \text{numbers.length} \leq 3 * 10^4$
- $-1000 \leq \text{numbers}[i] \leq 1000$
- numbers is sorted in non-decreasing order.
- $-1000 \leq \text{target} \leq 1000$
- The tests are generated such that there is exactly one solution.

Output Format

Print the two integers which is the answer to the question

Sample Input 0

```
3 6
2 3 4
```

Sample Output 0

```
1 3
```

Sample Input 1

```
4 9
2 7 11 15
```

Sample Output 1

```
1 2
```

```
code_28.java code_29.java code_30.java *code_31.java code_11.java *code_32.java *code_33.java
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_33 {
6     public static int[] twoSum(int[] numbers, int target) {
7         int left = 0;
8         int right = numbers.length - 1;
9
10        while (left < right) {
11            int sum = numbers[left] + numbers[right];
12
13            if (sum == target) {
14                return new int[] { left + 1, right + 1 };
15            } else if (sum < target) {
16                left++;
17            } else {
18                right--;
19            }
20        }
21
22        return new int[] {-1, -1}; // No solution found
23    }
24
25    public static void main(String[] args) {
26        Scanner scanner = new Scanner(System.in);
27        int n = scanner.nextInt();
28        int target = scanner.nextInt();
29        int[] numbers = new int[n];
30
31        for (int i = 0; i < n; i++) {
32            numbers[i] = scanner.nextInt();
33        }
34
35        int[] result = twoSum(numbers, target);
36        System.out.println(result[0] + " " + result[1]);
37    }
38 }
39
```

A133 Next Greater Number (code_34)

Given a circular integer array nums (i.e., the next element of nums[nums.length - 1] is nums[0]), return the next greater number for every element in nums.

The next greater number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return -1 for this number.

Input Format

- The First line contains a integer N denoting the size of array.
- The Next line contains N integers denoting the elements of the array.

Constraints

- $1 \leq \text{nums.length} \leq 5 \cdot 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

Output Format

Print the array of integers as the answer to the question

Sample Input 0

```
3
1 2 1
```

Sample Output 0

```
2 -1 2
```

Sample Input 1

```
5
1 2 3 4 3
```

Sample Output 1

```
2 3 4 -1 4
```

```

code_28.java code_29.java code_30.java *code_31.java code_11.java *code_
1 package Coding_Round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_34 {
6     public static int[] nextGreaterElements(int[] nums) {
7         int n = nums.length;
8         int[] result = new int[n];
9         Arrays.fill(result, -1);
10        Stack<Integer> stack = new Stack<>();
11
12        for (int i = 0; i < 2 * n; i++) {
13            int num = nums[i % n];
14            while (!stack.isEmpty() && nums[stack.peek()] < num) {
15                result[stack.pop()] = num;
16            }
17            if (i < n) {
18                stack.push(i);
19            }
20        }
21
22        return result;
23    }
24
25    public static void main(String[] args) {
26        Scanner scanner = new Scanner(System.in);
27        int n = scanner.nextInt();
28        int[] nums = new int[n];
29
30        for (int i = 0; i < n; i++) {
31            nums[i] = scanner.nextInt();
32        }
33
34        int[] result = nextGreaterElements(nums);
35        for (int num : result) {
36            System.out.print(num + " ");
37        }
38    }
39 }
40

```

A139 Return the Missing Number (code_35)

Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array.

Input Format

- The first line contains a integer N denoting the size of array.
- The Next line contains N integers representing the elements of the array.

Constraints

- $n == \text{nums.length}$
- $1 \leq n \leq 2 \cdot 10^5$
- $0 \leq \text{nums}[i] \leq n$
- All the numbers of nums are unique.

Output Format

Print a integer which is the answer to the question.

Sample Input 0

```
3
3 0 1
```

Sample Output 0

```
2
```

Sample Input 1

```
2
0 1
```

Sample Output 1

```
2
```

```
*code_33.java  *code_34.java  *code_35.java ×
1 package Coding_round;
2 import java.util.Scanner;
3
4 public class code_35 {
5     public static int findMissingNumber(int[] nums) {
6         int missing = nums.length;
7         for (int i = 0; i < nums.length; i++) {
8             missing ^= i ^ nums[i];
9         }
10        return missing;
11    }
12
13    public static void main(String[] args) {
14        Scanner scanner = new Scanner(System.in);
15        int n = scanner.nextInt();
16        int[] nums = new int[n];
17
18        for (int i = 0; i < n; i++) {
19            nums[i] = scanner.nextInt();
20        }
21
22        int result = findMissingNumber(nums);
23        System.out.println(result);
24    }
25 }
26
```

A138 Check for the Anagram (code_36)

Given two strings s and t, return true if t is an anagram of s, and false otherwise.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Input Format

- The First Line contains a string s
- The Second Line contains a string t.

Constraints

- $1 \leq s.length, t.length \leq 8 * 10^4$
- s and t consist of lowercase English letters.

Output Format

Print true if t is an anagram of s, and false otherwise.

Sample Input 0

anagram
nagaram

Sample Output 0

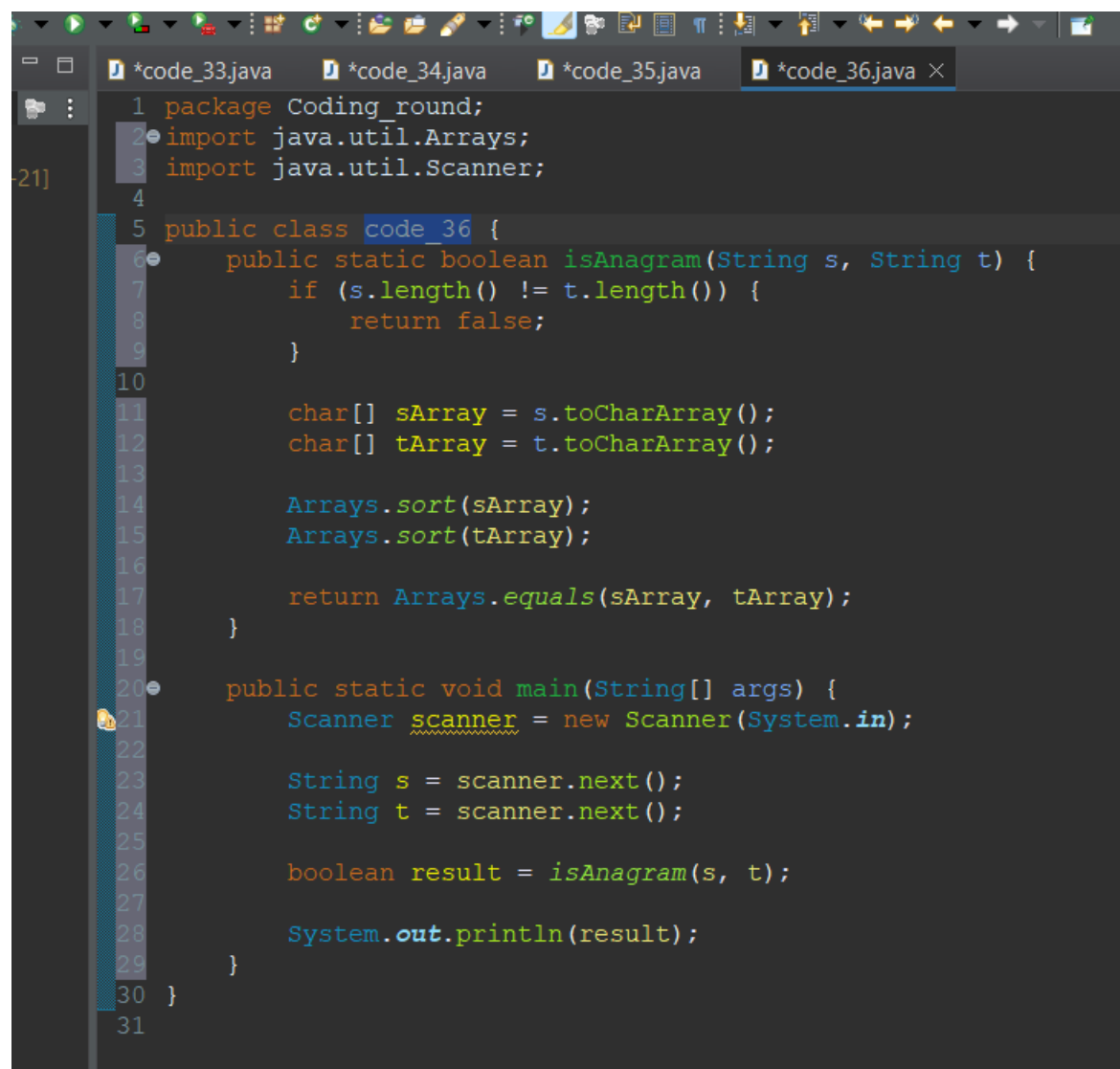
true

Sample Input 1

rat
car

Sample Output 1

false



```
1 package Coding_round;
2 import java.util.Arrays;
3 import java.util.Scanner;
4
5 public class code_36 {
6     public static boolean isAnagram(String s, String t) {
7         if (s.length() != t.length()) {
8             return false;
9         }
10
11         char[] sArray = s.toCharArray();
12         char[] tArray = t.toCharArray();
13
14         Arrays.sort(sArray);
15         Arrays.sort(tArray);
16
17         return Arrays.equals(sArray, tArray);
18     }
19
20     public static void main(String[] args) {
21         Scanner scanner = new Scanner(System.in);
22
23         String s = scanner.next();
24         String t = scanner.next();
25
26         boolean result = isAnagram(s, t);
27
28         System.out.println(result);
29     }
30 }
31
```

A137 Longest Contiguous Subarray

(code_37)

Given a binary array `nums`, return the maximum length of a contiguous subarray with an equal number of 0 and 1.

Input Format

- The First line contains a integer `N`
- The Next Line contains `N` integers.

Constraints

- $1 \leq \text{nums.length} \leq 10^5$
- `nums[i]` is either 0 or 1.

Output Format

print a integer which is the answer to the question.

Sample Input 0

```
8
1 1 0 0 0 0 1
```

Sample Output 0

```
4
```

Sample Input 1

```
4
1 1 0 0
```

Sample Output 1

```
4
```



```
*code_33.java *code_34.java *code_35.java *code_36.java *code_37.java x
1 package Coding_round;
2 import java.util.HashMap;
3 import java.util.Scanner;
4
5 public class code_37 {
6     public static int findMaxLength(int[] nums) {
7         int maxLen = 0;
8         int count = 0;
9         HashMap<Integer, Integer> countMap = new HashMap<>();
10        countMap.put(0, -1);
11
12        for (int i = 0; i < nums.length; i++) {
13            if (nums[i] == 0) {
14                count--;
15            } else {
16                count++;
17            }
18
19            if (countMap.containsKey(count)) {
20                maxLen = Math.max(maxLen, i - countMap.get(count));
21            } else {
22                countMap.put(count, i);
23            }
24        }
25
26        return maxLen;
27    }
28
29    public static void main(String[] args) {
30        Scanner scanner = new Scanner(System.in);
31        int n = scanner.nextInt();
32        int[] nums = new int[n];
33
34        for (int i = 0; i < n; i++) {
35            nums[i] = scanner.nextInt();
36        }
37
38        int result = findMaxLength(nums);
39        System.out.println(result);
40    }
41 }
```

A140 Solo Repeated Number (code_38)

Given an array of integers nums containing $n + 1$ integers where each integer is in the range $[1, n]$ inclusive.

There is only one repeated number in nums, return this repeated number.

You must solve the problem without modifying the array nums and uses only constant extra space.

Input Format

- The First line contains a integer N
- The Next line contains N +1 integers representing the array elements.

Constraints

- $1 \leq n \leq 10^5$
- `nums.length == n + 1`
- $1 \leq \text{nums}[i] \leq n$
- All the integers in `nums` appear only once except for precisely one integer which appears two or more times.

Output Format

Print a integer which is the answer to the question

Sample Input 0

```
4
1 3 4 2 2
```

Sample Output 0

```
2
```

Sample Input 1

```
2
1 1 2
```

Sample Output 1

```
1
```

```
*code_33.java *code_34.java *code_35.java *code_36.java *code_37.java *code_38.java
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_38 {
6     public static int findDuplicate(int[] nums) {
7         int slow = nums[0];
8         int fast = nums[0];
9         do {
10             slow = nums[slow];
11             fast = nums[nums[fast]];
12         } while (slow != fast);
13         slow = nums[0];
14         while (slow != fast) {
15             slow = nums[slow];
16             fast = nums[fast];
17         }
18
19         return slow;
20     }
21
22     public static void main(String[] args) {
23         Scanner scanner = new Scanner(System.in);
24         int n = scanner.nextInt();
25         int[] nums = new int[n + 1];
26
27         for (int i = 0; i <= n; i++) {
28             nums[i] = scanner.nextInt();
29         }
30
31         int result = findDuplicate(nums);
32         System.out.println(result + " ");
33     }
34 }
35
```

A142 Solve the Equation (code_39)

Given an array A of integers and integer K, return the maximum S such that there exists i < j with $A[i] + A[j] = S$ and $S < K$. If no i, j exist satisfying this equation, return -1.

Input Format

- The First line contains two integers N and K.
- The Next line contains N integer representing the array elements.

Constraints

- $1 \leq \text{nums.length} \leq 6 \cdot 10^4$
- $1 \leq \text{nums}[i] \leq 1000$

- $1 \leq k \leq 2000$

Output Format

Print a integer which is the answer to the question.

Sample Input 0

```
3 15
10 20 30
```

Sample Output 0

```
-1
```

Sample Input 1

```
8 60
34 23 1 24 75 33 54 8
```

Sample Output 1

```
58
```

```

1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_39 {
6     public static int maxSumLessThanK(int[] nums, int k) {
7         Arrays.sort(nums);
8         int left = 0;
9         int right = nums.length - 1;
10        int maxSum = -1;
11
12        while (left < right) {
13            int sum = nums[left] + nums[right];
14            if (sum < k) {
15                maxSum = Math.max(maxSum, sum);
16                left++;
17            } else {
18                right--;
19            }
20        }
21
22        return maxSum;
23    }
24
25    public static void main(String[] args) {
26        Scanner scanner = new Scanner(System.in);
27        int n = scanner.nextInt();
28        int k = scanner.nextInt();
29        int[] nums = new int[n];
30
31        for (int i = 0; i < n; i++) {
32            nums[i] = scanner.nextInt();
33        }
34
35        int result = maxSumLessThanK(nums, k);
36        System.out.println(result);
37    }
38 }
39

```

A143 Least Number of a Perfect Square (code_40)

Given an integer n , return the least number of perfect square numbers that sum to n .

A perfect square is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not.

Input Format

The only input contains a integer N.

Constraints

$1 \leq n \leq 10^4$.

Output Format

Print a integer which is the answer to the question

Sample Input 0

15

Sample Output 0

4

Sample Input 1

13

Sample Output 1

2

```
code_33.java code_34.java code_35.java code_36.java code_37.java
1 package Coding_round;
2 import java.util.Scanner;
3
4 public class code_40 {
5     public static int numSquares(int n) {
6         int[] s = new int[n + 1];
7
8         for (int i = 1; i <= n; i++) {
9             s[i] = i;
10
11             for (int j = 1; j * j <= i; j++) {
12                 s[i] = Math.min(s[i], s[i - j * j] + 1);
13             }
14         }
15
16         return s[n];
17     }
18
19     public static void main(String[] args) {
20         Scanner scanner = new Scanner(System.in);
21         int n = scanner.nextInt();
22
23         int result = numSquares(n);
24         System.out.println(result);
25     }
26 }
27
```

A147 Binary Representation (code_41)

Given an integer n , return an array `ans` of length $n + 1$ such that for each i ($0 \leq i \leq n$), `ans[i]` is the number of 1's in the binary representation of i .

Input Format

The only line contains a number.

Constraints

$0 \leq n \leq 10^5$

Output Format

Print the array of integers as answer to the question

Sample Input 0

```
5
```

Sample Output 0

```
0 1 1 2 1 2
```

Sample Input 1

```
2
```

Sample Output 1

```
0 1 1
```

```
code_33.java code_34.java code_35.java code_36.java cod
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_41 {
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8         int n = scanner.nextInt();
9         int[] ans = new int[n + 1];
10        for (int i = 1; i <= n; i++) {
11            ans[i] = ans[i >> 1] + (i & 1);
12        }
13        for (int i = 0; i <= n; i++) {
14            System.out.print(ans[i] + " ");
15        }
16        System.out.println("\n");
17    }
18 }
```

A163 : Happy Number (code_42)

Write an algorithm to determine if a number n is happy.

A happy number is a number defined by the following process:

- Starting with any positive integer, replace the number by the sum of the squares of its digits.
- Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.
- Those numbers for which this process ends in 1 are happy.

Return true if n is a happy number, and false if not.

Input Format

Given a single integer N.

Constraints

$1 \leq n \leq 2^{31} - 1$

Output Format

Print true if n is a happy number, and false if not.

Sample Input 0

2

Sample Output 0

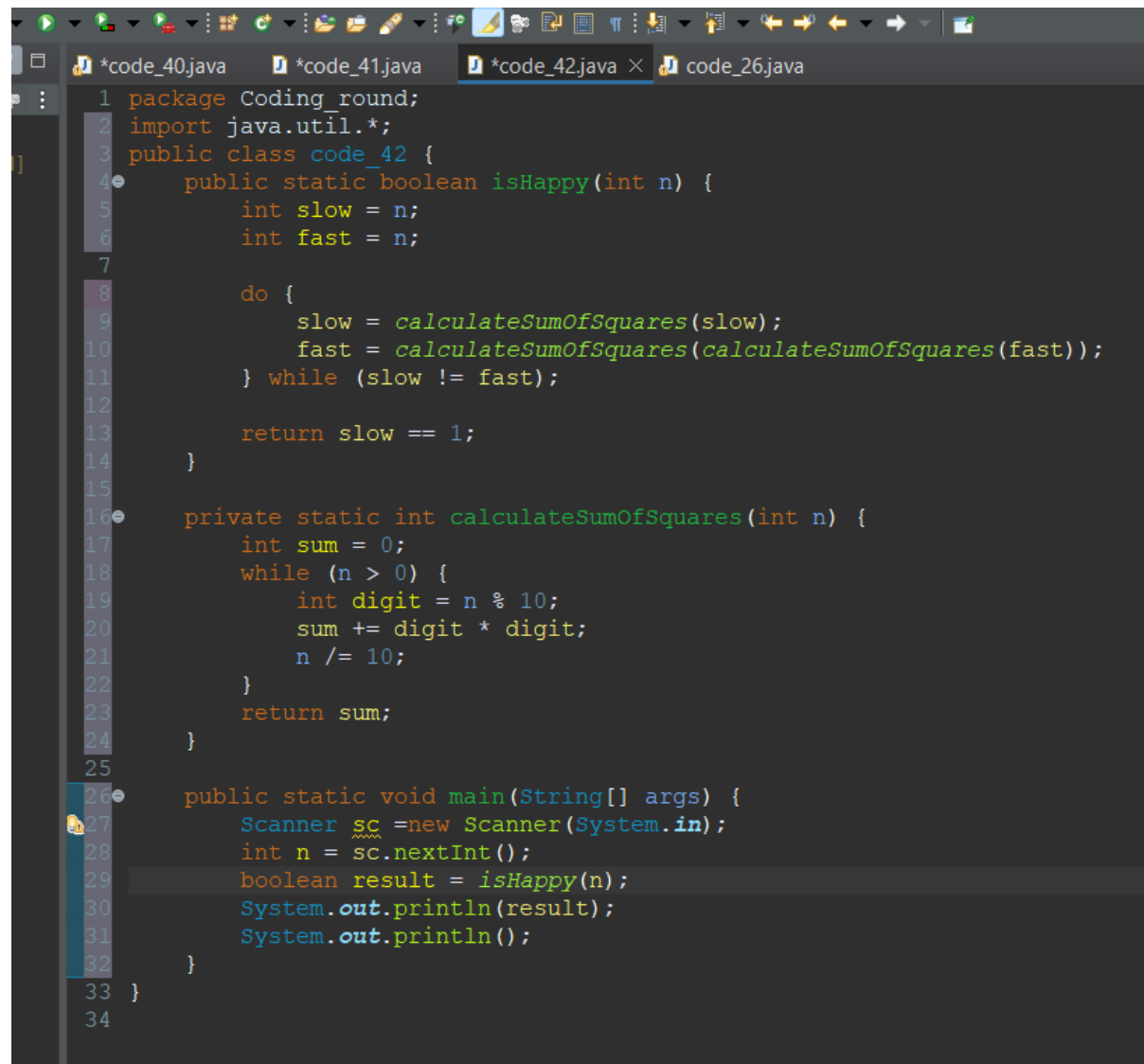
false

Sample Input 1

19

Sample Output 1

true

A screenshot of a Java IDE with a dark theme. The editor shows a file named 'code_42.java' with the following code:

```
1 package Coding_round;
2 import java.util.*;
3 public class code_42 {
4     public static boolean isHappy(int n) {
5         int slow = n;
6         int fast = n;
7
8         do {
9             slow = calculateSumOfSquares(slow);
10            fast = calculateSumOfSquares(calculateSumOfSquares(fast));
11        } while (slow != fast);
12
13        return slow == 1;
14    }
15
16    private static int calculateSumOfSquares(int n) {
17        int sum = 0;
18        while (n > 0) {
19            int digit = n % 10;
20            sum += digit * digit;
21            n /= 10;
22        }
23        return sum;
24    }
25
26    public static void main(String[] args) {
27        Scanner sc = new Scanner(System.in);
28        int n = sc.nextInt();
29        boolean result = isHappy(n);
30        System.out.println(result);
31        System.out.println();
32    }
33 }
34
```

The IDE's toolbar and tab bar are visible at the top.

A152 Value Appears atleast Twice

(code_43)

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Input Format

- The First line contains a integer N.
- The Next line contains N integers representing the array elements.

Constraints

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

Output Format

Print true if any value appears at least twice in the array, and return false if every element is distinct.

Sample Input 0

```
4
1 2 3 1
```

Sample Output 0

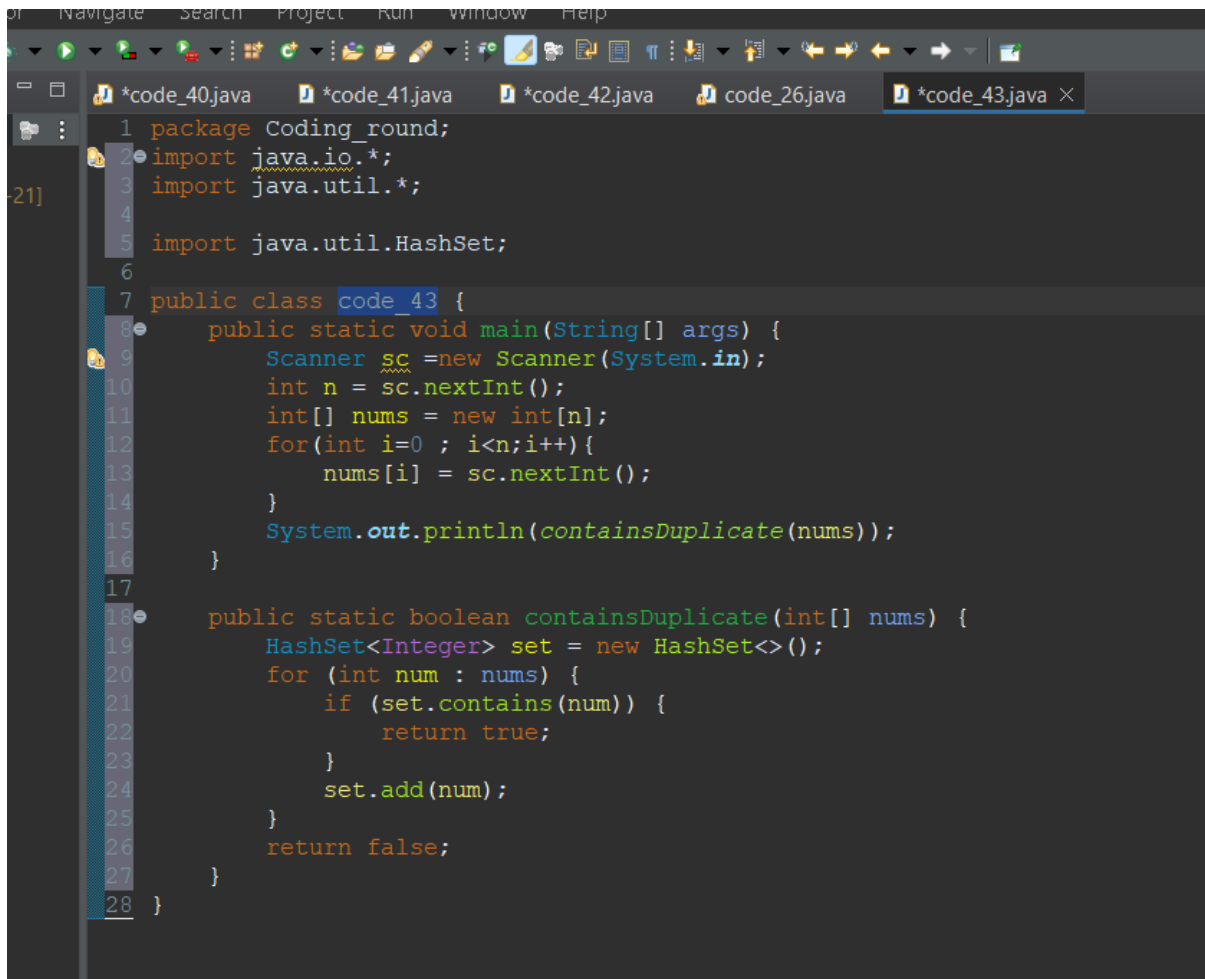
```
true
```

Sample Input 1

```
4
1 2 3 4
```

Sample Output 1

```
false
```

A screenshot of an IDE window showing a Java file named *code_43.java. The code is as follows:

```
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 import java.util.HashSet;
6
7 public class code_43 {
8     public static void main(String[] args) {
9         Scanner sc = new Scanner(System.in);
10         int n = sc.nextInt();
11         int[] nums = new int[n];
12         for(int i=0 ; i<n;i++){
13             nums[i] = sc.nextInt();
14         }
15         System.out.println(containsDuplicate(nums));
16     }
17
18     public static boolean containsDuplicate(int[] nums) {
19         HashSet<Integer> set = new HashSet<>();
20         for (int num : nums) {
21             if (set.contains(num)) {
22                 return true;
23             }
24             set.add(num);
25         }
26         return false;
27     }
28 }
```

A132 : Sliding Window of Size K (code_44)

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

Return the max sliding window.

Input Format

- The First line contains two integers N and k
- The Next Line contains N integers.

Constraints

- $1 \leq \text{nums.length} \leq 10^5$
- $-104 \leq \text{nums}[i] \leq 10^4$

- $1 \leq k \leq \text{nums.length}$

Output Format

Print the integer array which is the answer to the question.

Sample Input 0

```
1 1  
1
```

Sample Output 0

```
1
```

Sample Input 1

```
7 4  
1 2 8 -3 2 8 32
```

Sample Output 1

```
8 8 8 32
```

```

# *code_40.java *code_41.java *code_42.java code_26.java *code_43.java *code_44.java x
1 package Coding_round;
2 import java.util.ArrayDeque;
3 import java.util.Deque;
4 import java.util.Scanner;
5
6 public class code_44 {
7     public static int[] maxSlidingWindow(int[] nums, int k) {
8         int n = nums.length;
9         if (n == 0) return new int[0];
10        int[] result = new int[n - k + 1];
11        Deque<Integer> deque = new ArrayDeque<>();
12
13        for (int i = 0; i < n; i++) {
14            // Remove elements that are out of the current window
15            while (!deque.isEmpty() && deque.peekFirst() < i - k + 1) {
16                deque.pollFirst();
17            }
18            while (!deque.isEmpty() && nums[deque.peekLast()] < nums[i]) {
19                deque.pollLast();
20            }
21            deque.offerLast(i);
22            if (i - k + 1 >= 0) {
23                result[i - k + 1] = nums[deque.peekFirst()];
24            }
25        }
26        return result;
27    }
28    public static void main(String[] args) {
29        Scanner scanner = new Scanner(System.in);
30        int n = scanner.nextInt();
31        int k = scanner.nextInt();
32        int[] nums = new int[n];
33
34        for (int i = 0; i < n; i++) {
35            nums[i] = scanner.nextInt();
36        }
37
38        int[] result = maxSlidingWindow(nums, k);
39
40        for (int i : result) {
41            System.out.print(i + " ");
42        }
43    }
44 }
45

```

A483 Max product (code_45)

Given an integer n , break it into the sum of k positive integers, where $k \geq 2$, and maximize the product of those integers.

Return the maximum product you can get.

Input Format

The Input contains of only integer N

Constraints

$2 \leq n \leq 58$

Output Format

Print a integer as the answer to the question

Sample Input 0

2

Sample Output 0

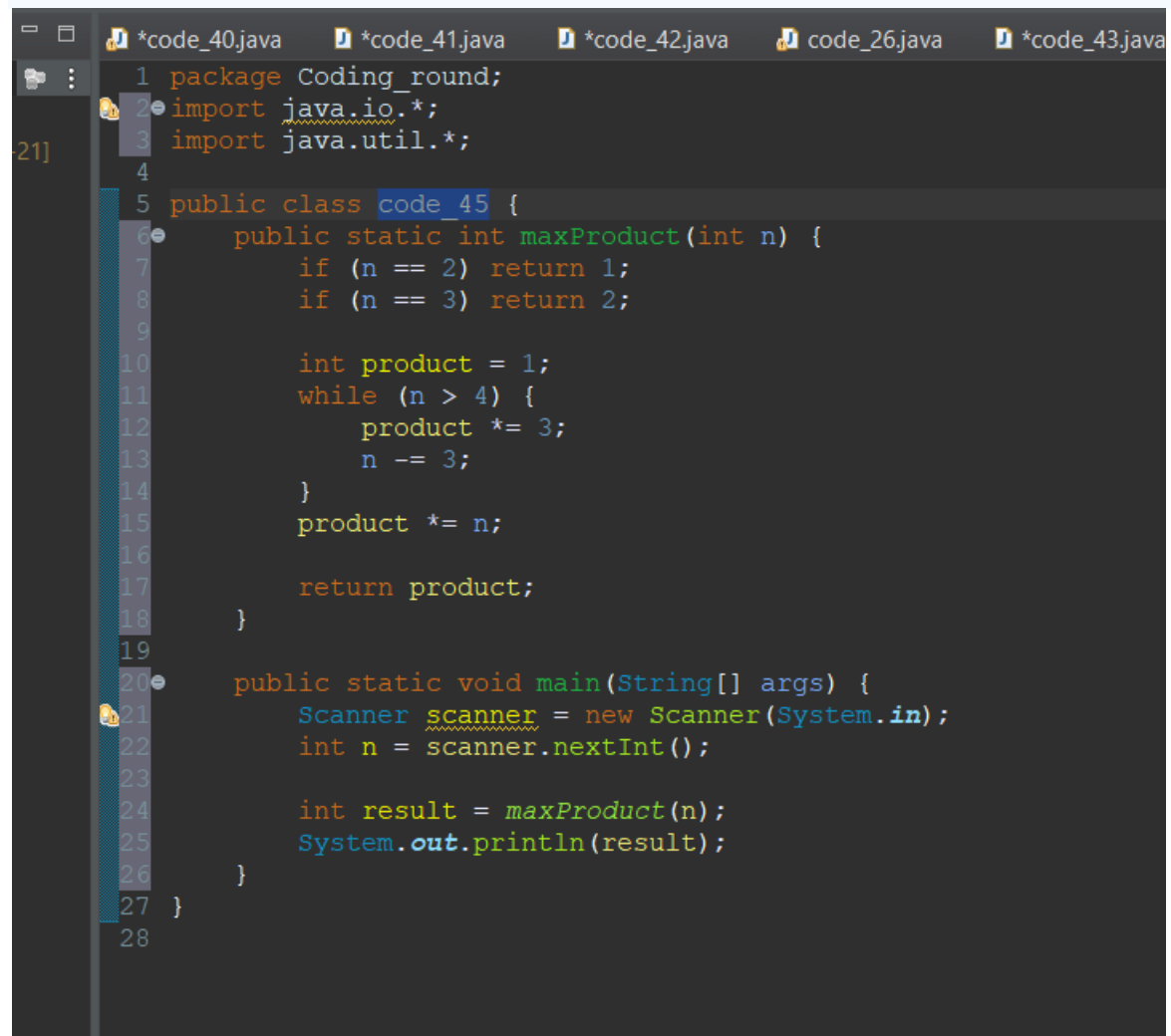
1

Sample Input 1

10

Sample Output 1

36



```
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_45 {
6     public static int maxProduct(int n) {
7         if (n == 2) return 1;
8         if (n == 3) return 2;
9
10        int product = 1;
11        while (n > 4) {
12            product *= 3;
13            n -= 3;
14        }
15        product *= n;
16
17        return product;
18    }
19
20    public static void main(String[] args) {
21        Scanner scanner = new Scanner(System.in);
22        int n = scanner.nextInt();
23
24        int result = maxProduct(n);
25        System.out.println(result);
26    }
27 }
28
```

A289 : The Tribonacci Sequence

(code_46)

The Tribonacci sequence T_n is defined as follows:

$T_0 = 0$, $T_1 = 1$, $T_2 = 1$, and $T_{n+3} = T_n + T_{n+1} + T_{n+2}$ for $n \geq 0$.

Given n , return the value of T_n .

Input Format

The Input contains a single integer N

Constraints

- $0 \leq n \leq 37$
- The answer is guaranteed to fit within a 32-bit integer, ie. $\text{answer} \leq 2^{31} - 1$.

Output Format

Print a integer as a answer to the question

Sample Input 0

4

Sample Output 0

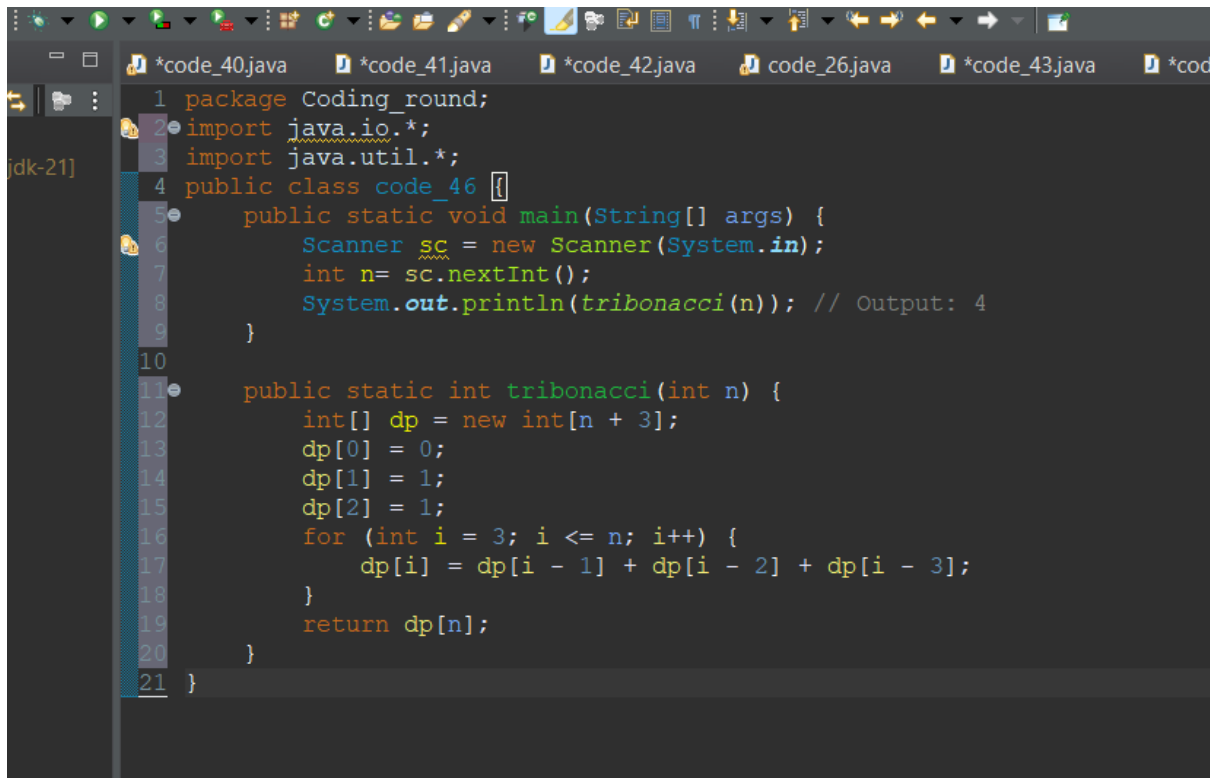
4

Sample Input 1

25

Sample Output 1

1389537

A screenshot of a Java IDE window showing a file named *code_40.java. The code defines a package Coding_round, imports java.io.* and java.util.*, and defines a public class code_46. The class has a main method that takes a String[] args, creates a Scanner for System.in, reads an integer n, and prints the result of tribonacci(n). The tribonacci method is a static function that uses dynamic programming with an array dp of size n+3. It initializes dp[0]=0, dp[1]=1, and dp[2]=1, then iterates from i=3 to n, calculating dp[i] as the sum of dp[i-1], dp[i-2], and dp[i-3]. The final result is dp[n].

```
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4 public class code_46 {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7         int n = sc.nextInt();
8         System.out.println(tribonacci(n)); // Output: 4
9     }
10
11     public static int tribonacci(int n) {
12         int[] dp = new int[n + 3];
13         dp[0] = 0;
14         dp[1] = 1;
15         dp[2] = 1;
16         for (int i = 3; i <= n; i++) {
17             dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3];
18         }
19         return dp[n];
20     }
21 }
```

A159 : Day between dates (code_47)

Write a program to count the number of days between two dates.

The two dates are given as strings, their format is YYYY-MM-DD as shown in the examples.

Input Format

- The First line contains a string representing date 1.
- The Second line contains a string representing date 2.

Constraints

The given dates are valid dates between the years 1971 and 2100.

Output Format

Print an integer which is the answer to the question

Sample Input 0

```
2019-06-29
```


2019-06-30

Sample Output 0

1

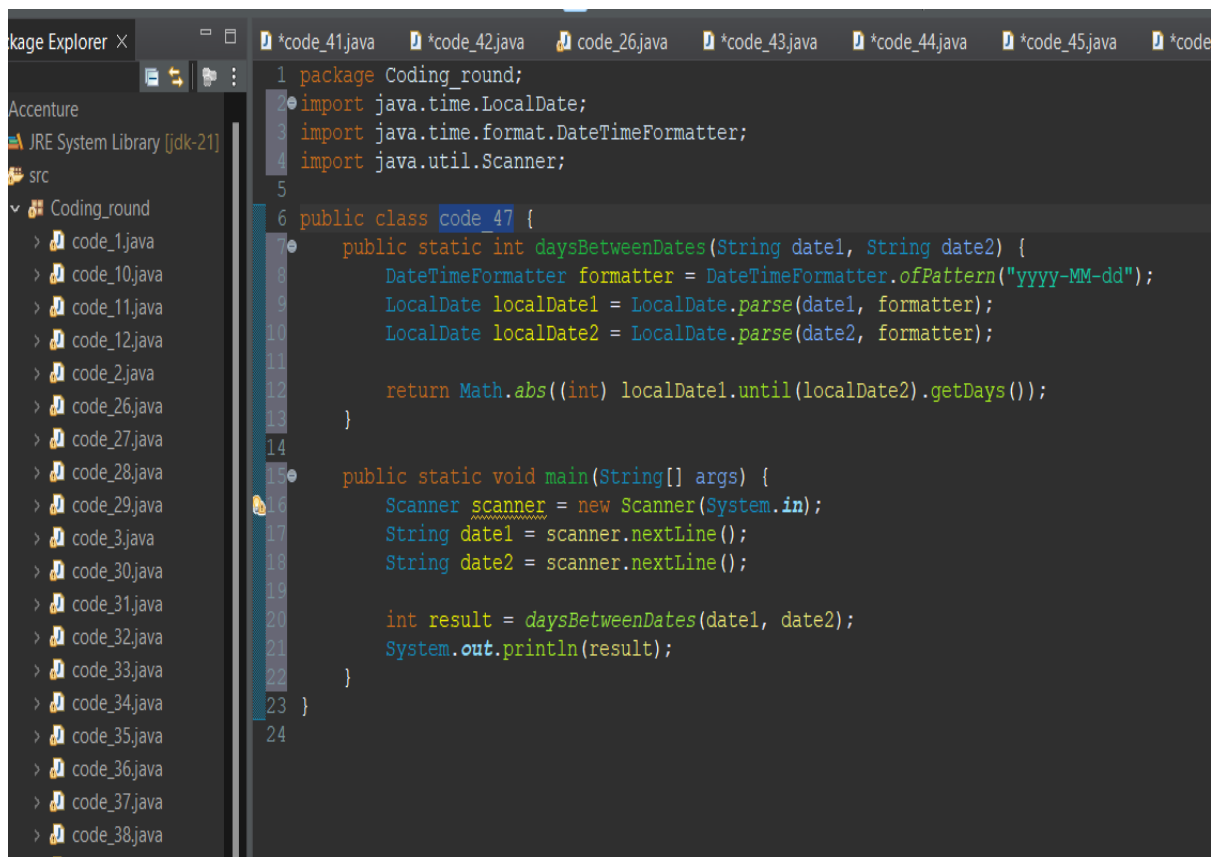
Sample Input 1

2019-07-29

2019-06-30

Sample Output 1

29



The screenshot shows an IDE with a project named 'Coding_round'. The left sidebar displays a file explorer with various Java files. The main editor window shows the code for 'code_47.java'. The code defines a package 'Coding_round', imports 'java.time.LocalDate', 'java.time.format.DateTimeFormatter', and 'java.util.Scanner'. It contains a public class 'code_47' with two methods: 'daysBetweenDates' which calculates the number of days between two dates using 'LocalDate.parse' and 'Math.abs', and a 'main' method which uses a 'Scanner' to read two dates from the user and prints the result.

```
1 package Coding_round;
2 import java.time.LocalDate;
3 import java.time.format.DateTimeFormatter;
4 import java.util.Scanner;
5
6 public class code_47 {
7     public static int daysBetweenDates(String date1, String date2) {
8         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
9         LocalDate localDate1 = LocalDate.parse(date1, formatter);
10        LocalDate localDate2 = LocalDate.parse(date2, formatter);
11
12        return Math.abs((int) localDate1.until(localDate2).getDays());
13    }
14
15    public static void main(String[] args) {
16        Scanner scanner = new Scanner(System.in);
17        String date1 = scanner.nextLine();
18        String date2 = scanner.nextLine();
19
20        int result = daysBetweenDates(date1, date2);
21        System.out.println(result);
22    }
23 }
24
```

A116 Running Sum (code_48)

Given an array nums. We define a running sum of an array as $\text{runningSum}[i] = \text{sum}(\text{nums}[0] \dots \text{nums}[i])$.

Return the running sum of nums.

Input Format

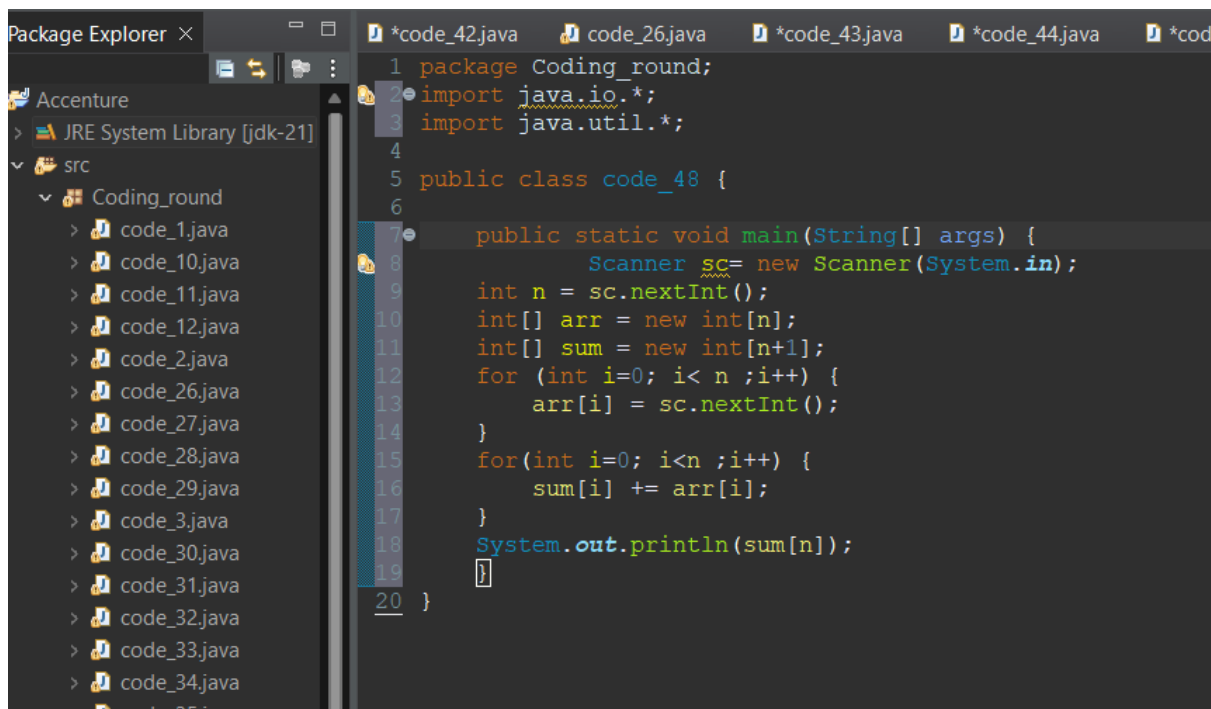
- The First Line contains a integer N
- The Next Line contains N integers seperated by a space

Constraints

- $1 \leq \text{nums.length} \leq 1000$
- $-10^6 \leq \text{nums}[i] \leq 10^6$

Output Format

Print the array of integers as the answer to the question

A screenshot of an IDE window. On the left is the 'Package Explorer' showing a project named 'Accenture' with a 'src' folder containing a 'Coding_round' package. Inside this package, several Java files are listed, including 'code_1.java' through 'code_35.java'. The main editor on the right displays the code for 'code_48.java'. The code is as follows:

```
1 package Coding_round;
2 import java.io.*;
3 import java.util.*;
4
5 public class code_48 {
6
7     public static void main(String[] args) {
8         Scanner sc= new Scanner(System.in);
9         int n = sc.nextInt();
10        int[] arr = new int[n];
11        int[] sum = new int[n+1];
12        for (int i=0; i< n ;i++) {
13            arr[i] = sc.nextInt();
14        }
15        for(int i=0; i<n ;i++) {
16            sum[i] += arr[i];
17        }
18        System.out.println(sum[n]);
19    }
20 }
```

A468 : Odd in Range (code_49)

Given two non-negative integers low and high. Return the count of odd numbers between low and high (inclusive).

Input Format

The Input contains two integers separated by a space representing the values of low and high respectively

Constraints

$0 \leq \text{low} \leq \text{high} \leq 10^9$

Output Format

Print an integer as the answer to the question

Sample Input 0

3 7

Sample Output 0

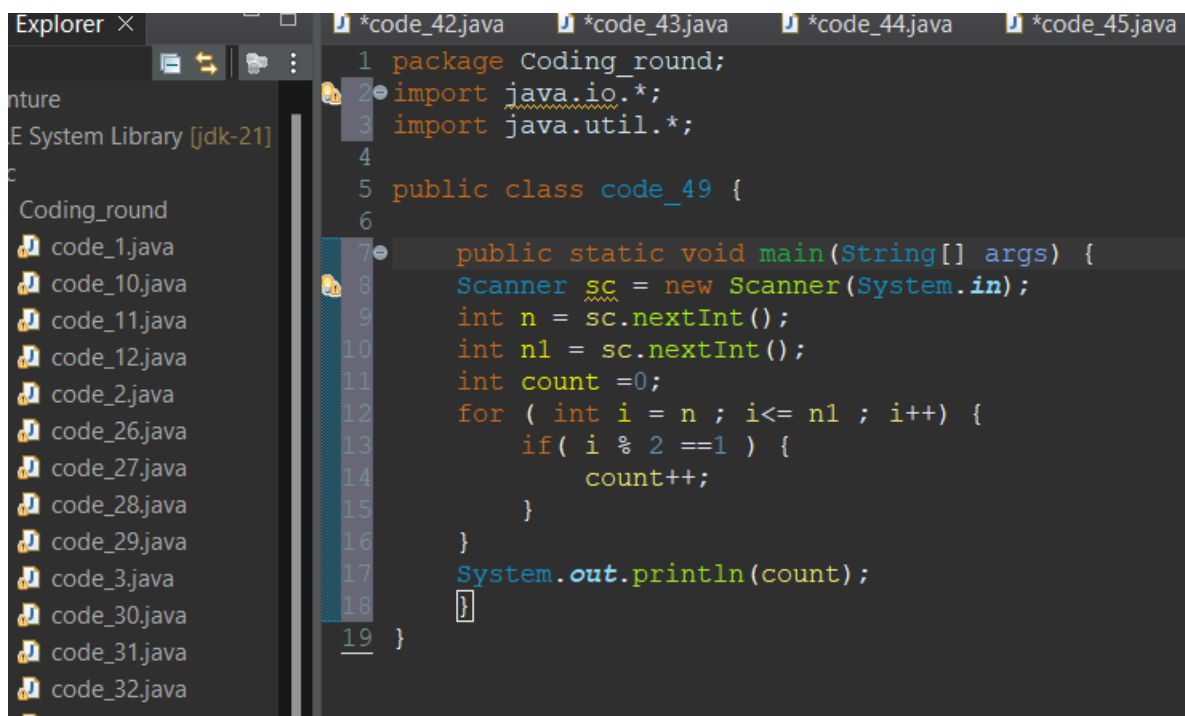
3

Sample Input 1

8 19

Sample Output 1

6

A screenshot of an IDE window titled 'Explorer' showing a file explorer on the left and a code editor on the right. The file explorer lists several Java files: code_1.java, code_10.java, code_11.java, code_12.java, code_2.java, code_26.java, code_27.java, code_28.java, code_29.java, code_3.java, code_30.java, code_31.java, and code_32.java. The code editor shows the content of *code_42.java, which is a Java program. The code defines a package 'Coding_round', imports 'java.io.*' and 'java.util.*', and defines a public class 'code_49'. Inside the class, there is a 'main' method that takes a 'String[] args' parameter. It creates a 'Scanner' object 'sc' from 'System.in', reads two integers 'n' and 'n1' using 'sc.nextInt()', initializes a 'count' variable to 0, and then enters a 'for' loop from 'i = n' to 'i = n1'. Inside the loop, it checks if 'i' is odd using 'i % 2 == 1' and increments 'count' if it is. Finally, it prints the 'count' using 'System.out.println(count)'.

A302 : Subsequence of numbers length k with Max sum (code_50)

You are given an integer array `nums` and an integer `k`. You want to find a subsequence of `nums` of length `k` that has the largest sum.

Return such subsequence as an integer array of length `k` and print it in a sorted order.

A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Input Format

- The First line contains a integer `N`
- The Next line contains `N` integers

Constraints

- $1 \leq \text{nums.length} \leq 1000$
- $-10^5 \leq \text{nums}[i] \leq 10^5$
- $1 \leq k \leq \text{nums.length}$

Output Format

Print a array of integers in sorted order.

Sample Input 0

```
4 2
2 1 3 3
```

Sample Output 0

```
3 3
```

Sample Input 1

```
3 2
4 3 3
```

Sample Output 1

```
3 4
```

```
1 package Coding_round;
2 import java.util.Arrays;
3 import java.util.Scanner;
4
5 public class code_50 {
6     public static int[] findLargestSumSubsequence(int[] nums, int k) {
7         int n = nums.length;
8         int[] subsequence = new int[k];
9
10        Arrays.sort(nums);
11
12        int sum = 0;
13        for (int i = n - 1; i >= n - k; i--) {
14            subsequence[i - (n - k)] = nums[i];
15            sum += nums[i];
16        }
17        return subsequence;
18    }
19
20    public static void main(String[] args) {
21        Scanner scanner = new Scanner(System.in);
22        int n = scanner.nextInt();
23        int k = scanner.nextInt();
24        int[] nums = new int[n];
25
26        for (int i = 0; i < n; i++) {
27            nums[i] = scanner.nextInt();
28        }
29
30        int[] result = findLargestSumSubsequence(nums, k);
31        Arrays.sort(result);
32
33        for (int num : result) {
34            System.out.print(num + " ");
35        }
36    }
37 }
38
```