



## 56. Merge Intervals

Solved

Medium

[Topics](#)[Companies](#)

Given an array of `intervals` where `intervals[i] = [starti, endi]`, merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

### Example 1:

Input: `intervals = [[1,3],[2,6],[8,10],[15,18]]`

Output: `[[1,6],[8,10],[15,18]]`

Explanation: Since intervals `[1,3]` and `[2,6]` overlap, merge them into `[1,6]`.

### Example 2:

Input: `intervals = [[1,4],[4,5]]`

Output: `[[1,5]]`

Explanation: Intervals `[1,4]` and `[4,5]` are considered overlapping.

### Constraints:

- `1 <= intervals.length <= 104`
- `intervals[i].length == 2`
- `0 <= starti <= endi <= 104`

### </> Code

Python3 Auto



```
1 class Solution:
2     def merge(self, intervals: List[List[int]]) -> List[List[int]]:
3
```

Description Editorial Solutions Submissions

## 56. Merge Intervals

Solved 

Medium Topics Companies

Given an array of `intervals` where `intervals[i] = [starti, endi]`, merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

### Example 1:

Input: `intervals = [[1,3],[2,6],[8,10],[15,18]]`

Output: `[[1,6],[8,10],[15,18]]`

Explanation: Since intervals `[1,3]` and `[2,6]` overlap, merge them into `[1,6]`.

### Example 2:

Input: `intervals = [[1,4],[4,5]]`

Output: `[[1,5]]`

Explanation: Intervals `[1,4]` and `[4,5]` are considered overlapping.

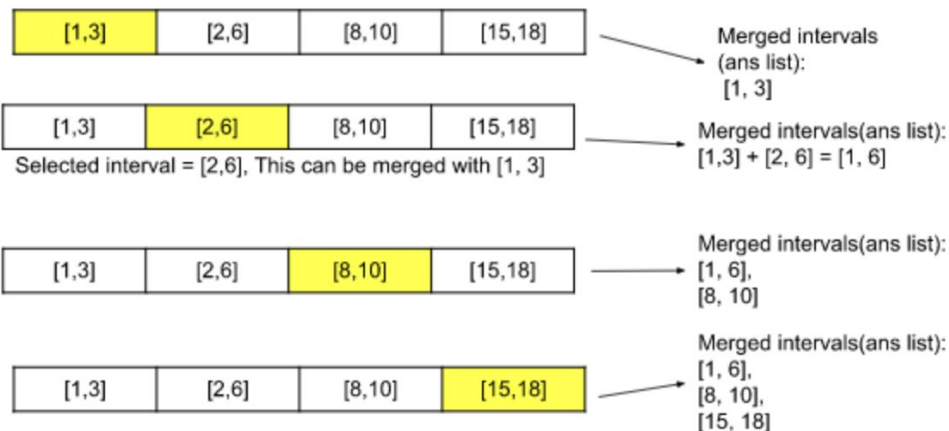
### Constraints:

- `1 <= intervals.length <= 104`
- `intervals[i].length == 2`
- `0 <= starti <= endi <= 104`

</> Code

Python3  Auto

```
1 class Solution:
2     def merge(self, intervals: List[List[int]]) -> List[List[int]]:
3
```



## 56. Merge Intervals

Solved

[Medium](#)
[Topics](#)
[Companies](#)

Given an array of intervals where  $intervals[i] = [start_i, end_i]$ , merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

### Example 1:

Input:  $intervals = [[1,3], [2,6], [8,10], [15,18]]$

Output:  $[[1,6], [8,10], [15,18]]$

Explanation: Since intervals  $[1,3]$  and  $[2,6]$  overlap, merge them into  $[1,6]$ .

### Example 2:

Input:  $intervals = [[1,4], [4,5]]$

Output:  $[[1,5]]$

Explanation: Intervals  $[1,4]$  and  $[4,5]$  are considered overlapping.

### Constraints:

- $1 \leq intervals.length \leq 10^4$
- $intervals[i].length == 2$
- $0 \leq start_i \leq end_i \leq 10^4$

Seen this question in a real interview before? 1/5

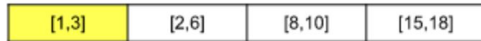
☐ Yes
 ☐ No

### </> Code

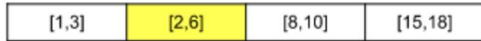
Python3 Auto

```

1 class Solution:
2     def merge(self, intervals: List[List[int]]) -> List[List[int]]:
3         merged = []
4         # Sort the intervals based on their start times. The lambda function lambda x: x[0] is used to extract the
5         # start time of each interval for comparison during sorting.
6         intervals.sort(key=lambda x: x[0])
7         # Initialize prev with the first interval from the sorted list.
8         prev = intervals[0]
9         # Iterate through the sorted intervals starting from the second interval (index 1) to the last interval.
10        for interval in intervals[1:]:
11            #If the start time of the current interval is less than or equal to the end time of the prev interval
12            if interval[0] <= prev[1]:
13                #Update the end time of the prev interval to maximum end time between prev and the current interval.
14                prev[1] = max(prev[1], interval[1])
15            else:
16                #Intervals don't overlap, add the prev interval to the merged list
17                merged.append(prev)
18                #Update prev to be the current interval, this interval will be used for further comparisons.
19                prev = interval
20        #Adding the Last Interval
21        merged.append(prev)
22        return merged
    
```

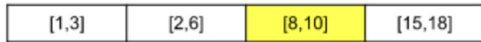


Merged intervals  
(ans list):  
[1, 3]

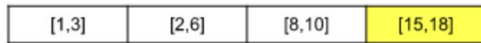


Selected interval = [2,6], This can be merged with [1, 3]

Merged intervals(ans list):  
[1,3] + [2, 6] = [1, 6]



Merged intervals(ans list):  
[1, 6],  
[8, 10]



Merged intervals(ans list):  
[1, 6],  
[8, 10],  
[15, 18]

[6,8]	[1,9]	[2,4]	[4,7]
-------	-------	-------	-------

Sorting the array

[1,9]	[2,4]	[4,7]	[6,8]
-------	-------	-------	-------

Compares 2 and 9, since  $2 < 9$  it merges and takes  $\max(4,9)$

[1,9]	[4,7]	[6,8]
-------	-------	-------

Compares 9 and 4, since  $4 < 9$  it merges and takes  $\max(7,9)$

[1,9]	[6,8]
-------	-------

Compares 9 and 6, since  $6 < 9$  it merges and takes  $\max(8,9)$

[1,9]	→	Output
-------	---	--------



## 121. Best Time to Buy and Sell Stock

Solved

Easy Topics Companies

You are given an array `prices` where `prices[i]` is the price of a given stock on the  $i^{\text{th}}$  day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

### Example 1:

Input: `prices = [7,1,5,3,6,4]`

Output: `5`

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

### Example 2:

Input: `prices = [7,6,4,3,1]`

Output: `0`

Explanation: In this case, no transactions are done and the max profit = 0.

### Constraints:

- $1 \leq \text{prices.length} \leq 10^5$

- $0 \leq \text{prices}[i] \leq 10^4$

### </> Code

Python3 Auto



```
1 class Solution:
2
3
```

Saving...

Ln 2, Col 1



## 121. Best Time to Buy and Sell Stock

Solved ✓

Easy Topics Companies

You are given an array `prices` where `prices[i]` is the price of a given stock on the  $i^{\text{th}}$  day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

### Example 1:

Input: `prices = [7,1,5,3,6,4]`

Output: 5

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

### Example 2:

Input: `prices = [7,6,4,3,1]`

Output: 0

Explanation: In this case, no transactions are done and the max profit = 0.

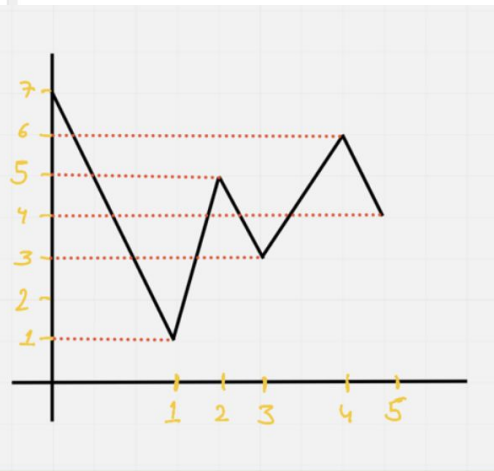
### Constraints:

- $1 \leq \text{prices.length} \leq 10^5$
- $0 \leq \text{prices}[i] \leq 10^4$

### Code

Python3 Auto

```
1 class Solution:
2
3
```



Left = 0  
Right = 1

Buy	Sell	Profit
7	1	-6
1	5	4
1	3	2
1	6	5

Saving...

Ln 2, Col 1

## 121. Best Time to Buy and Sell Stock

Solved

Easy Topics Companies

You are given an array `prices` where `prices[i]` is the price of a given stock on the  $i^{\text{th}}$  day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return the *maximum profit* you can achieve from this transaction. If you cannot achieve any profit, return 0.

### Example 1:

**Input:** `prices = [7,1,5,3,6,4]`

**Output:** 5

**Explanation:** Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.  
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

### Example 2:

**Input:** `prices = [7,6,4,3,1]`

**Output:** 0

**Explanation:** In this case, no transactions are done and the max profit = 0.

### Constraints:

- $1 \leq \text{prices.length} \leq 10^5$
- $0 \leq \text{prices}[i] \leq 10^4$

Seen this question in a real interview before? 1/5

### </> Code

Python3 Auto

```
1 class Solution:
2     def maxProfit(self, prices):
3         left = 0 #Buy
4         right = 1 #Sell
5         max_profit = 0
6         while right < len(prices):
7             currentProfit = prices[right] - prices[left] #our current Profit
8             #price[left] < price[right] which means we will get profit
9             if prices[left] < prices[right]:
10                 max_profit = max(currentProfit, max_profit)
11             else:
12                 #Price[left] > price[right] so we will move left pointer to the right
13                 left = right
14             right += 1
15         return max_profit
```

Saved

Ln 15, Col 26



## 11. Container With Most Water

Solved 🟢

Medium Topics Companies Hint

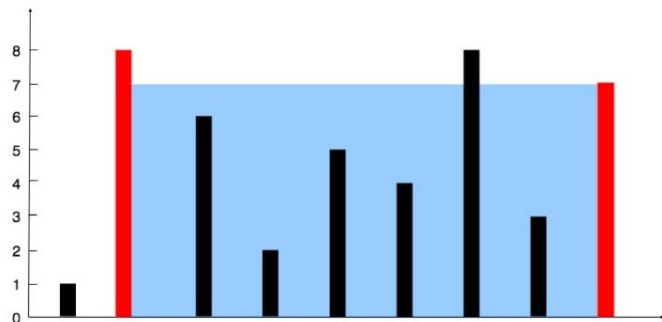
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the  $i^{\text{th}}$  line are  $(i, 0)$  and  $(i, \text{height}[i])$ .

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

### Code

Python3 Auto

```
1 class Solution:
2
```

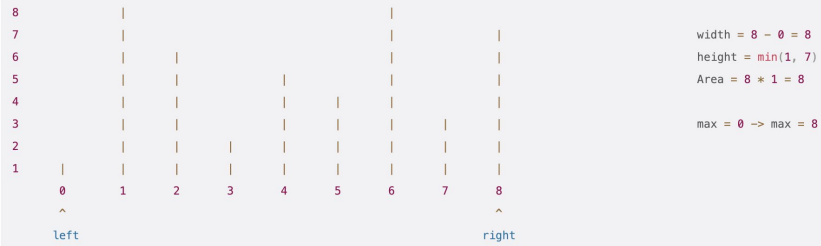
Saved

Ln 2, Col 1

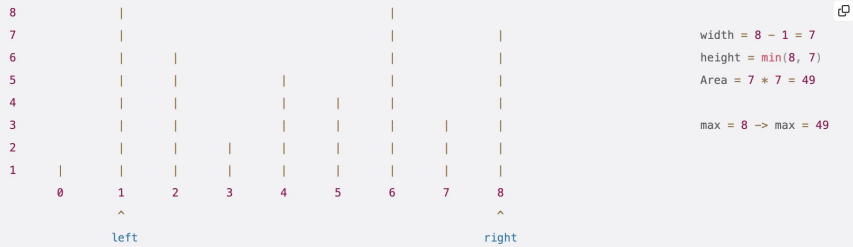
Testcase Test Result



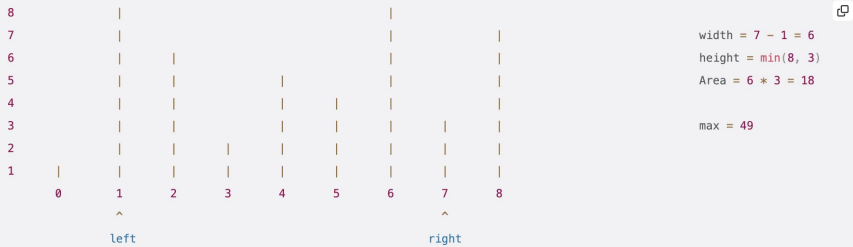
1.



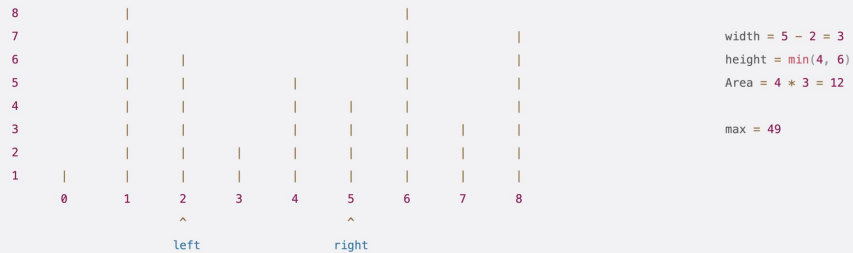
2.



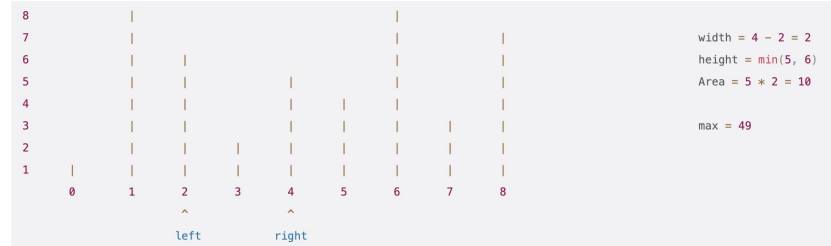
3.



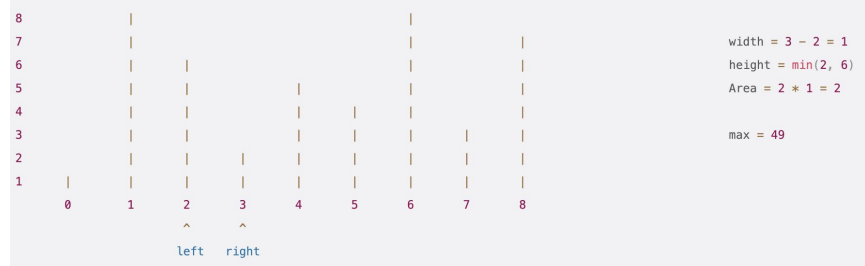
4.



5.



6.





## 11. Container With Most Water

Solved ✓

Medium Topics Companies Hint

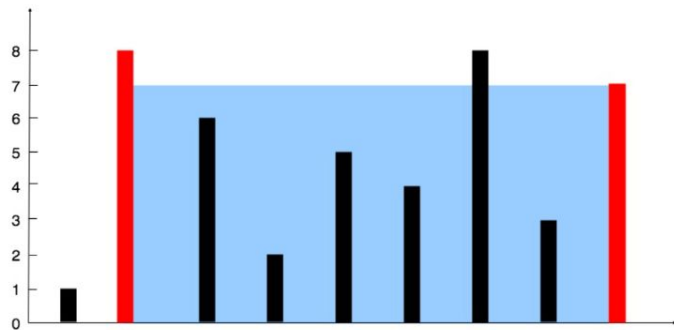
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the  $i^{\text{th}}$  line are  $(i, 0)$  and  $(i, \text{height}[i])$ .

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.

### </> Code

Python3 Auto

```
1 class Solution:
2     def maxArea(self, height: List[int]) -> int:
3         max_area = 0
4         left = 0
5         right = len(height) - 1
6
7         while left < right: #Continue looping until left pointer is not equal to or exceeds the right pointer
8             max_area = max(max_area, (right - left) * min(height[left], height[right]))
9             # Multiple width of the container and minimum of the heights of the two lines.
10            if height[left] < height[right]:
11                left += 1
12            else:
13                right -= 1
14        return max_area
```

Saved

Ln 14, Col



## 152. Maximum Product Subarray

Medium

Topics

Companies

Given an integer array `nums`, find a **subarray** that has the largest product, and return *the product*.

The test cases are generated so that the answer will fit in a **32-bit** integer.

### Example 1:

Input: `nums = [2,3,-2,4]`

Output: 6

Explanation: [2,3] has the largest product 6.

### Example 2:

Input: `nums = [-2,0,-1]`

Output: 0

Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

### Constraints:

- $1 \leq \text{nums.length} \leq 2 * 10^4$
- $-10 \leq \text{nums}[i] \leq 10$
- The product of any subarray of `nums` is **guaranteed** to fit in a **32-bit** integer.

```
1 class Solution:
2     def maxProduct(self, nums: List[int]) -> int:
3
```

2	3	-2	-5	6	-1	4
---	---	----	----	---	----	---

i = 0, left = 0, right = arr.length - 1

		ANS
$L = 1 * 2 = 2$	$R = 1 * 4 = 4$	4
$L = 2 * 3 = 6$	$R = 4 * -1 = -4$	6
$L = 6 * -2 = -12$	$R = -4 * 6 = -24$	6
$L = -12 * -5 = 60$	$R = -24 * -5 = 120$	120
$L = 60 * 6 = 360$	$R = 120 * -2 = -240$	360
$L = 360 * -1 = -360$	$R = -240 * 3 = -720$	360
$L = -360 * 4 = -1440$	$R = -720 * 2 = -1440$	360



## 152. Maximum Product Subarray

Medium

Topics

Companies

Given an integer array `nums`, find a **subarray** that has the largest product, and return *the product*.

The test cases are generated so that the answer will fit in a **32-bit** integer.

### Example 1:

Input: `nums = [2,3,-2,4]`

Output: 6

Explanation: [2,3] has the largest product 6.

### Example 2:

Input: `nums = [-2,0,-1]`

Output: 0

Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

### Constraints:

- $1 \leq \text{nums.length} \leq 2 * 10^4$
- $-10 \leq \text{nums}[i] \leq 10$
- The product of any subarray of `nums` is **guaranteed** to fit in a **32-bit** integer.

```
1 class Solution:
2     def maxProduct(self, nums: List[int]) -> int:
3
```