Problem 1580: Put Boxes Into the Warehouse II

Problem Information

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

Problem Description

You are given two arrays of positive integers,

boxes

and

warehouse

, representing the heights of some boxes of unit width and the heights of

n

rooms in a warehouse respectively. The warehouse's rooms are labeled from

0

to

n - 1

from left to right where

warehouse[i]

(0-indexed) is the height of the

i

th

room.

Boxes are put into the warehouse by the following rules:

Boxes cannot be stacked.

You can rearrange the insertion order of the boxes.

Boxes can be pushed into the warehouse from

either side

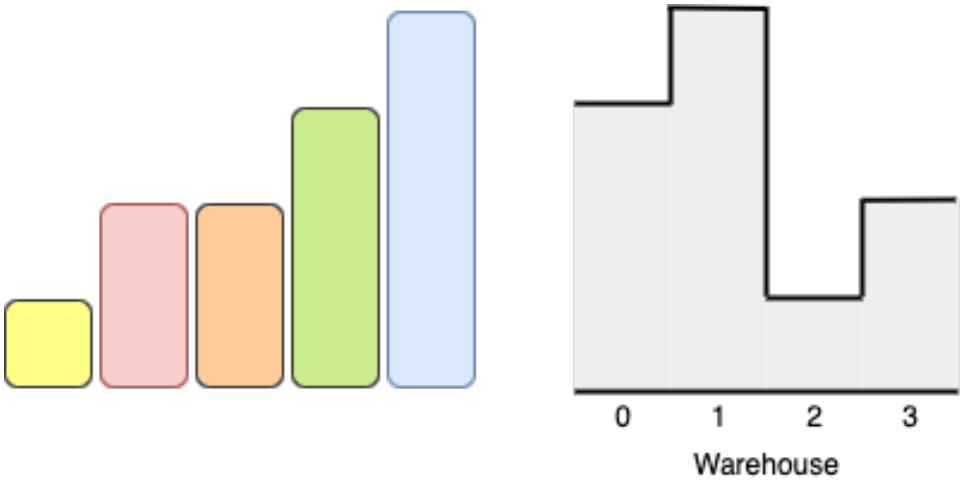
(left or right)

If the height of some room in the warehouse is less than the height of a box, then that box and all other boxes behind it will be stopped before that room.

Return

the maximum number of boxes you can put into the warehouse.

Example 1:



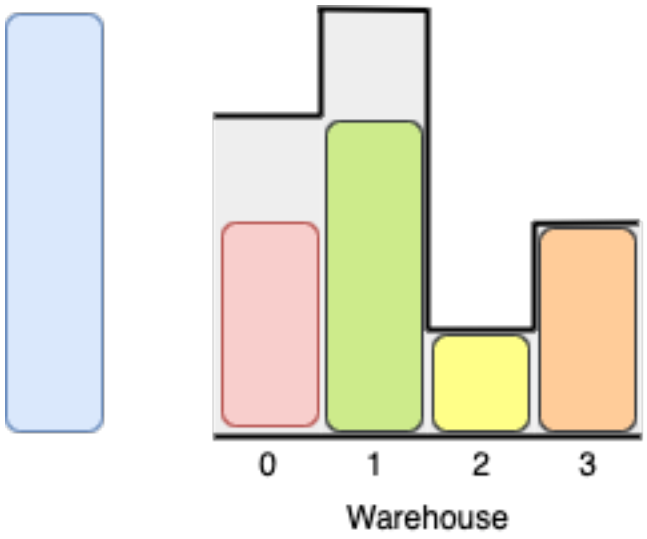
Input:

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

Output:

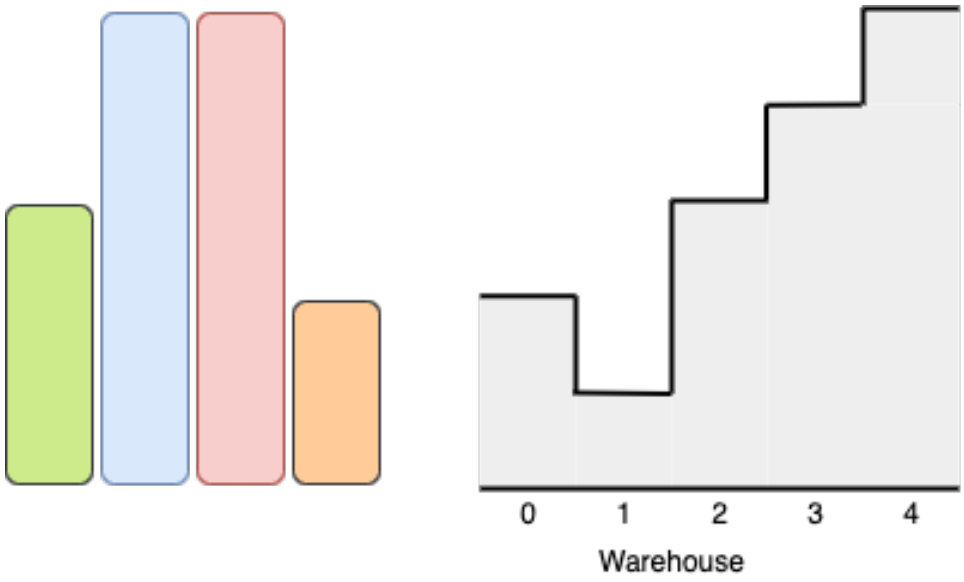
4

Explanation:



We can store the boxes in the following order: 1- Put the yellow box in room 2 from either the left or right side. 2- Put the orange box in room 3 from the right side. 3- Put the green box in room 1 from the left side. 4- Put the red box in room 0 from the left side. Notice that there are other valid ways to put 4 boxes such as swapping the red and green boxes or the red and orange boxes.

Example 2:



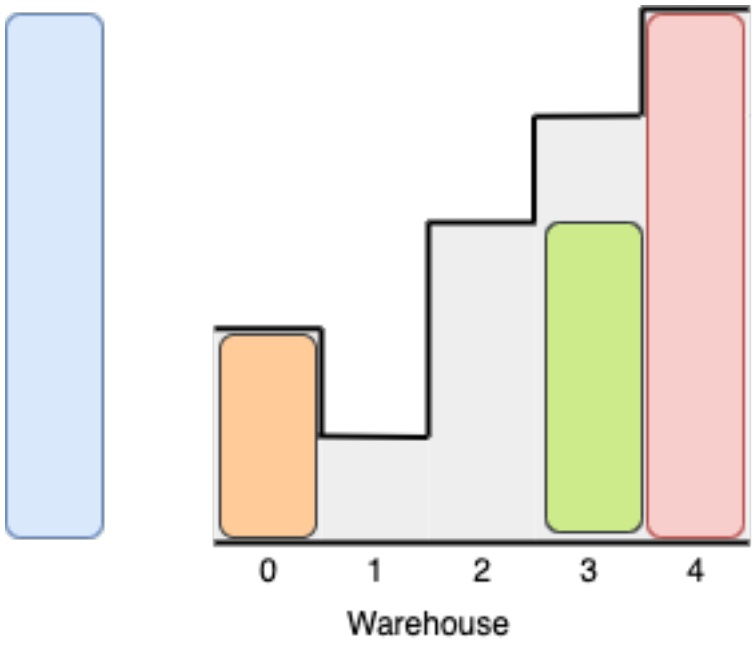
Input:

boxes = [3,5,5,2], warehouse = [2,1,3,4,5]

Output:

3

Explanation:



It is not possible to put the two boxes of height 5 in the warehouse since there's only 1 room of height >= 5. Other valid solutions are to put the green box in room 2 or to put the orange box first in room 2 before putting the green and red boxes.

Constraints:

n == warehouse.length

1 <= boxes.length, warehouse.length <= 10

5

1 <= boxes[i], warehouse[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxBoxesInWarehouse(vector<int>& boxes, vector<int>& warehouse) {

}
};
```

Java:

```
class Solution {
public int maxBoxesInWarehouse(int[] boxes, int[] warehouse) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} boxes
* @param {number[]} warehouse
* @return {number}
*/
var maxBoxesInWarehouse = function(boxes, warehouse) {

};
```

TypeScript:

```
function maxBoxesInWarehouse(boxes: number[], warehouse: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxBoxesInWarehouse(int[] boxes, int[] warehouse) {

}
}
```

C:

```
int maxBoxesInWarehouse(int* boxes, int boxesSize, int* warehouse, int
warehouseSize) {

}
```

Go:

```
func maxBoxesInWarehouse(boxes []int, warehouse []int) int {

}
```

Kotlin:

```
class Solution {
fun maxBoxesInWarehouse(boxes: IntArray, warehouse: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxBoxesInWarehouse(_ boxes: [Int], _ warehouse: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_boxes_in_warehouse(boxes: Vec<i32>, warehouse: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} boxes
# @param {Integer[]} warehouse
# @return {Integer}
def max_boxes_in_warehouse(boxes, warehouse)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $boxes
```

```
* @param Integer[] $warehouse
* @return Integer
*/
function maxBoxesInWarehouse($boxes, $warehouse) {

}
}
```

Dart:

```
class Solution {
int maxBoxesInWarehouse(List<int> boxes, List<int> warehouse) {

}
}
```

Scala:

```
object Solution {
def maxBoxesInWarehouse(boxes: Array[Int], warehouse: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_boxes_in_warehouse(boxes :: [integer], warehouse :: [integer]) ::
integer
def max_boxes_in_warehouse(boxes, warehouse) do

end
end
```

Erlang:

```
-spec max_boxes_in_warehouse(Boxes :: [integer()], Warehouse :: [integer()])
-> integer().
max_boxes_in_warehouse(Boxes, Warehouse) ->
.
```

Racket:

```
(define/contract (max-boxes-in-warehouse boxes warehouse)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Put Boxes Into the Warehouse II
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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 maxBoxesInWarehouse(vector<int>& boxes, vector<int>& warehouse) {

}
};
```

Java Solution:

```
/**
* Problem: Put Boxes Into the Warehouse II
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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 maxBoxesInWarehouse(int[] boxes, int[] warehouse) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Put Boxes Into the Warehouse II
Difficulty: Medium
Tags: array, greedy, sort, stack

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 maxBoxesInWarehouse(self, boxes: List[int], warehouse: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Put Boxes Into the Warehouse II
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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[]} boxes
* @param {number[]} warehouse
* @return {number}
*/
var maxBoxesInWarehouse = function(boxes, warehouse) {

};
```

TypeScript Solution:

```
/**
* Problem: Put Boxes Into the Warehouse II
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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 maxBoxesInWarehouse(boxes: number[], warehouse: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Put Boxes Into the Warehouse II
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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 MaxBoxesInWarehouse(int[] boxes, int[] warehouse) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Put Boxes Into the Warehouse II
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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 maxBoxesInWarehouse(int* boxes, int boxesSize, int* warehouse, int
warehouseSize) {

}
```

Go Solution:

```
// Problem: Put Boxes Into the Warehouse II
// Difficulty: Medium
// Tags: array, greedy, sort, stack
//
// 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 maxBoxesInWarehouse(boxes []int, warehouse []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxBoxesInWarehouse(boxes: IntArray, warehouse: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxBoxesInWarehouse(_ boxes: [Int], _ warehouse: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Put Boxes Into the Warehouse II
// Difficulty: Medium
// Tags: array, greedy, sort, stack
//
// 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_boxes_in_warehouse(boxes: Vec<i32>, warehouse: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} boxes
# @param {Integer[]} warehouse
# @return {Integer}
def max_boxes_in_warehouse(boxes, warehouse)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $boxes
* @param Integer[] $warehouse
* @return Integer
*/
```

```
function maxBoxesInWarehouse($boxes, $warehouse) {

}
}
```

Dart Solution:

```
class Solution {
int maxBoxesInWarehouse(List<int> boxes, List<int> warehouse) {

}
}
```

Scala Solution:

```
object Solution {
def maxBoxesInWarehouse(boxes: Array[Int], warehouse: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_boxes_in_warehouse(boxes :: [integer], warehouse :: [integer]) ::
integer
def max_boxes_in_warehouse(boxes, warehouse) do

end
end
```

Erlang Solution:

```
-spec max_boxes_in_warehouse(Boxes :: [integer()], Warehouse :: [integer()])
-> integer().
max_boxes_in_warehouse(Boxes, Warehouse) ->
.
```

Racket Solution:

```
(define/contract (max-boxes-in-warehouse boxes warehouse)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```