Problem 1561: Maximum Number of Coins You Can Get

Problem Information

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

Problem Description

There are

3n

piles of coins of varying size, you and your friends will take piles of coins as follows:

In each step, you will choose

any

3

piles of coins (not necessarily consecutive).

Of your choice, Alice will pick the pile with the maximum number of coins.

You will pick the next pile with the maximum number of coins.

Your friend Bob will pick the last pile.

Repeat until there are no more piles of coins.

Given an array of integers

piles

where

piles[i]

is the number of coins in the

i

th

pile.

Return the maximum number of coins that you can have.

Example 1:

Input:

piles = [2,4,1,2,7,8]

Output:

9

Explanation:

Choose the triplet (2, 7, 8), Alice Pick the pile with 8 coins, you the pile with

7

coins and Bob the last one. Choose the triplet (1, 2, 4), Alice Pick the pile with 4 coins, you the pile with

2

coins and Bob the last one. The maximum number of coins which you can have are: 7 + 2 = 9. On the other hand if we choose this arrangement (1,

2

, 8), (2,

4

, 7) you only get 2 + 4 = 6 coins which is not optimal.

Example 2:

Input:

piles = [2,4,5]

Output:

4

Example 3:

Input:

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

Output:

18

Constraints:

3 <= piles.length <= 10

5

piles.length % 3 == 0

1 <= piles[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int maxCoins(vector<int>& piles) {

}
};
```

Java:

```
class Solution {
public int maxCoins(int[] piles) {

}
}
```

Python3:

```
class Solution:
def maxCoins(self, piles: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxCoins(self, piles):
"""
:type piles: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} piles
* @return {number}
*/
var maxCoins = function(piles) {

};
```

TypeScript:

```
function maxCoins(piles: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxCoins(int[] piles) {

}
}
```

C:

```
int maxCoins(int* piles, int pilesSize) {

}
```

Go:

```
func maxCoins(piles []int) int {

}
```

Kotlin:

```
class Solution {
fun maxCoins(piles: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxCoins(_ piles: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_coins(piles: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} piles
# @return {Integer}
def max_coins(piles)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $piles
* @return Integer
*/
function maxCoins($piles) {

}
}
```

Dart:

```
class Solution {
int maxCoins(List<int> piles) {

}
}
```

Scala:

```
object Solution {
def maxCoins(piles: Array[Int]): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec max_coins(piles :: [integer]) :: integer
def max_coins(piles) do

end
end
```

Erlang:

```
-spec max_coins(Piles :: [integer()]) -> integer().
max_coins(Piles) ->
.
```

Racket:

```
(define/contract (max-coins piles)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Coins You Can Get
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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 maxCoins(vector<int>& piles) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Coins You Can Get
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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 maxCoins(int[] piles) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Coins You Can Get
Difficulty: Medium
Tags: array, greedy, math, sort

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 maxCoins(self, piles: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxCoins(self, piles):
"""
:type piles: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Coins You Can Get
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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[]} piles
* @return {number}
*/
var maxCoins = function(piles) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Coins You Can Get
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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 maxCoins(piles: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Coins You Can Get
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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 MaxCoins(int[] piles) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Coins You Can Get
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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 maxCoins(int* piles, int pilesSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Coins You Can Get
// Difficulty: Medium
// Tags: array, greedy, math, sort
//
// 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 maxCoins(piles []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxCoins(piles: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxCoins(_ piles: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Coins You Can Get
// Difficulty: Medium
// Tags: array, greedy, math, sort
//
// 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 max_coins(piles: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} piles
# @return {Integer}
def max_coins(piles)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $piles
* @return Integer
*/
function maxCoins($piles) {

}
}
```

Dart Solution:

```
class Solution {
int maxCoins(List<int> piles) {

}
}
```

Scala Solution:

```
object Solution {
def maxCoins(piles: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_coins(piles :: [integer]) :: integer
def max_coins(piles) do

end
```

```
end
```

Erlang Solution:

```
-spec max_coins(Piles :: [integer()]) -> integer().
max_coins(Piles) ->
.
```

Racket Solution:

```
(define/contract (max-coins piles)
(-> (listof exact-integer?) exact-integer?)
)
```