Problem 1423: Maximum Points You Can Obtain from Cards

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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 <= cardPoints.length <= 10

5

1 <= cardPoints[i] <= 10

4

1 <= k <= cardPoints.length

Code Snippets

C++:

```
class Solution {
public:
int maxScore(vector<int>& cardPoints, int k) {

}
};
```

Java:

```
class Solution {
public int maxScore(int[] cardPoints, int k) {
```

```
}
}
```

Python3:

```
class Solution:
def maxScore(self, cardPoints: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def maxScore(self, cardPoints, k):
"""
:type cardPoints: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} cardPoints
* @param {number} k
* @return {number}
*/
var maxScore = function(cardPoints, k) {

};
```

TypeScript:

```
function maxScore(cardPoints: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaxScore(int[] cardPoints, int k) {

}
```

```
}
```

C:

```
int maxScore(int* cardPoints, int cardPointsSize, int k) {

}
```

Go:

```
func maxScore(cardPoints []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxScore(cardPoints: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxScore(_ cardPoints: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_score(card_points: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} card_points
# @param {Integer} k
```

```
# @return {Integer}
def max_score(card_points, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $cardPoints
* @param Integer $k
* @return Integer
*/
function maxScore($cardPoints, $k) {

}
}
```

Dart:

```
class Solution {
int maxScore(List<int> cardPoints, int k) {

}
}
```

Scala:

```
object Solution {
def maxScore(cardPoints: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_score(card_points :: [integer], k :: integer) :: integer
def max_score(card_points, k) do

end
```

```
end
```

Erlang:

```
-spec max_score(CardPoints :: [integer()], K :: integer()) -> integer().
max_score(CardPoints, K) ->
.
```

Racket:

```
(define/contract (max-score cardPoints k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Points You Can Obtain from Cards
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxScore(vector<int>& cardPoints, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Points You Can Obtain from Cards
* Difficulty: Medium
```

```
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxScore(int[] cardPoints, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Points You Can Obtain from Cards
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxScore(self, cardPoints: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxScore(self, cardPoints, k):
"""
:type cardPoints: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Points You Can Obtain from Cards
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} cardPoints
* @param {number} k
* @return {number}
*/
var maxScore = function(cardPoints, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Points You Can Obtain from Cards
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxScore(cardPoints: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Points You Can Obtain from Cards
* Difficulty: Medium
* Tags: array, dp
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxScore(int[] cardPoints, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Points You Can Obtain from Cards
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxScore(int* cardPoints, int cardPointsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Points You Can Obtain from Cards
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxScore(cardPoints []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxScore(cardPoints: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxScore(_ cardPoints: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Points You Can Obtain from Cards
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_score(card_points: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} card_points
# @param {Integer} k
# @return {Integer}
def max_score(card_points, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $cardPoints
* @param Integer $k
* @return Integer
*/
function maxScore($cardPoints, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxScore(List<int> cardPoints, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxScore(cardPoints: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_score(card_points :: [integer], k :: integer) :: integer
def max_score(card_points, k) do

end
end
```

Erlang Solution:

```
-spec max_score(CardPoints :: [integer()], K :: integer()) -> integer().
max_score(CardPoints, K) ->
.
```

Racket Solution:

```
(define/contract (max-score cardPoints k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```