**Q1. Good Pair**

**Problem Description**

Given an array **A** and a integer **B**. A **pair(i,j)** in the array is a good pair if **i!=j** and **(A[i]+A[j]==B)**. Check if any good pair exist or not.

**Problem Constraints**

1 <= A.size() <= 104

1 <= A[i] <= 109

1 <= B <= 109

**Input Format**

First argument is an integer array A.

Second argument is an integer B.

**Output Format**

Return 1 if good pair exist otherwise return 0.

**Example Input**

Input 1:

A = [1,2,3,4]

B = 7

Input 2:

A = [1,2,4]

B = 4

Input 3:

A = [1,2,2]

B = 4

**Example Output**

Output 1:

1

Output 2:

0

Output 3:

1

**Example Explanation**

Explanation 1:

(i,j) = (3,4)

Explanation 2:

No pair has sum equal to 4.

Explanation 3:

(i,j) = (2,3)

public class Solution {

    public int solve(ArrayList<Integer> A, int B) {

        for (int i = 0; i < A.size(); i++) {

            for (int j = i + 1; j < A.size(); j++) {

                if ((A.get(i) + A.get(j)) == B) {

                    return 1;

                }

            }

        }

        return 0;

    }

}

**Q2. FizzBuzz**

**Problem Description**

Given a positive integer **A**, return an array of strings with all the integers from **1** to **N**. But for multiples of **3** the array should have **“Fizz”** instead of the number. For the multiples of **5**, the array should have **“Buzz”** instead of the number. For numbers which are multiple of **3** and **5** both, the array should have **"FizzBuzz"** instead of the number.  
  
Look at the example for more details.

**Problem Constraints**

* 1 <= A <= 500000

**Input Format**

The first argument has the integer A.

**Output Format**

Return an array of string.

**Example Input**

Input 1:

A = 5

**Example Output**

Output 1:

[1 2 Fizz 4 Buzz]

**Example Explanation**

Explanation 1:

3 is divisible by 3 so it is replaced by "Fizz".

Similarly, 5 is divisible by 5 so it is replaced by "Buzz".

public class Solution {

    public ArrayList<String> fizzBuzz(int A) {

        ArrayList<String> result = new ArrayList<String>();

        for (int i = 1; i <= A; i++) {

            if (i % 15 == 0) {

                result.add("FizzBuzz");

            } else if (i % 5 == 0) {

                result.add("Buzz");

            } else if (i % 3 == 0) {

                result.add("Fizz");

            } else {

                result.add(String.valueOf(i));

            }

        }

        return result;

    }

}

**Q3. Time to equality**

**Problem Description**  
<div id=problem\_description\_markdown\_content\_value style=”background-color: #f9f9f9; padding: 5px 10px; “>Given an integer array **A** of size **N**. In one second you can increase the value of one element by 1.  
  
  
Find the **minimum** time in seconds to make all elements of the array equal.<p></p>

</div>  
  
**Problem Constraints**  
<div id=problem\_constraints\_markdown\_content\_value style=”background-color: #f9f9f9; padding: 5px 10px; “>1 <= N <= 1000000<p></p><p></p>  
1 <= A[i] <= 1000

</div>  
  
**Input Format**  
<div id=input\_format\_markdown\_content\_value style=”background-color: #f9f9f9; padding: 5px 10px; “>First argument is an integer array A.</div>  
  
**Output Format**  
<div id=output\_format\_markdown\_content\_value style=”background-color: #f9f9f9; padding: 5px 10px; “>Return an integer denoting the minimum time to make all elements equal.</div>  
  
**Example Input**  
<div id=example\_input\_markdown\_content\_value style=”background-color: #f9f9f9; padding: 5px 10px; “>A = [2, 4, 1, 3, 2] </div>  
  
**Example Output**  
<div id=example\_output\_markdown\_content\_value style=”background-color: #f9f9f9; padding: 5px 10px; “>8 </div>  
  
**Example Explanation**  
<div id=example\_explanation\_markdown\_content\_value style=”background-color: #f9f9f9; padding: 5px 10px; “>We can change the array A = [4, 4, 4, 4, 4].  
The time required will be 8 seconds.</div>

public class Solution {

    public int solve(ArrayList<Integer> A) {

        Integer maxElement = Integer.MIN\_VALUE;

        Integer totalOperations = 0;

        for (Integer element : A) {

            if (element > maxElement ) {

                maxElement = element;

            }

        }

        for (Integer element : A) {

            totalOperations += Math.abs(maxElement - element);

        }

        return totalOperations;

    }

}

**Q4. Rotation Game**

**Problem Description**

You are given an integer array **A** and an integer **B**. You have to print the same array after rotating it **B** times towards right.

**NOTE:** You can use extra memory.

**Problem Constraints**

1 <= **|A|** <= 106

1 <= **A[i]** <= 109

1 <= **B** <= 109

**Input Format**

First line begins with an integer **|A|** denoting the length of array, and then **|A|** integers denote the array elements.  
Second line contains a single integer **B**

**Output Format**

Print an array of integers which is the **Bth** right rotation of input array **A**, on a separate line.

**Example Input**

Input 1:

4 1 2 3 4

2

Input 2:

3 1 2 2

3

**Example Output**

Output 1:

3 4 1 2

Output 2:

1 2 2

**Example Explanation**

Explanation 1:

[1,2,3,4] => [4,1,2,3] => [3,4,1,2]

Explanation 2:

[1,2,2] => [2,1,2] => [2,2,1] => [1,2,2]

import java.lang.\*;

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        // YOUR CODE GOES HERE

        // Please take input and print output to standard input/output (stdin/stdout)

        // DO NOT USE ARGUMENTS FOR INPUTS

        // E.g. 'Scanner' for input & 'System.out' for output

        Scanner scanner = new Scanner(System.in);

        int size = scanner.nextInt();

        int[] input = new int[size];

        for (int i = 0; i < size; i++) {

            input[i] = scanner.nextInt();

        }

        int B = scanner.nextInt();

        // because after size rotations the array will become same as initial input array

        B = B % size;

        for (int i = 0; i < size; i++) {

            if ( i < B) {

                System.out.print(input[size-B+i] + " ");

            } else {

                System.out.print(input[i - B] + " ");

            }

        }

    }

}

**Q5. Leaders in an array**

**Problem Description**

Given an integer array **A** containing **N** distinct integers, you have to find all the **leaders** in the array **A**.

An element is leader if it is strictly greater than all the elements to its right side.

**NOTE:**The rightmost element is always a leader.

**Problem Constraints**

1 <= N <= 105

1 <= A[i] <= 108

**Input Format**

First and only argument is an integer array **A**.

**Output Format**

Return an integer array denoting all the **leader elements** of the array.

**NOTE:** Ordering in the output doesn't matter.

**Example Input**

Input 1:

A = [16, 17, 4, 3, 5, 2]

Input 2:

A = [1, 2]

**Example Output**

Output 1:

[17, 2, 5]

Output 2:

[2]

**Example Explanation**

Explanation 1:

Element 17 is strictly greater than all the elements on the right side to it.

Element 2 is strictly greater than all the elements on the right side to it.

Element 5 is strictly greater than all the elements on the right side to it.

So we will return this three elements i.e [17, 2, 5], we can also return [2, 5, 17] or [5, 2, 17] or any other ordering.

Explanation 2:

Only 2 the rightmost element is the leader in the array.

public class Solution {

    public ArrayList<Integer> solve(ArrayList<Integer> A) {

        ArrayList<Integer> result = new ArrayList<Integer>();

        int size = A.size();

        // base condition - 1

        if (size == 0) {

            return result;

        }

        // base condition - 2

        if (size == 1) {

            result.add(A.get(0));

            return result;

        }

        // add the right most element to result array

        result.add(A.get(size - 1));

        int currentMax = A.get(size - 1);

        for (int i = size - 2; i >= 0; i--) {

            if (A.get(i) > currentMax) {

                result.add(A.get(i));

                currentMax = A.get(i);

            }

        }

        return result;

    }

}

**Q6. Max Sum Contiguous Subarray**

**Problem Description**

Find the **contiguous** subarray within an array, **A** of length **N** which has the **largest sum**.

**Problem Constraints**

1 <= N <= 1e6  
-1000 <= A[i] <= 1000

**Input Format**

The first and the only argument contains an integer array, A.

**Output Format**

Return an integer representing the maximum possible sum of the contiguous subarray.

**Example Input**

Input 1:

A = [1, 2, 3, 4, -10]

Input 2:

A = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

\*\*Example Output\*\*

Output 1:

10

Output 2:

6

\*\*Example Explanation\*\*

Explanation 1:

The subarray [1, 2, 3, 4] has the maximum possible sum of 10.

Explanation 2:

The subarray [4,-1,2,1] has the maximum possible sum of 6.

public class Solution {

    // DO NOT MODIFY THE LIST. IT IS READ ONLY

    public int maxSubArray(final List<Integer> A) {

        int length = A.size();

        int currentSum = A.get(0);

        int maxSum = A.get(0);

        for (int i = 1; i < length; i++) {

            if (currentSum < 0) {

                currentSum = 0;

            }

            currentSum += A.get(i);

            if (currentSum > maxSum) {

                maxSum = currentSum;

            }

        }

        return maxSum;

    }

}

**Q7. Multiple left rotations of the array**

**Problem Description**

Given an array of integers **A** and multiple values in **B** which represents number of times array A needs to be left rotated.

Find the rotated array for each value and return the result in the from of a matrix where i'th row represents the rotated array for the i'th value in B.

**Problem Constraints**

1 <= length of both arrays <= 2000 -10^9 <= A[i] <= 10^9 0 <= B[i] <= 2000

**Input Format**

The first argument given is the integer array **A**.  
The second argument given is the integer array **B**.

**Output Format**

Return the resultant matrix.

**Example Input**

Input 1:

A = [1, 2, 3, 4, 5]

B = [2, 3]

Input 2:

A = [5, 17, 100, 11]

B = [1]

**Example Output**

Output 1:

[ [3, 4, 5, 1, 2]

[4, 5, 1, 2, 3] ]

Output 2:

[ [17, 100, 11, 5] ]

**Example Explanation**

for input 1 -> B[0] = 2 which requires 2 times left rotations

1: [2, 3, 4, 5, 1]

2: [3, 4, 5, 1, 2]

B[1] = 3 which requires 3 times left rotation

1: [2, 3, 4, 5, 1]

2: [3, 4, 5, 1, 2]

2: [4, 5, 1, 2, 4]

public class Solution {

    public ArrayList<ArrayList<Integer>> solve(ArrayList<Integer> A, ArrayList<Integer> B) {

        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();

        int length = A.size();

        for (Integer eachB : B) {

            eachB = eachB % length;

            ArrayList<Integer> eachRotation = new ArrayList<Integer>();

            for (int i = 0; i < length; i++) {

                eachRotation.add(A.get((i + eachB) % length));

                // if ( i <= eachB) {

                //     eachRotation.add(A.get(i + eachB));

                // } else {

                //     eachRotation.add(A.get(i - eachB - 1));

                // }

            }

            result.add(eachRotation);

        }

        return result;

    }

}

**Q8. Noble Integer**

**Problem Description**

Given an integer array **A**, find if an integer **p** exists in the array such that the number of integers greater than **p** in the array equals to **p**.

**Input Format**

First and only argument is an integer array A.

**Output Format**

Return 1 if any such integer p is found else return -1.

**Example Input**

Input 1:

A = [3, 2, 1, 3]

Input 2:

A = [1, 1, 3, 3]

**Example Output**

Output 1:

1

Output 2:

-1

**Example Explanation**

Explanation 1:

For integer 2, there are 2 greater elements in the array. So, return 1.

Explanation 2:

There is no such integer exists.

public class Solution {

    public int solve(ArrayList<Integer> A) {

        int len = A.size();

        Collections.sort(A);

        // base condition where all the elements in the array are negative and one element is zero

        if (A.get(len - 1) == 0) {

            return 1;

        }

        for (int i = 0; i < len - 1; i++) {

            int currElement = A.get(i);

            if (currElement == (len - i -1) && currElement != A.get(i + 1)) {

                return 1;

            }

        }

        // return -1;

        // for (int i = 0; i < len; i++) {

        //     int currElement = A.get(i);

        //     int totalElementsGreaterThanCurrElement = 0;

        //     for (int j = 0; j < len; j++) {

        //         if (A.get(j) > currElement) {

        //             totalElementsGreaterThanCurrElement++;

        //         }

        //     }

        //     if (totalElementsGreaterThanCurrElement == currElement) {

        //         return 1;

        //     }

        // }

        return -1;

    }

}

**Q9. Reverse the Array**

**Problem Description**

You are given a constant array **A**.

You are required to return another array which is the reversed form of input array.

**Problem Constraints**

1 <= A.size() <= 10000

1 <= A[i] <= 10000

**Input Format**

First argument is a constant array A.

**Output Format**

You have to return an integer array.

**Example Input**

Input 1:

A = [1,2,3,2,1]

Input 2:

A = [1,1,10]

**Example Output**

Output 1:

[1,2,3,2,1]

Output 2:

[10,1,1]

**Example Explanation**

Explanation 1:

Reversed form of input array is same as original array

Explanation 2:

Clearly, Reverse of [1,1,10] is [10,1,1]

public class Solution {

    // DO NOT MODIFY THE LIST. IT IS READ ONLY

    public ArrayList<Integer> solve(final List<Integer> A) {

        ArrayList<Integer> result = new ArrayList<Integer>();

        int len = A.size();

        if (len == 0) {

            return result;

        }

        for (int i = len - 1; i >= 0; i--) {

            result.add(A.get(i));

        }

        return result;

    }

}

**Q10. Subarray with least average**

**Problem Description**

Given an array of size **N**, Find the subarray with least average of size **k**.

**Problem Constraints**

1<=k<=N<=1e5

-1e5<=A[i]<=1e5

**Input Format**

First argument contains an array A of integers of size N.  
Second argument contains integer k.

**Output Format**

Return the index of the first element of the subarray of size k that has least average.  
Array indexing starts from 0.

**Example Input**

Input 1:

A=[3, 7, 90, 20, 10, 50, 40]

B=3

Input 2:

A=[3, 7, 5, 20, -10, 0, 12]

B=2

**Example Output**

Output 1:

3

Output 2:

4

**Example Explanation**

Explanation 1:

Subarray between indexes 3 and 5

The subarray {20, 10, 50} has the least average

among all subarrays of size 3.

Explanation 2:

Subarray between [4, 5] has minimum average

public class Solution {

    public int solve(ArrayList<Integer> A, int B) {

        int len = A.size();

        if (B > len) {

            return 0;

        }

        int currentSum = 0;

        double minAverage = Integer.MAX\_VALUE;

        int index = 0;

        for (int i = 0; i < len; i++) {

            currentSum += A.get(i);

            if (i >= (B - 1)) {

                if (((currentSum) \* 1.0 / B) < minAverage) {

                    minAverage = (currentSum) \* 1.0 / B;

                    index = (i - B + 1);

                }

                // minAverage = (minAverage < (currentSum) / B) ? minAverage : (currentSum) / B;

                currentSum = currentSum - A.get(i - B + 1);

            }

        }

        return index;

    }

}