1. Points on X Axis

You are given **N** points on the X-axis. The x coordinate of i^{th} point is X_i. Find a point on the X-axis such the sum of the distances of that point from all the N points is the minimum. If there are many such points then find the leftmost point.

Distance of point (a, 0) to (b, 0) is **|a - b|**.

Note:

1. |x| is the mod function that has value x if x is greater than equals to 0 otherwise its value is -x.

Function description

Complete the ***pointsOnXAxis*** function in the editor below. It has the following parameter(s):

Name	Type	Description
X	INTEGER ARRAY	X coordinates of all N points.

Return	The function must return an *INTEGER* denoting the X coordinate of point on the X-axis such the sum of the distances of that point from all the N points is the minimum.

Constraints

- $1 \leq \mathbf{N} \leq 10^5$
- $1 \leq \mathbf{X[i]} \leq 10^5$

Input format for debugging

- The first line contains an integer, N, denoting the number of elements in X.
- Each line i of the N subsequent lines (where $0 \leq i < N$) contains an integer describing X[i].

Sample Testcases

Input	Output	Output Description
5 2 1 3 5 4	3	The optimal point would be (3, 0) as its sum of distances from all 5 points would be |1 - 3| + |2 - 3| + |3 - 3| + |4 - 3| + |5 - 3| = 2 + 1 + 0 + 1 + 2 = 6.
1 1	1	The optimal point would be (1, 0) as its distance from all points is 0.
3 1 1 1	1	The optimal point would be (1, 0) as its distance from all points is 0.

Program

```
import java.io.*;
import java.util.*;
import java.lang.Math;
public class Solution {
    // public static int pointsOnXAxis(int[] X){
    //     for(int i=0; i<x.length-1; i++ ){
    //         for(int j=0; j<length-1; j++){
```

```
    //             sum = sum+ (a[i])
    //         }
    //     }
    // }
public static int pointsOnXAxis(int[] X){
        Arrays.sort(X);

        int medianIndex = X.length / 2;
        if (X.length % 2 == 0) {
            medianIndex--;
        }
        return X[medianIndex];
    }
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        int N;
        N=scan.nextInt();
        int[] X = new int[N];
        for(int j=0;j<N;j++){
            X[j]=scan.nextInt();
        }
        int result;
        result = pointsOnXAxis(X);
        System.out.print(result);
        return ;
    }
}
```

2. Bitwise AND Pairs

You are given a string **S** of length **N**. The string **S** is a base-64 string. A base-64 string is a string that denotes a number in base 64 (the leftmost digit is most significant). Each digit is represented as follows:

1. Digits from '0' to '9' correspond to integers from 0 to 9.
2. Letters from 'A' to 'Z' correspond to integers from 10 to 35.
3. Letters from 'a' to 'z' correspond to integers from 36 to 61.
4. The character '-' correspond to integer 62.
5. The character '_' correspond to integer 63.

Find the number of ordered pairs of base-64 strings of length **N** such that the **Bitwise AND** of the number represented by them equals the number represented by string **S**. Since the answer could be very large print it modulo $10^9 + 7$.

Function description

Complete the ***bitwiseANDPairs*** function in the editor below. It has the following parameter(s):

Name	Type	Description
N	INTEGER	Length of string
S	STRING	The string representing the number id 64 bit.
Return		The function must return an *INTEGER* denoting the number of ordered pairs of numbers whose Bitwise AND is equal to the number represented by string S modulo 10^9 + 7.

Constraints

- $1 \leq N \leq 10^5$
- $1 \leq len(S) \leq 10^5$

Input format for debugging

- The first line contains an integer, N, denoting the length of string.
- The next line contains a string, S, denoting the string representing the number id 64 bit.

Sample Testcases

Input	Output	Output Description
1 z	3	"z" equals 61 in base 10. Thus, possible

Input	Output	Output Description
		pairs of bit-64 string are 1. "z", "_" 2. "_", "_" 3. "z", "z" "_" equals 63 in base 10.
3 V_V	9	"V_V" equals 131039 in base 10. Thus, possible pairs of bit-64 string are 1. "V_V", "V_V" 2. "__V", "V_V" 3. "V_V", "__V" 4. "V_V", "V__" 5. "V__", "V_V" 6. "__V", "V__" 7. "___", "V_V" 8. "V__", "__V" 9. "V_V", "___" "___" equals 262143, "__V" equals 262111, "V__" equals 131071 in base 10
16 ProgrammingIsFun	412233812	Large Sample Case for Debugging

Program

```
import java.io.*;
import java.util.*;
import java.lang.Math;
public class Solution {
    public static int bitwiseANDPairs(int N,String S){
```

```
//Code neede ☹
    }

    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        int N;
        N=scan.nextInt();
        String S;
        S=scan.next();
        int result;
        result = bitwiseANDPairs(N,S);
        System.out.print(result);
        return ;
    }
}
```

3. Twice Match

You are given an array **A** of length **N**. Find the number of pairs of **(i, j)** such that **i < j** and $\mathbf{A_i + A_j = 2^x}$ where **x** is an integer. Since the answer could be large return it modulo 10^9 + 7.

Function description

Complete the ***twiceMatch*** function in the editor below. It has the following parameter(s):

Name	Type	Description
A	INTEGER ARRAY	array if non-negative integers.
Return	The function must return an *INTEGER* denoting the number of ordered pairs satisfying the condition.	

Constraints

- $1 \leq \mathbf{N} \leq 10^5$
- $1 \leq \mathbf{A[i]} \leq 10^9$

Input format for debugging

- The first line contains an integer, N, denoting the number of elements in A.
- Each line i of the N subsequent lines (where $0 \leq i < N$) contains an integer describing array element Ai.

Sample Testcases

Input	Output	Output Description
4 1 15 2 7	2	ordered pairs of indices satisfying the conditions are: 1. (1, 2) 2. (1, 4)
5 1 2 3 4	2	ordered pairs of indices satisfying the conditions are: 1. (1, 3) 2. (3, 5)

Input	Output	Output Description
5		
5 2 2 2 2 2	10	All pairs of i, j satisfies the condition.

Program

```
import java.io.*;
import java.util.*;
import java.lang.Math;
public class Solution {
    // public static boolean isMatch(int sum){
    //     // boolean res = false;
    //     // for(int i = 0; i<sum; i++){
    //     //     if(Math.pow(2,i)==sum){
    //     //      res =true;
    //     //      break;
    //     //     }
    //     // }
    //     return sum != 0 && (sum & (sum - 1)) == 0;
    // }
    public static int twiceMatch(int[] A){
        int sum = 0, count = 0;
        //boolean case;
        for(int i =0; i<A.length-1; i++){
                for(int j = i+1; j<A.length; j++){
                sum = A[i] + A[j];
                if(sum != 0 && (sum & (sum - 1)) == 0)
                    count++;
            }
        }
        return count;
    }
```

```
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        int N;
        N=scan.nextInt();
        int[] A = new int[N];
        for(int j=0;j<N;j++){
            A[j]=scan.nextInt();
        }
        int result;
        result = twiceMatch(A);
        System.out.print(result);
        return ;
    }
}
```

Note: Out of 15 Test Cases 6 cases are failed all other are time limit exceeded

4. ODD BITS

Alice has received an array **AR** consisting of **N** numbers.

Alice defines the score of an array as the length of the largest good subarray that can be obtained using this array. Bob knows that Alice can find the score of an array very fast. Hence, he asks her **Q** queries each of the form **[L,R]** where L,R are integers**.**

You are given two arrays **LEFT** and **RIGHT**, each of size **Q** where the **i^{th}** query is **[LEFT[i],RIGHT[i]]**, in each query. Alice has to find the score of the subarray **[AR[LEFT[i]] ... AR[RIGHT[i]]** and report it to Bob.

Your task is to find the sum of all values reported by Alice in response to the **Q** queries.

Note:

A **good array** can be defined as follows:

An array is considered as a ***good array***, if XOR of all the elements in this array has odd number of set bits in it's binary representation. We can say [2,5,3] is a ***good array*** because, XOR of all it's elements is 4 => "100"(in binary), which has odd number of set bits(bits with value 1).

Function description

Complete the ***oddXor*** function in the editor below. It has the following parameter(s):

Name	Type	Description
N	INTEGER	Number of elements in the array AR
AR	INTEGER ARRAY	Integer array consisting of N numbers
Q	INTEGER	Number of Queries
LEFT	INTEGER ARRAY	LEFT[i] is the left end of the subarray for ith query
RIGHT	INTEGER ARRAY	RIGHT[i] is the Right end of the subarray for ith query
Return	The function must return an *INTEGER* denoting the Sum of scores reported by Alice in all the queries	

Constraints

- $1 \leq \mathbf{N} \leq 10^5$
- $1 \leq \mathbf{AR[i]} \leq 10^6$
- $1 \leq \mathbf{Q} \leq 10^4$
- $1 \leq \mathbf{LEFT[i]} \leq N$
- $1 \leq \mathbf{RIGHT[i]} \leq N$

Input format for debugging

- The first line contains an integer, N, denoting the number of elements in AR.
- Each line i of the N subsequent lines (where $0 \leq i < N$) contains an integer describing ARi.
- The next line contains an integer, Q, denoting the number of elements in LEFT.
- Each line i of the Q subsequent lines (where $0 \leq i < Q$) contains an integer describing LEFTi.
- Each line i of the Q subsequent lines (where $0 \leq i < Q$) contains an integer describing RIGHTi.

Sample Testcases

Input	Output	Output Description
3 2 3 4 1 1 3	2	L=1 , R= 3 subarray [2,3] is the largest subarray which is a good array Ans=2
3 2 3 5 2 1 2 3 2	3	L=1 , R= 3 subarray [2,3,5] is the largest subarray which is a good array L=2 , R=2 There is no subarray which is good Ans=3+0 =3
3 2 3 5 2 1	4	L=1 , R= 3 subarray [2,3,5] is the largest subarray which is a good array L=1 , R=1 subarray [2] is the largest good subarray Ans=3+1 =4

Input	Output	Output Description
1 3 1		

Program

```
import java.io.*;
import java.util.*;
import java.lang.Math;
public class Solution {
    public static int oddXor(int N,int[] AR,int Q,int[] LEFT,int[] RIGHT){

    }

    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        int N;
        N=scan.nextInt();
        int[] AR = new int[N];
        for(int j=0;j<N;j++){
            AR[j]=scan.nextInt();
        }
        int Q;
        Q=scan.nextInt();
        int[] LEFT = new int[Q];
        for(int j=0;j<Q;j++){
            LEFT[j]=scan.nextInt();
        }
        int[] RIGHT = new int[Q];
        for(int j=0;j<Q;j++){
            RIGHT[j]=scan.nextInt();
        }
        int result;
        result = oddXor(N,AR,Q,LEFT,RIGHT);
        System.out.print(result);
        return ;
    }
}
```

5.Collect in Order

You are given a permutation of **N** integers called **Perm**. You have to collect the numbers in the increasing order that is 1, 2,3...**N**.

Initially, you traverse the array from **left to right** collecting the numbers until the end. Then, change the direction and traverse it from **right to left** and so on until you have collected all the numbers.

It is given that you are allowed to change the direction of traversal while collecting the numbers at any position.

You have to tell the **minimum number of times**, you would have to change the direction while collecting all the numbers from 1 to **N** in sequential order.

NOTE:

- Permutation of first **N** natural numbers is an array of size **N** where each number from 1 to **N** occurs exactly once.

Function description
Complete the ***minTurns*** function in the editor below. It has the following parameter(s):

Name	Type	Description
N	INTEGER	Length of Perm array
Perm	INTEGER ARRAY	The given permutation array
Return	The function must return an *INTEGER* denoting the Minimum number of turns required to pick the elements in right order	

Constraints

- 1 **≤ N ≤** 10^5

- 1 **≤ Perm[i] ≤** 10^5

Input format for debugging

- The first line contains an integer, N, denoting the number of elements in Perm.
- Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing Permi.

Sample Testcases

Input	Output	Output Description
2 2 1	1	Pick 1 in the first traversal from left to right , then Pick two in the second traversal from right to left. There was only 1 turn required
3 1 2 3	0	Pick 1 2 3 in the first traversal from left to right . There were 0 turns required
3 2 3 1	2	Pick 1 in the first traversal from left to right , then Pick two in the second traversal from right to left. Pick 2 in the first traversal from left to right There were only 2 turns required

Program

```
import java.io.*;
import java.util.*;
import java.lang.Math;
public class Solution {
    public static int minTurns(int N,int[] Perm){
        int flag =0,count =0,t;
        for(int k = 0; k<Perm.length-1; k++){
        if(flag == 0){
            for(int i =0; i<Perm.length-1; i++){
                if(Perm[i]>Perm[i+1]){
                    t = Perm[i];
```

```
                    Perm[i] = Perm[i+1];
                    Perm[i+1] = t;
                    count++;
                    flag = 1;
                }
            }
        }
        if(flag == 1){
            for(int i =Perm.length-1; i>0; i--){
                if(Perm[i]<Perm[i-1]){
                    t = Perm[i];
                    Perm[i] = Perm[i-1];
                    Perm[i-1] = t;
                    count++;
                    flag = 0;
                }
            }
        }
        }
        return count;

    }

    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        int N;
        N=scan.nextInt();
        int[] Perm = new int[N];
        for(int j=0;j<N;j++){
            Perm[j]=scan.nextInt();
        }
        int result;
        result = minTurns(N,Perm);
        System.out.print(result);
        return ;
    }
}
```

Note: Only 3 test case are passed some of them are wrong

6. Happy Numbers

Bob likes to work with extremely large numbers, one such number is **LARGE.** Since **LARGE** cannot be comprehended easily, he factors the number **LARGE** into **N** factors and stores them in array of size **N** called **AR** .

There are certain digits that make Bob happy, These digits are *3,6,9* and these three digits are called **Happy digits**.

*A **Happy Number*** is any number that consists of only **Happy digits.** This means that 3, 36, 936 are all examples of **Happy Numbers** while 25, 360,13 are not **Happy Numbers.**

Bob wants to perform some operations on the number **LARGE**, such that the number of trailing zeroes in the final value of **LARGE** can be maximized.
The operation that Bob can perform to maximize trailing zeroes in **LARGE** is described below

- Bob can multiply **LARGE** by any **Happy Number** and assign their product to **LARGE**. Thus, **LARGE** can be assigned the value of **LARGE x (any Happy Number)**

It is given that Bob can perform the above given operations any number of times.

Your task is to find the maximum possible number of trailing zeroes in **LARGE.**

Note:

- AR[1] x AR[2] . . . x AR[N] = LARGE
- The array AR is referenced using 1-based indexing
- It is given that 70100 has two trailing zeroes.

Function description
Complete the ***trailingZeroes*** function in the editor below. It has the following parameter(s):

Name	Type	Description
N	INTEGER	Number of factors LARGE is divided into
AR	INTEGER ARRAY	An array consisting of N factors
Return	The function must return an *INTEGER* denoting the Maximum number of trailing zeroes possible.	

Constraints

- 1 **≤ N ≤** 2 x 10^5
- 1 **≤ AR[i] ≤** 10^9

Input format for debugging

- The first line contains an integer, N, denoting the number of elements in AR.
- Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing ARi.

Sample Testcases

Input	Output	Output Description
2 1 17	0	value of LARGE is 1x17 = 17 multiplying 17 with any Happy number does not produce any trailing zero. ANS=0
2 15 7	1	value of LARGE is 15x7 = 105 multiplying 105 with an Happy number(36) makes the value of LARGE as 3780.

Input	Output	Output Description
		which has 1 trailing zero. ANS=1
3 1 5 6	1	value of LARGE is 1x5x6 = 30 There is already 1 trailing zero. Multiplying any other happy number does not increase trailing zeroes. ANS=1

Program

```
import java.io.*;
import java.util.*;
import java.lang.Math;
public class Solution {
    public static int trailingZeroes(int N,int[] AR){
    int trailingZeroes=0;
    for(int num : AR){
        trailingZeroes+=countFactorsOfFive(num);
    }
    return trailingZeroes;
    }
    private static int countFactorsOfFive(int num){
        int factorsOfFive=0;
        while(num>0&&num%5==0){
            factorsOfFive++;
            num/=5;
        }
        return factorsOfFive;
    }

    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        int N;
        N=scan.nextInt();
        int[] AR = new int[N];
        for(int j=0;j<N;j++){
            AR[j]=scan.nextInt();
```

```
        }
        int result;
        result = trailingZeroes(N,AR);
        System.out.print(result);
        return ;
    }
}
```

7. Farthest Pairs

You are given an array **A** of length **N** representing a **permutation**. Find **two indices i** and **j** such that:

1. **i<j**
2. $\mathbf{A_i<A_j}$
3. **j−i** is **maximized**

If there are **no** such pairs you should return -1 else return the **maximum possible** value of **j-i**.

Note: An array of **N** integers is called a **permutation** if it contains **all** integers from 1 to **N** **exactly once**.
Function description
Complete the ***farthestPairs*** function in the editor below. It has the following parameter(s):

Name	Type	Description
N	INTEGER	The size of the given array.
A	INTEGER ARRAY	The given array.
Return	The function must return an *INTEGER* denoting the maximum possible value of j-i such that Ai<Aj and i<j.	

Constraints

- $1 \leq \mathbf{N} \leq 10^5$
- $1 \leq \mathbf{A[i]} \leq N$

Input format for debugging

- The first line contains an integer, N, denoting the number of elements in A.
- Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing A[i].

Sample Testcases

Input	Output	Output Description
3 3 2 1	-1	Here there are no such pairs.
4 1 2 3 4	3	The farthest pair satisfying the given property is (A[0], A[3]) hence answer is 3-0=3.
5 5 4 1 3 2	2	The farthest pair satisfying the given property is (A[2], A[4]) hence answer is 4-2=2.

Program

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


public class Solution {
    public static int farthestPairs(int N, List<Integer> A) {
        int max = 0, count =0;
        for(int i = 0; i<N-1; i++){
            // System.out.println("in for loop");
            for(int j = i+1; j<=N-1; j++)
            {
```

```
                // System.out.println(A.get(i)+"<"+A.get(j));
                if(A.get(i)<A.get(j) && i<j){
                    // System.out.println(A.get(i)+"<"+A.get(j));
                   max = Math.max(max,j-i);
                   // max =max<j-i?j-1:max;
                    count++;
                    //  System.out.println(max+"if");
                }
            }
        }
        if(count == 0)
            return -1;
        else
            return max;
  }


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

        int N = Integer.parseInt(scan.nextLine().trim());

        List<Integer> A = new ArrayList<>(N);
        for(int j=0; j<N; j++) {
            A.add(Integer.parseInt(scan.nextLine().trim()));
        }

        int result = farthestPairs(N, A);

        System.out.println(result);
    }
}
```

Note: 11 Test cases are passed out of 15 all other are Time limit exceed