Problem 330: Patching Array

Problem Information

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

Problem Description

Given a sorted integer array

nums

and an integer

n

, add/patch elements to the array such that any number in the range

[1, n]

inclusive can be formed by the sum of some elements in the array.

Return

the minimum number of patches required

.

Example 1:

Input:

nums = [1,3], n = 6

Output:

1 Explanation: Combinations of nums are [1], [3], [1,3], which form possible sums of: 1, 3, 4. Now if we add/patch 2 to nums, the combinations are: [1], [2], [3], [1,3], [2,3], [1,2,3]. Possible sums are 1, 2, 3, 4, 5, 6, which now covers the range [1, 6]. So we only need 1 patch.

Example 2:

Input:

nums = [1,5,10], n = 20

Output:

2 Explanation: The two patches can be [2, 4].

Example 3:

Input:

nums = [1,2,2], n = 5

Output:

0

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 10

4

nums

is sorted in

ascending order

.

1 <= n <= 2

31

- 1

Code Snippets

C++:

```
class Solution {
public:
int minPatches(vector<int>& nums, int n) {

}
};
```

Java:

```
class Solution {
public int minPatches(int[] nums, int n) {

}
}
```

Python3:

```
class Solution:
def minPatches(self, nums: List[int], n: int) -> int:
```

Python:

```
class Solution(object):
def minPatches(self, nums, n):
"""
:type nums: List[int]
:type n: int
:rtype: int
```

```
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} n
* @return {number}
*/
var minPatches = function(nums, n) {

};
```

TypeScript:

```
function minPatches(nums: number[], n: number): number {

};
```

C#:

```
public class Solution {
public int MinPatches(int[] nums, int n) {

}
}
```

C:

```
int minPatches(int* nums, int numsSize, int n) {

}
```

Go:

```
func minPatches(nums []int, n int) int {

}
```

Kotlin:

```
class Solution {
fun minPatches(nums: IntArray, n: Int): Int {

}
}
```

Swift:

```
class Solution {
func minPatches(_ nums: [Int], _ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_patches(nums: Vec<i32>, n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} n
# @return {Integer}
def min_patches(nums, n)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $n
* @return Integer
*/
function minPatches($nums, $n) {

}
}
```

Dart:

```
class Solution {
int minPatches(List<int> nums, int n) {

}
}
```

Scala:

```
object Solution {
def minPatches(nums: Array[Int], n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_patches(nums :: [integer], n :: integer) :: integer
def min_patches(nums, n) do

end
end
```

Erlang:

```
-spec min_patches(Nums :: [integer()], N :: integer()) -> integer().
min_patches(Nums, N) ->
.
```

Racket:

```
(define/contract (min-patches nums n)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Patching Array
* Difficulty: Hard
* Tags: array, greedy, 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 minPatches(vector<int>& nums, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Patching Array
* Difficulty: Hard
* Tags: array, greedy, 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 minPatches(int[] nums, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Patching Array
Difficulty: Hard
Tags: array, greedy, 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 minPatches(self, nums: List[int], n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minPatches(self, nums, n):
"""
:type nums: List[int]
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Patching Array
* Difficulty: Hard
* Tags: array, greedy, 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[]} nums
* @param {number} n
* @return {number}
*/
var minPatches = function(nums, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Patching Array
* Difficulty: Hard
* Tags: array, greedy, 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 minPatches(nums: number[], n: number): number {

};
```

C# Solution:

```
/*
* Problem: Patching Array
* Difficulty: Hard
* Tags: array, greedy, 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 MinPatches(int[] nums, int n) {

}
}
```

C Solution:

```
/*
* Problem: Patching Array
* Difficulty: Hard
* Tags: array, greedy, 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 minPatches(int* nums, int numsSize, int n) {

}
```

Go Solution:

```
// Problem: Patching Array
// Difficulty: Hard
// Tags: array, greedy, 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 minPatches(nums []int, n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minPatches(nums: IntArray, n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minPatches(_ nums: [Int], _ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Patching Array
// Difficulty: Hard
// Tags: array, greedy, 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 min_patches(nums: Vec<i32>, n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} n
# @return {Integer}
def min_patches(nums, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $n
* @return Integer
*/
function minPatches($nums, $n) {

}
}
```

Dart Solution:

```
class Solution {
int minPatches(List<int> nums, int n) {

}
}
```

Scala Solution:

```
object Solution {
def minPatches(nums: Array[Int], n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_patches(nums :: [integer], n :: integer) :: integer
def min_patches(nums, n) do

end
end
```

Erlang Solution:

```
-spec min_patches(Nums :: [integer()], N :: integer()) -> integer().
min_patches(Nums, N) ->
.
```

Racket Solution:

```
(define/contract (min-patches nums n)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```