REPORT

Q1:

At first, I was thinking about using ordinary methods to solve this problem, which means to calculate how many times the same element has show up. If >=2 then +2, cuz there are positive and negative ways, if =1 then 1 way, if the element is 0 then only 1 distinct way, however, the time is too long, the last 5 questions are all out of the limit. Then I think about some particular algorithm which can help me shorten the time, so I use segment tree to change the element in the list, which is operation 2. then I accomplish 80% of the questions. But for the large numbers, the proceeding time is still too much. Then I look at the hint, I choose to import sys to simplify I/O procedures to improve efficiency, it didn't work too well.

The turning point would be I figure out that there should be someway to link the operations, which means to actually update every time when ak changes. then I really understand what this question really want us to do. I use cpp to write the code again using a brand new thought(for I learned cpp in high school kind of a first language more familiar to me), here s my final thinking:

I can create a list simply where the index is the number, and the element is the times that it is shown then I can look at it whenever I want. Here I notice that a is between P and -P!!! and when ak changes the circumstances will differ when the original and transferring element has shown how many times. The count updates evertime when ak is added. Because ak is deleted first, so we first let countarr of ak declines by 1. As for the total counts, When ak equals 0 , if now there are no 0(abs 0 count change is 1 to 0) then totalcount -1, if ak does not equals 0 ,then if now there are one or 0 of ak(it means that abs number change is 2 to one, or 1 to 0), totalcount -1, as for other situations, no need to change.
After adding ak into the permutation, we first add the countarr of ak element, Now we count totalcounts again, also separated into 0 and other consequences to calculate, if ak = 0 and now there are one 0, then add one. If ak is not 0 and now there are 1 or 2 of this absolute ak, totalcount add 1. In other situations, there are no change because we have already 2 of them.

So this solution will highly improve the efficiency and is easy to understand. And I only used 83 lines to accomplish.

Q2:

For q2, at first I did not understand that every subarrays can be cut, so I followed one line to identify whether the different index in the next index pairs. However, after realizing the question, the index pairs should be much more, and the situations are a lot more. So basically the logic is:
From the bottom to the top, when is the next index pair available? When 1, they are totally out of this index pair, 2, they include one side of this index pair, and enlarge the other side. When does it go wrong? When the next pair is inside, or cross or containing the index pair.

So we can create a dictionary left_map which is important to store all the left indexes which has already been tested. Also, we create a list to record all the times that the number has been covered. After taking input, the indexes pairs are put in the reverse order. So if in this pair, the left side has been covered but not in the left_map, this means that the left side is already inside previous range, WRONG!　If the right side is inside the right side of the left side, this means that it is like [(……)…],() means the current pair, it is still WRONG. If left side has not been covered, then we look at every index in the current range to find if there is one element has been covered(here we want it to be out of the previous range), if there is one element has been covered, maybe we have a situation like([……]…), it is still available, so we check the first covered element's right side is it the right side of the current right side, if not it is WRONG.

Except these three circumstances, it is accepted. And I uploaded the code in to he system only to find that only 60% accuracy.
For every pair, the leftside index should be smaller than the rightside index!!! I didn't identify this one!
After making it up, my code is finally 100 accepted, and I think this code is super easy to understand and super efficient. Only using list and dictionary, we can implement the whole function, and I only used 50 lines.