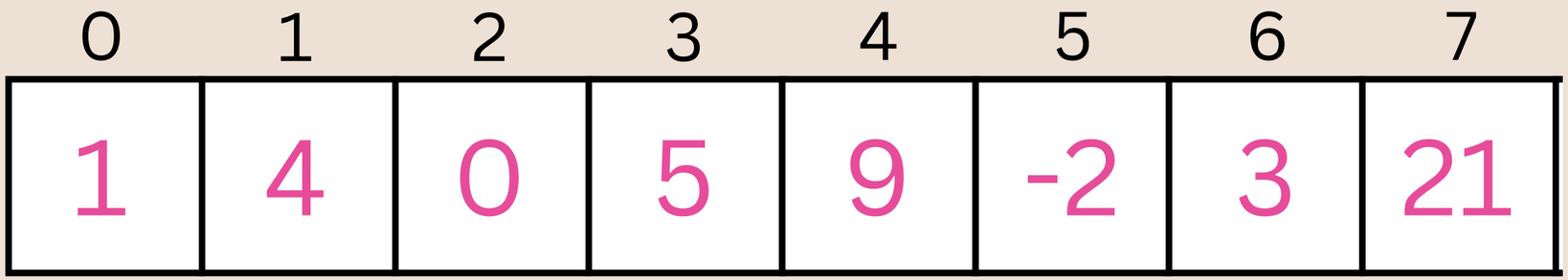
Topic 7.2: Revisiting Merge Sort

Merge Sort - Now with Recursion!

- We looked at merging two sorted arrays when we discussed Searching and Sorting
- We looked at a little not-so-animated break down of this sort
- Let’s actually take a look at the code now

Quick recap of Merge Sort's concept

- What you already know:
  - We've seen the merging step (go back to the sorting videos for this)
  - We've seen how the algorithm would split data into smaller parts.
    - *theoretically*
- *What we're adding today: The full recursive implementation!*
  - *this is actually going to be really easy :)*

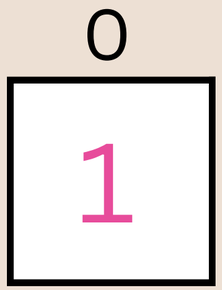


Why now? Why Recursion?

- Why Recursion?
  - What's the base case?

Why now? Why Recursion?

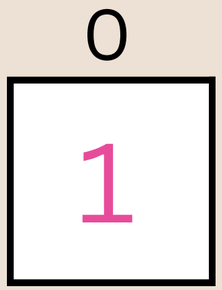
- Why Recursion?
  - What's the base case?



    - 1 element is always sorted!

Why now? Why Recursion?

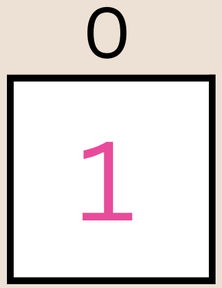
- Why Recursion?
  - What's the base case?



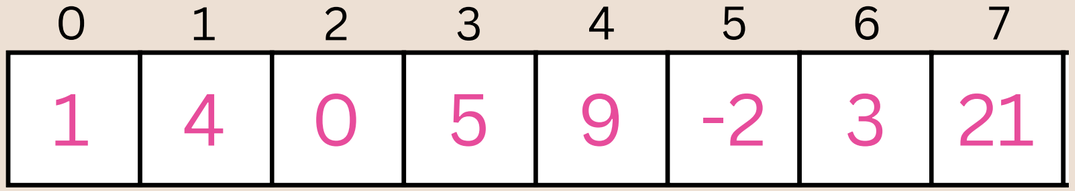
    - 1 element is always sorted!
- How do we break down the problem?

Why now? Why Recursion?

- Why Recursion?
  - What's the base case?

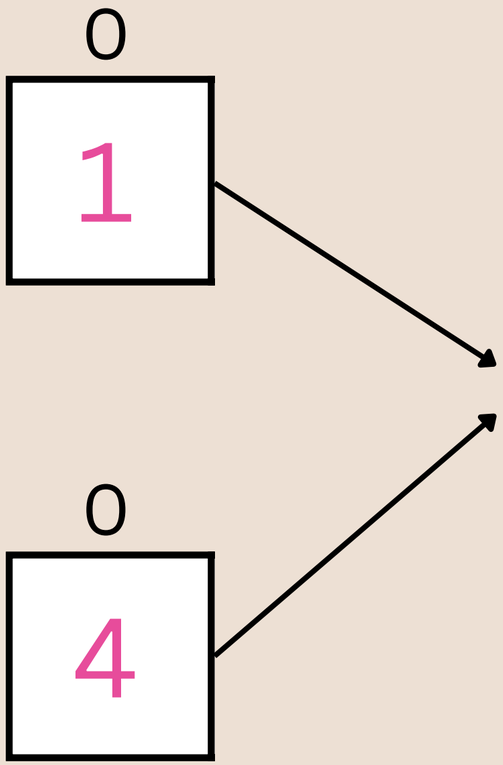


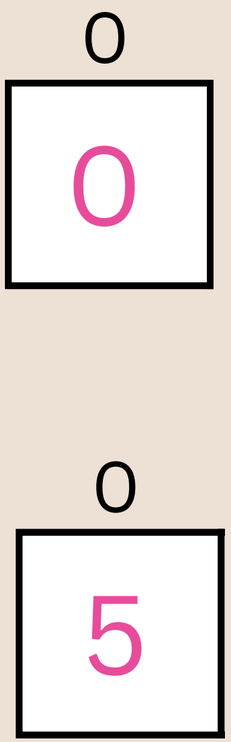
    - 1 element is always sorted!
- How do we break down the problem?
  - How do we get to 1 element?



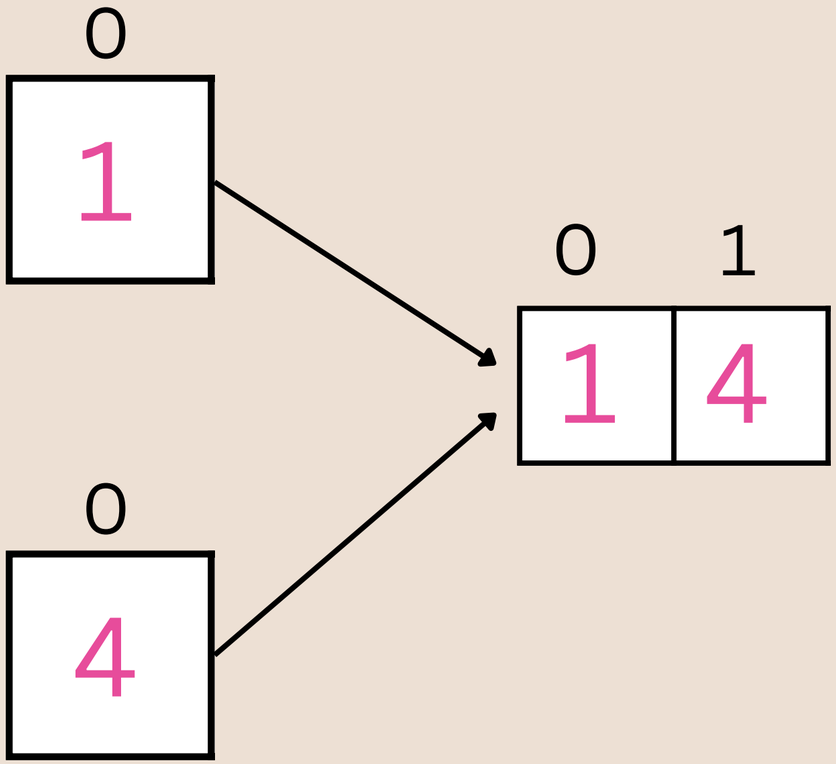
Revisiting the Merge Step

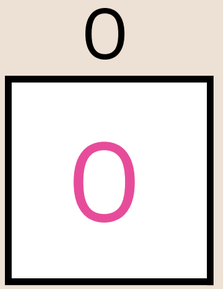
0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21

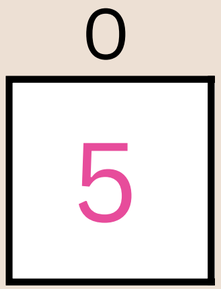




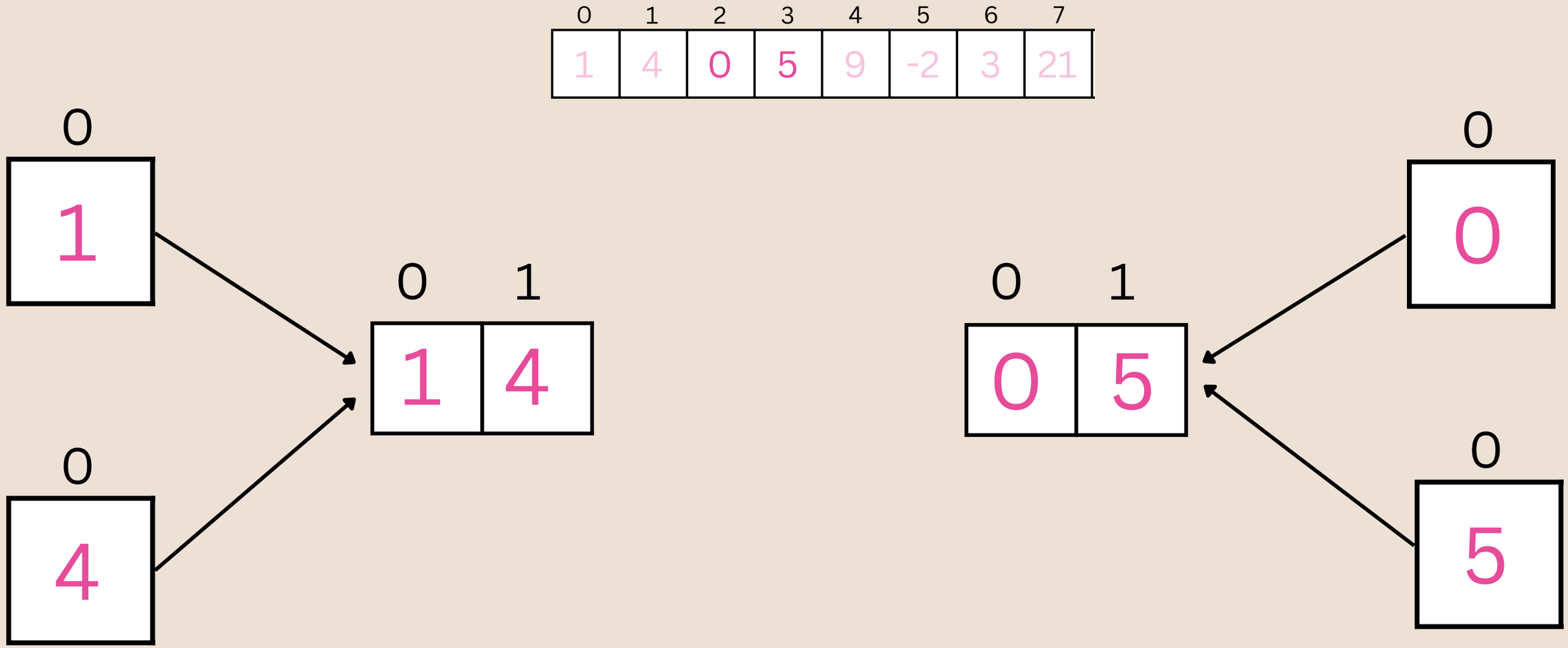
Revisiting the Merge Step



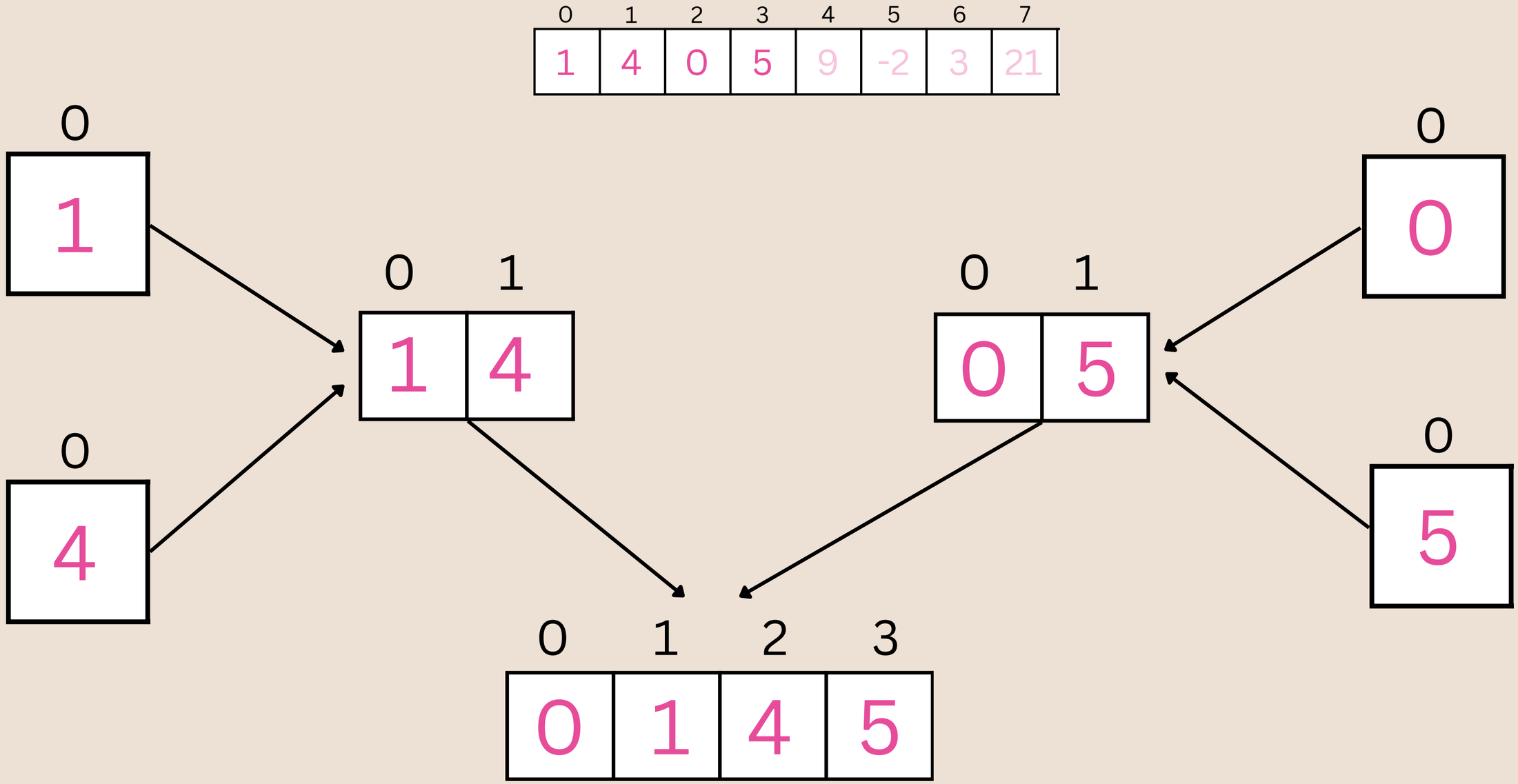




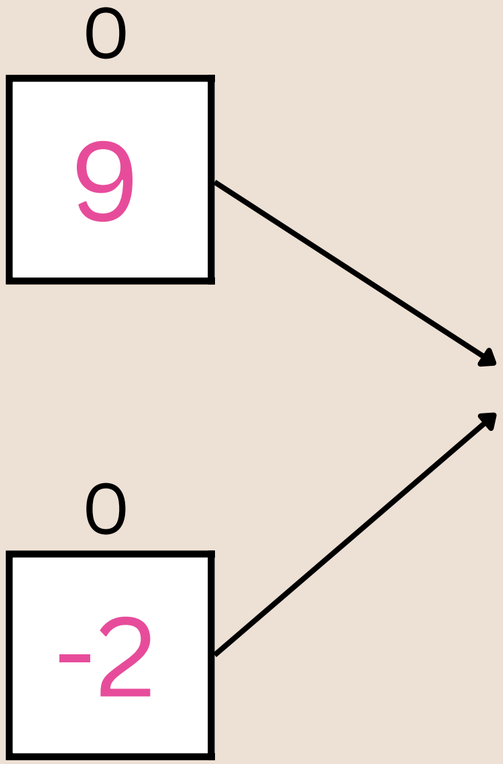
Revisiting the Merge Step

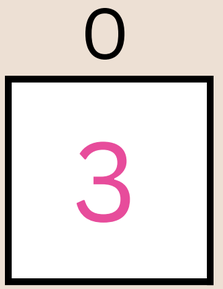


Revisiting the Merge Step

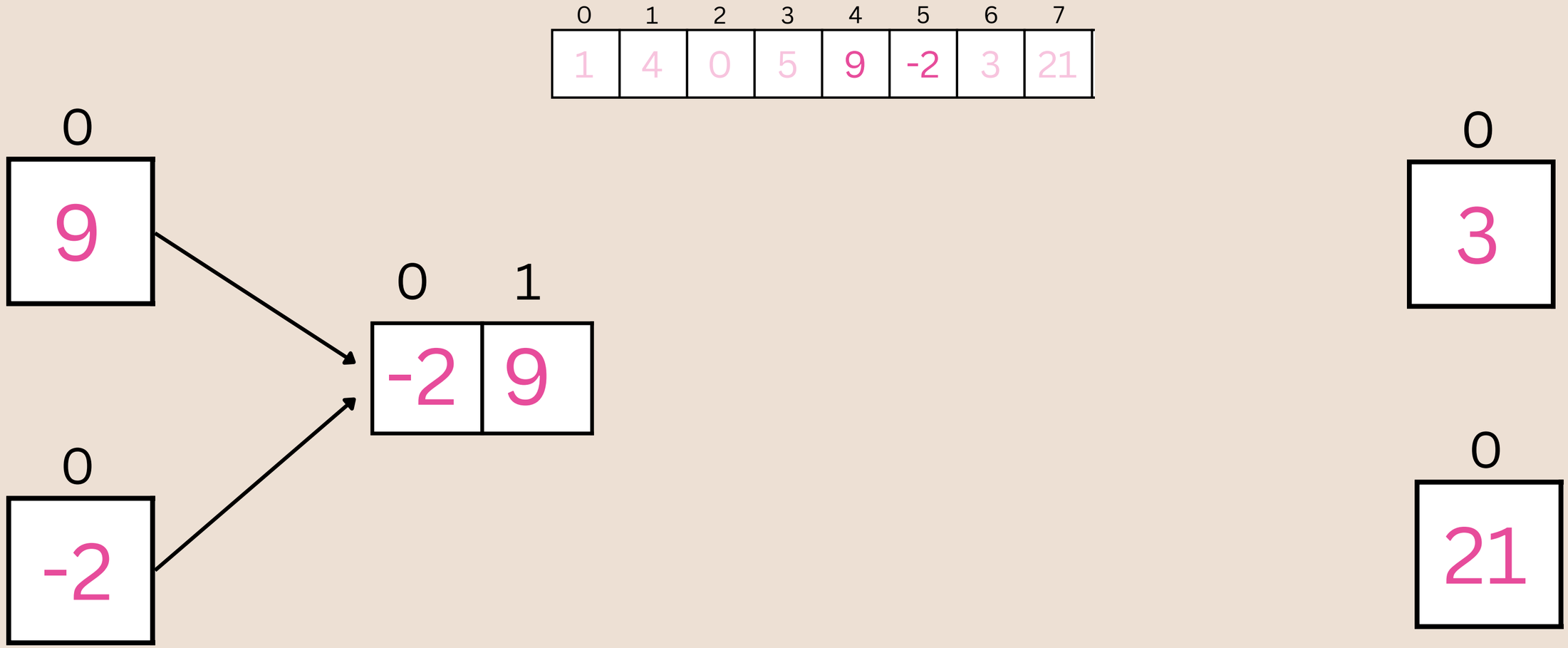


Revisiting the Merge Step



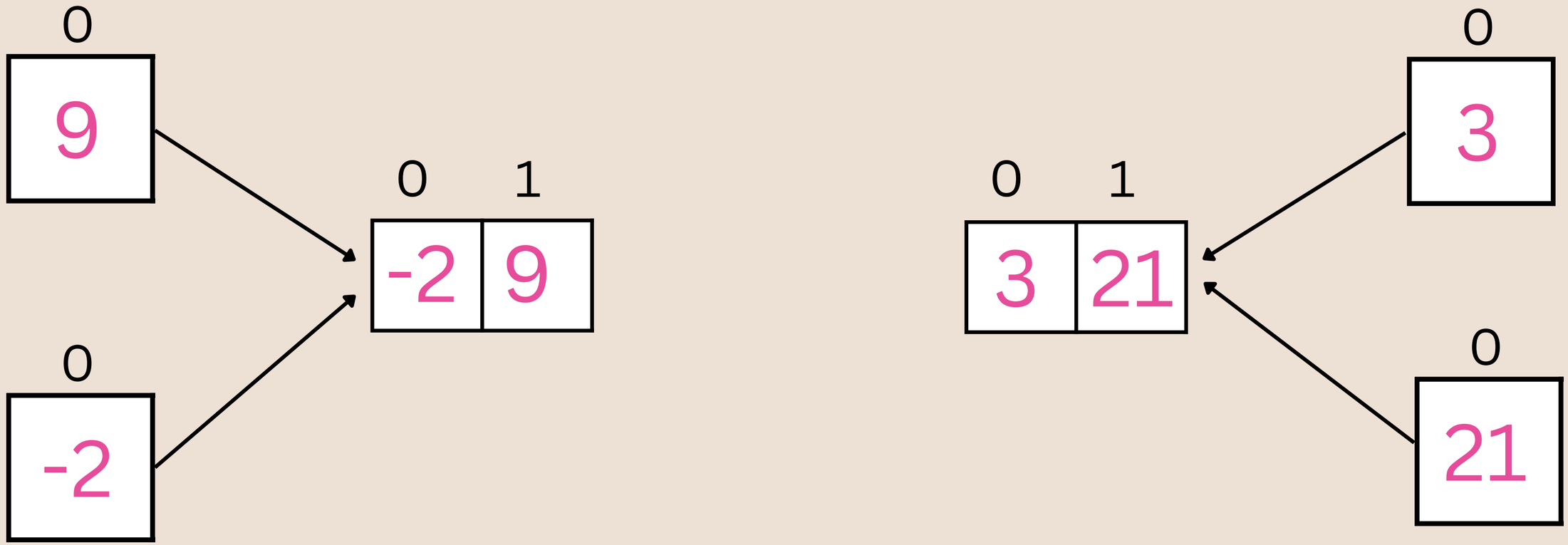


Revisiting the Merge Step

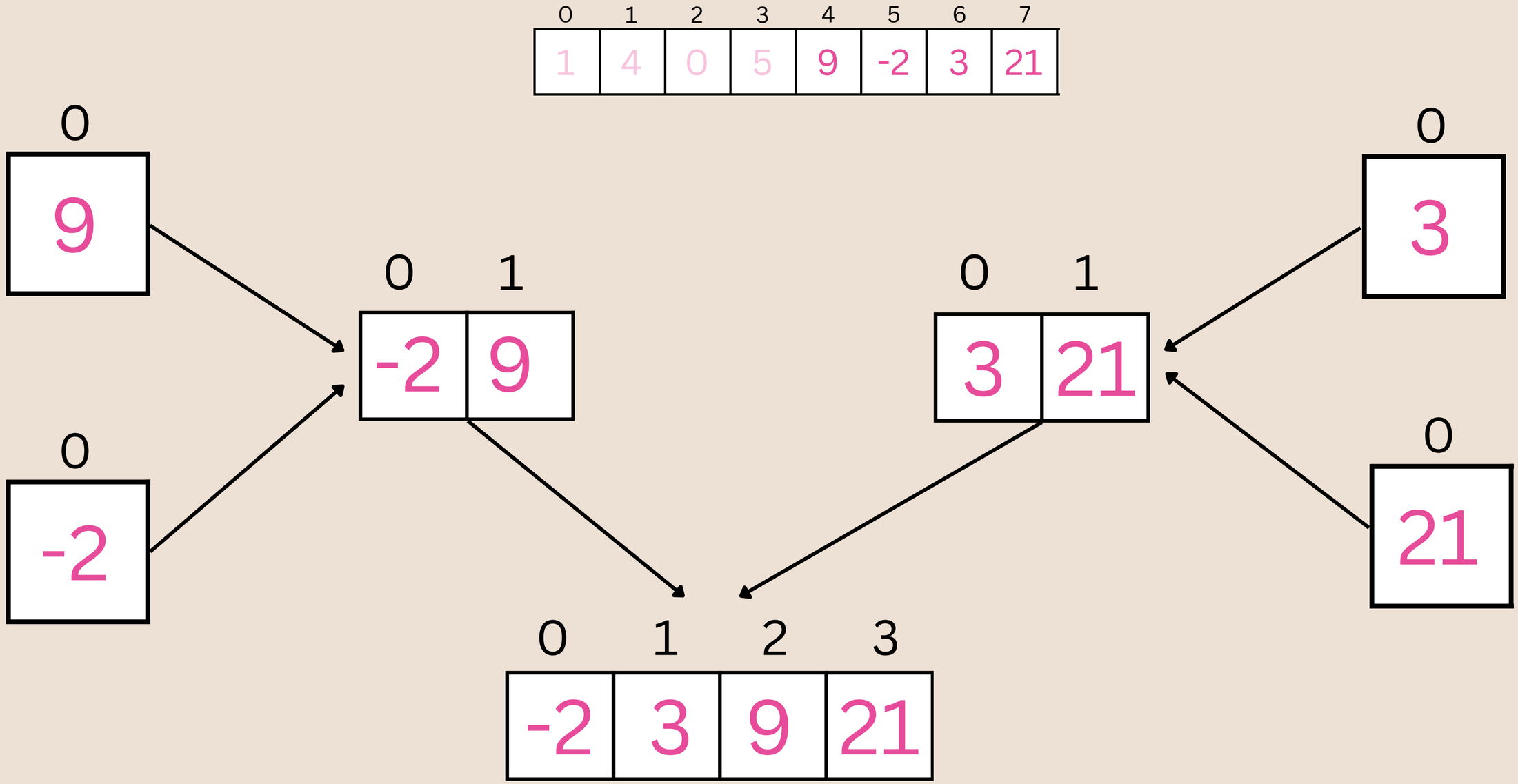


Revisiting the Merge Step

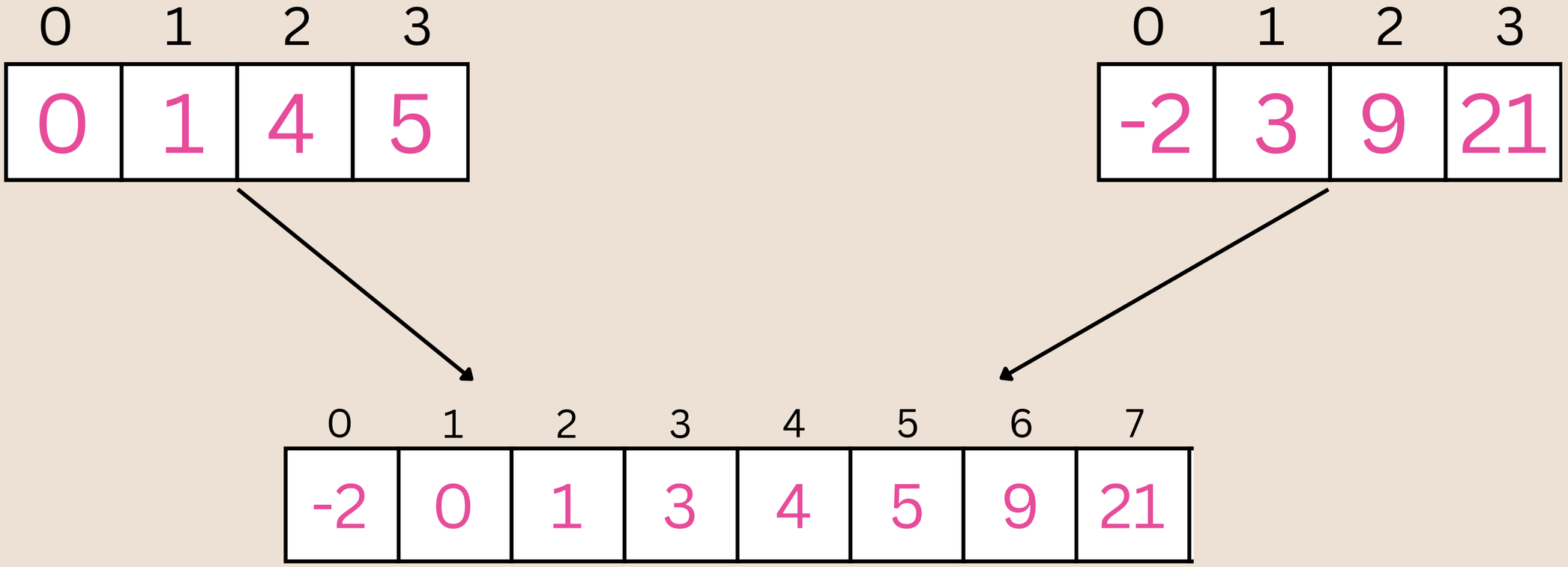
0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



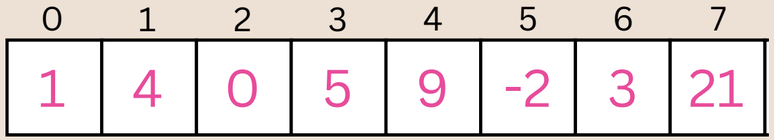
Revisiting the Merge Step



Revisiting the Merge Step



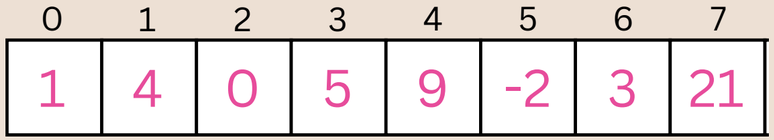
What did the call stack look like?



split([1,4,0,5])

split([1,4,0,5,9,-2,3,21])

What did the call stack look like?

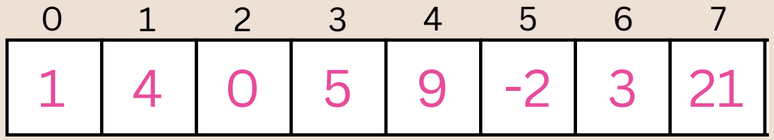


split([1,4])

split([1,4,0,5])

split([1,4,0,5,9,-2,3,21])

What did the call stack look like?



split([1])

split([1,4])

split([1,4,0,5])

split([1,4,0,5,9,-2,3,21])

What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21

BASE = return[1]

split([1])

split([1,4])

split([1,4,0,5])

split([1,4,0,5,9,-2,3,21])

What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21

split([4])

split([1,4])

split([1,4,0,5])

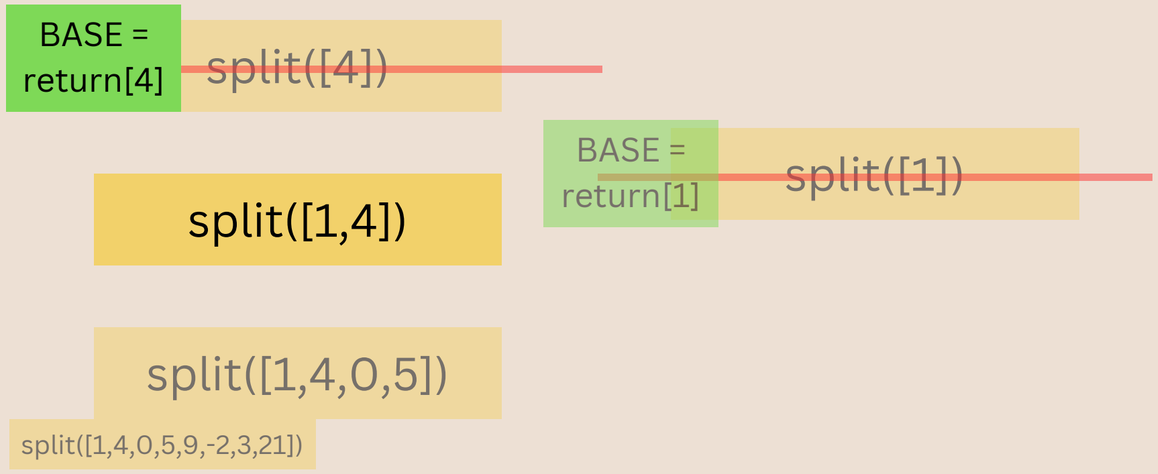
split([1,4,0,5,9,-2,3,21])

BASE = return[1]

split([1])

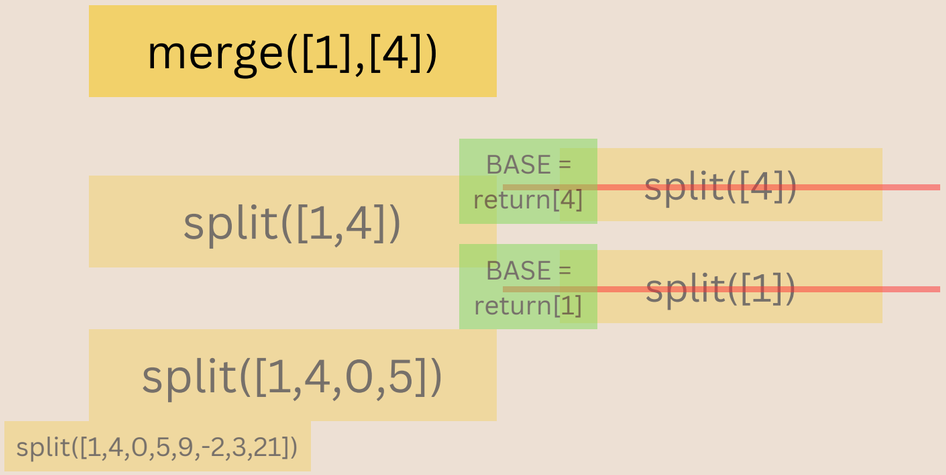
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



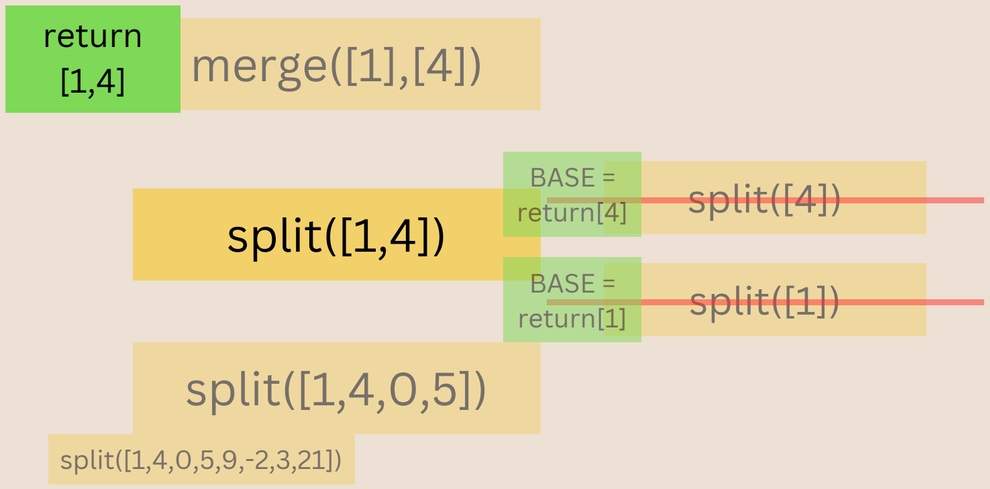
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



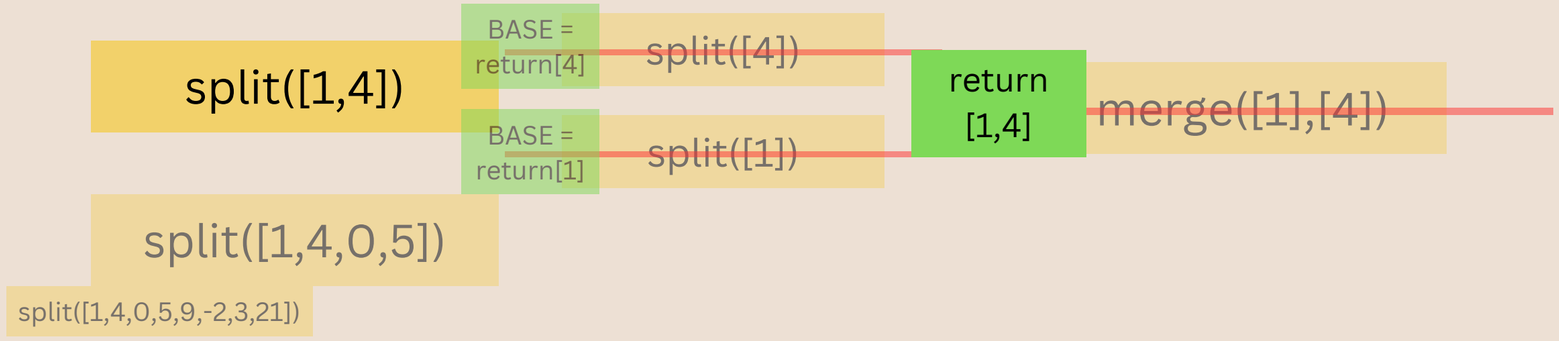
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



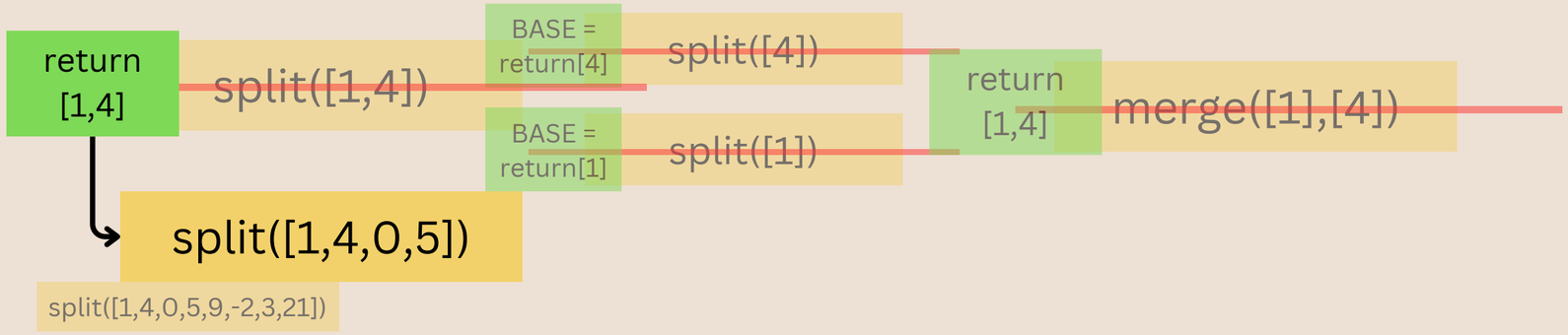
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21

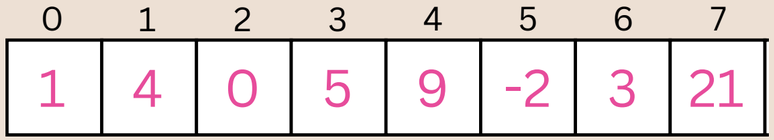


What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



What did the call stack look like?

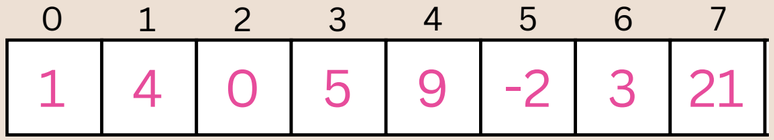


split([1,4,0,5])

rightSplit = [1,4]

split([1,4,0,5,9,-2,3,21])

What did the call stack look like?



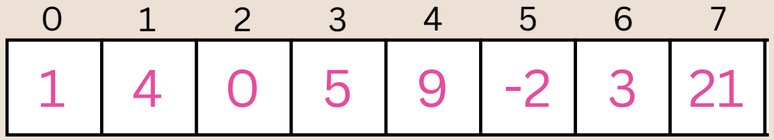
split([0,5])

split([1,4,0,5])

rightSplit = [1,4]

split([1,4,0,5,9,-2,3,21])

What did the call stack look like?



split([0])

split([0,5])

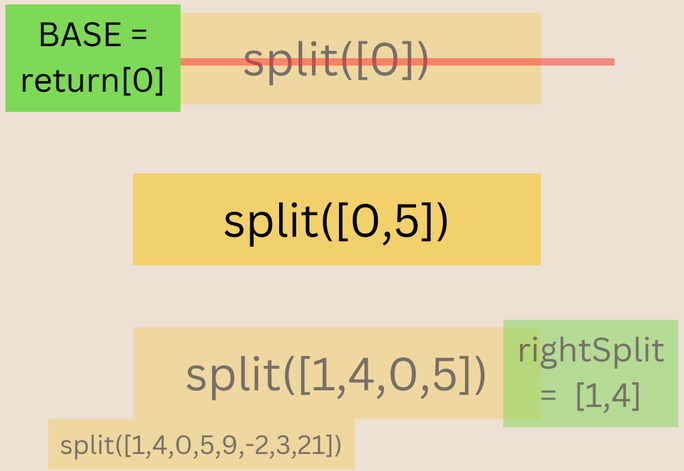
split([1,4,0,5])

rightSplit = [1,4]

split([1,4,0,5,9,-2,3,21])

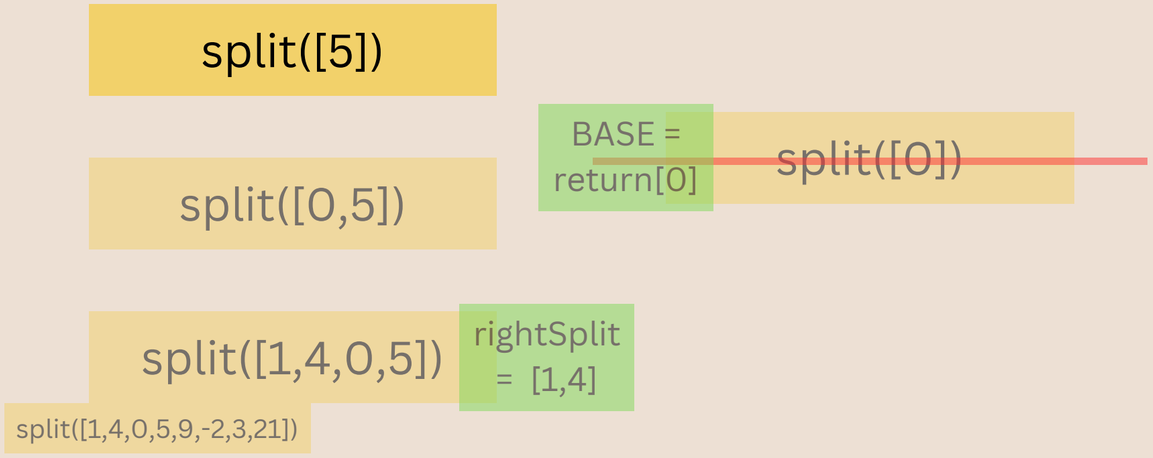
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



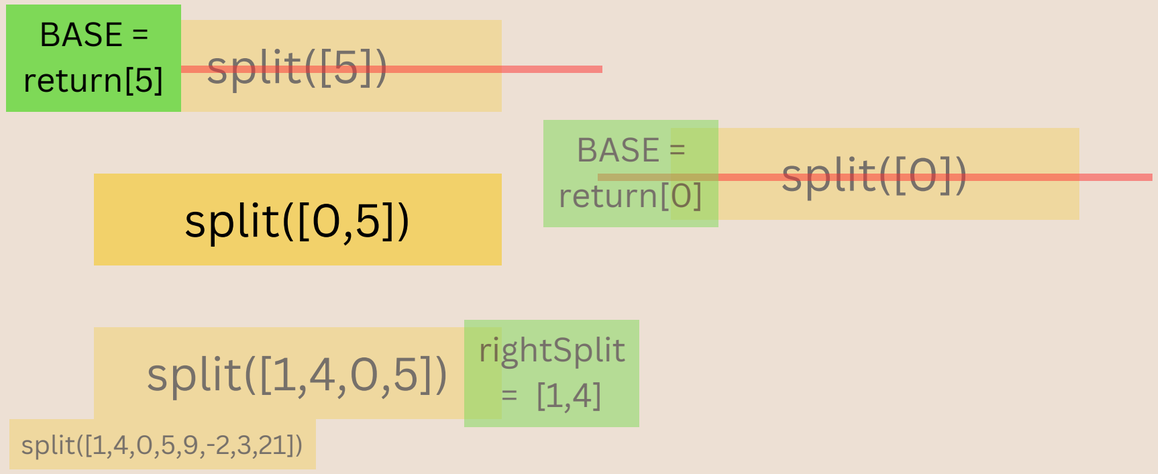
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



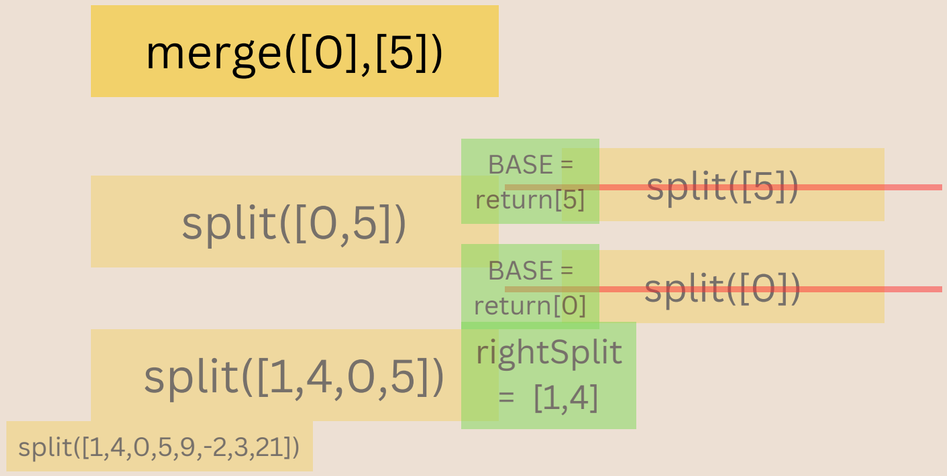
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



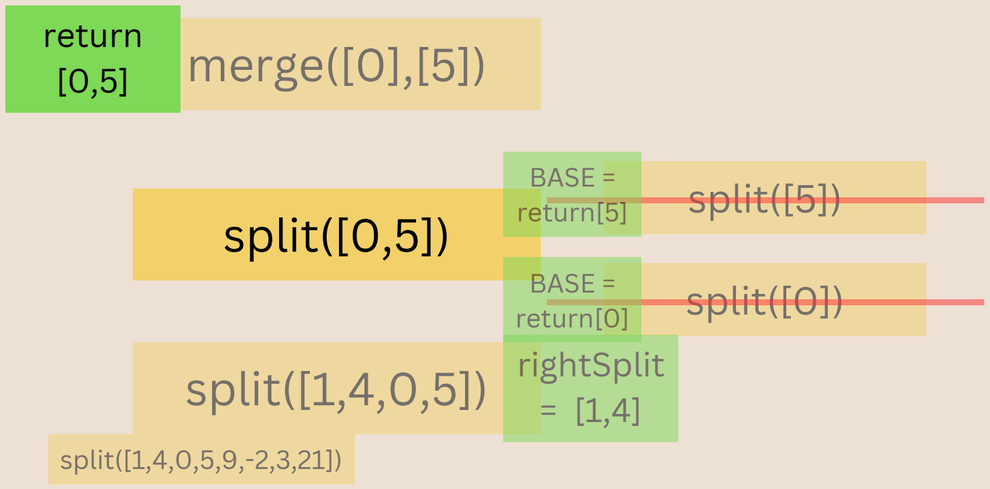
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



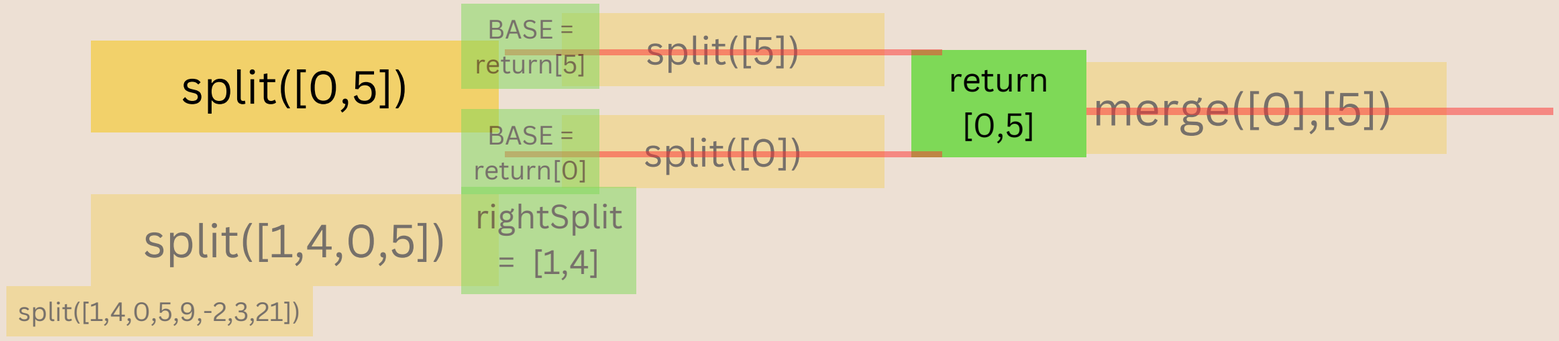
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



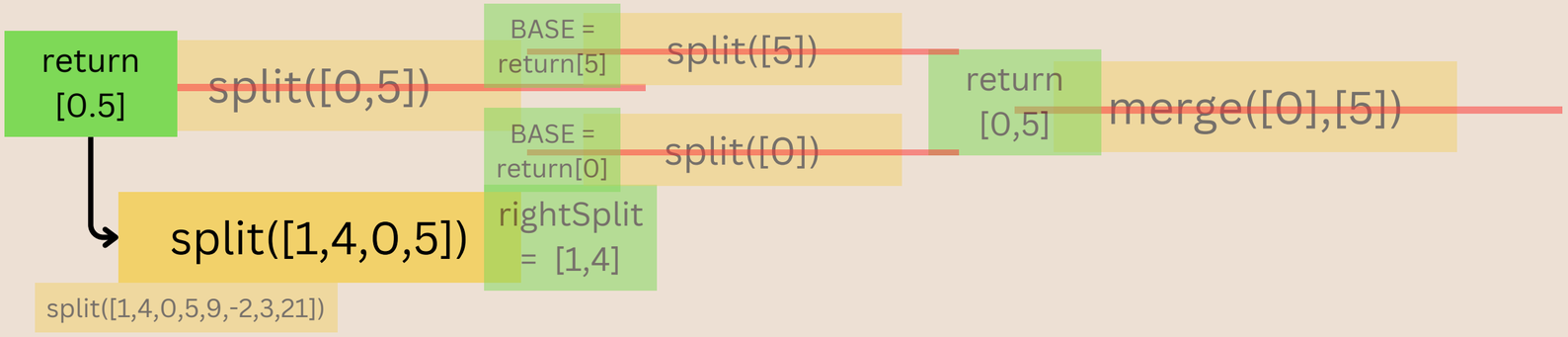
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21

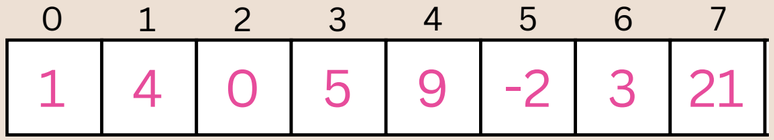


What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



What did the call stack look like?

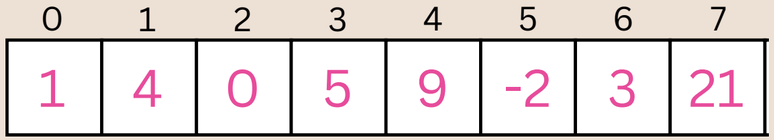


split([1,4,0,5])

rightSplit = [1,4]
leftSplit = [0,5]

split([1,4,0,5,9,-2,3,21])

What did the call stack look like?



merge([1,4],[0,5])

split([1,4,0,5])

rightSplit = [1,4]
leftSplit = [0,5]

split([1,4,0,5,9,-2,3,21])

What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21

return [0,1,4,5]

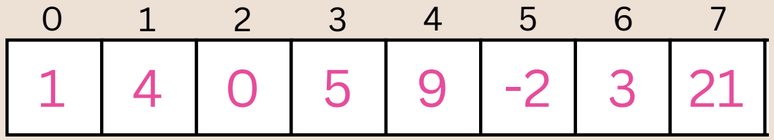
merge([1,4],[0,5])

split([1,4,0,5])

rightSplit = [1,4]
leftSplit = [0,5]

split([1,4,0,5,9,-2,3,21])

What did the call stack look like?



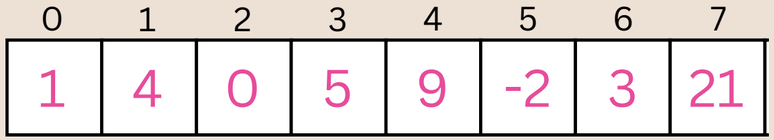
split([9,-2,3,21]

split([1,4,0,5,9,-2,3,21])

rightSplit = [0,1,4,5]

And then we do the other side...

What did the call stack look like?



merge([0,1,4,5],[-2,3,9,21])

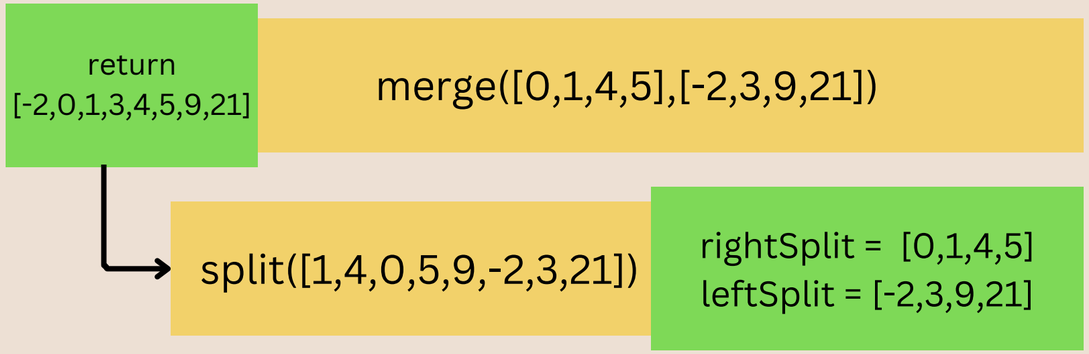
split([1,4,0,5,9,-2,3,21])

rightSplit = [0,1,4,5]
leftSplit = [-2,3,9,21]

and then we merge it all together

What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21



aaannnnddddd....

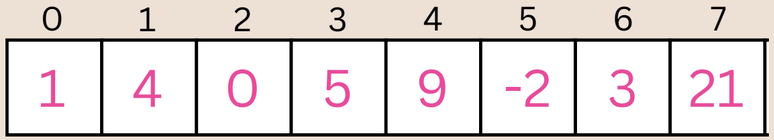
What did the call stack look like?

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21

split([1,4,0,5,9,-2,3,21])

result = [-2,0,1,3,4,5,9,21]
// this is returned to main or wherever

PSEUDO CODE?




```
function mergeSort(arr):
    if length of arr is 1:
        return arr
    mid = find middle of arr
    left = mergeSort(left half)
    right = mergeSort(right half)
    return merge(left, right)
```

This function is *recursive*

So what sorts do we have now?

We know:

- insertion sort
- selection sort
- merge sort

But wait! There's more!!!

- We will see one more sort in just a few weeks when we talk about measuring how efficient our code really is

Pause and Practice

0	1	2	3	4	5	6	7
1	4	0	5	9	-2	3	21

- Write the code! Implement Merge Sort and sort this array:
- How many recursive calls did your program make?
  - Count the number of times mergeSort is called.
- *Challenge*: Modify your code to count the total number of recursive calls automatically.
- How many comparisons occur during the merge step?
  - Count the number of element comparisons made only in the merging process.
- *Challenge*: Can you generalize this?
  - If an array has m elements, what is the maximum number of comparisons we might make during merging?
- How many comparisons would we make using Selection Sort instead?
  - Compare your result with Selection Sort's total comparisons.