Problem 2551: Put Marbles in Bags

Problem Information

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

Problem Description

You have

k

bags. You are given a

0-indexed

integer array

weights

where

weights[i]

is the weight of the

i

th

marble. You are also given the integer

k.

Divide the marbles into the

k

bags according to the following rules:

No bag is empty.

If the

i

th

marble and

j

th

marble are in a bag, then all marbles with an index between the

i

th

and

j

th

indices should also be in that same bag.

If a bag consists of all the marbles with an index from

i

to

j

inclusively, then the cost of the bag is

weights[i] + weights[j]

.

The

score

after distributing the marbles is the sum of the costs of all the

k

bags.

Return

the

difference

between the

maximum

and

minimum

scores among marble distributions

.

Example 1:

Input:

weights = [1,3,5,1], k = 2

Output:

4

Explanation:

The distribution [1],[3,5,1] results in the minimal score of (1+1) + (3+1) = 6. The distribution [1,3],[5,1], results in the maximal score of (1+3) + (5+1) = 10. Thus, we return their difference 10 - 6 = 4.

Example 2:

Input:

weights = [1, 3], k = 2

Output:

0

Explanation:

The only distribution possible is [1],[3]. Since both the maximal and minimal score are the same, we return 0.

Constraints:

1 <= k <= weights.length <= 10

5

1 <= weights[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long putMarbles(vector<int>& weights, int k) {

}
};
```

Java:

```
class Solution {
public long putMarbles(int[] weights, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function putMarbles(weights: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long PutMarbles(int[] weights, int k) {

}
}
```

C:

```
long long putMarbles(int* weights, int weightsSize, int k) {

}
```

Go:

```
func putMarbles(weights []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun putMarbles(weights: IntArray, k: Int): Long {

}
}
```

Swift:

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

```
}
}
```

Rust:

```
impl Solution {
pub fn put_marbles(weights: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} weights
# @param {Integer} k
# @return {Integer}
def put_marbles(weights, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $weights
* @param Integer $k
* @return Integer
*/
function putMarbles($weights, $k) {

}
}
```

Dart:

```
class Solution {
int putMarbles(List<int> weights, int k) {

}
}
```

Scala:

```
object Solution {
def putMarbles(weights: Array[Int], k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec put_marbles(weights :: [integer], k :: integer) :: integer
def put_marbles(weights, k) do

end
end
```

Erlang:

```
-spec put_marbles(Weights :: [integer()], K :: integer()) -> integer().
put_marbles(Weights, K) ->
.
```

Racket:

```
(define/contract (put-marbles weights k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Put Marbles in Bags
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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:
long long putMarbles(vector<int>& weights, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Put Marbles in Bags
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 long putMarbles(int[] weights, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Put Marbles in Bags
Difficulty: Hard
Tags: array, greedy, sort, queue, heap

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 putMarbles(self, weights: List[int], k: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Put Marbles in Bags
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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[]} weights
* @param {number} k
* @return {number}
*/
var putMarbles = function(weights, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Put Marbles in Bags
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 putMarbles(weights: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Put Marbles in Bags
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 long PutMarbles(int[] weights, int k) {

}
}
```

C Solution:

```
/*
* Problem: Put Marbles in Bags
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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
*/

long long putMarbles(int* weights, int weightsSize, int k) {

}
```

Go Solution:

```
// Problem: Put Marbles in Bags
// Difficulty: Hard
// Tags: array, greedy, sort, queue, heap
//
// 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 putMarbles(weights []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun putMarbles(weights: IntArray, k: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Put Marbles in Bags
// Difficulty: Hard
// Tags: array, greedy, sort, queue, heap
//
// 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 put_marbles(weights: Vec<i32>, k: i32) -> i64 {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer[]} weights
# @param {Integer} k
# @return {Integer}
def put_marbles(weights, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $weights
* @param Integer $k
* @return Integer
*/
function putMarbles($weights, $k) {

}
}
```

Dart Solution:

```
class Solution {
int putMarbles(List<int> weights, int k) {

}
}
```

Scala Solution:

```
object Solution {
def putMarbles(weights: Array[Int], k: Int): Long = {

}
```

```
}
```

Elixir Solution:

```
defmodule Solution do
@spec put_marbles(weights :: [integer], k :: integer) :: integer
def put_marbles(weights, k) do

end
end
```

Erlang Solution:

```
-spec put_marbles(Weights :: [integer()], K :: integer()) -> integer().
put_marbles(Weights, K) ->
.
```

Racket Solution:

```
(define/contract (put-marbles weights k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```