Qiyao Zhou

Z5379852

Question 4

4.1:

Create a new array R to store the number of red flowers, set the initial number of red flowers r=0, iterate through the flowers array from the beginning and make a judgement each time, if it is red flower then r is added to 1 and put into the array, otherwise r is put directly into the array.

The whole algorithm only needs to traverse the array once, the time complexity is O (n), and after this pre-processing, each time to calculate the number of red flowers in the subarray only need to subtract the value of the index of the first and last element in the subarray in R to get, the time complexity is O (1). Note that the first element of the subarray referred to here is actually the previous element at the start of the subarray (unless the subarray starts at the first element of the flower array).

4.2:

1. First perform the array preprocessing shown in 4.1 to obtain the array R. This way the number of safflowers inside the array can be obtained by the difference between the values corresponding to the indexes of the first and last elements in R (noted as count_r=r_b-r_a), and the number of yellow flowers outside the subarray can be easily found (noted as count_y=n-b+a-r_n+r_b-r_a).

2. Iterate through the array R from the beginning, taking the point taken in each iteration as the starting point of the subarray, then apply a dichotomous lookup in the section from the starting point to the end of the array R to find the index minimum point where count_r>count_y, so that the index point i can be used as the end point of the subarray from then on, adding (n-i) to the total number of subarrays(Because a subarray with this point as the end of the subarray already ensures that the subarray condition is met, and moving the end further back ensures that the question condition is met, regardless of whether a red or yellow flower follows).

3.When the traversal is complete, the total number of subarrays is obtained.

The preprocessing time in the first step is O (n), the second step requires n iterations, each of which requires a dichotomous lookup of O (logn), and the value addition requires O (1), so the total time complexity is O (n)+O (nlogn+n) = O (nlogn)