Problem 2177: Find Three Consecutive Integers That Sum to a Given Number

Problem Information

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

Problem Description

Given an integer

num

, return

three consecutive integers (as a sorted array)

that

sum

to

num

. If

num

cannot be expressed as the sum of three consecutive integers, return

an

empty

array.

Example 1:

Input:

num = 33

Output:

[10,11,12]

Explanation:

33 can be expressed as 10 + 11 + 12 = 33. 10, 11, 12 are 3 consecutive integers, so we return [10, 11, 12].

Example 2:

Input:

num = 4

Output:

[]

Explanation:

There is no way to express 4 as the sum of 3 consecutive integers.

Constraints:

0 <= num <= 10

Code Snippets

C++:

```
class Solution {
public:
vector<long long> sumOfThree(long long num) {

}
};
```

Java:

```
class Solution {
public long[] sumOfThree(long num) {

}
}
```

Python3:

```
class Solution:
def sumOfThree(self, num: int) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function sumOfThree(num: number): number[] {

};
```

C#:

```
public class Solution {
public long[] SumOfThree(long num) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* sumOfThree(long long num, int* returnSize) {

}
```

Go:

```
func sumOfThree(num int64) []int64 {

}
```

Kotlin:

```
class Solution {
fun sumOfThree(num: Long): LongArray {

}
}
```

Swift:

```
class Solution {
func sumOfThree(_ num: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn sum_of_three(num: i64) -> Vec<i64> {

}
}
```

Ruby:

```
# @param {Integer} num
# @return {Integer[]}
def sum_of_three(num)

end
```

PHP:

```
class Solution {

/**
* @param Integer $num
* @return Integer[]
*/
function sumOfThree($num) {

}
}
```

Dart:

```
class Solution {
List<int> sumOfThree(int num) {

}
}
```

Scala:

```
object Solution {
def sumOfThree(num: Long): Array[Long] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sum_of_three(num :: integer) :: [integer]
def sum_of_three(num) do

end
end
```

Erlang:

```
-spec sum_of_three(Num :: integer()) -> [integer()].
sum_of_three(Num) ->
.
```

Racket:

```
(define/contract (sum-of-three num)
(-> exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Three Consecutive Integers That Sum to a Given Number
* Difficulty: Medium
* Tags: array, 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:
vector<long long> sumOfThree(long long num) {

}
};
```

Java Solution:

```
/**
* Problem: Find Three Consecutive Integers That Sum to a Given Number
* Difficulty: Medium
* Tags: array, 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 long[] sumOfThree(long num) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Three Consecutive Integers That Sum to a Given Number
Difficulty: Medium
Tags: array, 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 sumOfThree(self, num: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Three Consecutive Integers That Sum to a Given Number
* Difficulty: Medium
* Tags: array, 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} num
* @return {number[]}
*/
var sumOfThree = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Three Consecutive Integers That Sum to a Given Number
* Difficulty: Medium
* Tags: array, 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 sumOfThree(num: number): number[] {
};
```

C# Solution:

```
/*
* Problem: Find Three Consecutive Integers That Sum to a Given Number
* Difficulty: Medium
* Tags: array, 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 long[] SumOfThree(long num) {

}
}
```

C Solution:

```
/*
* Problem: Find Three Consecutive Integers That Sum to a Given Number
* Difficulty: Medium
* Tags: array, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* sumOfThree(long long num, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find Three Consecutive Integers That Sum to a Given Number
// Difficulty: Medium
// Tags: array, 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 sumOfThree(num int64) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun sumOfThree(num: Long): LongArray {

}
}
```

Swift Solution:

```
class Solution {
func sumOfThree(_ num: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find Three Consecutive Integers That Sum to a Given Number
// Difficulty: Medium
// Tags: array, 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 sum_of_three(num: i64) -> Vec<i64> {

}
}
```

Ruby Solution:

```
# @param {Integer} num
# @return {Integer[]}
def sum_of_three(num)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $num
* @return Integer[]
*/
function sumOfThree($num) {

}
}
```

Dart Solution:

```
class Solution {
List<int> sumOfThree(int num) {

}
}
```

Scala Solution:

```
object Solution {
def sumOfThree(num: Long): Array[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sum_of_three(num :: integer) :: [integer]
def sum_of_three(num) do

end
end
```

Erlang Solution:

```
-spec sum_of_three(Num :: integer()) -> [integer()].
sum_of_three(Num) ->
.
```

Racket Solution:

```
(define/contract (sum-of-three num)
(-> exact-integer? (listof exact-integer?))
)
```