Day 81 coding Statement :

You have a **binary** string *S* of length *N*. In one operation you can select a substring of *S* and **reverse** it. For example, on reversing the substring [2,4]*S*[2,4] for *S*=11000, we change 11000→10010.

Find the **minimum** number of operations required to sort this binary string.
It can be proven that the string can always be sorted using the above operation finite number of times.

Input Format

- The first line of input will contain a single integer *T*, denoting the number of test cases.
- Each test case consists of 22 lines of input.
  - The first line of each test case contains a single integer *N* — the length of the binary string.
  - The second line of each test case contains a binary string *S* of length *N*.

Output Format

For each test case, output on a new line — the minimum number of operations required to sort the binary string.

Sample Input

4

3

000

4

1001

4

1010

6

010101

Sample Output

0

1

2

2

Explanation:

Test case 1: The string is already sorted, hence, zero operations are required to sort it.

Test case 2: We can sort the string in the following way: 1001 →0011.

Test case 3: We can sort the string in the following way:
1010→ 1100 → 0011.
It can be proven that this string cannot be sorted in less than 2 operations.

Test case 4: We can sort the string in the following way:
010101 → 001011→ 000111.
It can be proven that this string cannot be sorted in less than 2 operations.

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

class Main {
	public static void main(String[] args) throws java.lang.Exception {
		Scanner scan = new Scanner(System.in);
		int times = scan.nextInt();

		while (times-- > 0) {
			int val = scan.nextInt();
			scan.nextLine();

			String s = scan.nextLine();
			int count = 0;
			for (int i = s.length(); i >= 2; i--) {
				if (s.substring(i - 2, i).equals("10"))
					count++;
			}
			System.out.println(count);
		}
	}
}
```