2461. Maximum Sum of Distinct Subarrays With Length K

You are given an integer array `nums` and an integer `k`. Find the maximum subarray sum of all the subarrays of `nums` that meet the following conditions:

- The length of the subarray is `k`, and
- All the elements of the subarray are **distinct**.

Return *the maximum subarray sum of all the subarrays that meet the conditions.* If no subarray meets the conditions, return `0`.

*A **subarray** is a contiguous non-empty sequence of elements within an array.*

Example 1:

```
Input: nums = [1,5,4,2,9,9,9], k = 3
Output: 15
Explanation: The subarrays of nums with length 3 are:
- [1,5,4] which meets the requirements and has a sum of 10.
- [5,4,2] which meets the requirements and has a sum of 11.
- [4,2,9] which meets the requirements and has a sum of 15.
- [2,9,9] which does not meet the requirements because the element 9 is repeated.
- [9,9,9] which does not meet the requirements because the element 9 is repeated.
We return 15 because it is the maximum subarray sum of all the subarrays that meet
the conditions
```

Example 2:

```
Input: nums = [4,4,4], k = 3
Output: 0
Explanation: The subarrays of nums with length 3 are:
- [4,4,4] which does not meet the requirements because the element 4 is repeated.
We return 0 because no subarrays meet the conditions.
```

Constraints:

- $1 <= k <= nums.length <= 10^5$
- $1 <= nums[i] <= 10^5$

2461. Maximum Sum of Distinct Subarrays With Length K

```
/*
   Sliding window (bad version)
   Time complexity:O(n.k)=O(n^2)
   Space complexity: O(1)
*/
class Solution {
public:
   // Function to check if the all the frequencies are equal to 1
   // means, check if all elements are distinct
   bool is_distinct(std::unordered_map<int,int>& freq){
      for(auto& [e,f]: freq){
         if(f>1) return false;
      }
      return true;
   }
   long long maximumSubarraySum(std::vector<int>& nums, int k) {
      int n=nums.size();
      long long ans=0;
      std::unordered_map<int,int> freq;
      // Create the window of size k
      long long win=0;
      for(int i=0;i<k;++i){
         win+=nums[i];
         freq[nums[i]]++;
      }
      // If all elemnts of the window are distincts, ans= sum of all elements of the window
      if(is_distinct(freq)) ans=win;
```

```
        // For each window
        for(int i=0;i<n-k;++i){
            // Slide the window to the right
            win-=nums[i];
            win+=nums[i+k];

            // Update frequencies
            freq[nums[i]]--;
            freq[nums[i+k]]++;

            // If all elemnts of the window are distincts, maximize the ans
            if(is_distinct(freq)) ans=std::max(ans,win);
        }
        return ans;
    }
};
```

2461. Maximum Sum of Distinct Subarrays With Length K

```
/*
   Sliding window (optimal version)
   Time complexity:O(k+n)=O(n)
   Space complexity: O(n)
*/
class Solution {
   public:
      long long maximumSubarraySum(std::vector<int>& nums, int k) {
         int n=nums.size();
         long long ans=0;
         std::unordered_map<int,int> freq;

         // Create the window of size k
         long long win=0;
         for(int i=0;i<k;++i){
            win+=nums[i];
            freq[nums[i]]++;
         }

         // If all elemnts of the window are distincts, ans= sum of all elements of the window
         if(freq.size()==k) ans=win;

         // For each window
         for(int i=0;i<n-k;++i){
            // Slide the window to the right
            win-=nums[i];
            win+=nums[i+k];

            // Update frequencies
            freq[nums[i]]--;
            freq[nums[i+k]]++;

            // if the frequency of the kicked element become 0
            // means it does not exist in the actual window
            // so remove it
            if(freq[nums[i]]==0) freq.erase(nums[i]);

            // If all elemnts of the window are distincts, maximize the ans
            if(freq.size()==k) ans=std::max(ans,win);
         }
         return ans;
      }
};
```