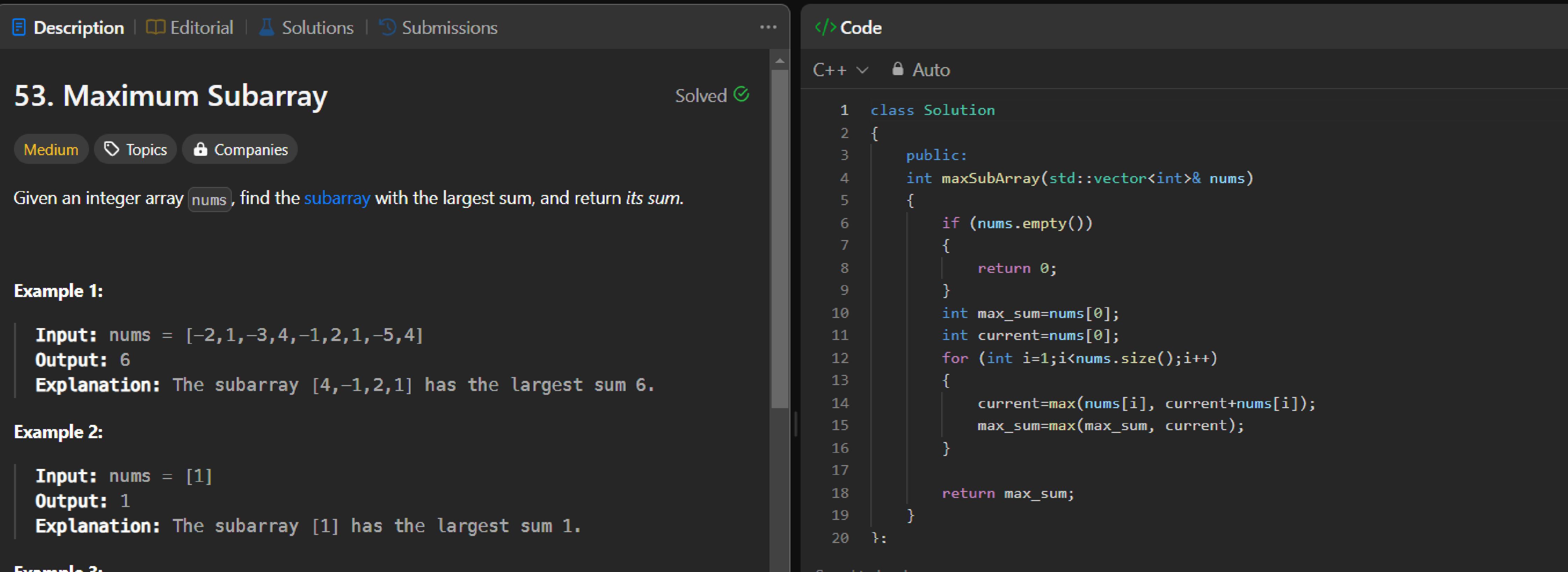


53. Maximum Subarray

Given an integer array `nums`, find the subarray with the largest sum, and return *its sum*.

Example 1:

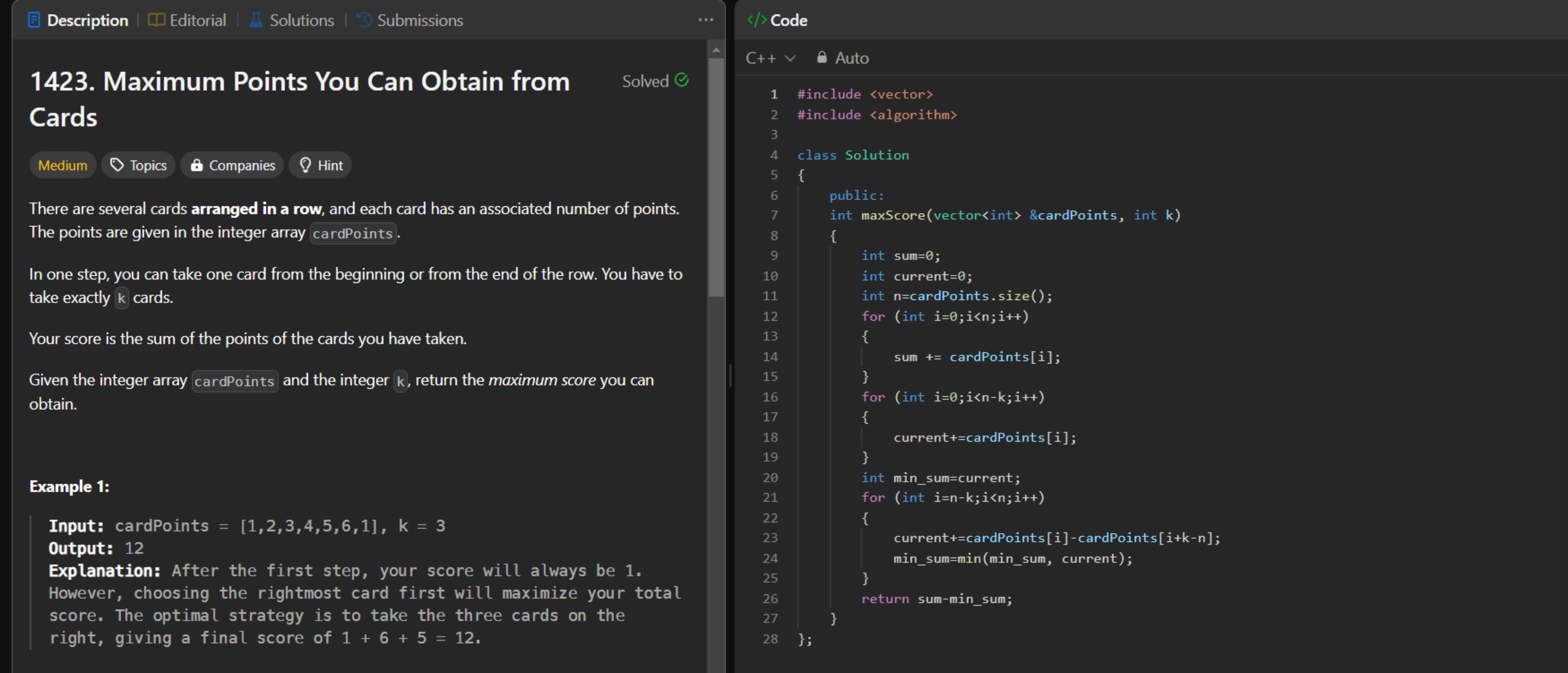
```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.
```

```cpp
class Solution
{
    public:
    int maxSubArray(std::vector<int>& nums)
    {
        if (nums.empty())
        {
            return 0;
        }
        int max_sum=nums[0];
        int current=nums[0];
        for (int i=1;i<nums.size();i++)
        {
            current=max(nums[i], current+nums[i]);
            max_sum=max(max_sum, current);
        }

        return max_sum;
    }
};
```



1423. Maximum Points You Can Obtain from Cards

Solved ✓

Medium | Topics | Companies | Hint

There are several cards **arranged in a row**, and each card has an associated number of points. The points are given in the integer array `cardPoints`.

In one step, you can take one card from the beginning or from the end of the row. You have to take exactly `k` cards.

Your score is the sum of the points of the cards you have taken.

Given the integer array `cardPoints` and the integer `k`, return the *maximum score* you can obtain.

Example 1:

```
Input: cardPoints = [1,2,3,4,5,6,1], k = 3
Output: 12
Explanation: After the first step, your score will always be 1.
However, choosing the rightmost card first will maximize your total
score. The optimal strategy is to take the three cards on the
right, giving a final score of 1 + 6 + 5 = 12.
```

```cpp
#include <vector>
#include <algorithm>

class Solution
{
    public:
    int maxScore(vector<int> &cardPoints, int k)
    {
        int sum=0;
        int current=0;
        int n=cardPoints.size();
        for (int i=0;i<n;i++)
        {
            sum += cardPoints[i];
        }
        for (int i=0;i<n-k;i++)
        {
            current+=cardPoints[i];
        }
        int min_sum=current;
        for (int i=n-k;i<n;i++)
        {
            current+=cardPoints[i]-cardPoints[i+k-n];
            min_sum=min(min_sum, current);
        }
        return sum-min_sum;
    }
};
```