# Problem 546: Remove Boxes

## Problem Information

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

## Problem Description

You are given several

boxes

with different colors represented by different positive numbers.

You may experience several rounds to remove boxes until there is no box left. Each time you can choose some continuous boxes with the same color (i.e., composed of

k

boxes,

k >= 1

), remove them and get

k * k

points.

Return

the maximum points you can get

.

Example 1:

Input:

boxes = [1,3,2,2,2,3,4,3,1]

Output:

23

Explanation:

[1, 3, 2, 2, 2, 3, 4, 3, 1] ----> [1, 3, 3, 4, 3, 1] (3*3=9 points) ----> [1, 3, 3, 3, 1] (1*1=1 points) ----> [1, 1] (3*3=9 points) ----> [] (2*2=4 points)

Example 2:

Input:

boxes = [1,1,1]

Output:

9

Example 3:

Input:

boxes = [1]

Output:

1

Constraints:

1 <= boxes.length <= 100

1 <= boxes[i] <= 100

## Code Snippets

### C++:

```
class Solution {
public:
int removeBoxes(vector<int>& boxes) {

}
};
```

### Java:

```
class Solution {
public int removeBoxes(int[] boxes) {

}
}
```

### Python3:

```
class Solution:
def removeBoxes(self, boxes: List[int]) -> int:
```

### Python:

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

### JavaScript:

```
/**
* @param {number[]} boxes
* @return {number}
*/
```

```
var removeBoxes = function(boxes) {

};
```

**TypeScript:**

```
function removeBoxes(boxes: number[]): number {

};
```

**C#:**

```
public class Solution {
public int RemoveBoxes(int[] boxes) {

}
}
```

**C:**

```
int removeBoxes(int* boxes, int boxesSize) {

}
```

**Go:**

```
func removeBoxes(boxes []int) int {

}
```

**Kotlin:**

```
class Solution {
fun removeBoxes(boxes: IntArray): Int {

}
}
```

**Swift:**

```
class Solution {
func removeBoxes(_ boxes: [Int]) -> Int {



}
}
```

**Rust:**

```
impl Solution {
pub fn remove_boxes(boxes: Vec<i32>) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} boxes
# @return {Integer}
def remove_boxes(boxes)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $boxes
* @return Integer
*/
function removeBoxes($boxes) {

}
}
```

**Dart:**

```
class Solution {
int removeBoxes(List<int> boxes) {

}
}
```

**Scala:**

```
object Solution {
def removeBoxes(boxes: Array[Int]): Int = {

}
}
```

### Elixir:

```
defmodule Solution do
@spec remove_boxes(boxes :: [integer]) :: integer
def remove_boxes(boxes) do

end
end
```

### Erlang:

```
-spec remove_boxes(Boxes :: [integer()]) -> integer().
remove_boxes(Boxes) ->
.
```

### Racket:

```
(define/contract (remove-boxes boxes)
(-> (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Remove Boxes
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
int removeBoxes(vector<int>& boxes) {

}
};
```

**Java Solution:**

```
/**
* Problem: Remove Boxes
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int removeBoxes(int[] boxes) {

}
}
```

**Python3 Solution:**

```
"""
Problem: Remove Boxes
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def removeBoxes(self, boxes: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Remove Boxes
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} boxes
* @return {number}
*/
var removeBoxes = function(boxes) {

};
```

### TypeScript Solution:

```
/**
* Problem: Remove Boxes
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function removeBoxes(boxes: number[]): number {
};
```

**C# Solution:**

```
/*
* Problem: Remove Boxes
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int RemoveBoxes(int[] boxes) {

}
}
```

**C Solution:**

```
/*
* Problem: Remove Boxes
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int removeBoxes(int* boxes, int boxesSize) {

}
```

**Go Solution:**

```
// Problem: Remove Boxes
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func removeBoxes(boxes []int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun removeBoxes(boxes: IntArray): Int {

}
}
```

### Swift Solution:

```
class Solution {
func removeBoxes(_ boxes: [Int]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Remove Boxes
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn remove_boxes(boxes: Vec<i32>) -> i32 {

}
}
```

## Ruby Solution:

```
# @param {Integer[]} boxes
# @return {Integer}
def remove_boxes(boxes)

end
```

## PHP Solution:

```
class Solution {

/**
* @param Integer[] $boxes
* @return Integer
*/
function removeBoxes($boxes) {

}
}
```

## Dart Solution:

```
class Solution {
int removeBoxes(List<int> boxes) {

}
}
```

## Scala Solution:

```
object Solution {
def removeBoxes(boxes: Array[Int]): Int = {

}
}
```

## Elixir Solution:

```
defmodule Solution do
@spec remove_boxes(boxes :: [integer]) :: integer
def remove_boxes(boxes) do
```

```
end
end
```

## Erlang Solution:

```
-spec remove_boxes(Boxes :: [integer()]) -> integer().
remove_boxes(Boxes) ->
.
```

## Racket Solution:

```
(define/contract (remove-boxes boxes)
(-> (listof exact-integer?) exact-integer?)
)
```