Problem 1889: Minimum Space Wasted From Packaging

Problem Information

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

Problem Description

You have

n

packages that you are trying to place in boxes,

one package in each box

. There are

m

suppliers that each produce boxes of

different sizes

(with infinite supply). A package can be placed in a box if the size of the package is

less than or equal to

the size of the box.

The package sizes are given as an integer array

packages

, where

packages[i]

is the

size

of the

i

th

package. The suppliers are given as a 2D integer array

boxes

, where

boxes[j]

is an array of

box sizes

that the

j

th

supplier produces.

You want to choose a

single supplier

and use boxes from them such that the

total wasted space

is

minimized

. For each package in a box, we define the space

wasted

to be

size of the box - size of the package

. The

total wasted space

is the sum of the space wasted in

all

the boxes.

For example, if you have to fit packages with sizes

[2,3,5]

and the supplier offers boxes of sizes

[4,8]

, you can fit the packages of size-

2

and size-

3

into two boxes of size-

4

and the package with size-

5

into a box of size-

8

. This would result in a waste of

(4-2) + (4-3) + (8-5) = 6

.

Return

the

minimum total wasted space

by choosing the box supplier

optimally

, or

-1

if it is

impossible

to fit all the packages inside boxes.

Since the answer may be

large

, return it

modulo

10

9

+ 7

.

Example 1:

Input:

packages = [2,3,5], boxes = [[4,8],[2,8]]

Output:

6

Explanation

: It is optimal to choose the first supplier, using two size-4 boxes and one size-8 box. The total waste is (4-2) + (4-3) + (8-5) = 6.

Example 2:

Input:

packages = [2,3,5], boxes = [[1,4],[2,3],[3,4]] Output:

-1

Explanation:

There is no box that the package of size 5 can fit in.

Example 3:

Input:

packages = [3,5,8,10,11,12], boxes = [[12],[11,9],[10,5,14]]

Output:

9

Explanation:

It is optimal to choose the third supplier, using two size-5 boxes, two size-10 boxes, and two size-14 boxes. The total waste is (5-3) + (5-5) + (10-8) + (10-10) + (14-11) + (14-12) = 9.

Constraints:

n == packages.length

m == boxes.length

1 <= n <= 10

5

1 <= m <= 10

5

1 <= packages[i] <= 10

5

1 <= boxes[j].length <= 10

5

1 <= boxes[j][k] <= 10

5

sum(boxes[j].length) <= 10

5

The elements in

boxes[j]

are

distinct

.

Code Snippets

C++:

```
class Solution {
public:
int minWastedSpace(vector<int>& packages, vector<vector<int>>& boxes) {

}
};
```

Java:

```
class Solution {
public int minWastedSpace(int[] packages, int[][] boxes) {



}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minWastedSpace(packages: number[], boxes: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinWastedSpace(int[] packages, int[][] boxes) {

}


}
```

C:

```
int minWastedSpace(int* packages, int packagesSize, int** boxes, int
boxesSize, int* boxesColSize) {

}
```

Go:

```
func minWastedSpace(packages []int, boxes [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minWastedSpace(packages: IntArray, boxes: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} packages
# @param {Integer[][]} boxes
# @return {Integer}
def min_wasted_space(packages, boxes)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $packages
* @param Integer[][] $boxes
* @return Integer
*/
function minWastedSpace($packages, $boxes) {

}
}
```

Dart:

```
class Solution {
int minWastedSpace(List<int> packages, List<List<int>> boxes) {

}
}
```

Scala:

```
object Solution {
def minWastedSpace(packages: Array[Int], boxes: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_wasted_space(packages :: [integer], boxes :: [[integer]]) ::
integer
def min_wasted_space(packages, boxes) do
```

```
end
end
```

Erlang:

```
-spec min_wasted_space(Packages :: [integer()], Boxes :: [[integer()]]) ->
integer().
min_wasted_space(Packages, Boxes) ->
.
```

Racket:

```
(define/contract (min-wasted-space packages boxes)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Space Wasted From Packaging
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 minWastedSpace(vector<int>& packages, vector<vector<int>>& boxes) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Space Wasted From Packaging
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 minWastedSpace(int[] packages, int[][] boxes) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Space Wasted From Packaging
Difficulty: Hard
Tags: array, sort, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Space Wasted From Packaging
* Difficulty: Hard
* Tags: array, sort, search
*
* 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[]} packages
* @param {number[][]} boxes
* @return {number}
*/
var minWastedSpace = function(packages, boxes) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Space Wasted From Packaging
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 minWastedSpace(packages: number[], boxes: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Space Wasted From Packaging
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 MinWastedSpace(int[] packages, int[][] boxes) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Space Wasted From Packaging
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 minWastedSpace(int* packages, int packagesSize, int** boxes, int
boxesSize, int* boxesColSize) {

}
```

Go Solution:

```
// Problem: Minimum Space Wasted From Packaging
// Difficulty: Hard
// Tags: array, sort, search
//
// 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 minWastedSpace(packages []int, boxes [][]int) int {
}
```

Kotlin Solution:

```
class Solution {
fun minWastedSpace(packages: IntArray, boxes: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Space Wasted From Packaging
// Difficulty: Hard
// Tags: array, sort, search
//
// 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_wasted_space(packages: Vec<i32>, boxes: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} packages
# @param {Integer[][]} boxes
# @return {Integer}
def min_wasted_space(packages, boxes)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $packages
* @param Integer[][] $boxes
* @return Integer
*/
function minWastedSpace($packages, $boxes) {

}
}
```

Dart Solution:

```
class Solution {
int minWastedSpace(List<int> packages, List<List<int>> boxes) {

}
}
```

Scala Solution:

```
object Solution {
def minWastedSpace(packages: Array[Int], boxes: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_wasted_space(packages :: [integer], boxes :: [[integer]]) ::
integer
def min_wasted_space(packages, boxes) do

end
```

```
end
```

Erlang Solution:

```
-spec min_wasted_space(Packages :: [integer()], Boxes :: [[integer()]]) ->
integer().
min_wasted_space(Packages, Boxes) ->
.
```

Racket Solution:

```
(define/contract (min-wasted-space packages boxes)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```