Problem 881: Boats to Save People

Problem Information

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

Problem Description

You are given an array

people

where

people[i]

is the weight of the

i

th

person, and an

infinite number of boats

where each boat can carry a maximum weight of

limit

. Each boat carries at most two people at the same time, provided the sum of the weight of those people is at most

limit

.

Return

the minimum number of boats to carry every given person

.

Example 1:

Input:

people = [1,2], limit = 3

Output:

1

Explanation:

1 boat (1, 2)

Example 2:

Input:

people = [3,2,2,1], limit = 3

Output:

3

Explanation:

3 boats (1, 2), (2) and (3)

Example 3:

Input:

people = [3,5,3,4], limit = 5

Output:

4

Explanation:

4 boats (3), (3), (4), (5)

Constraints:

1 <= people.length <= 5 * 10

4

1 <= people[i] <= limit <= 3 * 10

4

Code Snippets

C++:

```
class Solution {
public:
int numRescueBoats(vector<int>& people, int limit) {

}
};
```

Java:

```
class Solution {
public int numRescueBoats(int[] people, int limit) {

}
```

```
}
```

Python3:

```
class Solution:
def numRescueBoats(self, people: List[int], limit: int) -> int:
```

Python:

```
class Solution(object):
def numRescueBoats(self, people, limit):
"""
:type people: List[int]
:type limit: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} people
* @param {number} limit
* @return {number}
*/
var numRescueBoats = function(people, limit) {

};
```

TypeScript:

```
function numRescueBoats(people: number[], limit: number): number {

};
```

C#:

```
public class Solution {
public int NumRescueBoats(int[] people, int limit) {

}
}
```

C:

```
int numRescueBoats(int* people, int peopleSize, int limit) {

}
```

Go:

```
func numRescueBoats(people []int, limit int) int {

}
```

Kotlin:

```
class Solution {
fun numRescueBoats(people: IntArray, limit: Int): Int {

}
}
```

Swift:

```
class Solution {
func numRescueBoats(_ people: [Int], _ limit: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_rescue_boats(people: Vec<i32>, limit: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} people
# @param {Integer} limit
# @return {Integer}
def num_rescue_boats(people, limit)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $people
* @param Integer $limit
* @return Integer
*/
function numRescueBoats($people, $limit) {

}
}
```

Dart:

```
class Solution {
int numRescueBoats(List<int> people, int limit) {

}
}
```

Scala:

```
object Solution {
def numRescueBoats(people: Array[Int], limit: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_rescue_boats(people :: [integer], limit :: integer) :: integer
def num_rescue_boats(people, limit) do

end
end
```

Erlang:

```
-spec num_rescue_boats(People :: [integer()], Limit :: integer()) ->
integer().
num_rescue_boats(People, Limit) ->
.
```

Racket:

```
(define/contract (num-rescue-boats people limit)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Boats to Save People
* Difficulty: Medium
* 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 numRescueBoats(vector<int>& people, int limit) {

}
};
```

Java Solution:

```
/**
* Problem: Boats to Save People
* Difficulty: Medium
* 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 numRescueBoats(int[] people, int limit) {

}
}
```

Python3 Solution:

```
"""
Problem: Boats to Save People
Difficulty: Medium
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 numRescueBoats(self, people: List[int], limit: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numRescueBoats(self, people, limit):
"""
:type people: List[int]
:type limit: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Boats to Save People
* Difficulty: Medium
```

```
* 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[]} people
* @param {number} limit
* @return {number}
*/
var numRescueBoats = function(people, limit) {

};
```

TypeScript Solution:

```
/**
* Problem: Boats to Save People
* Difficulty: Medium
* 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 numRescueBoats(people: number[], limit: number): number {

};
```

C# Solution:

```
/*
* Problem: Boats to Save People
* Difficulty: Medium
* 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 NumRescueBoats(int[] people, int limit) {

}
}
```

C Solution:

```
/*
* Problem: Boats to Save People
* Difficulty: Medium
* 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 numRescueBoats(int* people, int peopleSize, int limit) {

}
```

Go Solution:

```
// Problem: Boats to Save People
// Difficulty: Medium
// 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 numRescueBoats(people []int, limit int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numRescueBoats(people: IntArray, limit: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numRescueBoats(_ people: [Int], _ limit: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Boats to Save People
// Difficulty: Medium
// 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 num_rescue_boats(people: Vec<i32>, limit: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} people
# @param {Integer} limit
# @return {Integer}
def num_rescue_boats(people, limit)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $people
* @param Integer $limit
* @return Integer
*/
function numRescueBoats($people, $limit) {

}
}
```

Dart Solution:

```
class Solution {
int numRescueBoats(List<int> people, int limit) {

}
}
```

Scala Solution:

```
object Solution {
def numRescueBoats(people: Array[Int], limit: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_rescue_boats(people :: [integer], limit :: integer) :: integer
def num_rescue_boats(people, limit) do

end
end
```

Erlang Solution:

```
-spec num_rescue_boats(People :: [integer()], Limit :: integer()) ->
integer().
num_rescue_boats(People, Limit) ->
```

```
.
```

Racket Solution:

```
(define/contract (num-rescue-boats people limit)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```