1423. Maximum Points You Can Obtain from Cards

Leetcode Link

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 $1 + 6 + 5 = 12$.

Example 2:

Input: `cardPoints = [2,2,2], k = 2`
Output: 4
Explanation: Regardless of which two cards you take, your score will always be 4.

Example 3:

Input: `cardPoints = [9,7,7,9,7,7,9], k = 7`
Output: 55
Explanation: You have to take all the cards. Your score is the sum of points of all cards.

Constraints:

- $1 \leq$ `cardPoints.length` $\leq 10^5$
- $1 \leq$ `cardPoints[i]` $\leq 10^4$
- $1 \leq k \leq$ `cardPoints.length`

Solution

Brute Force

First, we can make the observation that there are only $\mathcal{O}(K)$ different choices of cards if we take exactly K cards. Let's assume we took exactly L $(0 \leq L \leq K)$ cards from the left. The number of cards we take from the right will be fixed as we'll take $K - L$ cards on the right to reach a total of K cards.

For each choice of K cards, we can just iterate through the K cards and find the sum. To find the maximum sum, we'll repeat this process for all $\mathcal{O}(K)$ choices. This solution will run in $\mathcal{O}(K^2)$.

Full Solution

Let's look at two different choices of K cards. The first choice will be to take L cards from the left and $K - L$ cards from the right. The second choice will be to take $L - 1$ cards from the left and $K - L + 1$ cards from the right. Instead of recalculating the sum for the second choice, we can adjust the sum from the first choice to be the second choice. We can notice that the difference between these two choices is that we added a card on the right side and we removed a card from the left side. By applying these changes, we can transition between two choices in $\mathcal{O}(1)$ instead of $\mathcal{O}(K)$.

Example

`cardPoints = [1,2,3,4,5,6,1], k = 3`

Let's look at the transition between two different choices in this example. Our first choice consists of 2 cards on the left and 1 card on the right. Our second choice will consist of 1 card on the left and 2 cards on the right.

The sum with our first choice is currently 4. Going from our first choice to our second choice, our left side lost a card and our right side gained a card. Specifically, we lost `cardPoints[1] = 2` and we gained `cardPoints[5] = 6`. After removing `cardPoints[1]` and adding `cardPoints[5]`, we obtain a sum of 8 for our second choice.

We can implement this with the idea of two pointers. We'll use the pointers to indicate which cards we picked on the left and right side. A simple way to implement this with two pointers is to start with picking all K cards on the left. For each transition, we'll remove one card on the left and add one card on the right. We keep repeating this process until we reach the choice where all our K cards are picked from the right.

Time Complexity

We'll require $\mathcal{O}(K)$ to calculate the sum for the first choice. In addition, it will take another $\mathcal{O}(K)$ process all $\mathcal{O}(K)$ choices. Thus, our final time complexity will be $\mathcal{O}(K)$.

Time Complexity: $\mathcal{O}(K)$.

Space Complexity

Since we use two pointers to maintain the sum of every choice, our space complexity is $\mathcal{O}(1)$.

Space Complexity: $\mathcal{O}(1)$.

C++ Solution

```cpp
class Solution {
    public:
    int maxScore(vector<int>& cardPoints, int k) {
        int n = cardPoints.size();
        int leftSum = 0;
        for (int i = 0; i < k;
             i++) {  // calculate sum where all cards on the left side
            leftSum += cardPoints[i];
        }
        int rightSum = 0;
        int rightIndex = n;  // pointer for the right cards
        int ans = leftSum;
        for (int leftIndex = k - 1; leftIndex >= 0;
             leftIndex--) {                  // pointer for the left cards
            leftSum -= cardPoints[leftIndex];  // transition between choices
            rightIndex--;
            rightSum += cardPoints[rightIndex];
            ans = max(ans, leftSum + rightSum);
        }
        return ans;
    }
};
```

Java Solution

```java
class Solution {
    public int maxScore(int[] cardPoints, int k) {
        int n = cardPoints.length;
        int leftSum = 0;
        for (int i = 0; i < k;
             i++) { // calculate sum where all cards on the left side
            leftSum += cardPoints[i];
        }
        int rightSum = 0;
        int rightIndex = n; // pointer for the right cards
        int ans = leftSum;
        for (int leftIndex = k - 1; leftIndex >= 0;
             leftIndex--) { // pointer for the left cards
            leftSum -= cardPoints[leftIndex]; // transition between choices
            rightIndex--;
            rightSum += cardPoints[rightIndex];
            ans = Math.max(ans, leftSum + rightSum);
        }
        return ans;
    }
}
```

Python Solution

```python
class Solution:
    def maxScore(self, cardPoints: List[int], k: int) -> int:
        n = len(cardPoints)
        leftSum = 0
        for i in range(k):  # calculate sum where all cards on the left side
            leftSum += cardPoints[i]
        rightSum = 0
        rightIndex = n  # pointer for the right cards
        ans = leftSum
        for leftIndex in range(k - 1, -1, -1):  # pointer for the left cards
            leftSum -= cardPoints[leftIndex]  # transition between choices
            rightIndex -= 1
            rightSum += cardPoints[rightIndex]
            ans = max(ans, leftSum + rightSum)
        return ans


```

Got a question? Ask the Teaching Assistant anything you don't understand.