

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`

Example 3:

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

Output - `[1,9]`

</> Code

Python3 🔒 Auto

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

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

Sorting the array

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

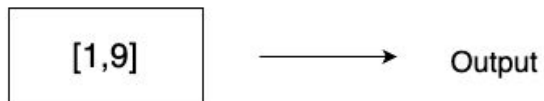
Compares 2 and 9, since $2 < 9$ it merges and takes $\max(4,9)$

[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 $\max(8,9)$



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.

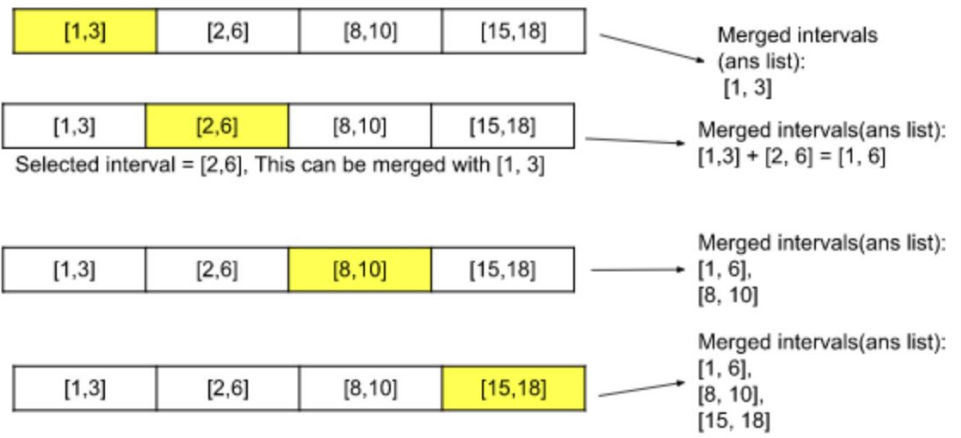
Constraints:

- `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

</> Code

Python3 Auto

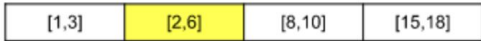
```

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

```

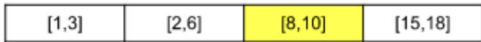


Merged intervals
(ans list):
[1, 3]



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]



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^{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^{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.

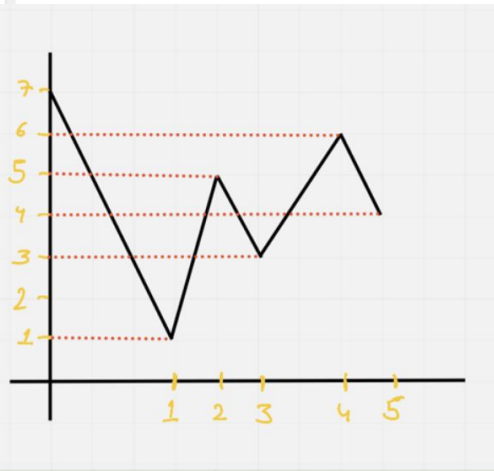
Constraints:

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

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^{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

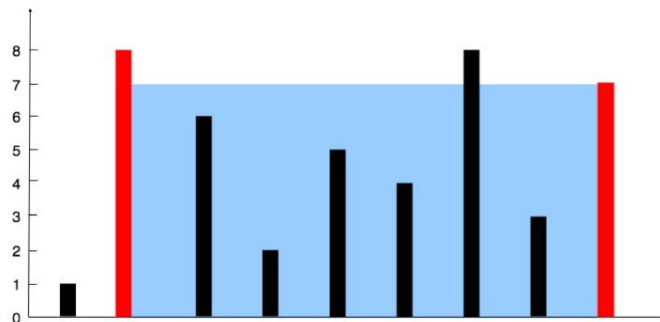
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the i^{th} line are $(i, 0)$ and $(i, \text{height}[i])$.

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. `0,1,2,3,4,5,6,7,8` (= index)
`[1,8,6,2,5,4,8,3,7]`
L R

Left pointer starts from index 0
Right pointer starts from the last index

2. `0,1,2,3,4,5,6,7,8` (= index)
`[1,8,6,2,5,4,8,3,7]`
L R

`max_area = 8`

3. `0,1,2,3,4,5,6,7,8` (= index)
`[1,8,6,2,5,4,8,3,7]`
L R

`max_area = 8`
`current_area = 7 * 7 = 49`
`max_area = max(8, 49) = 49`
Move R to next

4. `0,1,2,3,4,5,6,7,8` (= index)
`[1,8,6,2,5,4,8,3,7]`
L R

`max_area = 49`
`current_area = 6 * 3 = 18`
`max_area = max(49, 18) = 49`
Move R to next

5. `0,1,2,3,4,5,6,7,8` (= index)
`[1,8,6,2,5,4,8,3,7]`
L R

`max_area = 49`
`current_area = 5 * 8 = 40`
`max_area = max(49, 40) = 49`
Move L to next

6. `0,1,2,3,4,5,6,7,8` (= index)
`[1,8,6,2,5,4,8,3,7]`
L R

`max_area = 49`
`current_area = 4 * 6 = 14`
`max_area = max(49, 14) = 49`
Move L to next

7. `0,1,2,3,4,5,6,7,8` (= index)
`[1,8,6,2,5,4,8,3,7]`
L R

`max_area = 49`
`current_area = 3 * 2 = 6`
`max_area = max(49, 6) = 49`
Move L to next

8. `0,1,2,3,4,5,6,7,8` (= index)
`[1,8,6,2,5,4,8,3,7]`
L R

`max_area = 49`
`current_area = 2 * 5 = 10`
`max_area = max(49, 10) = 49`
Move L to next

9. `0,1,2,3,4,5,6,7,8` (= index)
`[1,8,6,2,5,4,8,3,7]`
L R

`max_area = 49`
`current_area = 1 * 4 = 4`
`max_area = max(49, 4) = 49`
Move L to next

Now L and R are the same index. We stop iteration.

`0,1,2,3,4,5,6,7,8` (= index)
`[1,8,6,2,5,4,8,3,7]`
L
R

return 49



11. Container With Most Water

Solved ✓

Medium Topics Companies Hint

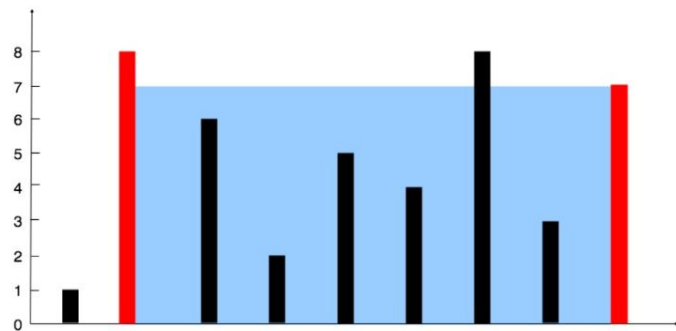
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the i^{th} line are $(i, 0)$ and $(i, \text{height}[i])$.

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