Problem 11: Container With Most Water

Problem Information

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

Problem Description

You are given an integer array

height

of length

n

. There are

n

vertical lines drawn such that the two endpoints of the

i

th

line are

(i, 0)

and

(i, height[i])

.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return

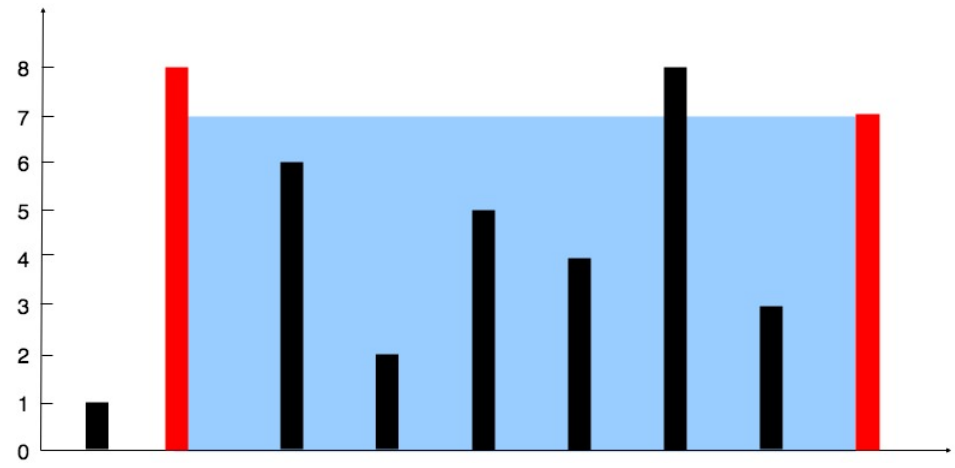
the maximum amount of water a container can store

.

Notice

that you may not slant the container.

Example 1:



Input:

height = [1,8,6,2,5,4,8,3,7]

Output:

49

Explanation:

The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input:

height = [1,1]

Output:

1

Constraints:

n == height.length

2 <= n <= 10

5

0 <= height[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int maxArea(vector<int>& height) {

}
};
```

Java:

```
class Solution {
public int maxArea(int[] height) {

}
}
```

Python3:

```
class Solution:
def maxArea(self, height: List[int]) -> int:
```

Python:

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

JavaScript:

```
/**
* @param {number[]} height
* @return {number}
*/
var maxArea = function(height) {

};
```

TypeScript:

```
function maxArea(height: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxArea(int[] height) {

}
}
```

C:

```
int maxArea(int* height, int heightSize) {

}
```

Go:

```
func maxArea(height []int) int {

}
```

Kotlin:

```
class Solution {
fun maxArea(height: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxArea(_ height: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_area(height: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} height
# @return {Integer}
def max_area(height)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $height
* @return Integer
*/
function maxArea($height) {

}
}
```

Dart:

```
class Solution {
int maxArea(List<int> height) {

}
}
```

Scala:

```
object Solution {
def maxArea(height: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_area(height :: [integer]) :: integer
def max_area(height) do

end
end
```

Erlang:

```
-spec max_area(Height :: [integer()]) -> integer().
max_area(Height) ->
.
```

Racket:

```
(define/contract (max-area height)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Container With Most Water
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 maxArea(vector<int>& height) {

}
};
```

Java Solution:

```
/**
* Problem: Container With Most Water
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 maxArea(int[] height) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Container With Most Water
Difficulty: Medium
Tags: array, dp, greedy

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Container With Most Water
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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[]} height
* @return {number}
*/
var maxArea = function(height) {

};
```

TypeScript Solution:

```
/**
* Problem: Container With Most Water
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 maxArea(height: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Container With Most Water
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 MaxArea(int[] height) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Container With Most Water
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 maxArea(int* height, int heightSize) {

}
```

Go Solution:

```
// Problem: Container With Most Water
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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 maxArea(height []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxArea(height: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxArea(_ height: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Container With Most Water
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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_area(height: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} height
# @return {Integer}
def max_area(height)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $height
* @return Integer
*/
function maxArea($height) {

}
}
```

Dart Solution:

```
class Solution {
int maxArea(List<int> height) {

}
}
```

Scala Solution:

```
object Solution {
def maxArea(height: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_area(height :: [integer]) :: integer
def max_area(height) do

end
end
```

Erlang Solution:

```
-spec max_area(Height :: [integer()]) -> integer().
max_area(Height) ->
.
```

Racket Solution:

```
(define/contract (max-area height)
(-> (listof exact-integer?) exact-integer?)
)
```