**Fill array with 1's**

Show Topic Tags   

[Amazon](http://practice.geeksforgeeks.org/company/Amazon/)

Given an array of 0 and 1, in how many iterations the whole array be filled with 1's if in a single iteration immediate neighbours of 1's can be filled.

**NOTE:**If you cannot fill array with 1's then print "-1" .

**Input:**  
The first line contains a single integer T i.e. the number of test cases. The first line consists of a single integer N(length of array) . The next line consists of a N spaced integers(either 0 or 1).

**Output:**  
In one line print the minimum number of iterations to fill the whole string with 1's.

**Constraints:**  
1<=T<=100  
1<=N<=1000  
0<=A[i]<=1  
  
**Examples:**  
**Input:**  
2  
15  
1 0 1 0 0 1 0 1 1 0 1 1 0 0 1  
17  
0 0 1 1 0 0 1 1 0 1 1 1 1 0 0 0 1

**Output:**  
1  
2  
  
**Explanation:**  
**Test Case 1:**  
To convert the whole array into 1's, one iteration is required . Between index i=2 and i=5 , the zero at i=3 would be converted to '1' due to its neighbour at i=2 similarly the zero at i=4 would be converted into '1' due to its neighbour at i=5 , all this can be done in a single iteration.

\*\*For More Examples Use Expected Output\*\*

<http://practice.geeksforgeeks.org/problems/fill-array-by-1s/0>

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class GFG {

public static void main(String[] args) throws IOException {

// TODO code application logic here

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int t = Integer.parseInt(br.readLine());

while(t-- > 0) {

int n = Integer.parseInt(br.readLine());

//int[] arr = { 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1 };

//int[] arr = { 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1 };

String[] input = br.readLine().trim().split(" ");

int[] arr = new int[n];

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

arr[i] = Integer.parseInt(input[i]);

}

//String[] input = "0 1 1 1 1 0 0 1 1 0 1 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 1 0 1 0 1 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 1 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 0 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 1 1 1".trim().split(" ");

//String[] input = "0 0 0 0 0 0".trim().split(" ");

//String[] input = "1 1 0 0 1 0 0 0 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 1 1 1 1 1 0 1 0 1 0 0 1 0 1 1 1 0 1 0 1 0 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 0 0 1 1 0 0 1 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 0 0 1 1 0 1 0 1 0 1 1 1 1 0 1 0 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 1 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 0 0 1 0 1 1 0 1 0 1 0 1 0 1 0 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 0 1 0 1 1 1 0 1 1 0 0 1 1 0 1 1 0 1 0 0 1 0 0 0 0 1 1 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 1 1 0 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 1 0 0 0 1 1 0 1 1 1 0 0 0 0 1 1 1 1 0 1 1 1 1 0 0 0 0".trim().split( " ");

/\*

int[] arr = new int[input.length];

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

arr[i] = Integer.parseInt(input[i]);

}\*/

int primer1 =-1; // Arrays.asList(arr).indexOf(1); //Array.IndexOf(arr, 1);

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

if(arr[i] == 1) {

primer1 = i;

break;

}

}

//System.out.println(primer1);

//int primer1 = Array.IndexOf(arr, 1);

if(primer1 == -1) {

System.out.println(-1);

//return;

//System.out.println(1);

continue;

}

//max ceros consecutivos en extremos

int i = 0;

int izq = 0;

while (i < arr.length && arr[i] == 0)

{

i++;

izq++;

}

i = arr.length - 1;

int der = 0;

while (i >= 0 && arr[i] == 0)

{

i--;

der++;

}

int medio = 0, max\_medio=0;

i = -1; // Arrays.asList(arr).indexOf(1); //Array.IndexOf(arr, 1);

for(int k =0; k<arr.length; k++) {

if(arr[k] == 1) {

i=k;

break;

}

}

int j = arr.length-1; //Array.LastIndexOf(arr, 1);

for(;j>=0; j-- ){

if(arr[j] == 1){

break;

}

}

boolean hay\_unos = false;

while (i < arr.length && arr[i] == 1)

{

hay\_unos = true;

i++;

}

if (hay\_unos)

{

for (j = primer1; j < arr.length; j++)

{

if (arr[j] == 0)

{

break;

}

}

i =j;

while (i < arr.length)

{

medio = 0;

while (i < arr.length && arr[i] == 0)

{

medio++;

i++;

}

max\_medio = Math.max(medio, max\_medio);

while (i < arr.length && arr[i] == 1)

{

i++;

}

}

}

//Console.WriteLine(izq + " " + max\_medio + " " + der);

if (max\_medio > izq\*2 && max\_medio > der\*2)

{

System.out.println((int)Math.ceil((double) (max\_medio /2.0)));

}

else

{

int max = Math.max(izq, der);

System.out.println(max);

}

}

}

}