A contiguous subarray is defined as unique if all the integers contained within it occur exactly once. There is a unique weight associated with each of the subarray. Unique weight for any subarray equals it's length if it's unique, 0 otherwise. Your task is to calculate the sum of unique weights of all the contiguous subarrays contained within a given array.

In computer science, the maximum **subarray** problem is the task of finding the**contiguous subarray** within a one-dimensional array of numbers which has the largest sum. For example, for the sequence of values −2, 1, −3, 4, −1, 2, 1, −5, 4; the**contiguous subarray** with the largest sum is 4, −1, 2, 1, with sum 6.

**Input**  
First line of the input contains an integer TT, denoting the number of testcases.  
2∗T2∗T lines follow, where first line of each testcase contains an integer NN denoting the number of integers in the given array. Last line of each testcase then contains NN single space separated integers

**Output**  
Print the summation of unique weights of all the subarrays for each testcase in a separate line.

**Constraints**

* 1≤T,N≤1051≤T,N≤105
* 0≤Ai≤1090≤Ai≤109
* Summation of NN for all TT does not exceed 105105

**SAMPLE INPUT**

1

5

1 2 3 4 5

**SAMPLE OUTPUT**

35

**Explanation**

Sample Case 1: Since, all integers are distinct within any contiguous subarray, therefore the unique weight will be the summation of lengths of all subarrays. Hence, this sums upto 5+4∗2+3∗3+2∗4+1∗5=355+4∗2+3∗3+2∗4+1∗5=35

**Time Limit:**1.0 sec(s) for each input file.

**Memory Limit:**256 MB

**Source Limit:**1024 KB

A:

**Solution** For each index ii in the array, we can find the corresponding index j such that j >= i and A[i..j] contains all distinct elements. Then, the number of subarrays contributing to the answer and starting from index i will be (j - i + 1). For each of these subarrays, their lengths will be of the form [1, 2, 3,. ... j - i + 1]. Sum of all these lengths will be ((j - i + 1)\*(j - i + 2))/2. Add up the answer for each of the index i similarly. Next question - How to find the longest contiguous subarray starting from index i? This can be done by maintaining two pointers i and j which convey the same meaning as described above. So, let's say you have current i and j already calculated for some subarray, now you increment i and correspondingly increment j unless (i + 1, j) are distinct. If not possible, then you increment i again and move forward the same way. This is typical two pointers approach.

Overall Complexity of the solution will be O(NlogN)

Note: Do not forget to take the answer in the long long data type.

For more details on the implementation, have a look at the setter's solution. To know more about the two pointers algorithm, see [here](https://tp-iiita.quora.com/The-Two-Pointer-Algorithm).

A:

**Tester Solution** by [FatalEagle](https://www.hackerearth.com/@fataleagle" \t "_blank)

1. #include <bits/stdc++.h>
2. using namespace std;
3. int T, N;
4. int main()
5. {
6. scanf("%d", &T);
7. assert(1<=T && T<=100000);
8. int S=0;
9. while(T--)
10. {
11. scanf("%d", &N);
12. assert(1<=N && N<=100000);
13. S+=N;
14. vector<int> A(N);
15. for(int i=0; i<N; i++)
16. {
17. scanf("%d", &A[i]);
18. assert(0<=A[i] && A[i]<=1000000000);
19. }
20. long long ans=0;
21. set<int> s;
22. for(int i=0, j=0; i<N; i++)
23. {
24. for(; j<N && !s.count(A[j]); j++)
25. s.insert(A[j]);
26. ans+=1LL\*(j-i)\*(j-i+1)/2;
27. s.erase(A[i]);
28. }
29. printf("%lld\n", ans);
30. }
31. assert(S<=100000);
32. return 0;
33. }