In-place Algorithm

Motivation

- Some devices don't have enough space
  - Embedded system like PDA, cell phone……
- I/O spends much more time than calculation, and less space usually means fewer I/O
  - Database
- Reducing space usage is important

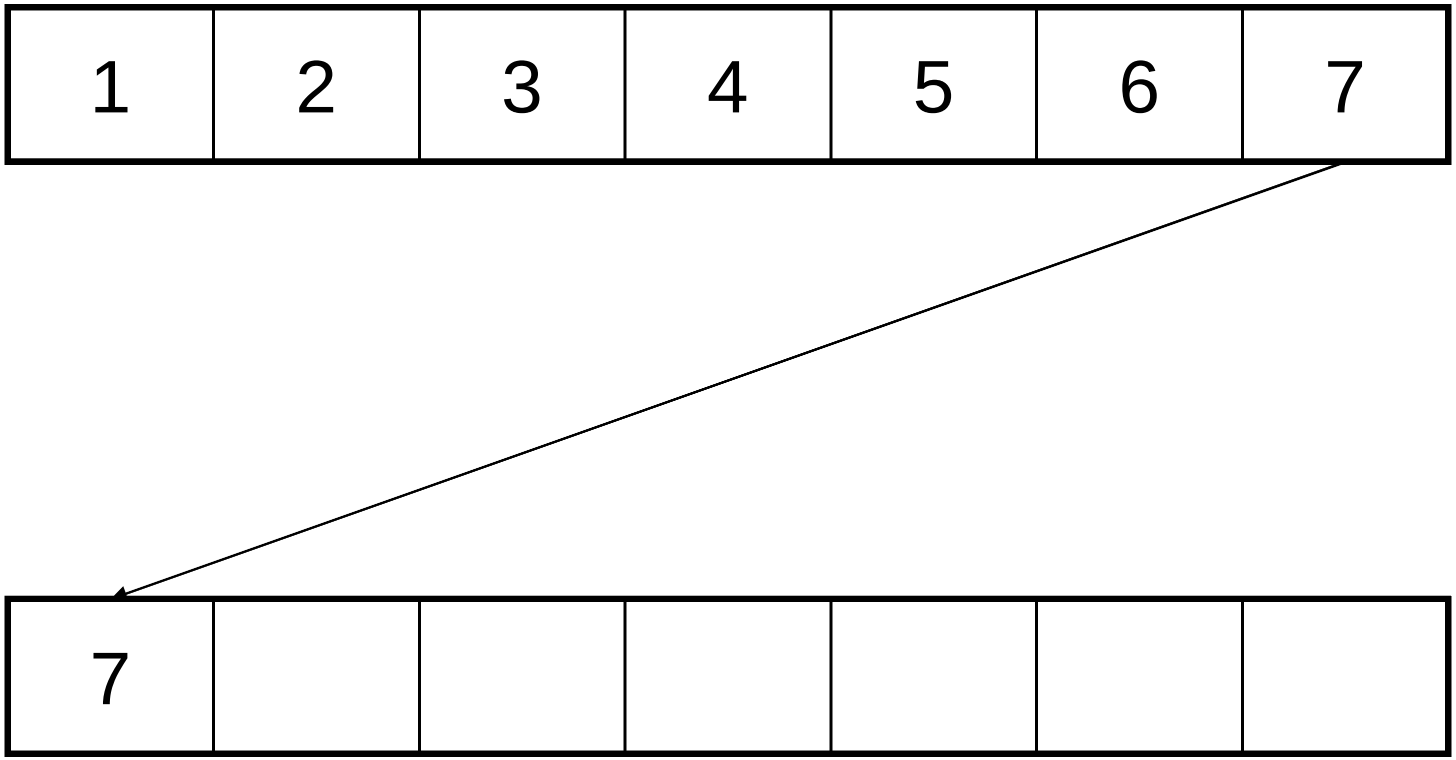
Simple Reverse

- Problem definition:
  - Given an array $A[0..n]$
  - Output the "reverse" of A
    - That is, output an array $B[0..n]$ such that $B[k] = A[n-k]$ for every k
  - In this problem, we are not required to keep A after the processing

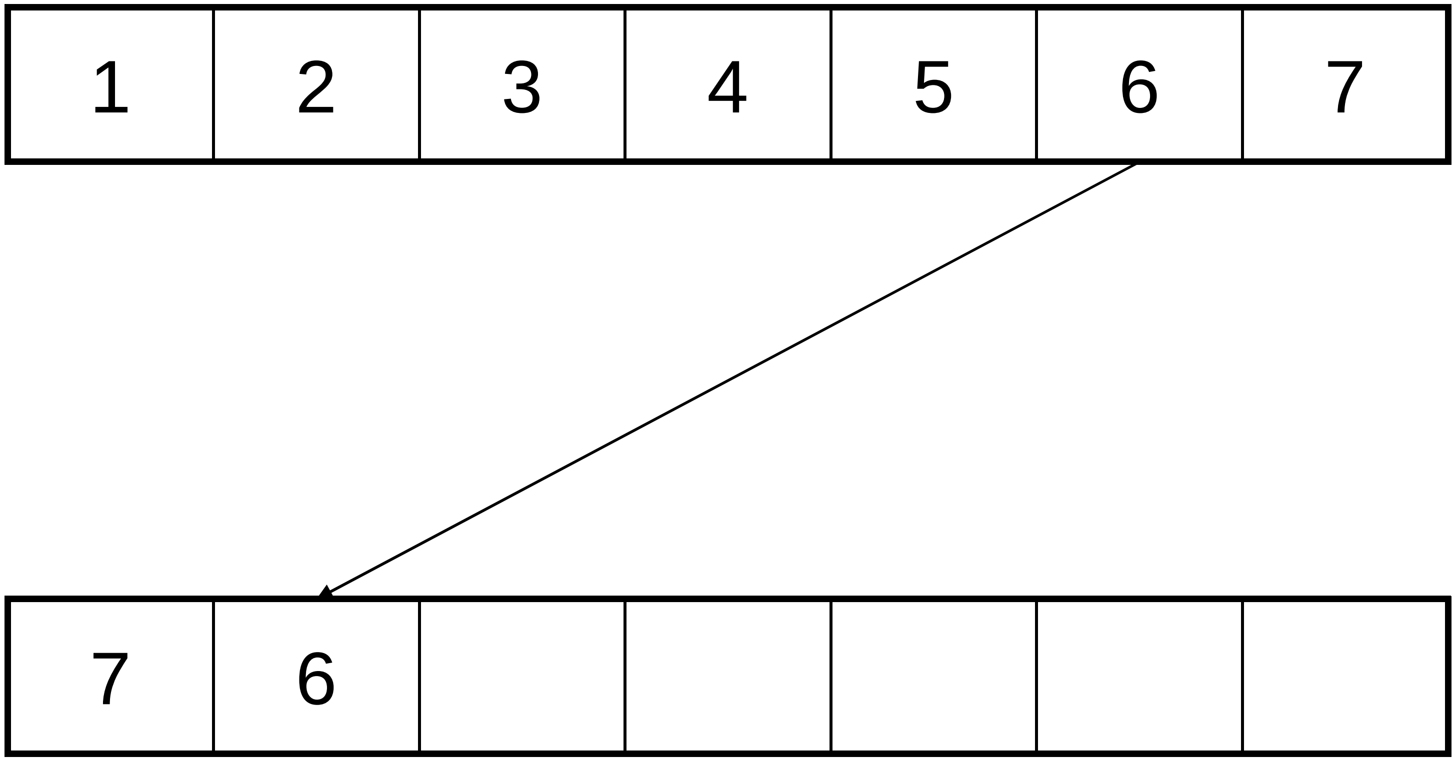
Simple Reverse

- Solution 1: Use a new array with size equal to the input array size

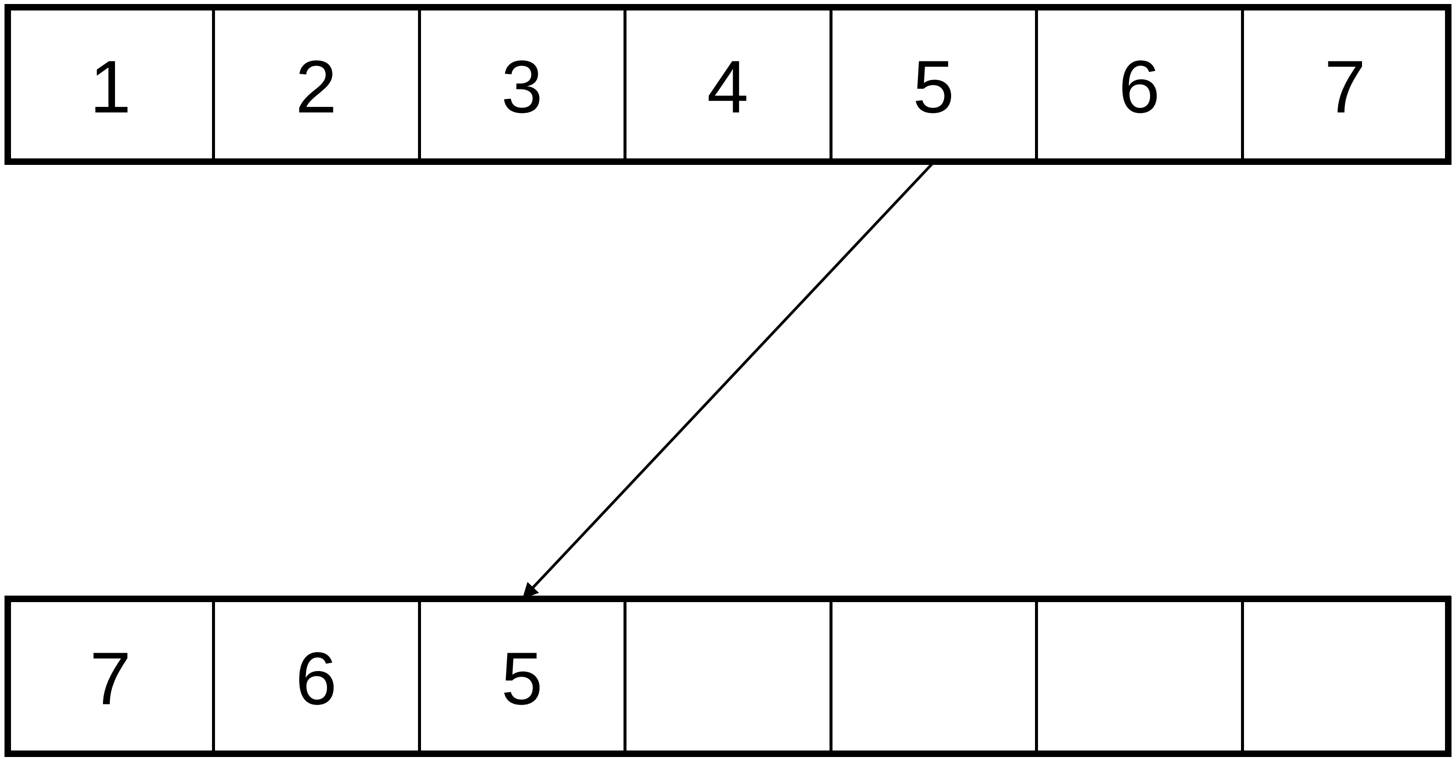
Simple Reverse



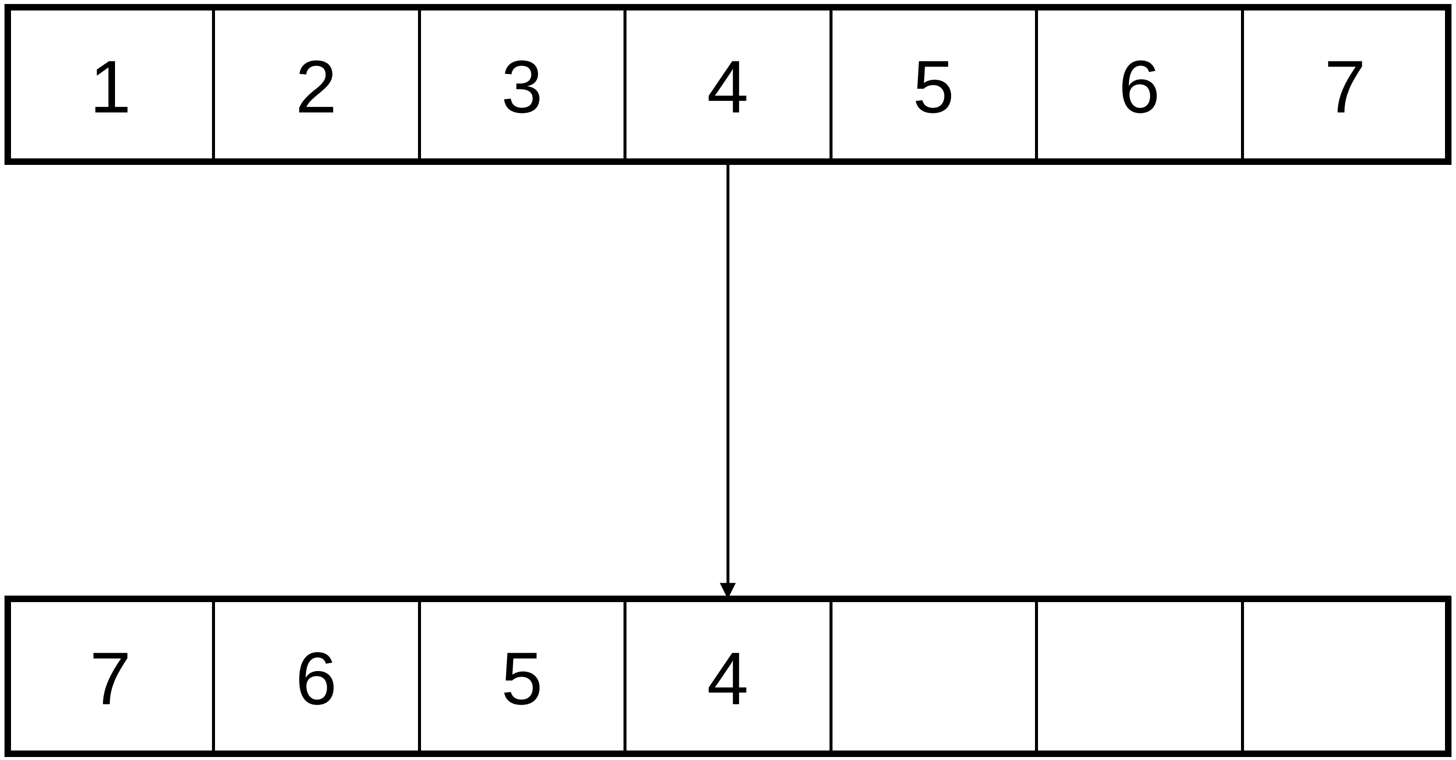
Simple Reverse



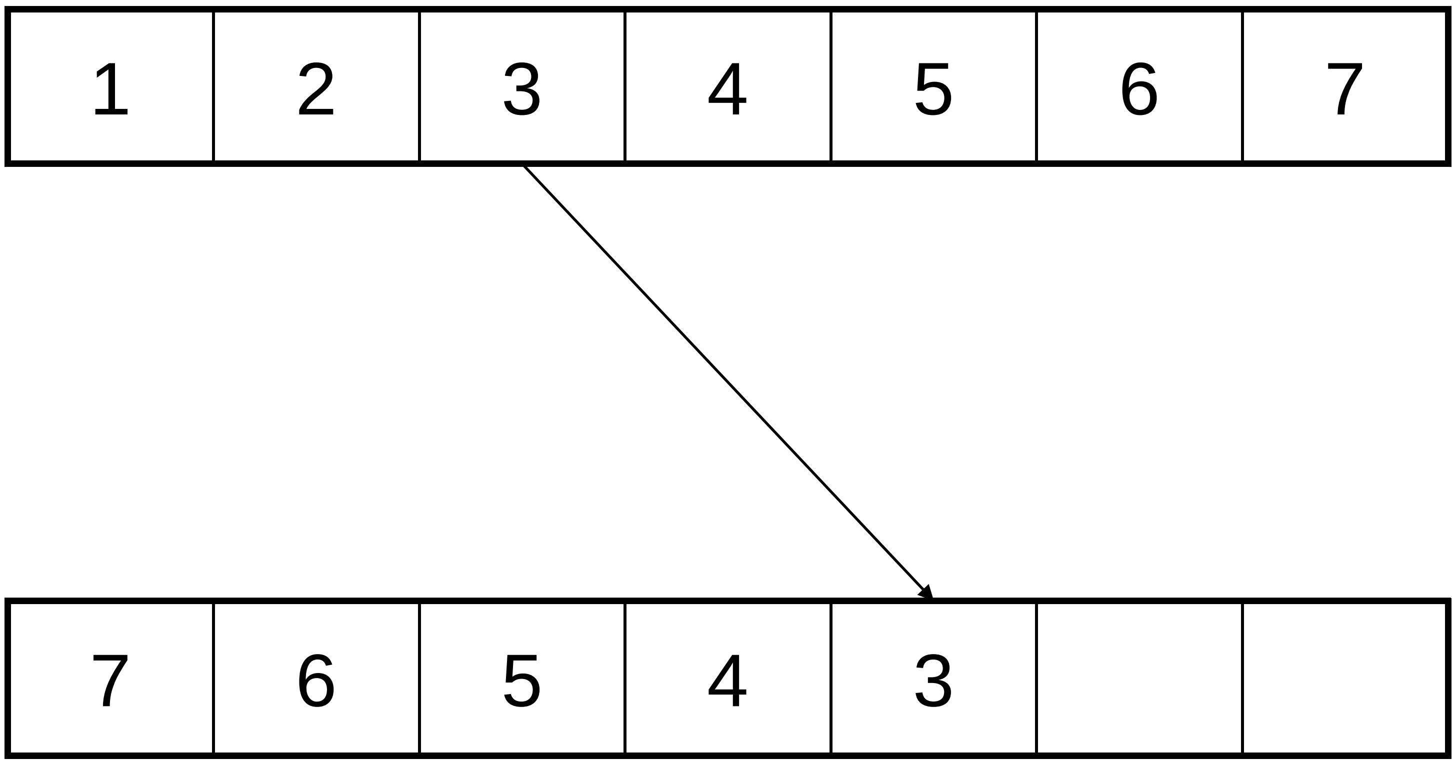
Simple Reverse



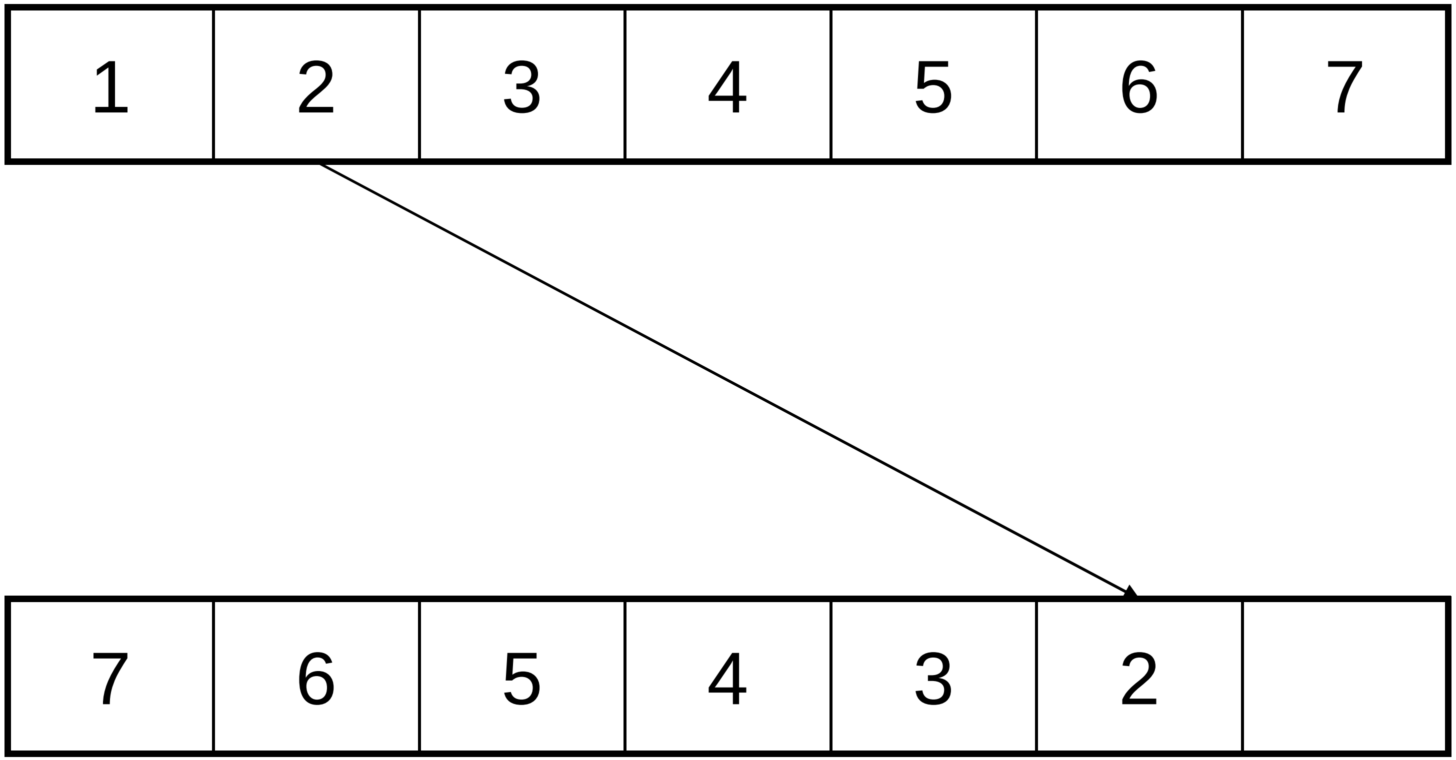
Simple Reverse



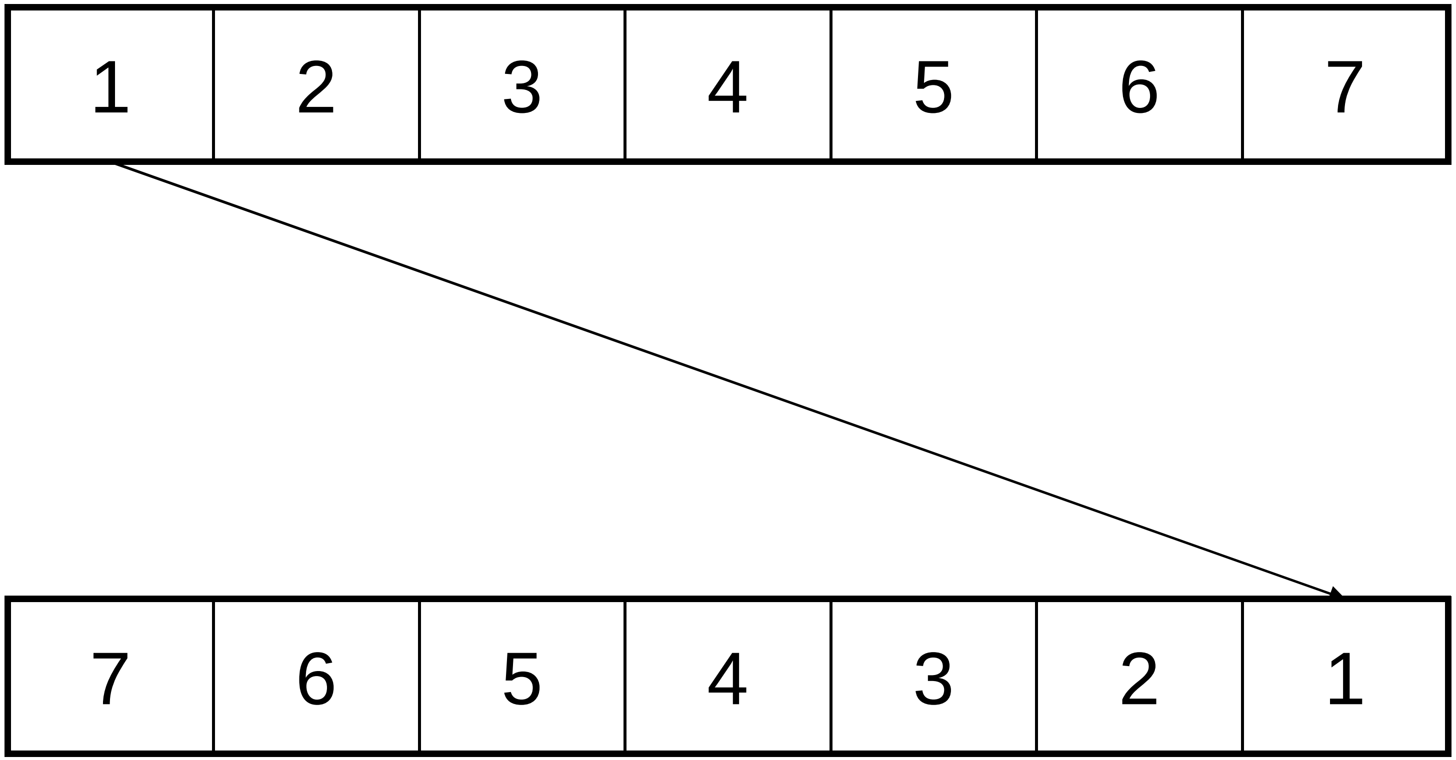
Simple Reverse



Simple Reverse



Simple Reverse



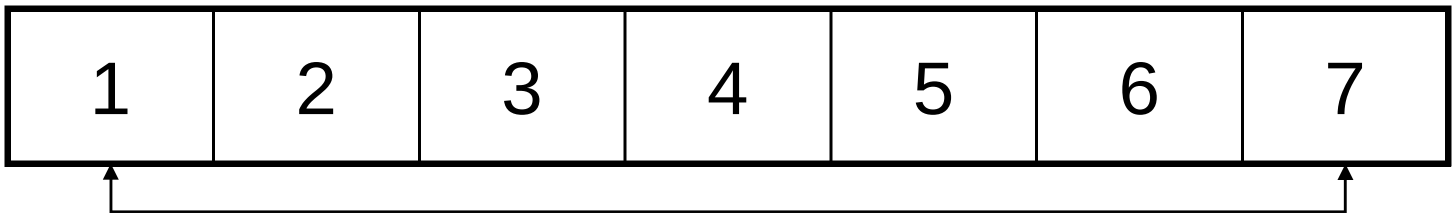
Simple Reverse

- Needs $O(n)$ extra space
- Can we use less space?

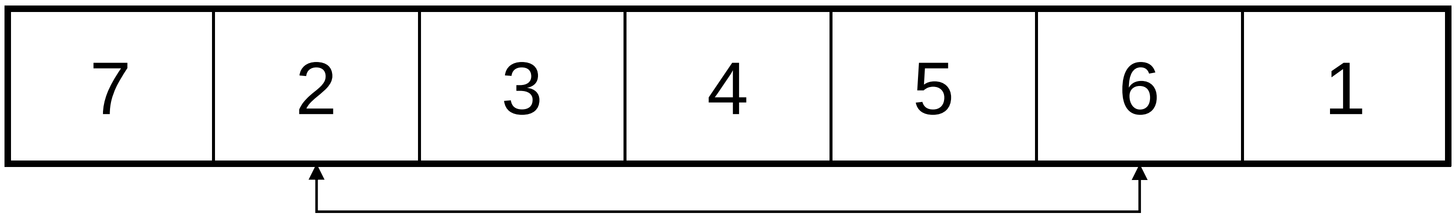
Smarter Reverse

- Solution 2: Exchange the first and the last elements (inside A), and then serve the remaining list

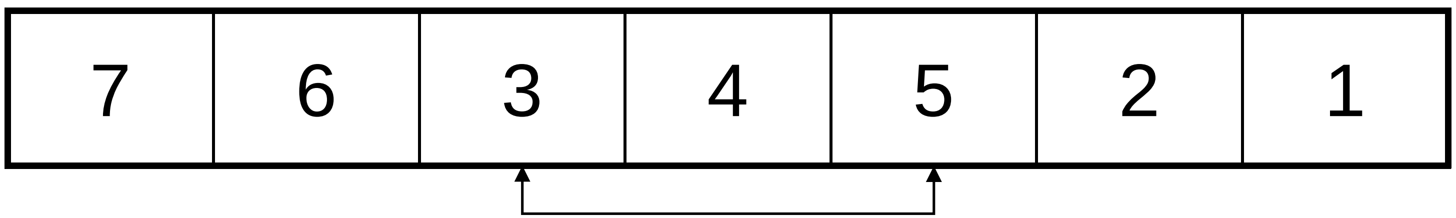
Smarter Reverse



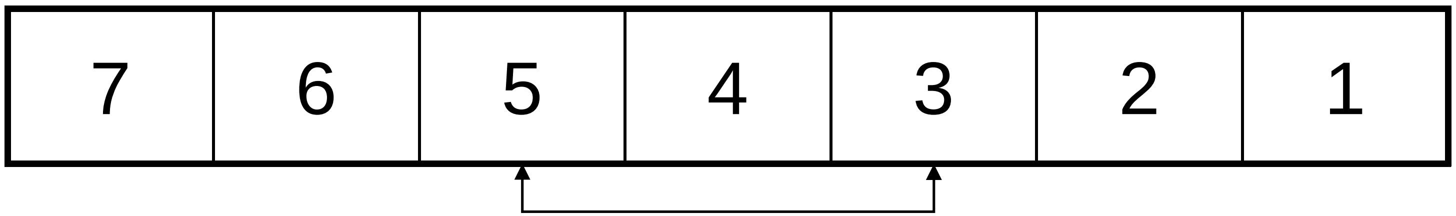
Smarter Reverse



Smarter Reverse



Smarter Reverse



Smarter Reverse

7	6	5	4	3	2	1

Smarter Reverse

- Needs O(1) extra space
  - One for exchanging elements

What is In-place Algorithm?

- Algorithm that uses a small constant amount of extra space in addition to the original input
- Usually overwrite the input space
  - Spend more time in some cases
- On the contrary: not-in-place or out-of-place

More Examples

- Do we know any algorithms which are in-place?
  - Insertion sort
  - Selection sort

Insertion Sort

4	2	3	6	7	1	8

Insertion Sort

4	2	3	6	7	1	8

Insertion Sort

2	**4**	3	6	7	1	8

Insertion Sort

2	**3**	**4**	6	7	1	8

Insertion Sort

2	**3**	**4**	**6**	7	1	8

Insertion Sort

2	**3**	**4**	**6**	**7**	1	8

Insertion Sort

1	**2**	**3**	**4**	**6**	**7**	8

Insertion Sort

1	2	3	4	6	7	8

Insertion Sort

1	2	3	4	6	7	8

- Only needs O(1) extra space
  - One for exchanging

Selection Sort

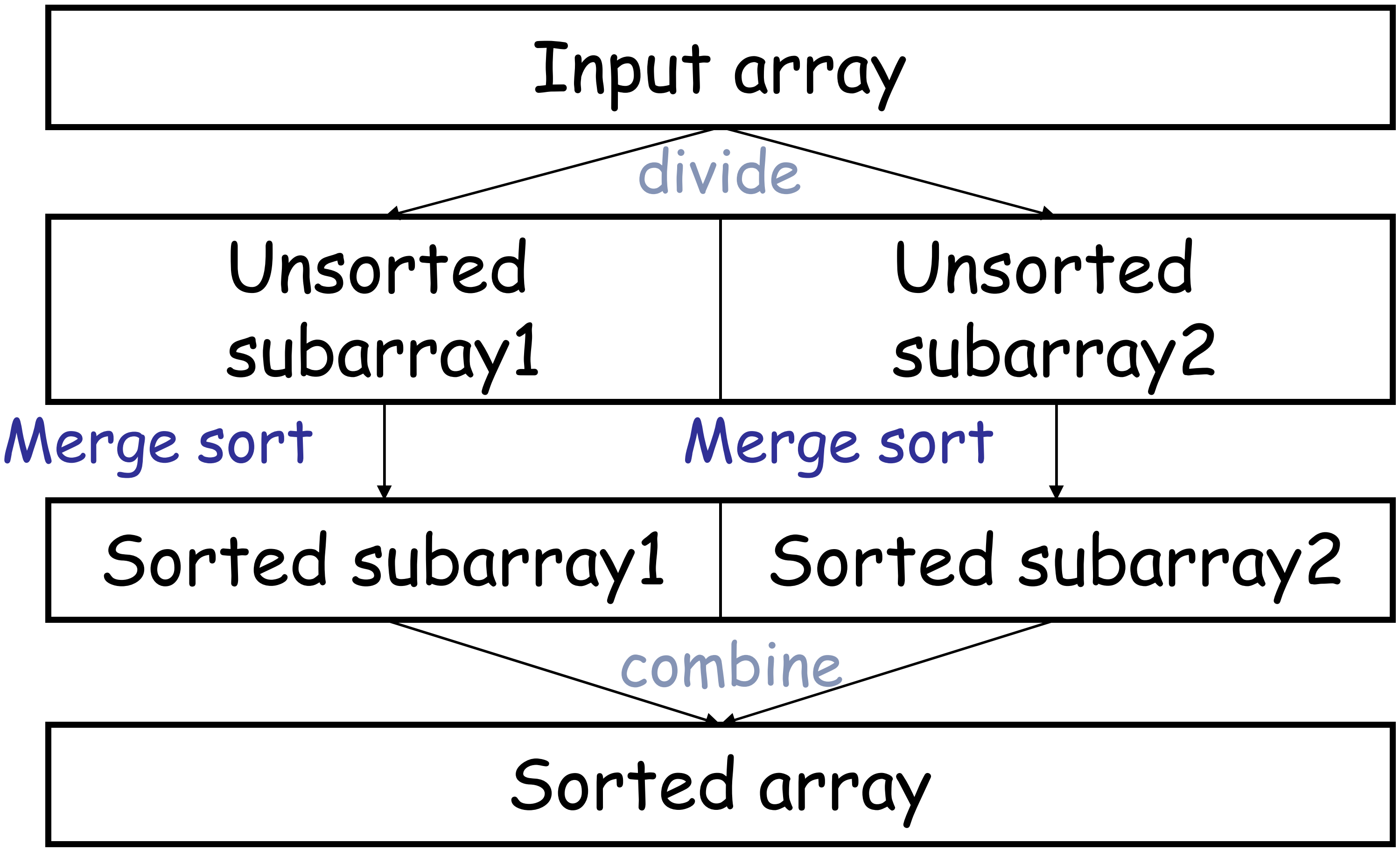
- How about Selection Sort?

- Needs only O(1) extra space
  - For exchanging

Not-in-place Algorithm

- Do we know any algorithms which are not-in-place?
  - Merge sort
    - O(n) extra space for merging

Simple Merge Sort



What's wrong in simple?

- In the merge step

2	6	8	9

1	3	4	7

1							

What's wrong in simple?

- In the merge step

2	6	8	9

1	3	4	7

1	2						

What's wrong in simple?

- In the merge step

2	6	8	9

1	3	4	7

1	2	3					

What's wrong in simple?

- In the merge step

2	6	8	9

1	3	4	7

1	2	3	4				

What's wrong in simple?

- In the merge step

2	6	8	9

1	3	4	7

1	2	3	4	6			

What's wrong in simple?

- In the merge step

2	6	8	9

1	3	4	7

1	2	3	4	6	7		

What's wrong in simple?

- In the merge step

2	6	8	9

1	3	4	7

1	2	3	4	6	7	8	

What's wrong in simple?

- In the merge step

2	6	8	9

1	3	4	7

1	2	3	4	6	7	8	9

What's wrong in simple?

- In the merge step, needs O(n) extra space

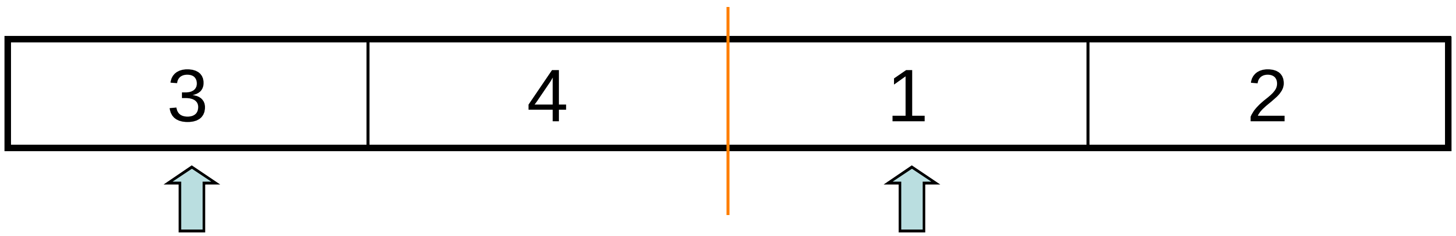
2	6	8	9

1	3	4	7

1	2	3	4	6	7	8	9

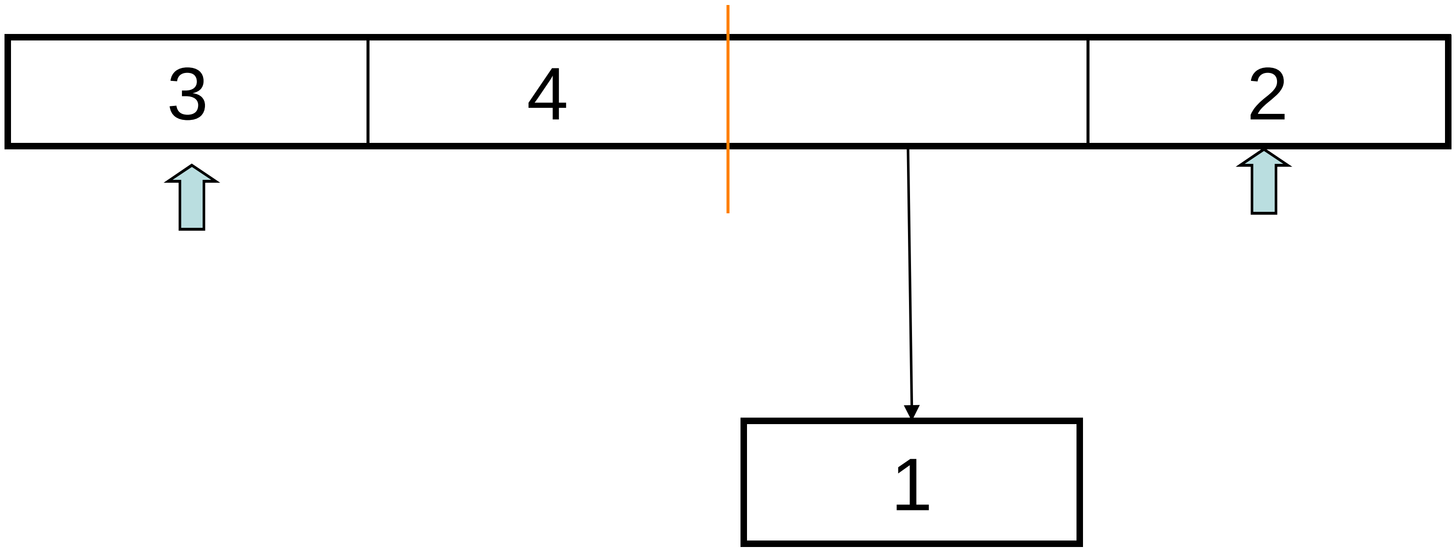
MergeSort2

- We design a new function called "inplaceMerge"



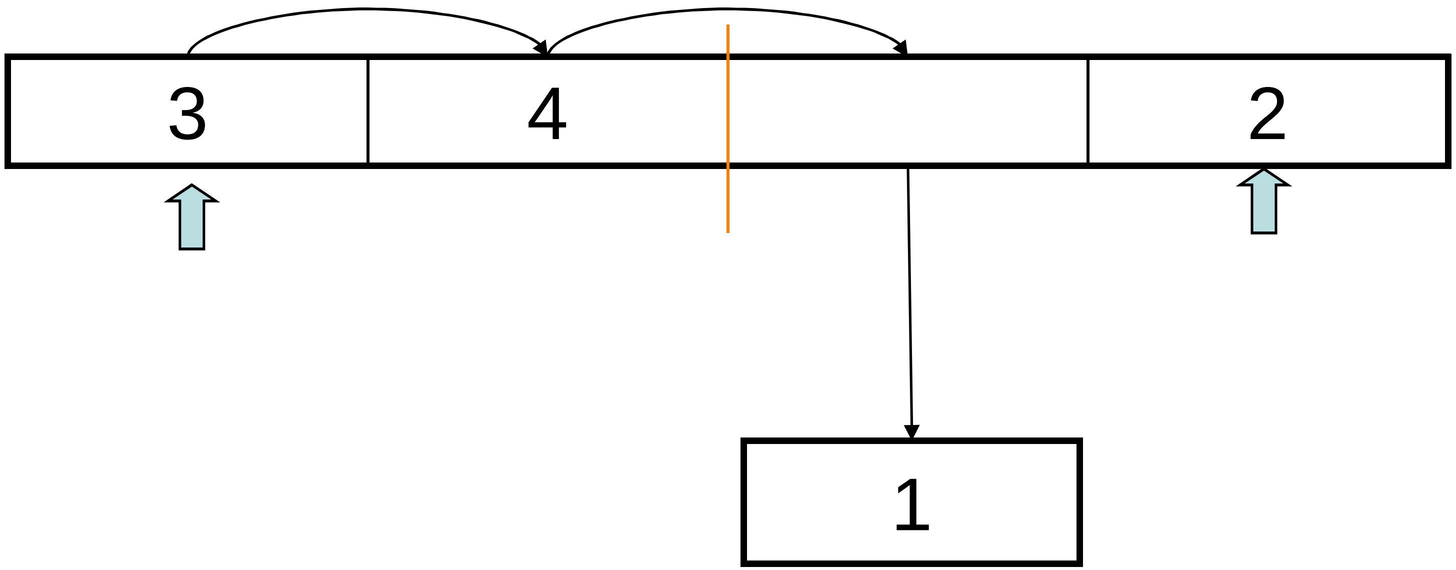
MergeSort2

- We design a new function called "inplaceMerge"



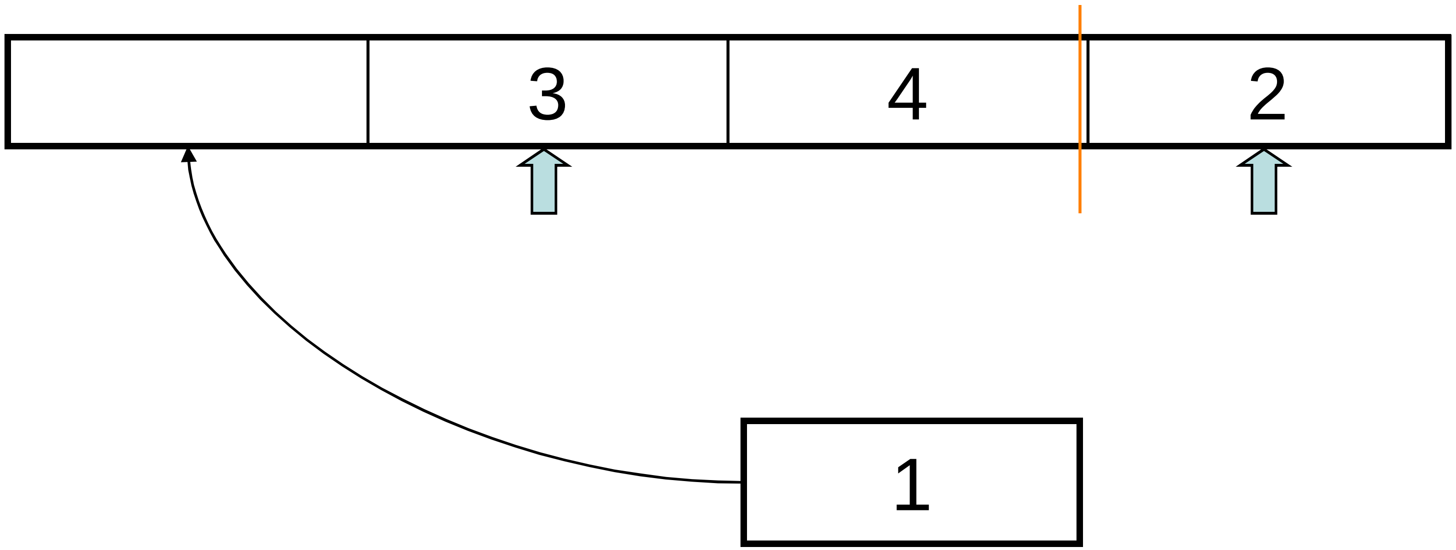
MergeSort2

- We design a new function called "inplaceMerge"



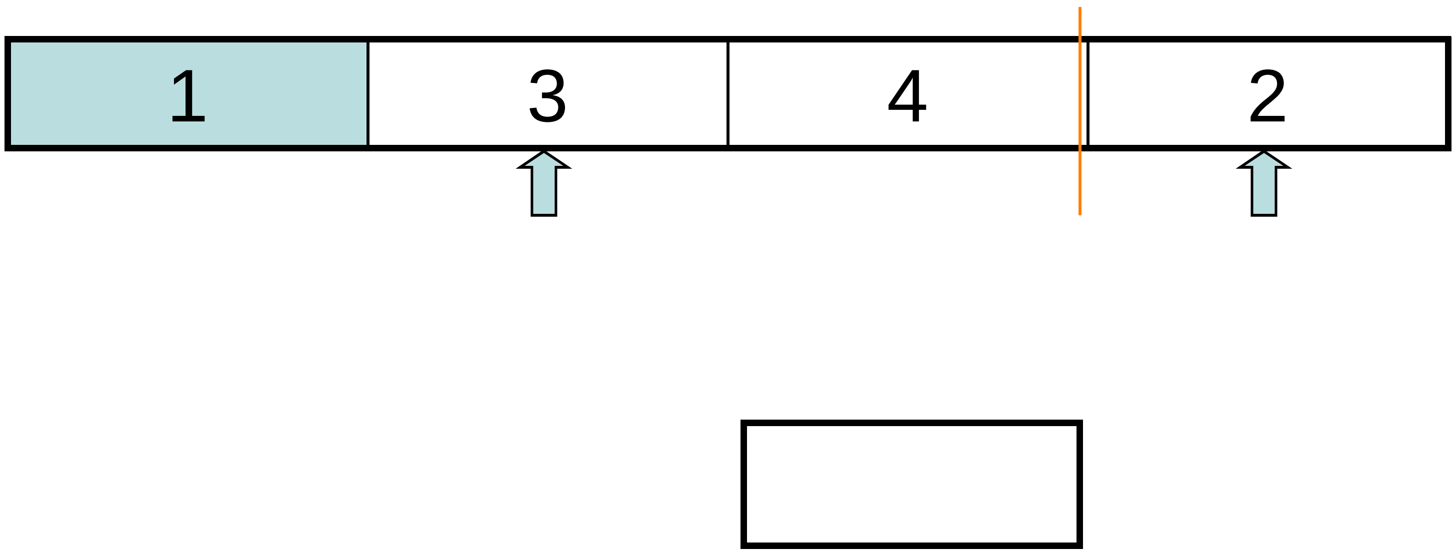
MergeSort2

- We design a new function called “inplaceMerge“



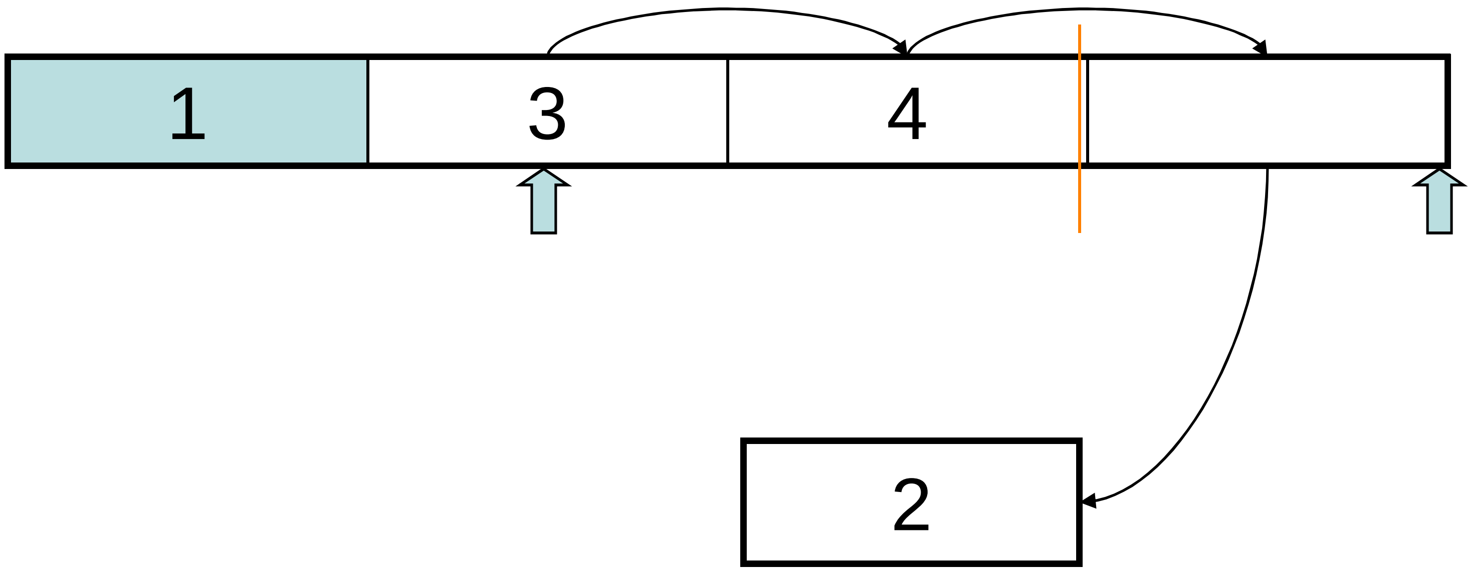
MergeSort2

- We design a new function called "inplaceMerge"



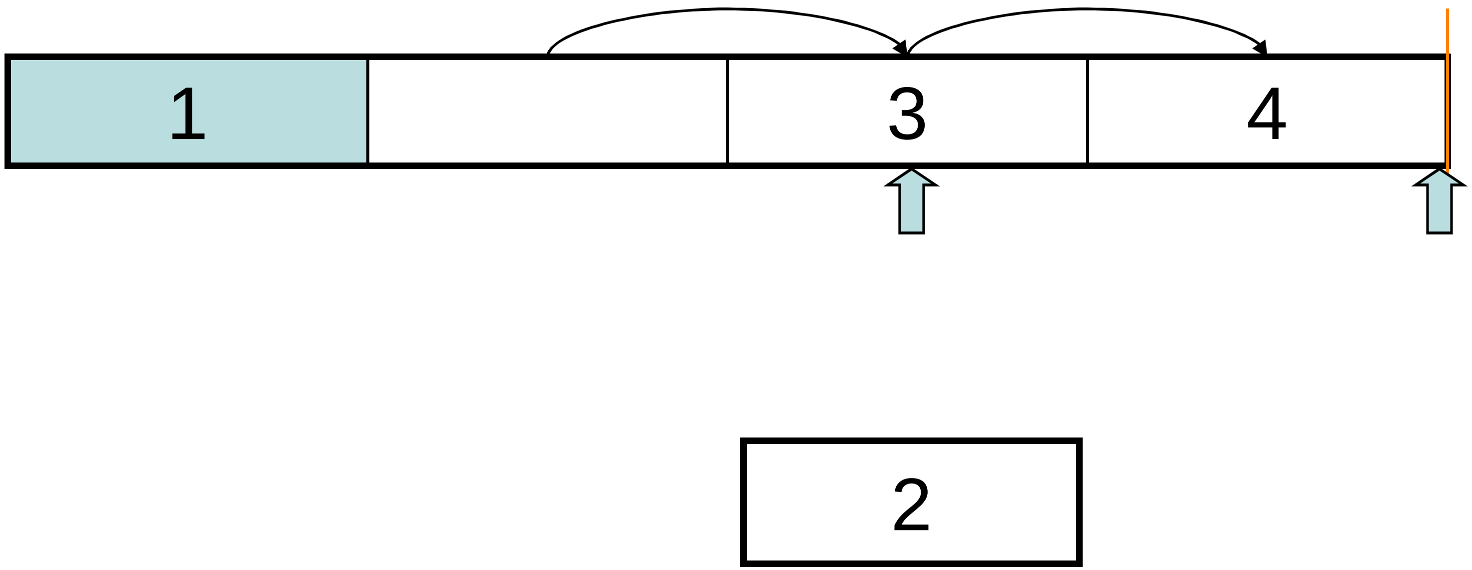
MergeSort2

- We design a new function called "inplaceMerge"



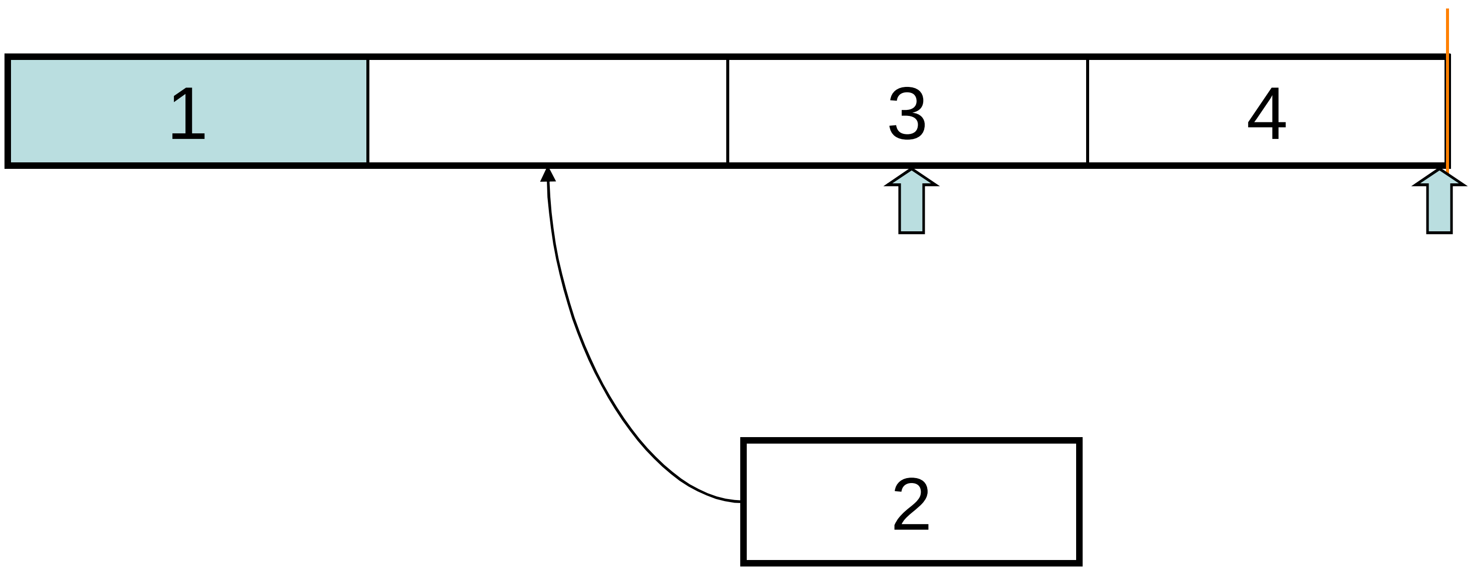
MergeSort2

- We design a new function called "inplaceMerge"



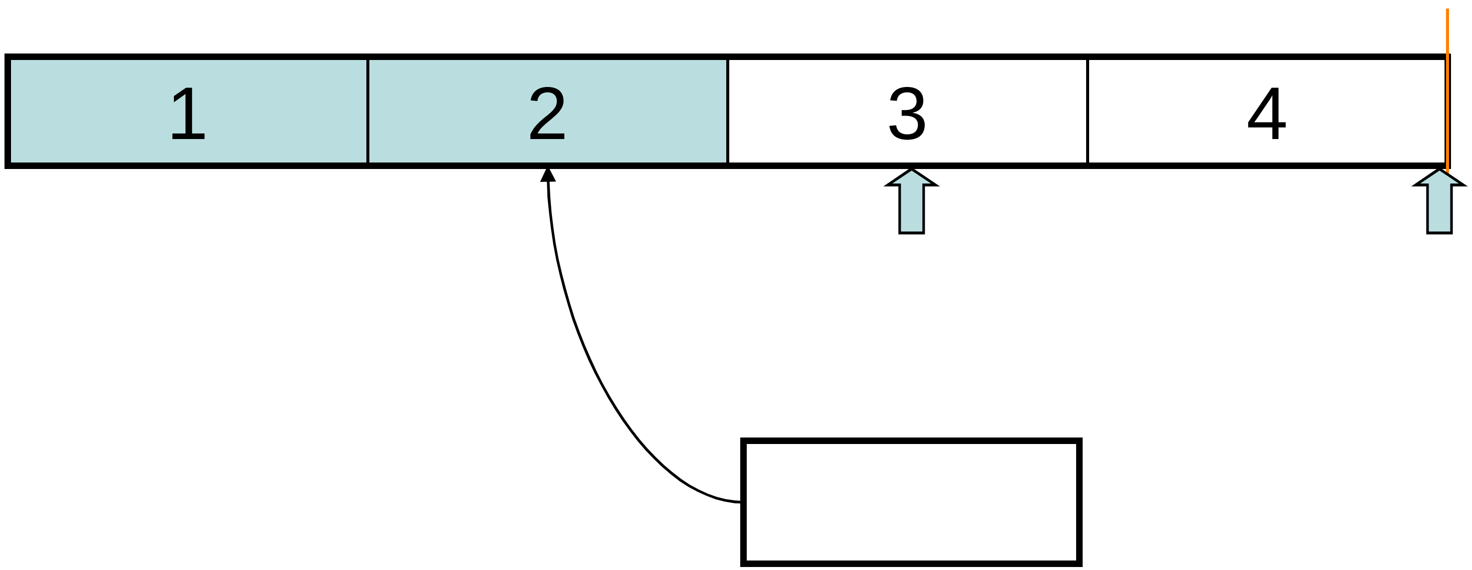
MergeSort2

- We design a new function called "inplaceMerge"



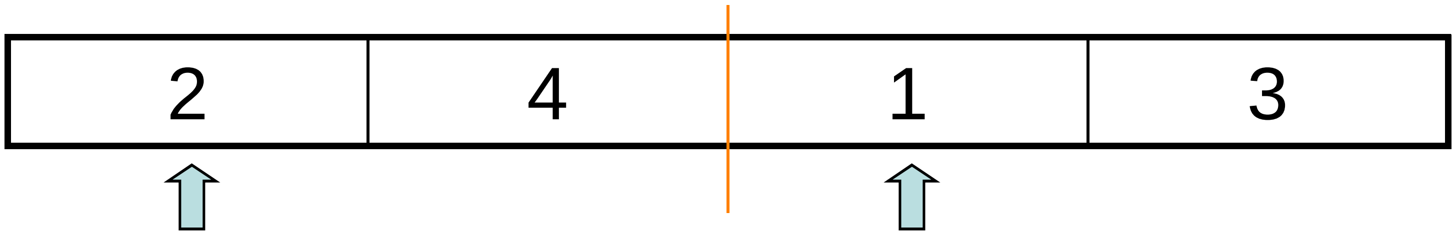
MergeSort2

- We design a new function called "inplaceMerge"



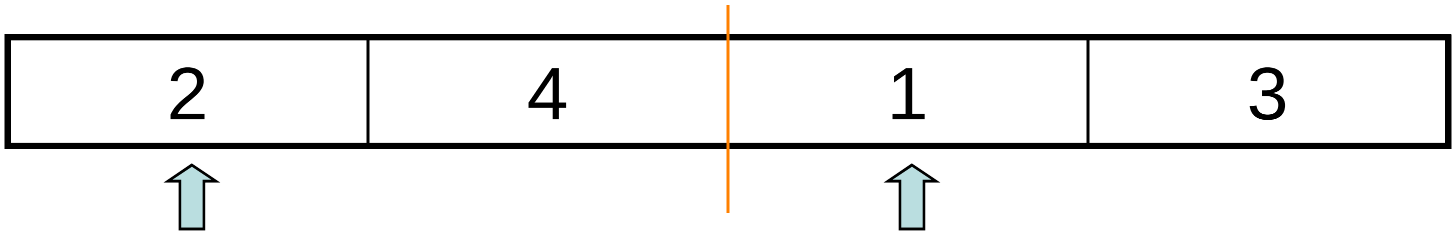
MergeSort2

- We design a new function called "inplaceMerge"



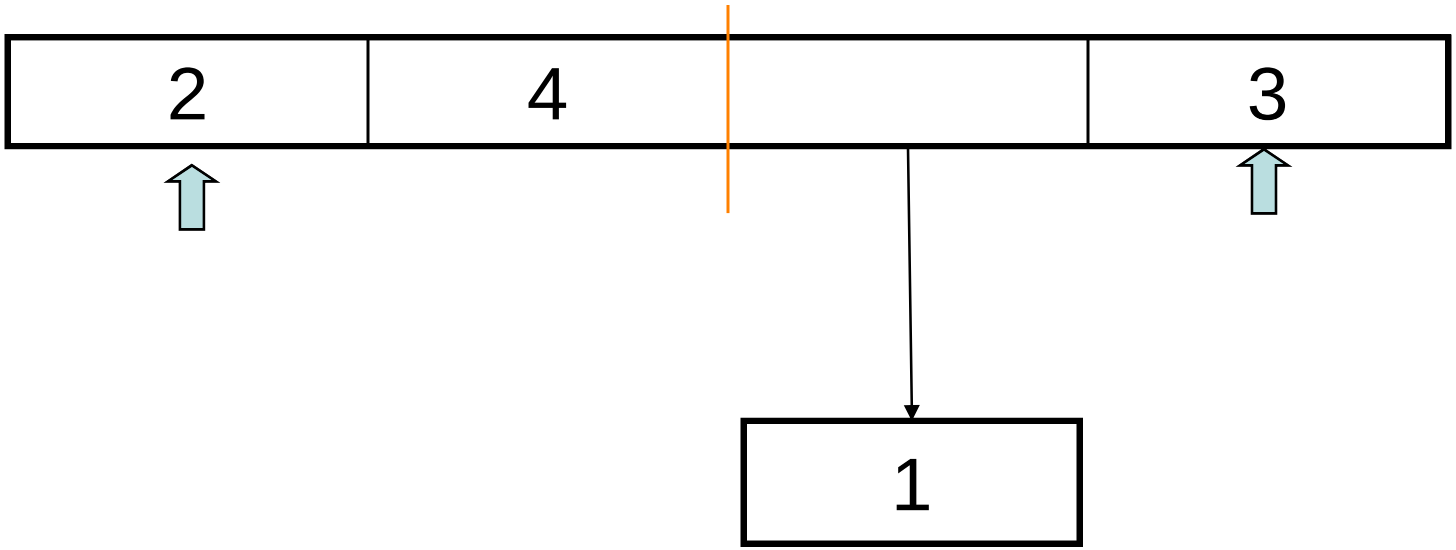
MergeSort2

- We design a new function called "inplaceMerge"



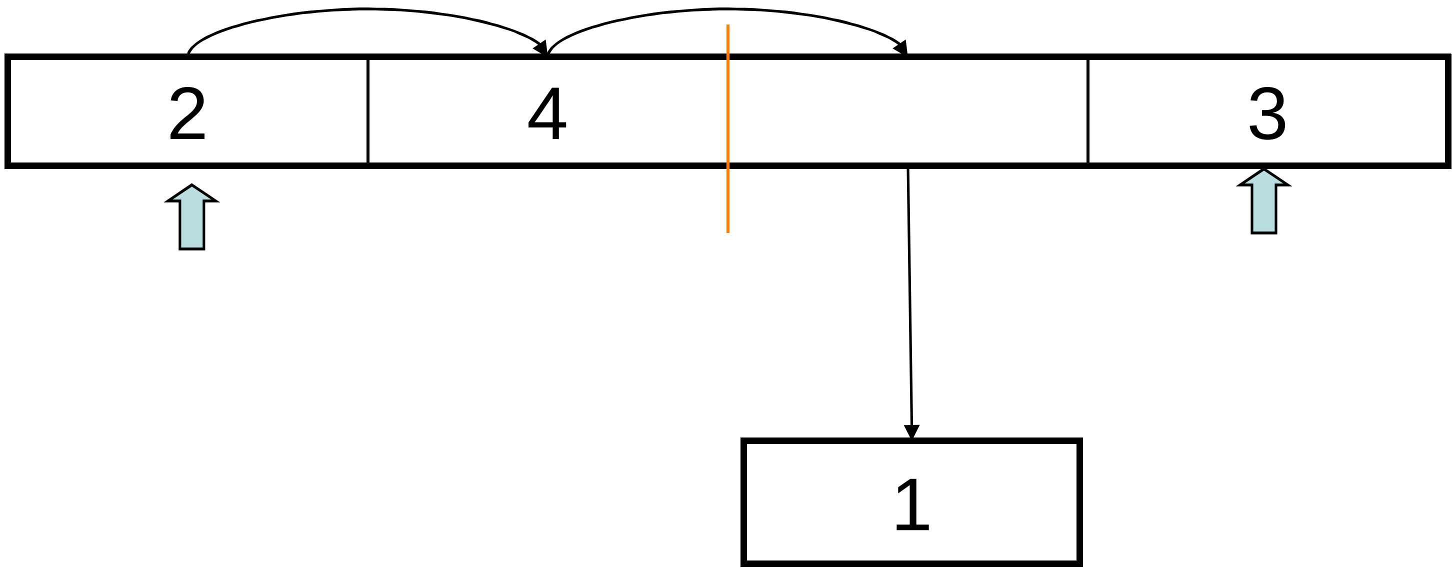
MergeSort2

- We design a new function called "inplaceMerge"



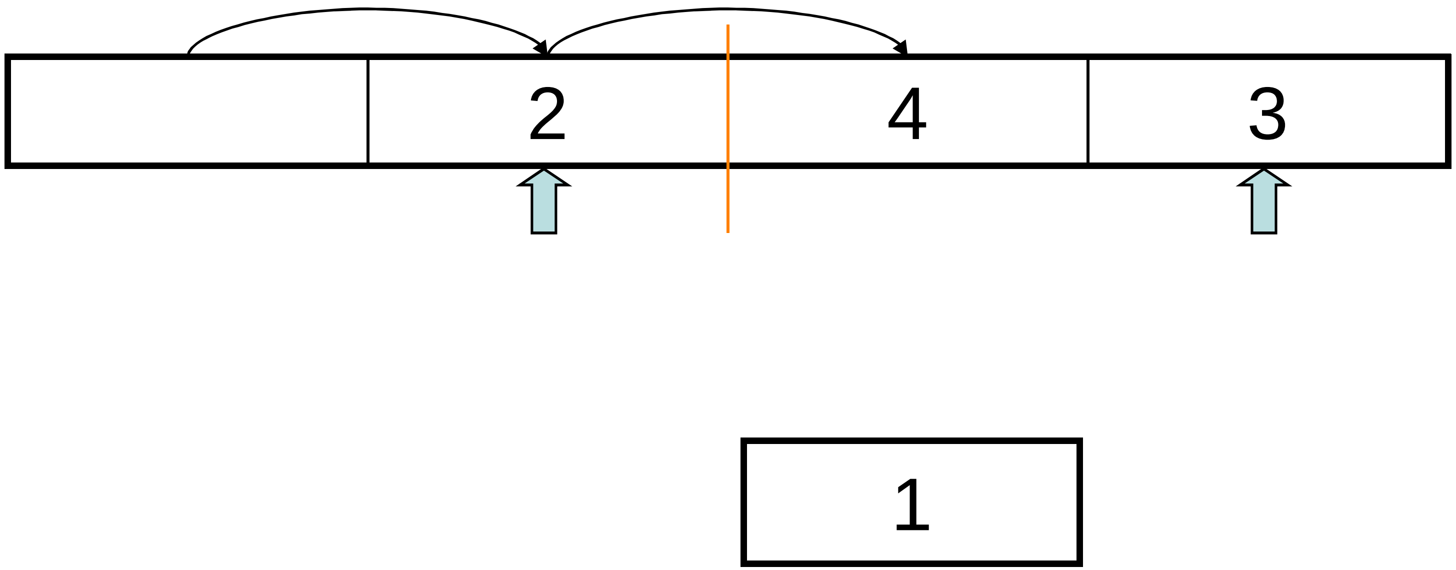
MergeSort2

- We design a new function called "inplaceMerge"



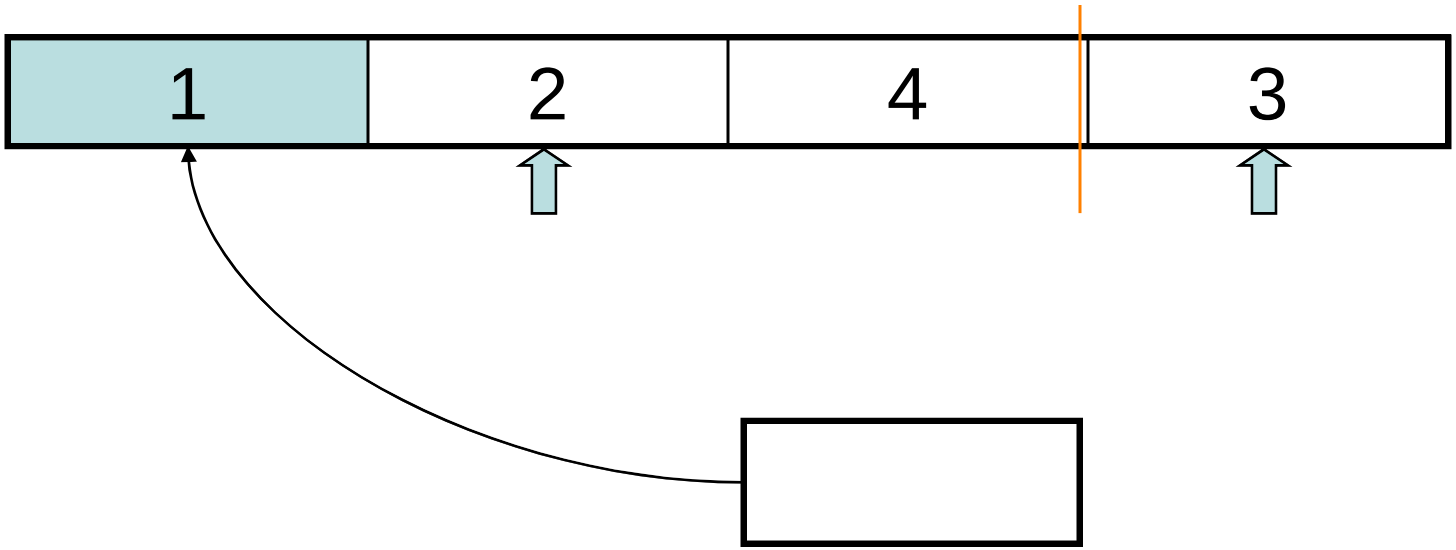
MergeSort2

- We design a new function called "inplaceMerge"



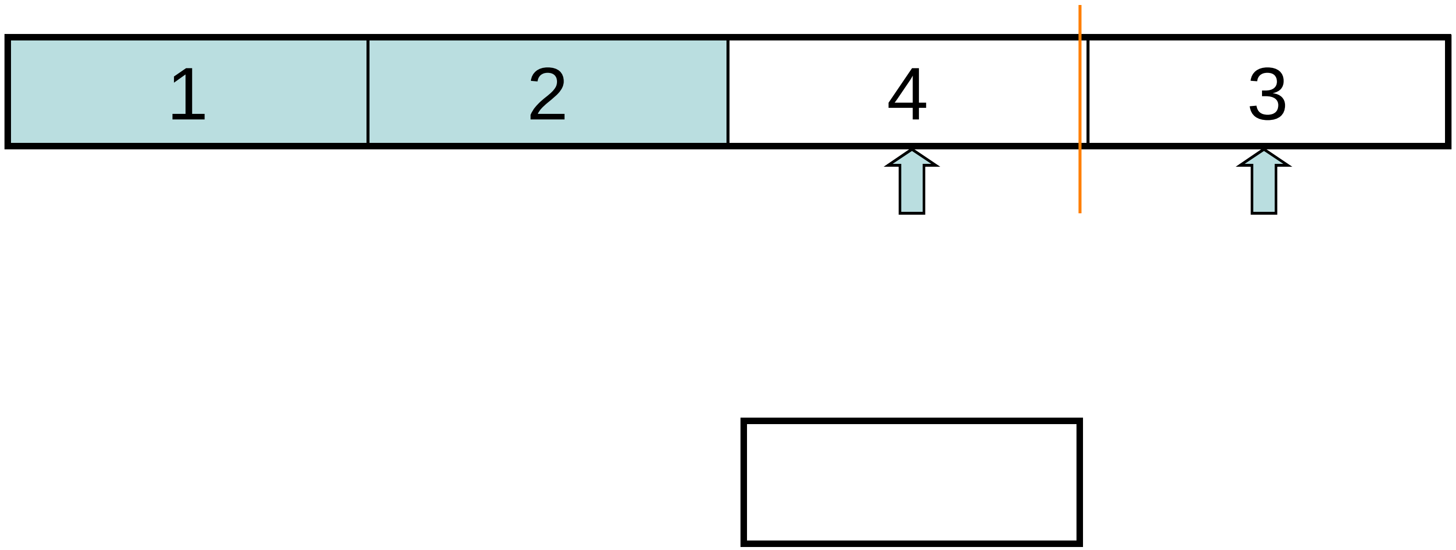
MergeSort2

- We design a new function called "inplaceMerge"



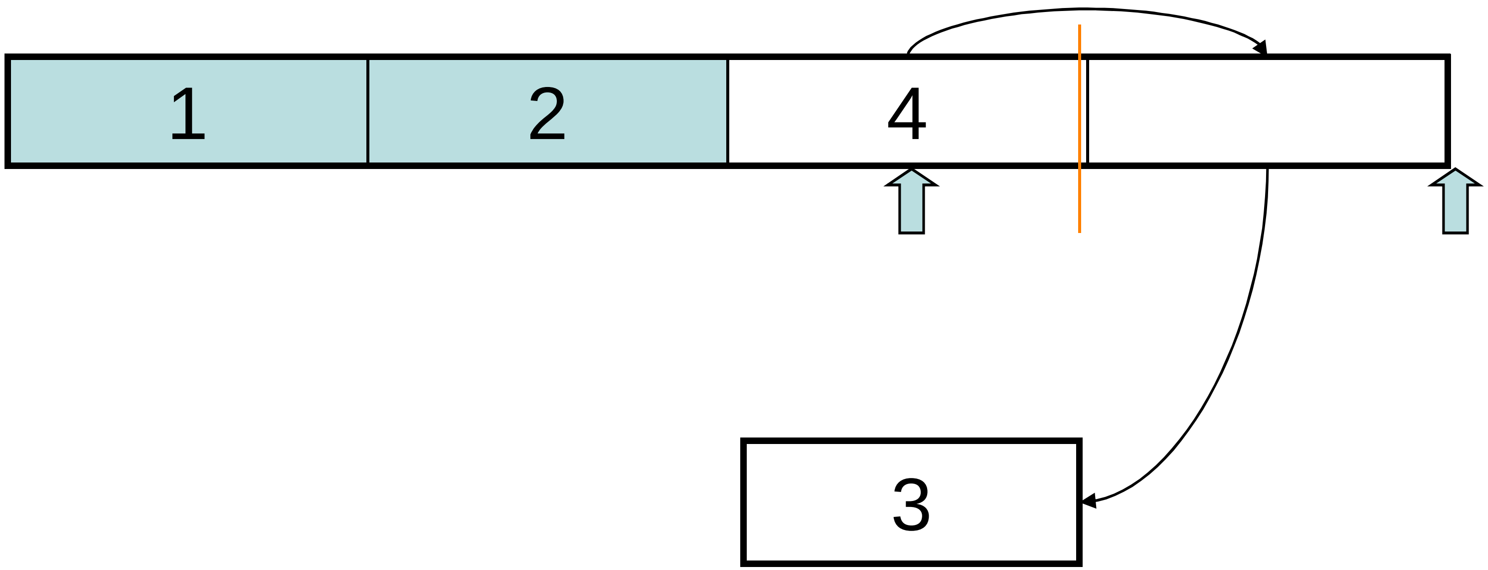
MergeSort2

- We design a new function called "inplaceMerge"



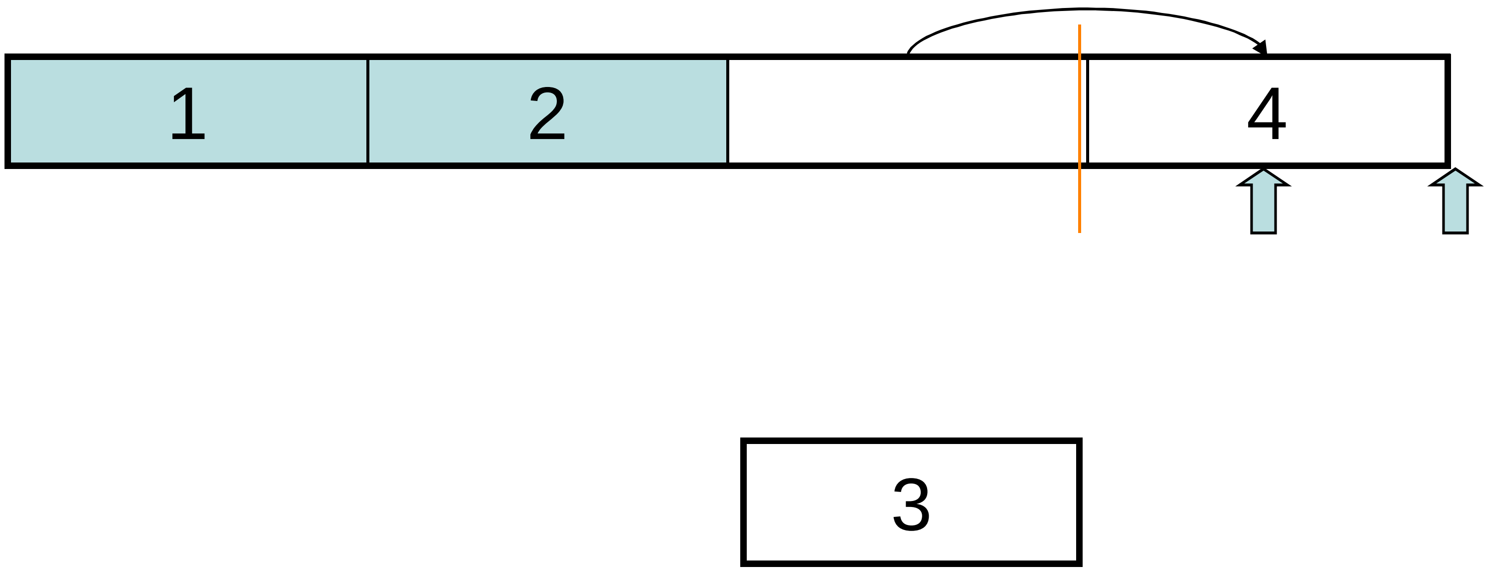
MergeSort2

- We design a new function called "inplaceMerge"



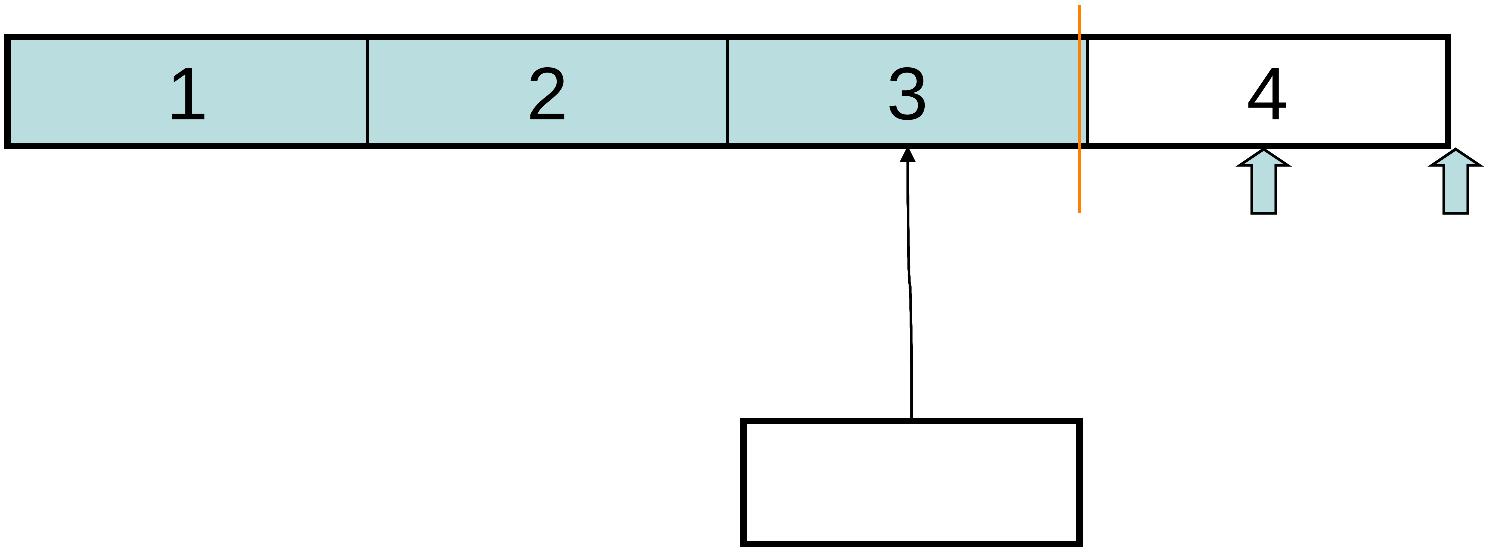
MergeSort2

- We design a new function called "inplaceMerge"



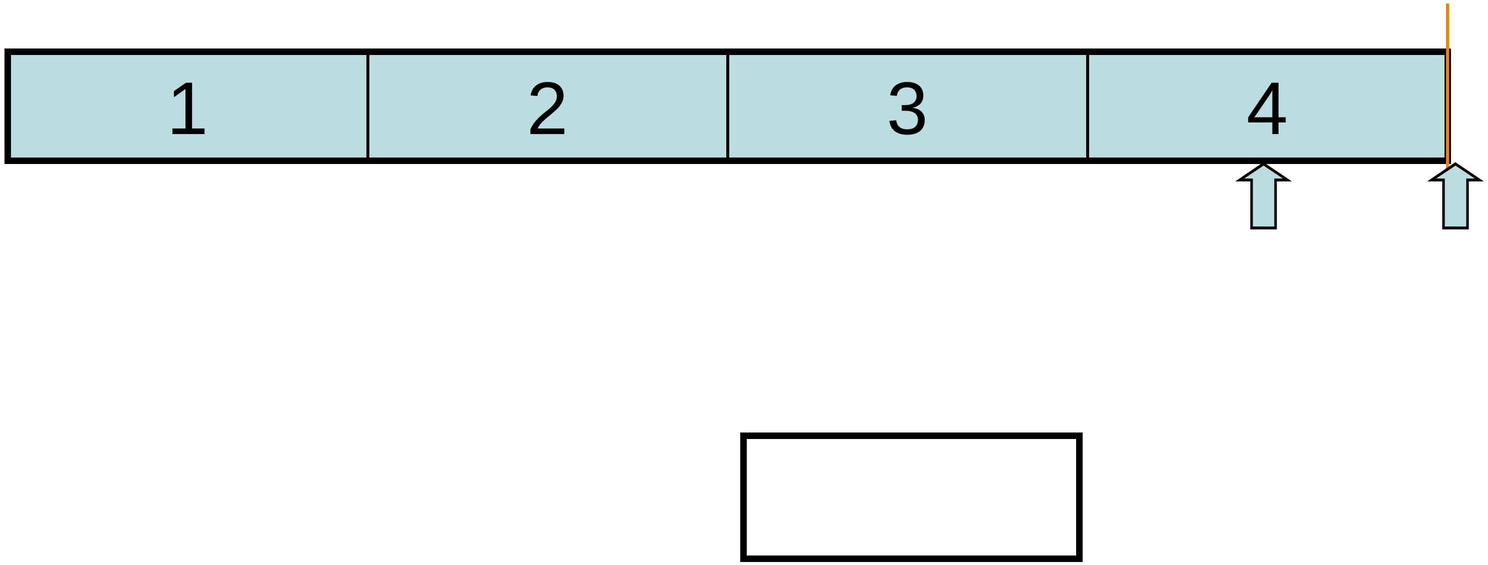
MergeSort2

- We design a new function called "inplaceMerge"



MergeSort2

- We design a new function called "inplaceMerge"



MergeSort2

- We design a new function called "inplaceMerge"
- Time complexity of inplaceMerge: $O(n^2)$

MergerSort2

- Replace the merge function in simple merge with **inplaceMerge**
- Time complexity:
  - $T(n) = 2T(n/2) + n^2$

    By Master theory, $T(n) = \Theta(n^2)$

MergerSort2

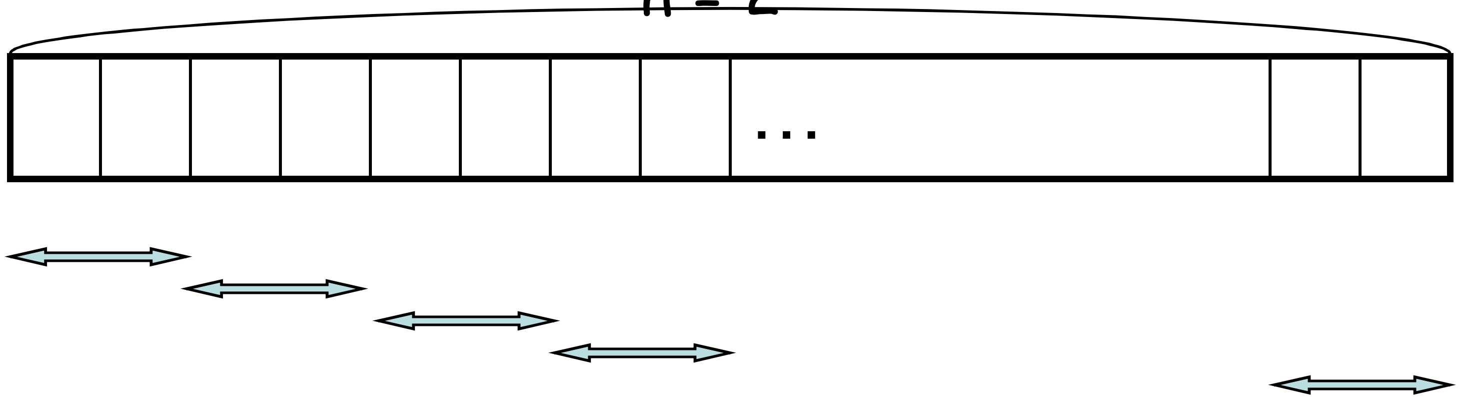
- Replace the merge function in simple merge with **inplaceMerge**

- Is the algorithm an in-place algorithm?
  - NO, because we recurrently call function
    - It require O(log n) function call

In-place MergeSort

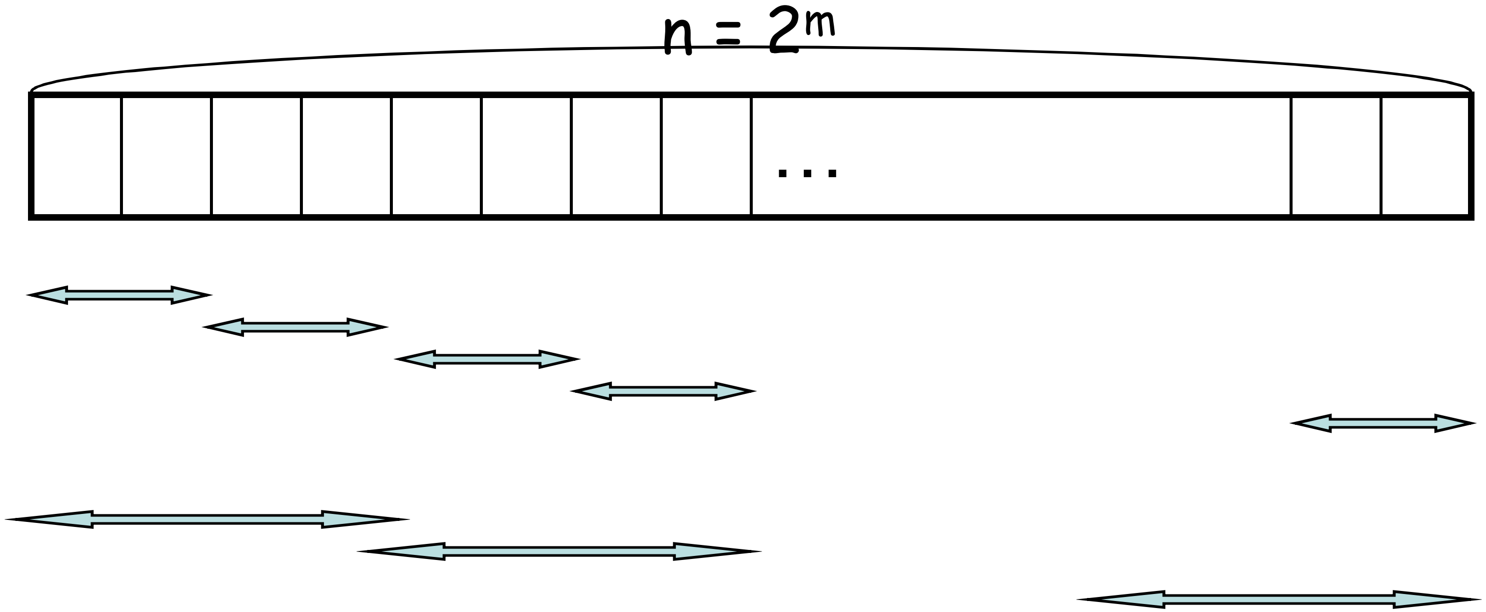
- So, we re-design the merge sort algorithm

$n = 2^m$



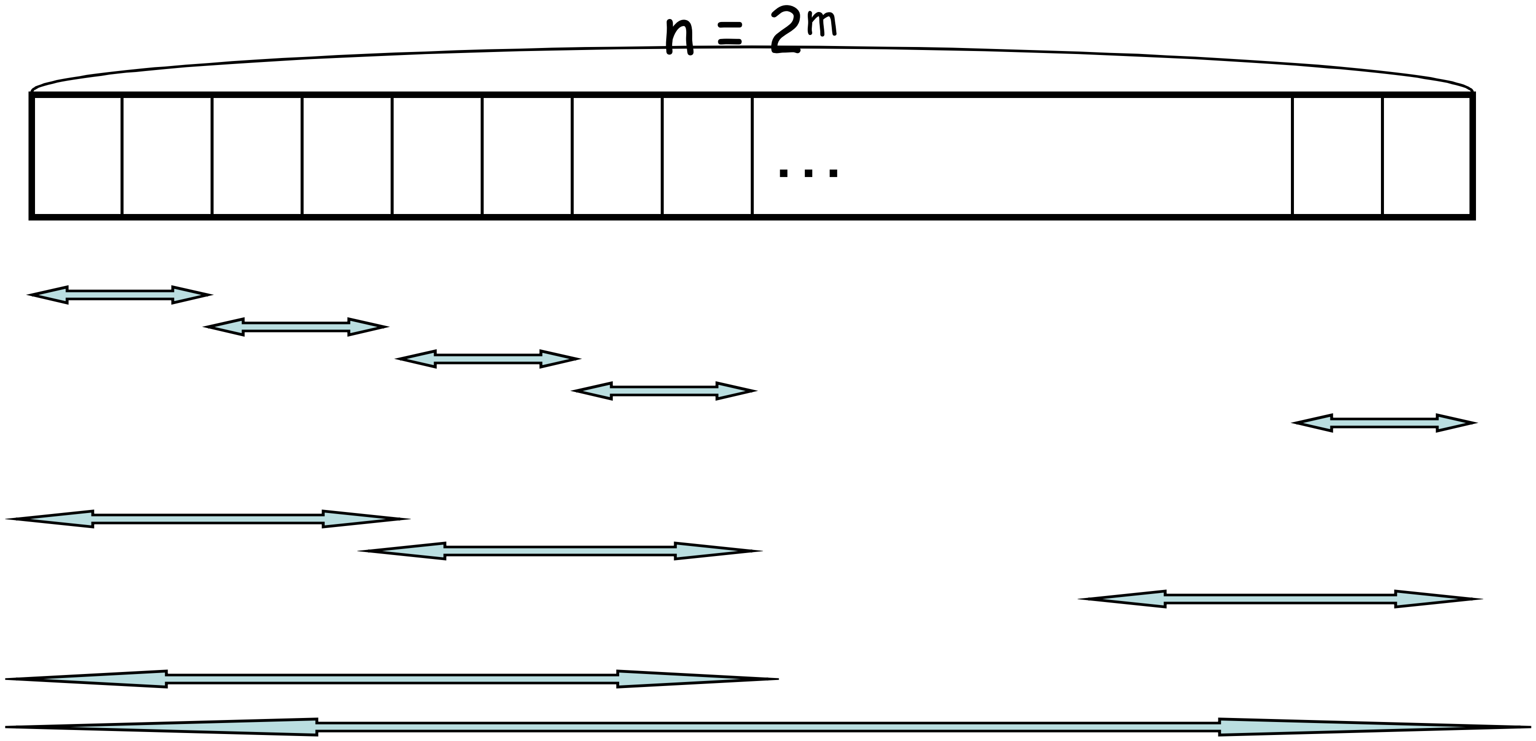
In-place MergeSort

- So, we re-design the merge sort algorithm



In-place MergeSort

- So, we re-design the merge sort algorithm



In-place Merge Sort

- Time complexity:

  $n/2 * O(2^2) + n/4 * O(4^2) + \ldots + 1 * O(n^2)$

  $= O(2n) + O(4n) + O(8n) + \ldots + O(n^2)$

  $= O(n^2)$

  ➔ still the same as MergeSort2, but avoid using O(log n) function calls

In-place Merge Sort

- Can we do better?
- In fact, there is an In-place Merge Sort algorithm that works faster, using only optimal $O(n \log n)$ time
  - The merging step is a bit complicated, so we do not introduce here …