Problem 1691: Maximum Height by Stacking Cuboids

Problem Information

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

Problem Description

Given

n

cuboids

where the dimensions of the

i

th

cuboid is

cuboids[i] = [width

i

, length

i

, height

i

]

(

0-indexed

). Choose a

subset

of

cuboids

and place them on each other.

You can place cuboid

i

on cuboid

j

if

width

i

<= width

j

and

length

i

<= length

j

and

height

i

<= height

j

. You can rearrange any cuboid's dimensions by rotating it to put it on another cuboid.

Return

the

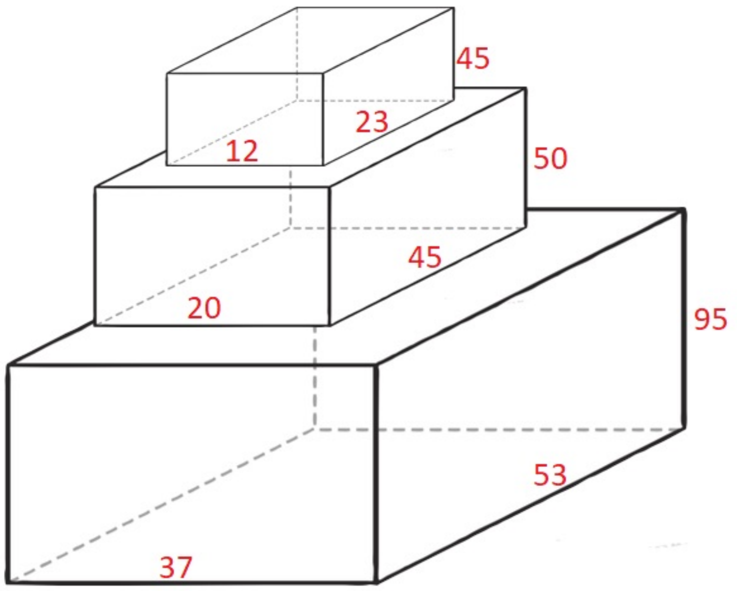
maximum height

of the stacked

cuboids

.

Example 1:



Input:

cuboids = [[50,45,20],[95,37,53],[45,23,12]]

Output:

190

Explanation:

Cuboid 1 is placed on the bottom with the 53x37 side facing down with height 95. Cuboid 0 is placed next with the 45x20 side facing down with height 50. Cuboid 2 is placed next with the 23x12 side facing down with height 45. The total height is 95 + 50 + 45 = 190.

Example 2:

Input:

cuboids = [[38,25,45],[76,35,3]]

Output:

76

Explanation:

You can't place any of the cuboids on the other. We choose cuboid 1 and rotate it so that the 35x3 side is facing down and its height is 76.

Example 3:

Input:

cuboids = [[7,11,17],[7,17,11],[11,7,17],[11,17,7],[17,7,11],[17,11,7]]

Output:

102

Explanation:

After rearranging the cuboids, you can see that all cuboids have the same dimension. You can place the 11x7 side down on all cuboids so their heights are 17. The maximum height of stacked cuboids is 6 * 17 = 102.

Constraints:

n == cuboids.length

1 <= n <= 100

1 <= width

i

, length

i

, height

i

<= 100

Code Snippets

C++:

```
class Solution {
public:
int maxHeight(vector<vector<int>>& cuboids) {

}
};
```

Java:

```
class Solution {
public int maxHeight(int[][] cuboids) {

}
}
```

Python3:

```
class Solution:
def maxHeight(self, cuboids: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

```
var maxHeight = function(cuboids) {

};
```

TypeScript:

```
function maxHeight(cuboids: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxHeight(int[][] cuboids) {

}
}
```

C:

```
int maxHeight(int** cuboids, int cuboidsSize, int* cuboidsColSize){

}
```

Go:

```
func maxHeight(cuboids [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxHeight(cuboids: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maxHeight(_ cuboids: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_height(cuboids: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} cuboids
# @return {Integer}
def max_height(cuboids)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $cuboids
* @return Integer
*/
function maxHeight($cuboids) {

}
}
```

Scala:

```
object Solution {
def maxHeight(cuboids: Array[Array[Int]]): Int = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Height by Stacking Cuboids
* Difficulty: Hard
* Tags: array, dp, sort, stack
*
* 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 maxHeight(vector<vector<int>>& cuboids) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Height by Stacking Cuboids
* Difficulty: Hard
* Tags: array, dp, sort, stack
*
* 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 maxHeight(int[][] cuboids) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Height by Stacking Cuboids
Difficulty: Hard
Tags: array, dp, sort, stack

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 maxHeight(self, cuboids: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Height by Stacking Cuboids
* Difficulty: Hard
* Tags: array, dp, sort, stack
*
* 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[][]} cuboids
* @return {number}
*/
var maxHeight = function(cuboids) {
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Height by Stacking Cuboids
* Difficulty: Hard
* Tags: array, dp, sort, stack
*
* 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 maxHeight(cuboids: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Height by Stacking Cuboids
* Difficulty: Hard
* Tags: array, dp, sort, stack
*
* 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 MaxHeight(int[][] cuboids) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Height by Stacking Cuboids
* Difficulty: Hard
* Tags: array, dp, sort, stack
*
* 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 maxHeight(int** cuboids, int cuboidsSize, int* cuboidsColSize){

}
```

Go Solution:

```
// Problem: Maximum Height by Stacking Cuboids
// Difficulty: Hard
// Tags: array, dp, sort, stack
//
// 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 maxHeight(cuboids [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxHeight(cuboids: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxHeight(_ cuboids: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Height by Stacking Cuboids
// Difficulty: Hard
// Tags: array, dp, sort, stack
//
// 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 max_height(cuboids: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} cuboids
# @return {Integer}
def max_height(cuboids)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $cuboids
* @return Integer
*/
function maxHeight($cuboids) {

}
}
```

Scala Solution:

```
object Solution {
def maxHeight(cuboids: Array[Array[Int]]): Int = {

}
}
```