Merge Interval

Leetcode Problem 56

Problem Description

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.

Constraints:

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

Example:

Input:

- intervals = [[1,3],[2,6],[8,10],[15,18]]

Output:

- [[1,6],[8,10],[15,18]]

Algorithm - A Clever Sorting Trick

1. Sort the array on the first element
2. Add the first element to the result
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Result:

Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

(1) Let's sort on the first element

Result:

Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [1,3] [2,6] [2,4] [8,10] [10,12] [13,14] [15,18] [17,18]

Result:

Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [1,3] [2,6] [2,4] [8,10] [10,12] [13,14] [15,18] [17,18]

Note: we can ignore the order of the second element

Result:

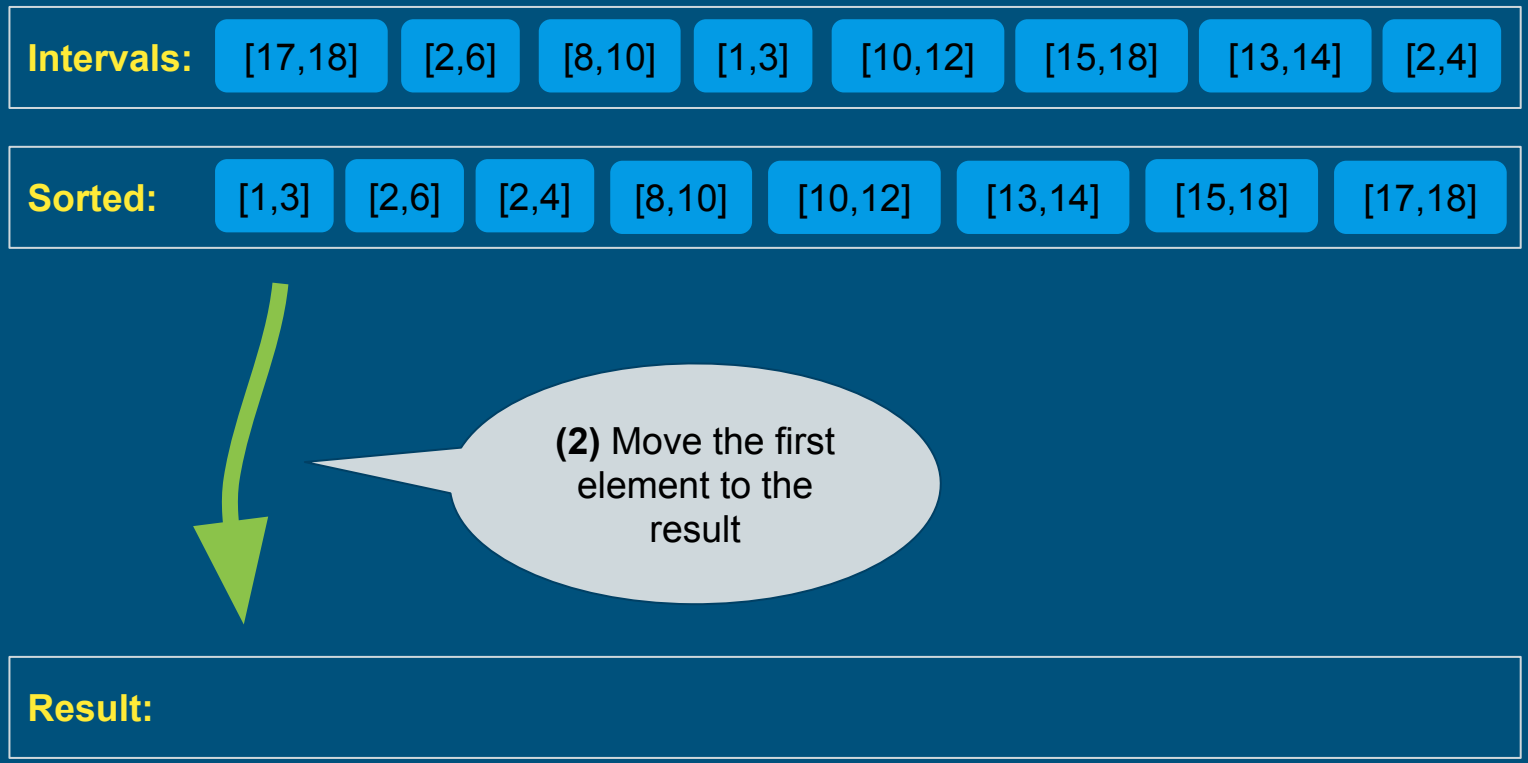
Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

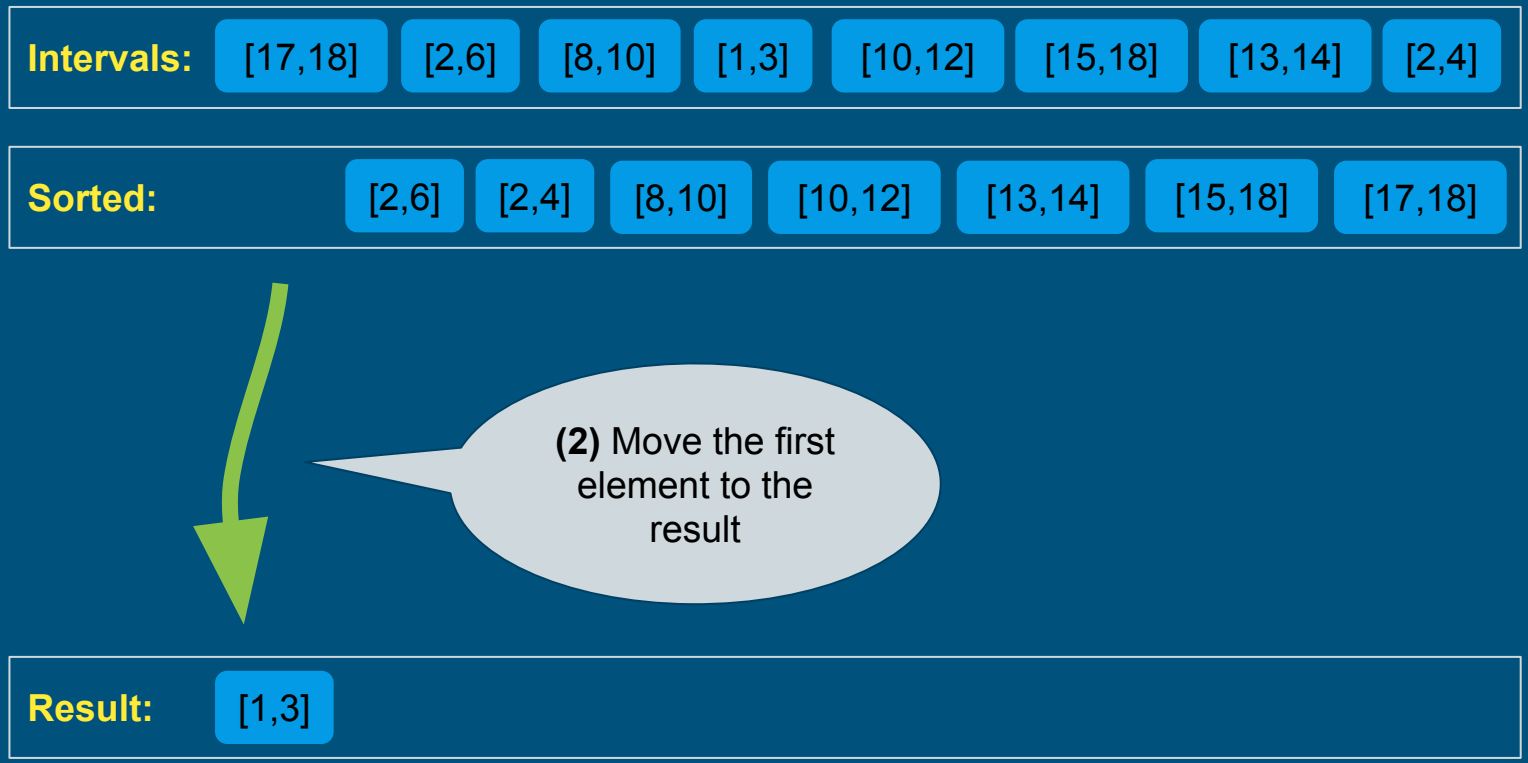
Sorted: [1,3] [2,6] [2,4] [8,10] [10,12] [13,14] [15,18] [17,18]

Result:

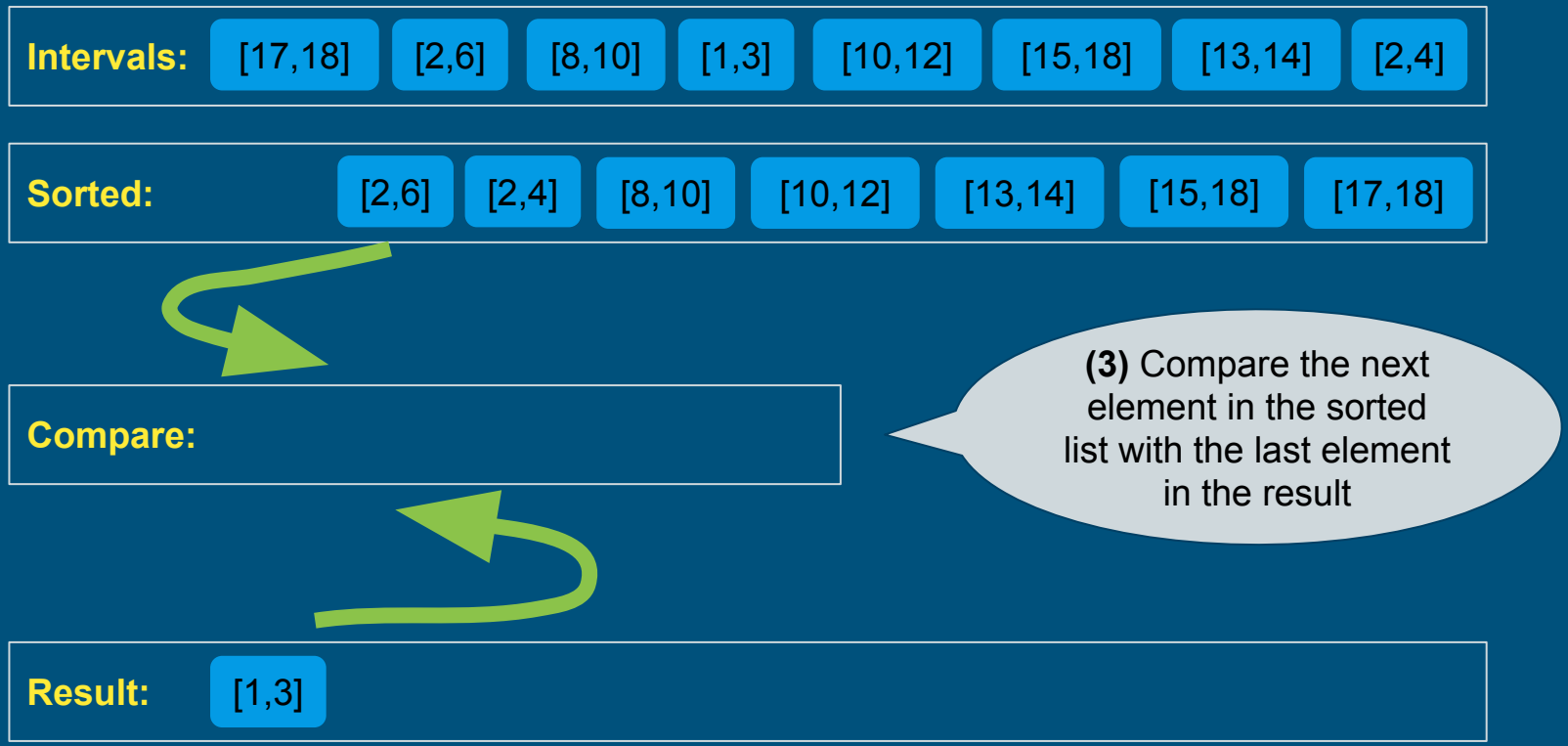
Algorithm - A Clever Sorting Trick



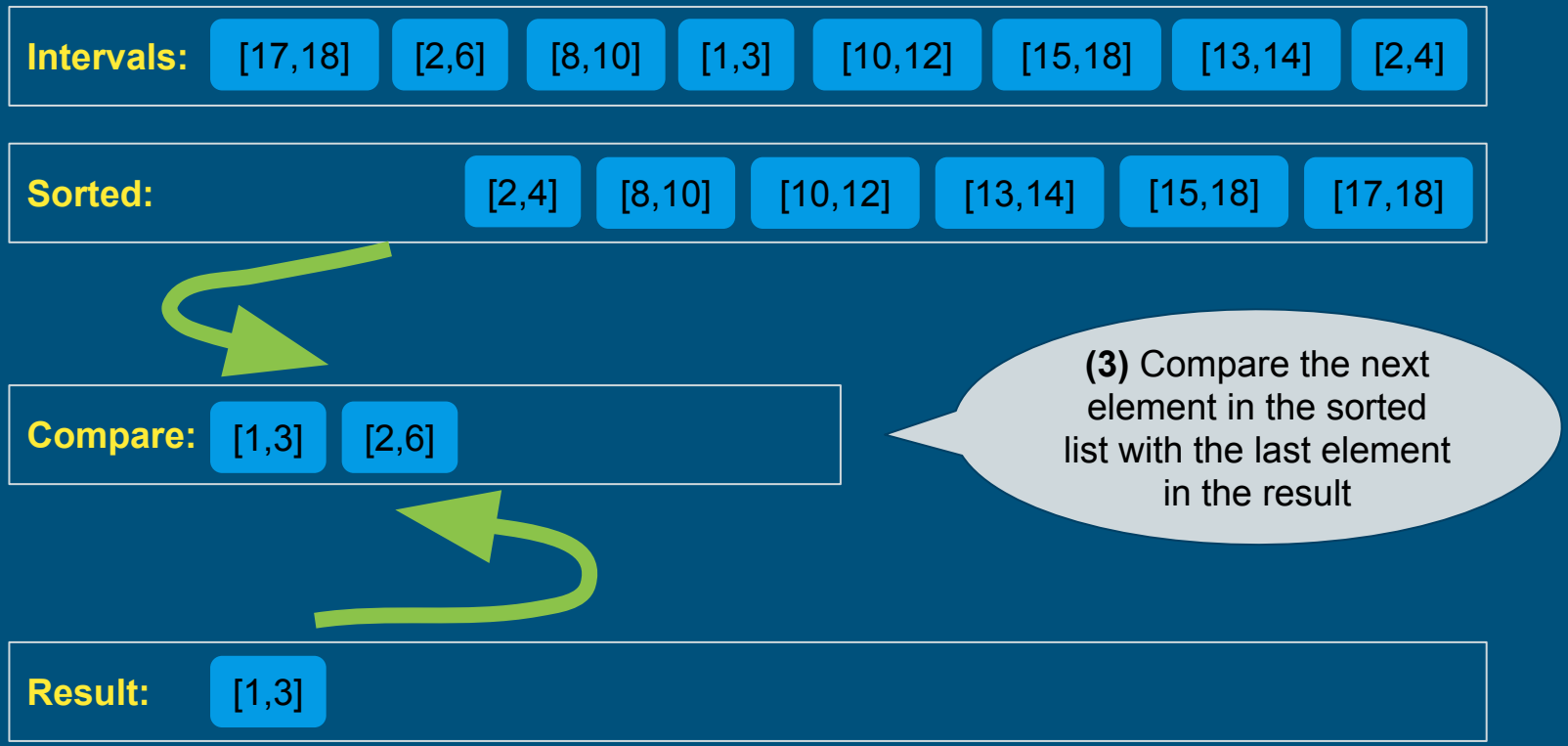
Algorithm - A Clever Sorting Trick



Algorithm - A Clever Sorting Trick



Algorithm - A Clever Sorting Trick

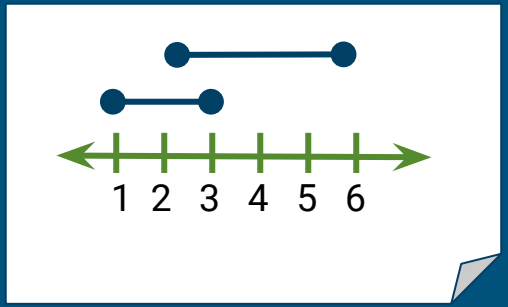


Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [2,4] [8,10] [10,12] [13,14] [15,18] [17,18]

Compare: [1,3] [2,6]



Result: [1,3]

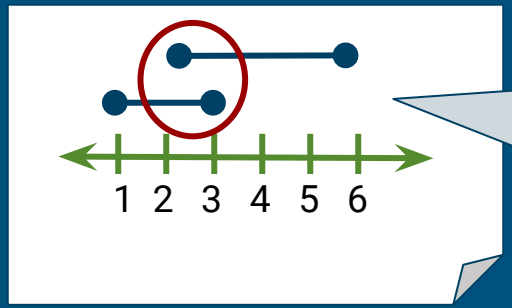
Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [2,4] [8,10] [10,12] [13,14] [15,18] [17,18]

Compare: [1,3] [2,6]

Result: [1,3]



Note: we only need to look at the end of the lower one and the start of the upper one

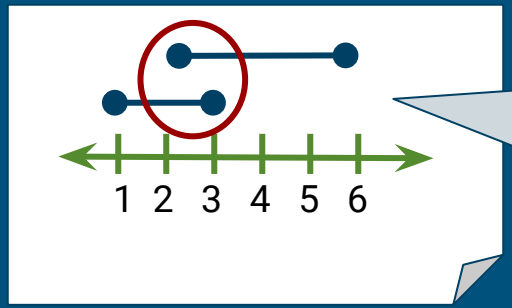
Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [2,4] [8,10] [10,12] [13,14] [15,18] [17,18]

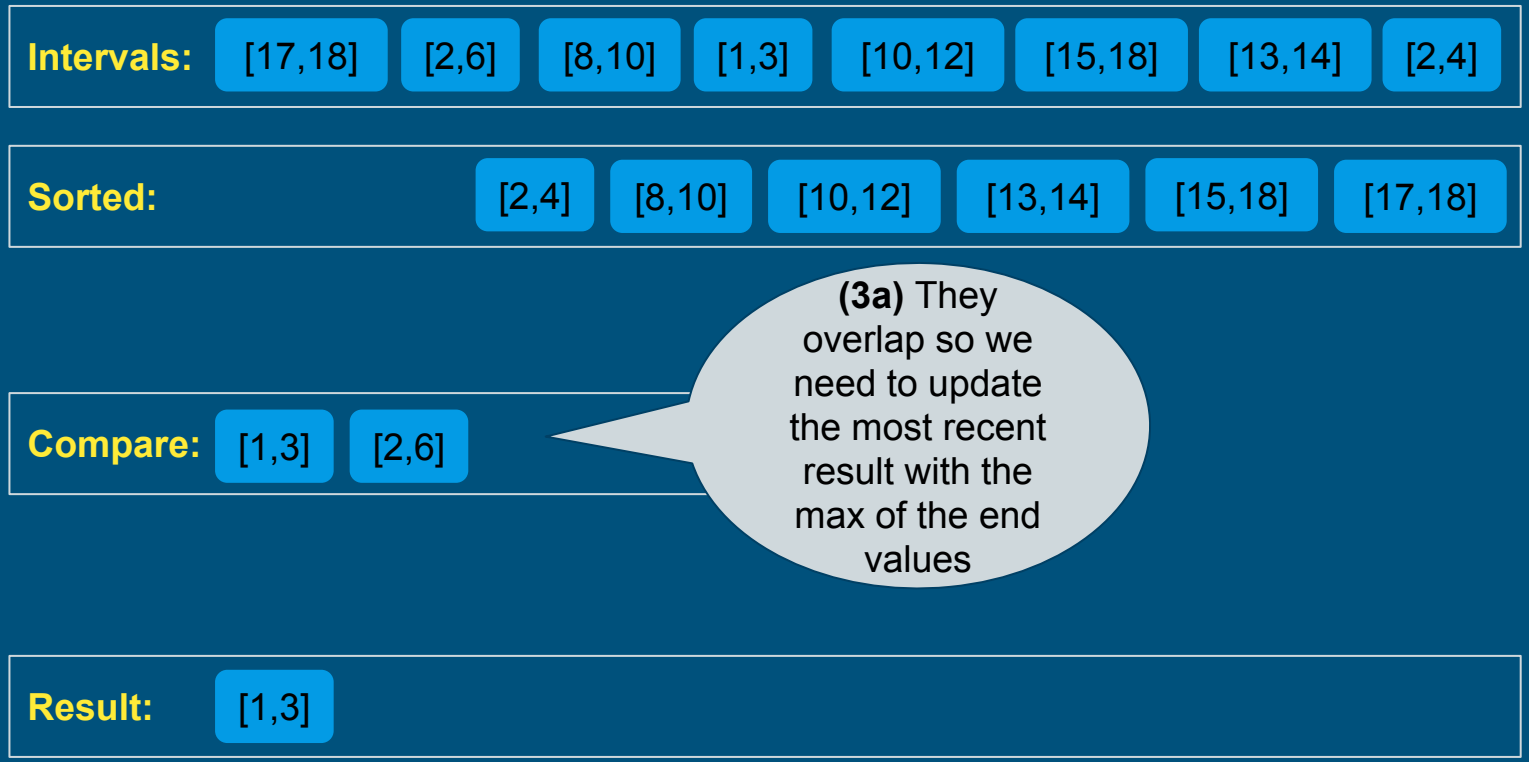
Compare: [1,3] [2,6]

Result: [1,3]



Note: if the end of the [1,**3**] is >= the start of [**2**,6], that means it overlaps

Algorithm - A Clever Sorting Trick



Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [2,4] [8,10] [10,12] [13,14] [15,18] [17,18]

Compare: [1,3] [2,6]

(3a) They overlap so we need to update the most recent result with the max of the end values

Determine the max:

[1,**3**] [2,**6**] = 6

Result: [1,3]

Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [2,4] [8,10] [10,12] [13,14] [15,18] [17,18]

Compare: [1,3] [2,6]

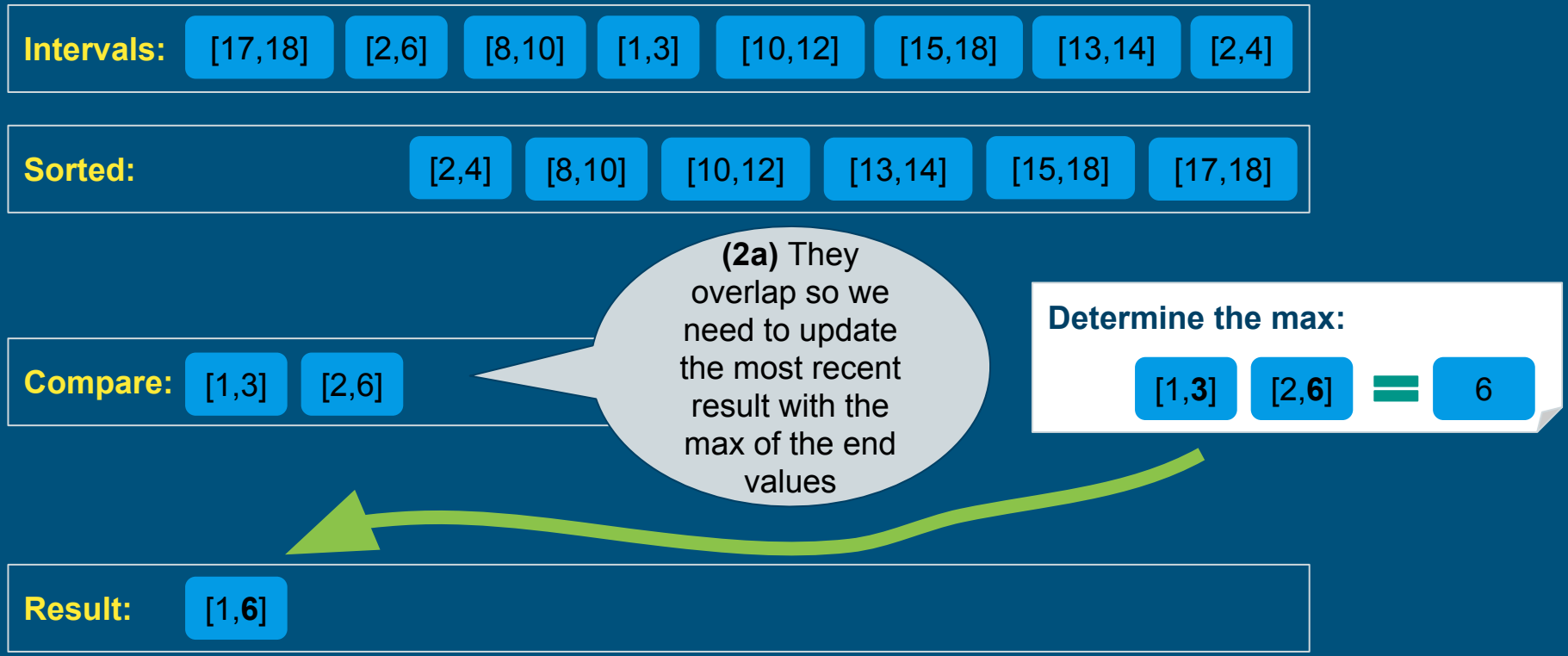
(3a) They overlap so we need to update the most recent result with the max of the end values

Determine the max:

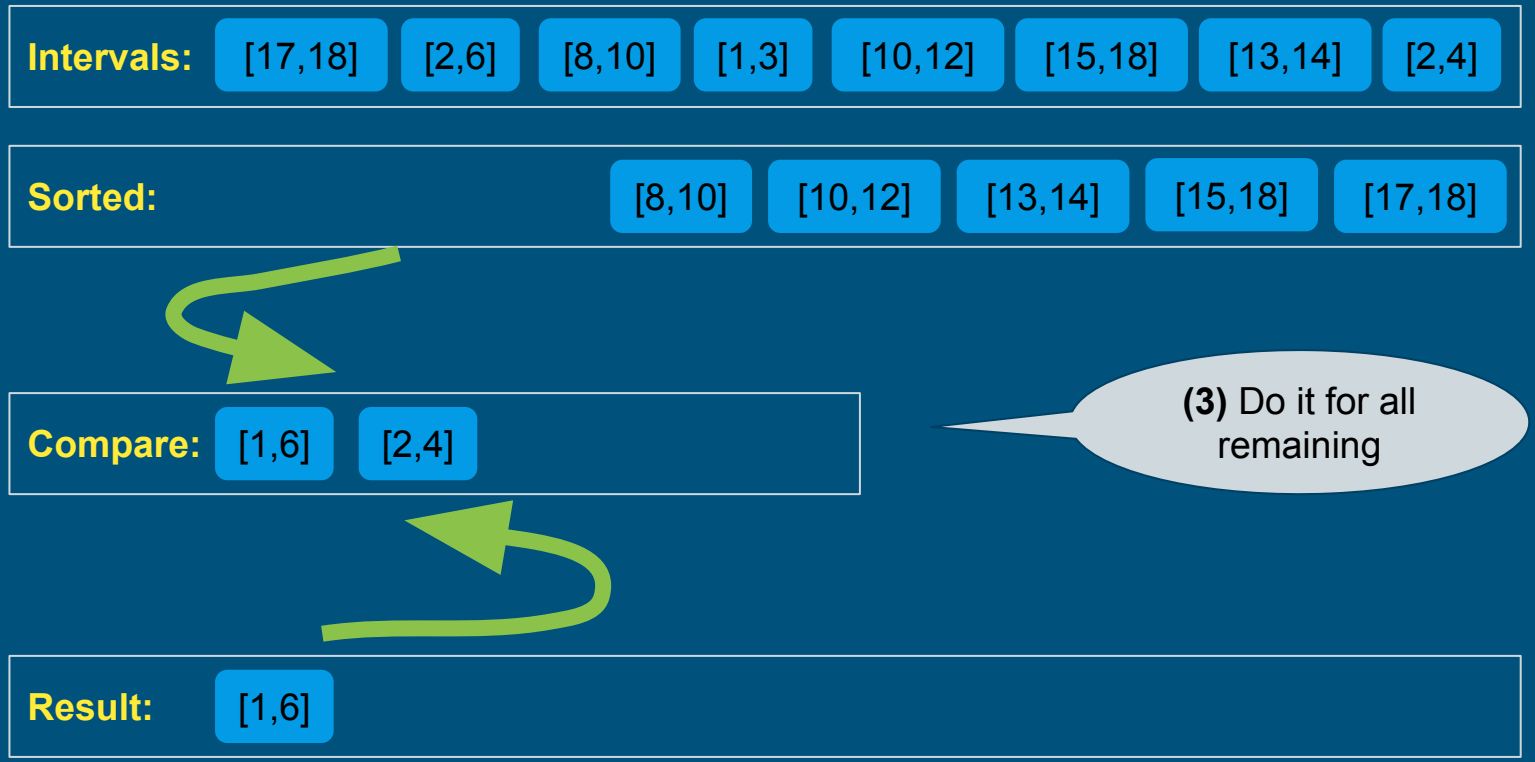
[1,**3**] [2,**6**] = 6

Result: [1,3]

Algorithm - A Clever Sorting Trick



Algorithm - A Clever Sorting Trick



Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [8,10] [10,12] [13,14] [15,18] [17,18]

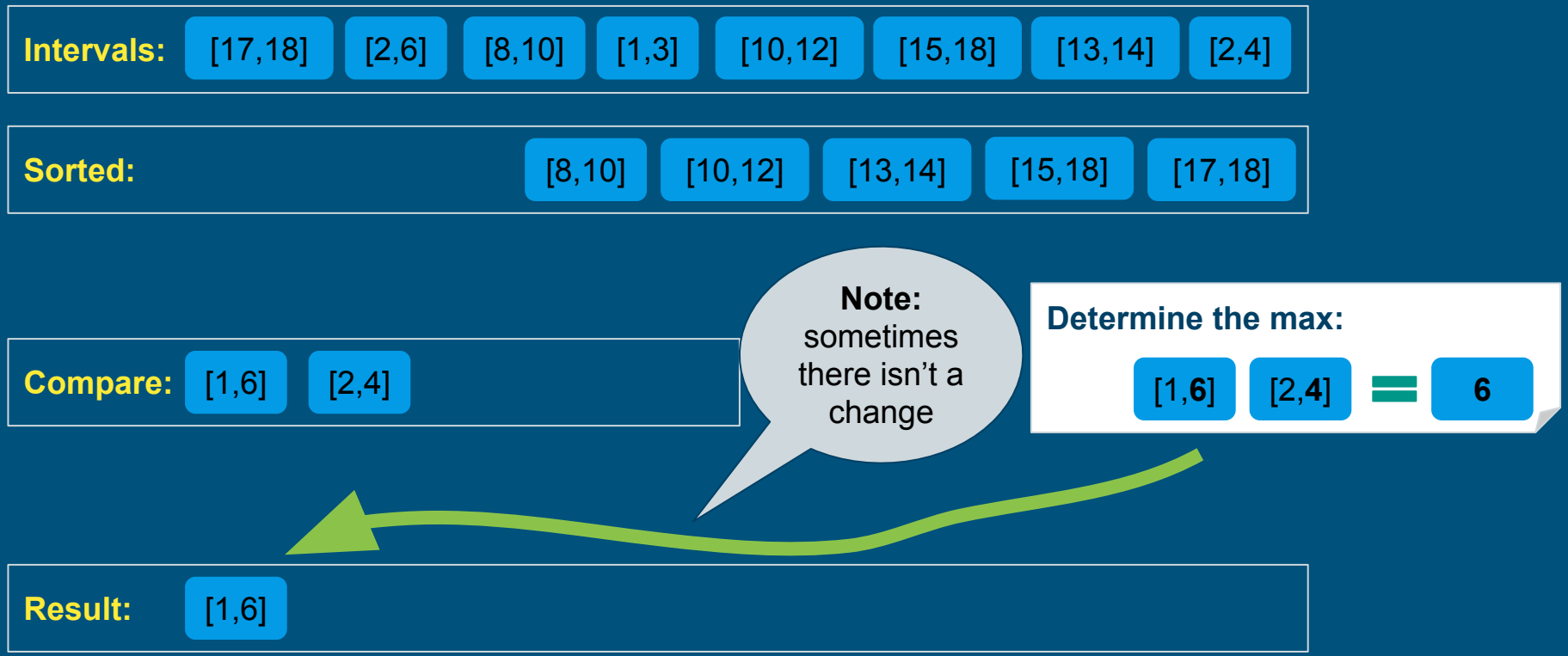
Compare: [1,6] [2,4]

Determine the max:

[1,**6**] [2,**4**] = 6

Result: [1,6]

Algorithm - A Clever Sorting Trick



Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [8,10] [10,12] [13,14] [15,18] [17,18]

Compare:

Result: [1,6]

Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

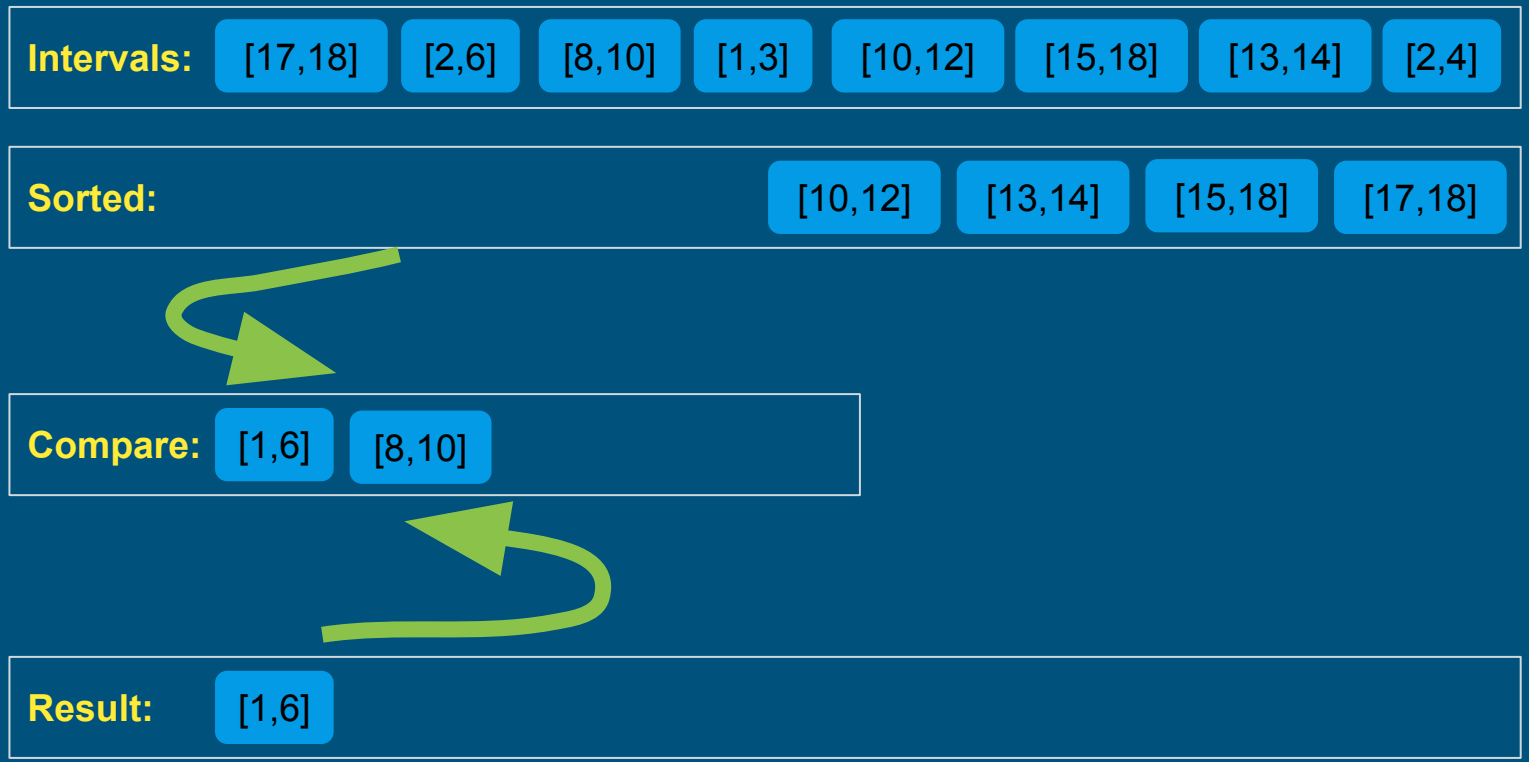
Sorted: [8,10] [10,12] [13,14] [15,18] [17,18]

Compare:

(3) And one more example for good measure

Result: [1,6]

Algorithm - A Clever Sorting Trick

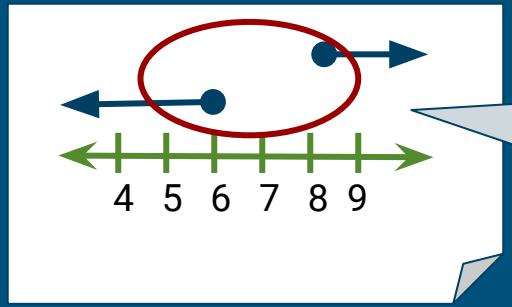


Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [10,12] [13,14] [15,18] [17,18]

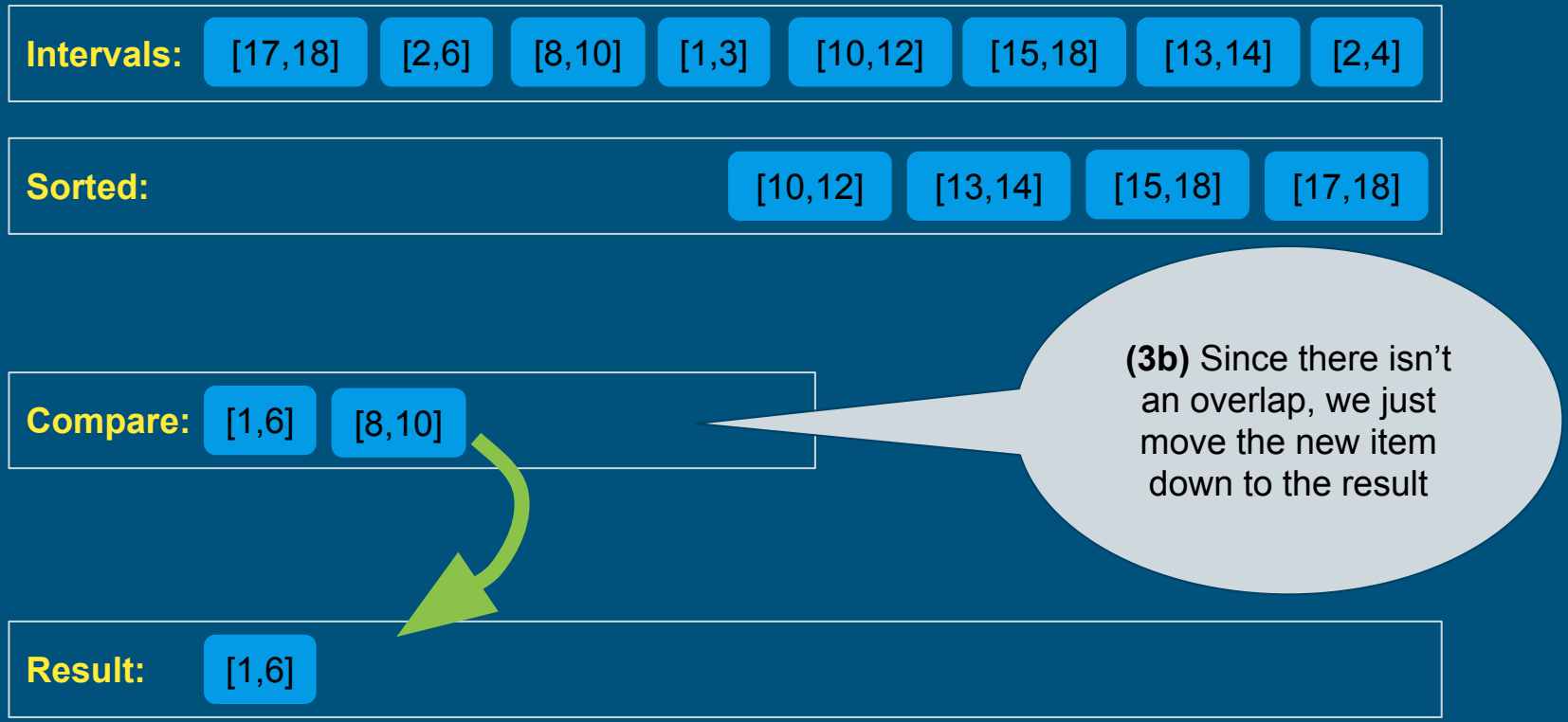
Compare: [1,6] [8,10]



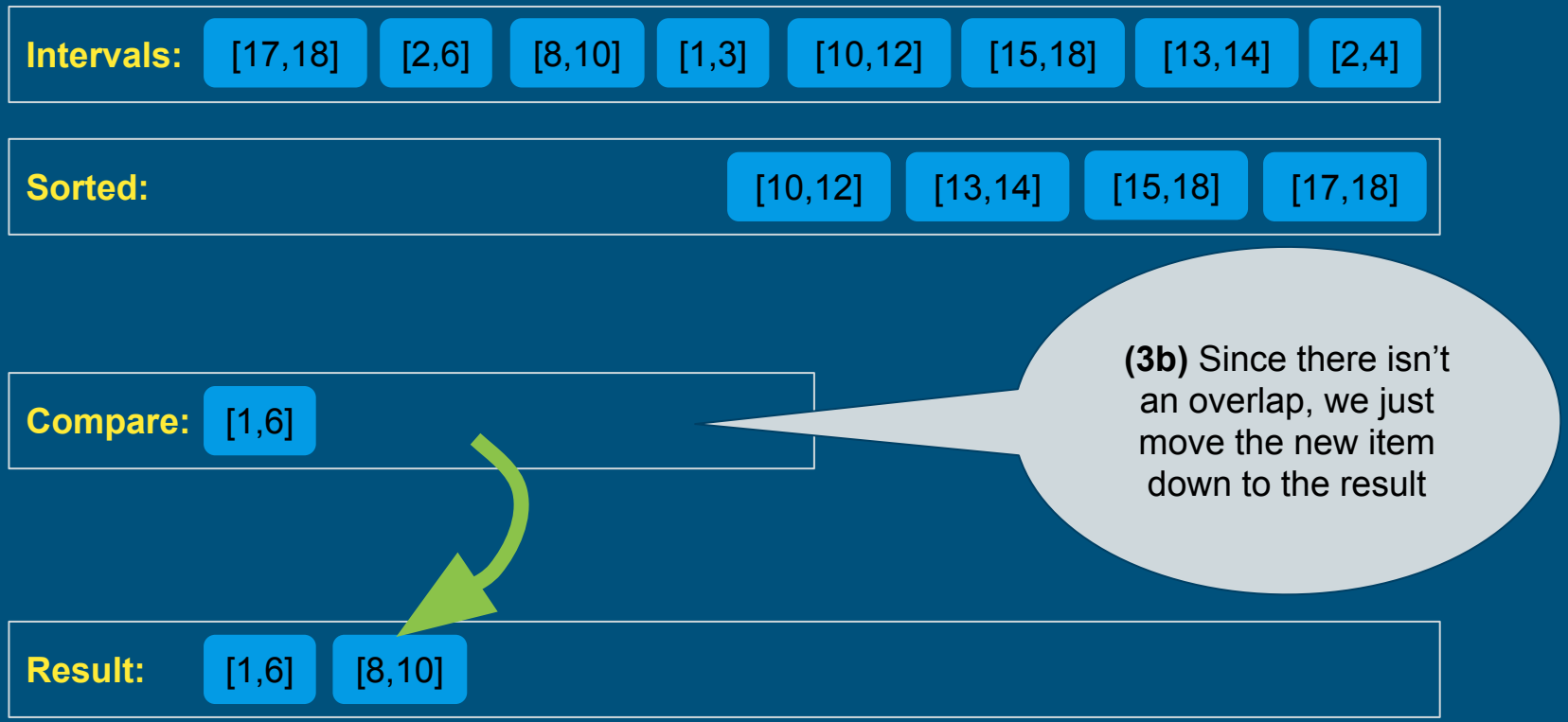
Note: no overlap since 6 is strictly less than 8

Result: [1,6]

Algorithm - A Clever Sorting Trick



Algorithm - A Clever Sorting Trick



Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted: [10,12] [13,14] [15,18] [17,18]

Compare:

Result: [1,6] [8,10]

Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted:

Compare:

Finishing it out!

Result: [1,6] [8,12] [13,14] [15,18]

Algorithm - A Clever Sorting Trick

Intervals: [17,18] [2,6] [8,10] [1,3] [10,12] [15,18] [13,14] [2,4]

Sorted:

Compare:

Result: [1,6] [8,12] [13,14] [15,18]

Algorithm - Time Complexity

1. Sort the array on the first element
2. Add the first element to the result
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Algorithm - Time Complexity

1. Sort the array on the first element
2. Add the first element to the result
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Let’s code it!

Algorithm - Time Complexity

1. Sort the array on the first element
2. Add the first element to the result
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Algorithm - Time Complexity

1. Sort the array on the first element O(nlogn)
2. Add the first element to the result
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Algorithm - Time Complexity

1. Sort the array on the first element
2. Add the first element to the result
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

O(nlogn)

Note: This can depend, but I'm using Java's Arrays.sort(...) in Java 8 which is duval-pivot quicksort

Algorithm - Time Complexity

1. Sort the array on the first element O(nlogn)
2. Add the first element to the result O(1)
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Algorithm - Time Complexity

1. Sort the array on the first element O(nlogn)
2. Add the first element to the result O(1)
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result O(n)
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Algorithm - Time Complexity

1. Sort the array on the first element O(nlogn)
2. Add the first element to the result O(1)
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result O(n)
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

O(nlogn)

Algorithm - Space Complexity

1. Sort the array on the first element O(logn)
2. Add the first element to the result
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Algorithm - Space Complexity

O(logn)

Note: This can depend, but I'm using Java's Arrays.sort(...) in Java 8 which is duval-pivot quicksort

1. Sort the array on the first element
2. Add the first element to the result
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Algorithm - Space Complexity

1. Sort the array on the first element O(logn)
2. Add the first element to the result O(1)
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Algorithm - Space Complexity

1. Sort the array on the first element O(logn)
2. Add the first element to the result O(1)
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result O(1)
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

Algorithm - Space Complexity

1. Sort the array on the first element O(logn)
2. Add the first element to the result O(1)
3. For all remaining elements in the sorted array, compare the start element of the element to the end of the most recent result O(1)
    a. **Case 1:** If they overlap, replace the most recent result element with the max end of the two elements
    b. **Case 2:** Otherwise, add the new element to the result list

O(logn)