PREREQUISITES:

Binary search or two-pointers

PROBLEM:

A binary string A is called good if it can be sorted by some sequence of adjacent swaps, such that any position is swapped with its right at most once.

Given a binary string S, how many of its substrings are good?

EXPLANATION:

The very first thing to do is to come up with a nice enough condition that tells us when a binary string is good.

▾ The condition

Let A be a binary string. We'd like to sort A using some adjacent swaps.

Suppose $A_i = 1$ and $A_j = 0$, where $i < j$.
We definitely need to swap these at some point to sort the string.
This means that, no matter what, we must use the swaps at positions $i, i+1, i+2, \ldots, j-1$.

In particular, if we had to use some of these positions to swap a different pair of 1 and 0, we'd be in trouble.
This should immediately tell you that if A contains 1100 as a subsequence, it cannot be good.

It's now not hard to see that if A doesn't contain 1100 as a subsequence, it will always be good.

This gives us a nice reduction in what we want to compute: all we need to do now is compute the number of substrings that don't contain 1100 as a subsequence.
While this is easy to do in $\mathcal{O}(N^2)$, that's obviously too slow.

Instead, we make one more observation. Let $S[L, R]$ denote the substring of S starting at L and ending at R.
Does knowing something about the goodness of $S[L, R]$ tell you something about $S[L, R-1]$ and/or $S[L, R+1]$?

Answer

If $S[L, R]$ is good, then so is $S[L, R-1]$.
Conversely, if $S[L, R]$ is not good, then neither is $S[L, R+1]$.

In particular, this tells us that if we fix L, the set of R such that $S[L, R]$ is good forms a continuous range starting at L.

So, let's fix L and try to find the maximum R such that $S[L, R]$ is good: then, we can add $R - L + 1$ to our answer since that's the number of good substrings starting at L.

To find R, we instead do the opposite: find the first time a 1100 subsequence forms, then end at the character right before that.
Finding the first time 1100 forms as a subsequence is not hard, and follows from the standard greedy algorithm to check whether one string is a subsequence of another:

- Let $i_1 \geq L$ be the first time we see a 1
- Let $i_2 > i_1$ be the first time we see a 1
- Let $i_3 > i_2$ be the first time we see a 0
- Let $i_4 > i_3$ be the first time we see a 0

Then, $R = i_4 - 1$.

Finding i_1, i_2, i_3, i_4 can each be done in $\mathcal{O}(\log N)$ if we have a sorted list of positions of the ones and zeros and simply binary search on this: for example, you can use `std::upper_bound` in C++ to simplify implementation.

So, we've found the optimal R for a fixed L in $\mathcal{O}(\log N)$. Do this for every L and add up the answers, giving us a solution in $\mathcal{O}(N \log N)$.

It is also possible to implement this in $\mathcal{O}(N)$ using two-pointers instead of binary search.

TIME COMPLEXITY

$\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ per test case.

```
for _ in range(int(input())):
    n = int(input())
    s = input()
    ans = 0
    p1 = p2 = p3 = p4 = n
    for i in reversed(range(n)):
        if s[i] == '1':
            zeros = []
            for j in range(p1, p2):
                if s[j] == '0': zeros.append(j)
            if len(zeros) > 1:
                p3 = zeros[0]
                p4 = zeros[1]
            elif len(zeros) == 1:
                p4 = p3
                p3 = zeros[0]
            p2 = p1
            p1 = i
        ans += p4 - i
    print(ans)
```