Problem 1231: Divide Chocolate

Problem Information

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

Problem Description

You have one chocolate bar that consists of some chunks. Each chunk has its own sweetness given by the array

sweetness

.

You want to share the chocolate with your

k

friends so you start cutting the chocolate bar into

k + 1

pieces using

k

cuts, each piece consists of some

consecutive

chunks.

Being generous, you will eat the piece with the minimum total sweetness

and give the other pieces to your friends.

Find the

maximum total sweetness

of the piece you can get by cutting the chocolate bar optimally.

Example 1:

Input:

sweetness = [1,2,3,4,5,6,7,8,9], k = 5

Output:

6

Explanation:

You can divide the chocolate to [1,2,3], [4,5], [6], [7], [8], [9]

Example 2:

Input:

sweetness = [5,6,7,8,9,1,2,3,4], k = 8

Output:

1

Explanation:

There is only one way to cut the bar into 9 pieces.

Example 3:

Input:

sweetness = [1,2,2,1,2,2,1,2,2], k = 2

Output:

5

Explanation:

You can divide the chocolate to [1,2,2], [1,2,2], [1,2,2]

Constraints:

0 <= k < sweetness.length <= 10

4

1 <= sweetness[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int maximizeSweetness(vector<int>& sweetness, int k) {

}
};
```

Java:

```
class Solution {
public int maximizeSweetness(int[] sweetness, int k) {
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximizeSweetness(sweetness: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaximizeSweetness(int[] sweetness, int k) {

}
}
```

C:

```
int maximizeSweetness(int* sweetness, int sweetnessSize, int k) {

}
```

Go:

```
func maximizeSweetness(sweetness []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maximizeSweetness(sweetness: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} sweetness
# @param {Integer} k
# @return {Integer}
def maximize_sweetness(sweetness, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $sweetness
* @param Integer $k
* @return Integer
*/
function maximizeSweetness($sweetness, $k) {

}
}
```

Dart:

```
class Solution {
int maximizeSweetness(List<int> sweetness, int k) {

}
}
```

Scala:

```
object Solution {
def maximizeSweetness(sweetness: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximize_sweetness(sweetness :: [integer], k :: integer) :: integer
def maximize_sweetness(sweetness, k) do

end
```

```
end
```

Erlang:

```
-spec maximize_sweetness(Sweetness :: [integer()], K :: integer()) ->
integer().
maximize_sweetness(Sweetness, K) ->
.
```

Racket:

```
(define/contract (maximize-sweetness sweetness k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Divide Chocolate
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int maximizeSweetness(vector<int>& sweetness, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Divide Chocolate
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int maximizeSweetness(int[] sweetness, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Divide Chocolate
Difficulty: Hard
Tags: array, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Divide Chocolate
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Divide Chocolate
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maximizeSweetness(sweetness: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Divide Chocolate
* Difficulty: Hard
* Tags: array, search
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MaximizeSweetness(int[] sweetness, int k) {

}
}
```

C Solution:

```
/*
* Problem: Divide Chocolate
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int maximizeSweetness(int* sweetness, int sweetnessSize, int k) {

}
```

Go Solution:

```
// Problem: Divide Chocolate
// Difficulty: Hard
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maximizeSweetness(sweetness []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximizeSweetness(sweetness: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Divide Chocolate
// Difficulty: Hard
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn maximize_sweetness(sweetness: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} sweetness
# @param {Integer} k
# @return {Integer}
def maximize_sweetness(sweetness, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $sweetness
* @param Integer $k
* @return Integer
*/
function maximizeSweetness($sweetness, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maximizeSweetness(List<int> sweetness, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maximizeSweetness(sweetness: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximize_sweetness(sweetness :: [integer], k :: integer) :: integer
def maximize_sweetness(sweetness, k) do

end
end
```

Erlang Solution:

```
-spec maximize_sweetness(Sweetness :: [integer()], K :: integer()) ->
integer().
maximize_sweetness(Sweetness, K) ->
.
```

Racket Solution:

```
(define/contract (maximize-sweetness sweetness k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```