1. Problem

- **K-th Element of Two Sorted Arrays**

Input: Two sorted arrays of size m and n respectively

Output: The element that would be at the k'th position of the final sorted array.

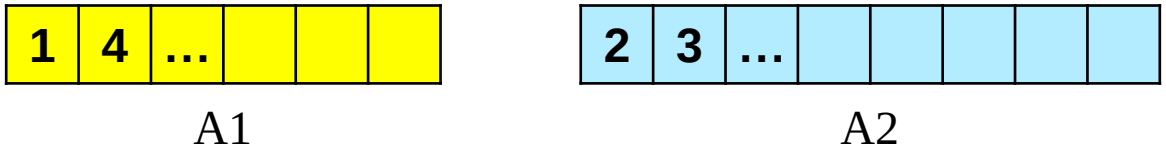
Example: A1 = [2, 3, 6, 7] A2 = [1, 4, 5] k = 5

Final sorted array: [1, 2, 3, 4, 5, 6, 7], 5^{th} element = 5

	Time complexity	Space complexity
Merge 2 sorted arrays:	O(k)	O(1)
Decrease and conquer:	O(min(m, n, k))	O(1)
Decrease and conquer 2 :	O(log(min(m, n, k)))	O(1)

m and n are the size of A1 and A2, respectively
k is the position of the required element

1) Merge 2 sorted arrays

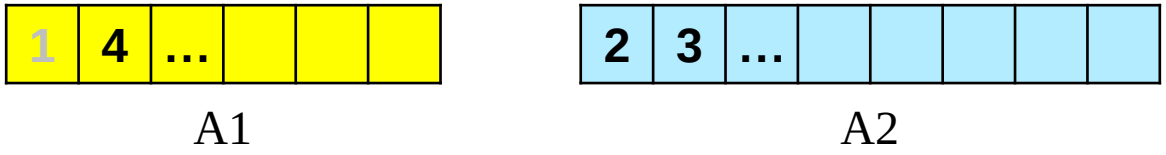


k = 3

Compare: 1st element of A1 — 1st element of A2

1st of final array: 1

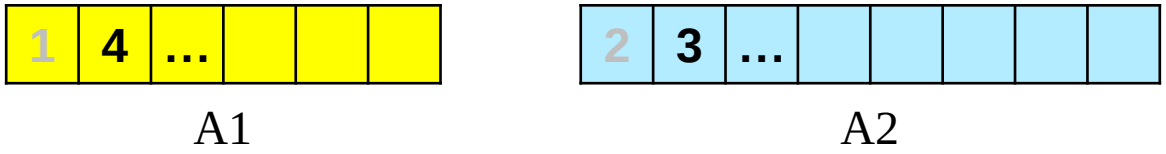
Remaining k = 2



Compare: 2nd element of A1 — 1st element of A2

2nd of final array: 2

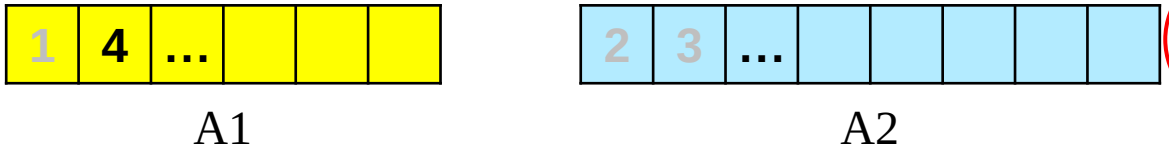
Remaining k = 1



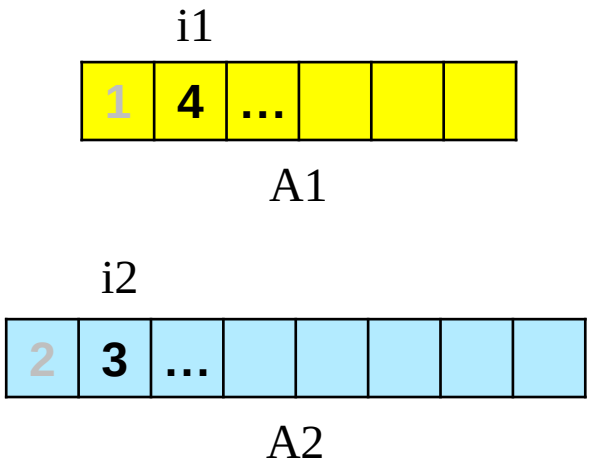
Compare: 2nd element of A1 — 2nd element of A2

3rd of final array: 3

Remaining k = 0



1) Merge 2 sorted arrays



Time complexity: O(*k*)

Space complexity: O(1)

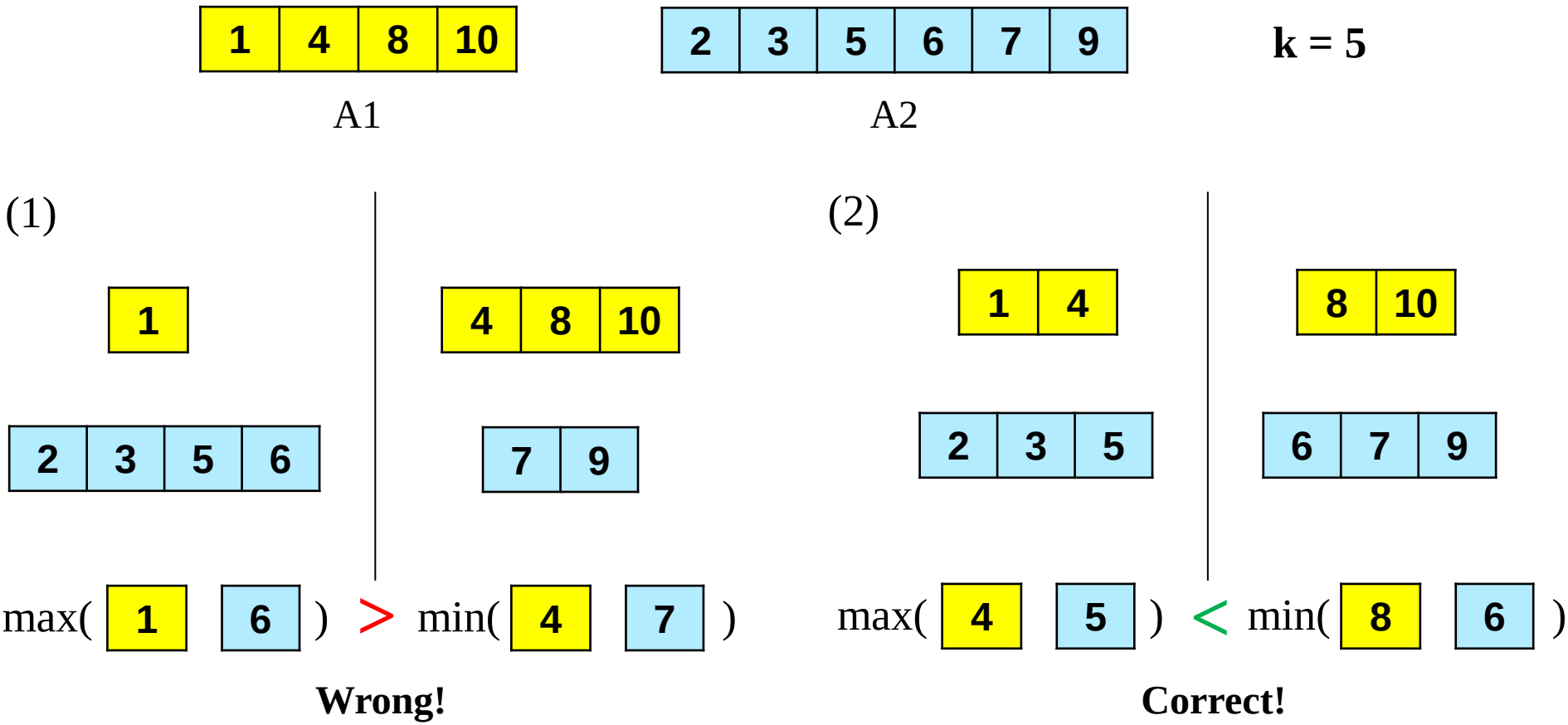
Pseudocode:

kthElement (A1[0..m-1], A2[0..n-1], k)

Input: : Two sorted arrays of size m and n respectively
Output: : The element that would be at the k'th position of the final sorted array.

```
1.  i1 ← 0; i2 ← 0; res = null
2.  while i1 < m and i2 < n do
3.      if k = 0 then return res
4.      if A1[i1] < A2[i2] then
5.          res ← A1[i1]; i1 ← i1 + 1; k = k - 1
6.      else
7.          res ← A2[i2]; i2 ← i2 + 1; k = k - 1
8.  while i1 < m do
9.      if k = 0 then return res
10.     res ← A1[i1]; i1 ← i1 + 1; k = k - 1
11. while i2 < n do
12.     if k = 0 then return res
13.     res ← A2[i2]; i2 ← i2 + 1; k = k - 1
```

2) Decrease and conquer



Condition: max_left <= min_right

2) Decrease and conquer

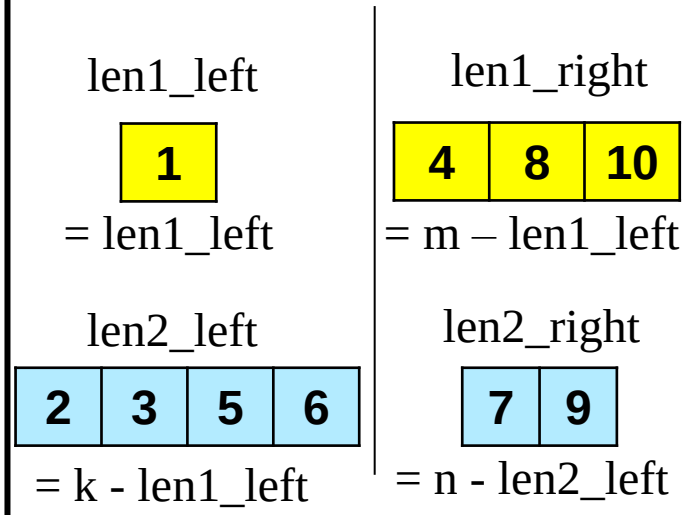
Pseudocode:

kthElement (A1[0..m-1], A2[0..n-1], k)

Input: : Two sorted arrays of size m and n. m <= n.
Output: : The element that would be at the k'th position of the final sorted array.

```
1.  for  len1_left ← 0 to min(m, k) do
2.      len1_right ← m – len1_left;
3.      len2_left = k - len1_left; len2_right = n - len2_left
4.      if len1_left = 0 then left_max =  A2[len2_left - 1]
5.      else if len2_left = 0 then left_max =  A1[len1_left  - 1]
6.      else left_max =  max(A1[len1_left-1],
7.                           A2[len2_left - 1])
8.      if len1_right = 0 then right_min = A2[len2_left]
9.      else if len2_right = 0 then right_min = A1[len1_left]
10.     else right_min =  min(A1[len1_left],
11.                           A2[len2_left])
12.     if left_max <= right_min then
13.         return left_max
```

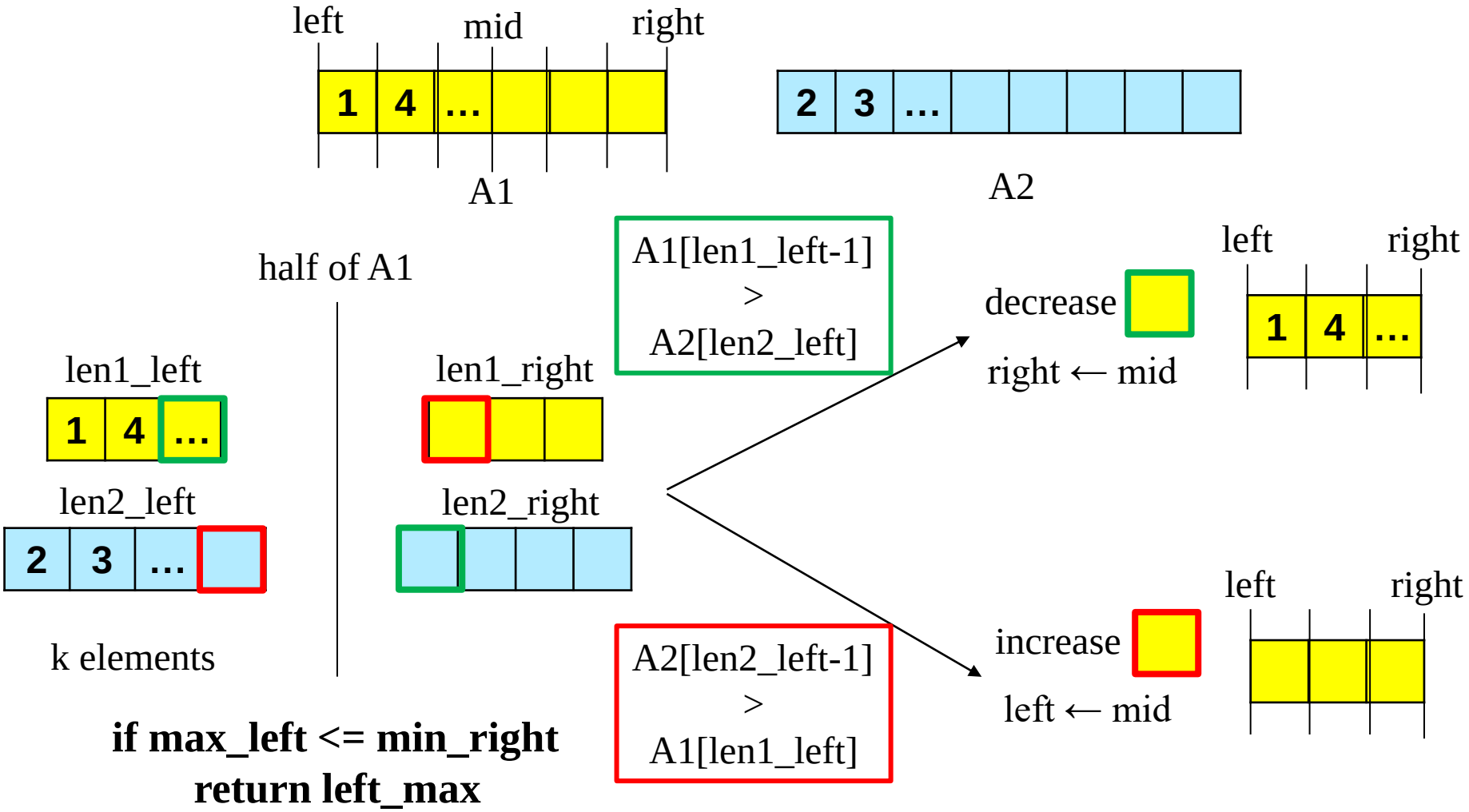
Why m <= n:
Ex. m = 10, n = 1, k = 8
len1_left = 5 ⟹ len2_left = 3



Time complexity:
O(min(*m*, *n*, *k*))

Space complexity: O(1)

3) Decrease and conquer 2



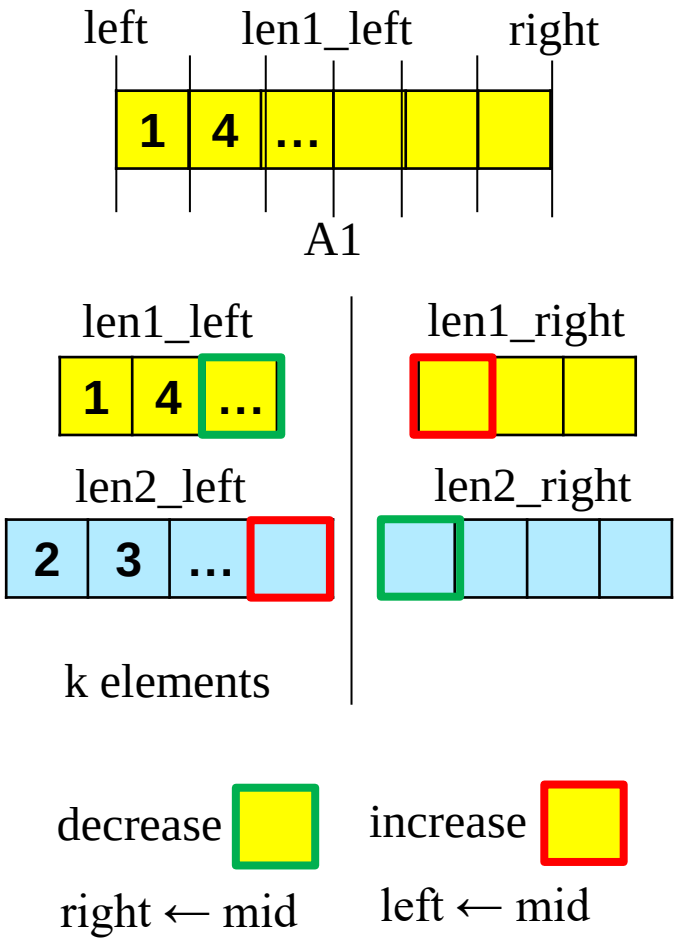
3) Decrease and conquer 2

kthElement (A1[0..m-1], A2[0..n-1], k)

Input: : Two sorted arrays of size m and n. m <= n.
Output: : The k-th element of the final sorted array.

```
1.  left ← 0; right ← min(k, m)
2.  while  left + 1 < end do
3.      len1_left ← (left + right) / 2; len1_left = m – i
4.      len2_left = k - len1_left; len2_right = n - len2_left
5.      if len1_left = 0 then left_max =  A2[len2_left - 1]
6.      else if len2_left = 0 then left_max =  A1[len1_left-1]
7.      else left_max =  max(A1[len1_left-1],
8.                           A2[len2_left - 1])
9.      if len1_right = 0 then right_min = A2[len2_left]
10.     else if len2_right = 0 then right_min = A1[len1_left]
11.     else right_min =  min(A1[len1_left],
12.                           A2[len2_left])
13.     if left_max <= right_min then
14.         return left_max
```

Continue on next page

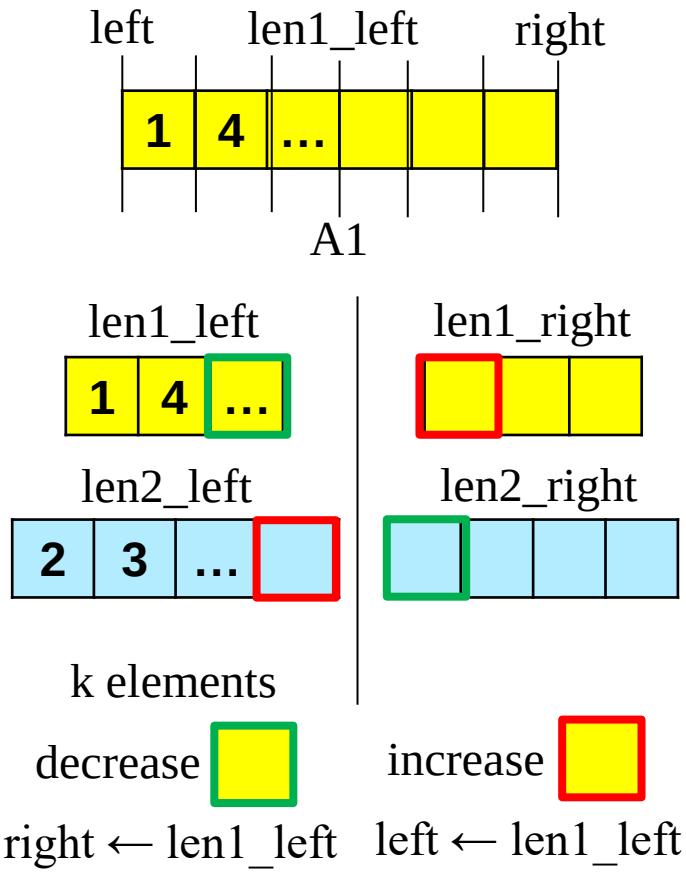


3) Decrease and conquer 2

kthElement (A1[0..m-1], A2[0..n-1], k)

Input: : Two sorted arrays of size m and n. m <= n.
Output: : The k-th element of the final sorted array.

```
15.     else
16.         if len1_left = 0 then left ← len1_left
17.         else if len1_right = 0 then right ← len1_left
18.         else if A1[len1_right - 1] > A2[len2_left] then
19.             right ← len1_left
20.         else if A2[len2_left - 1] > A1[len1_left] then
21.             left ← len1_left
22. if  leftMax <= rightMin then
23.     return leftMax
24. else
25.     return rightMin
```



Time complexity: O(log(min(*m*, *n*, *k*)))

Space complexity: O(1)

	Time complexity	Space complexity
Merge 2 sorted arrays:	O(k)	O(1)
Decrease and conquer:	O(min(m, n, k))	O(1)
Decrease and conquer 2 :	O(log(min(m, n, k)))	O(1)

m and n are the size of A1 and A2, respectively
k is the position of the required element