# CS 103000
# Prof. Madeline Blount

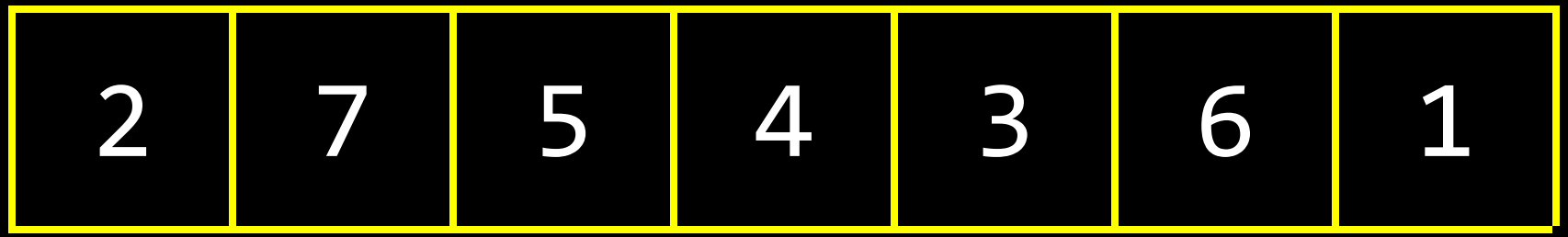
## Week 14:
## ALGORITHMS part 2

## attendance link:

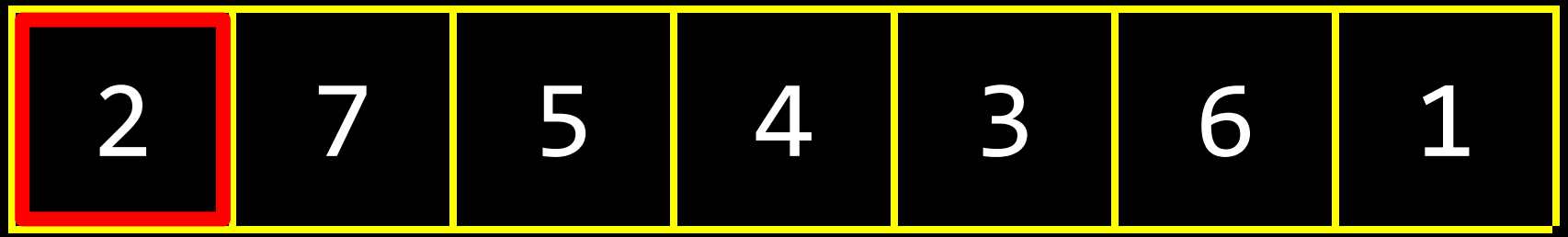
https://cs103-proton.glitch.me/

*Dall-E 2: cats learning C++ in the forest on '90's technology*

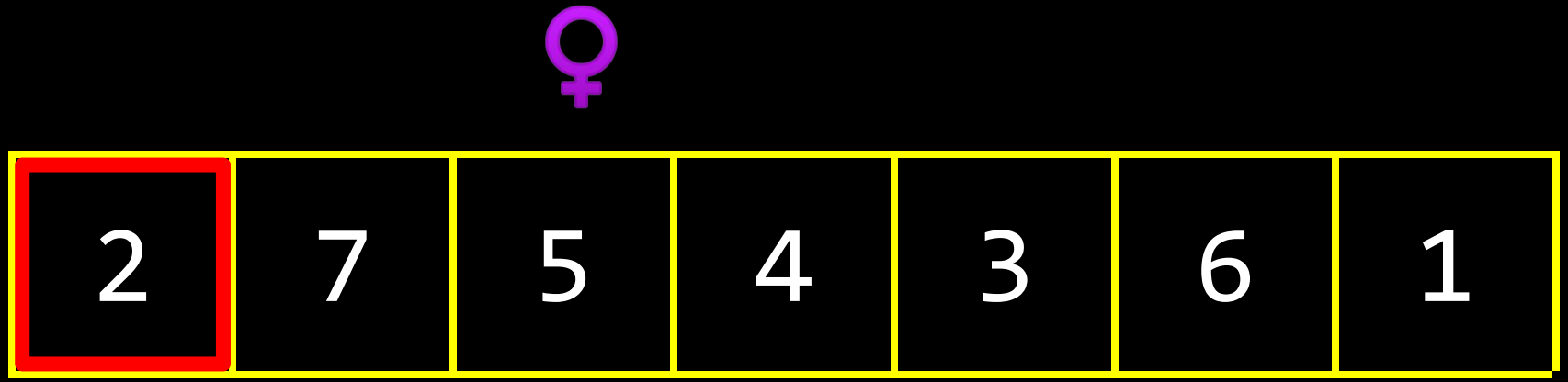
# SELECTION SORT



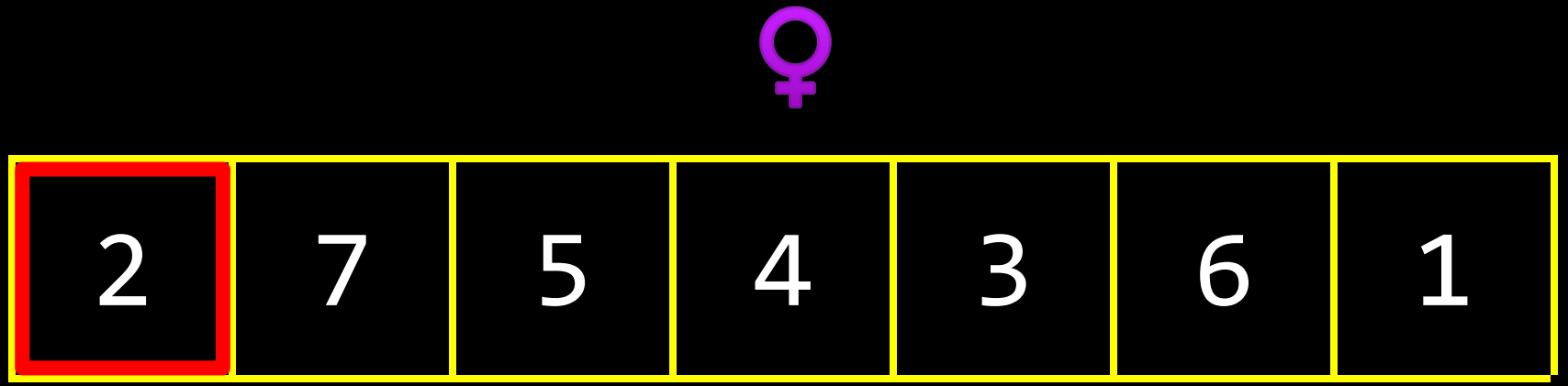
# SELECTION SORT

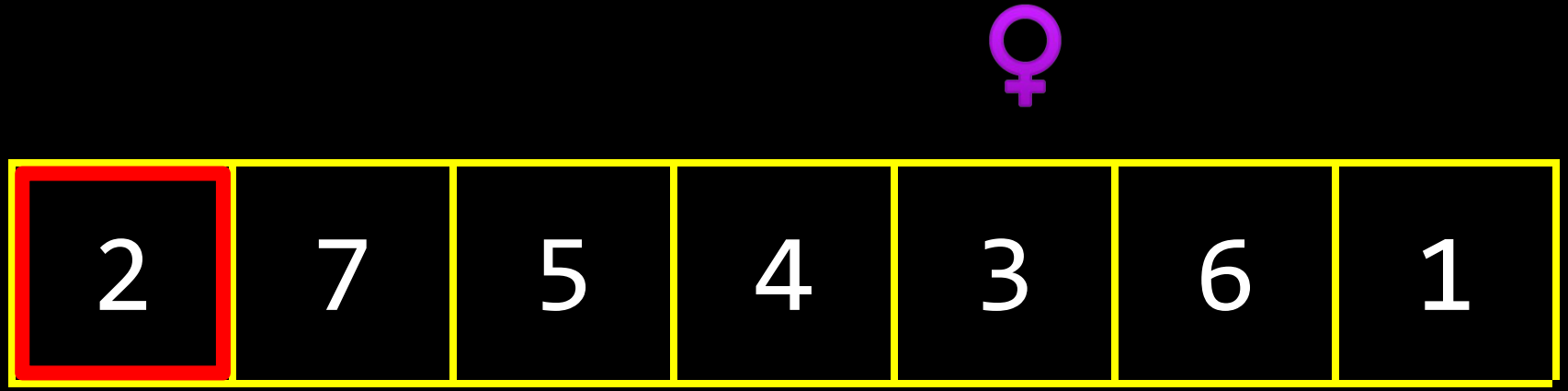


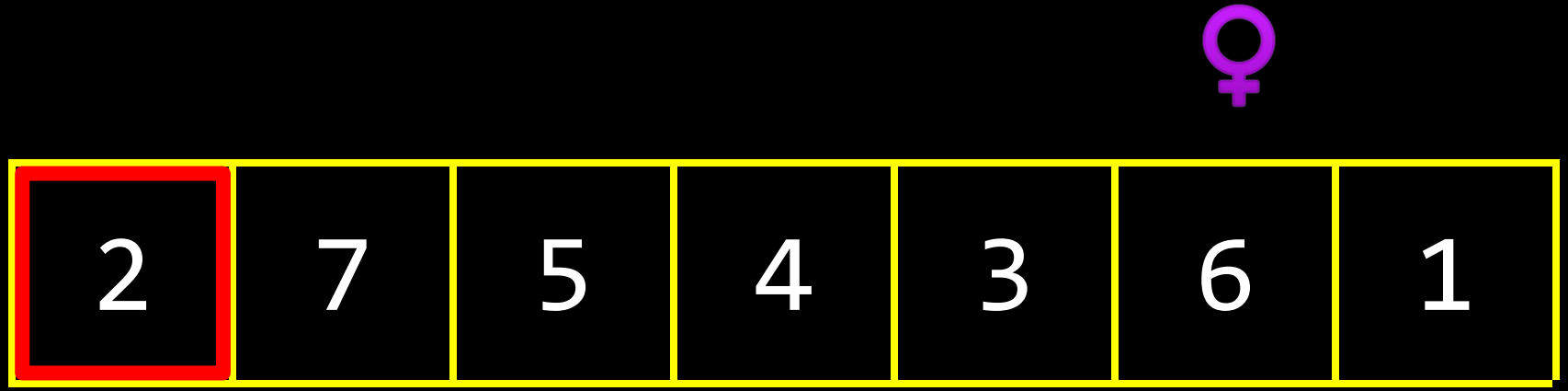
# SELECTION SORT

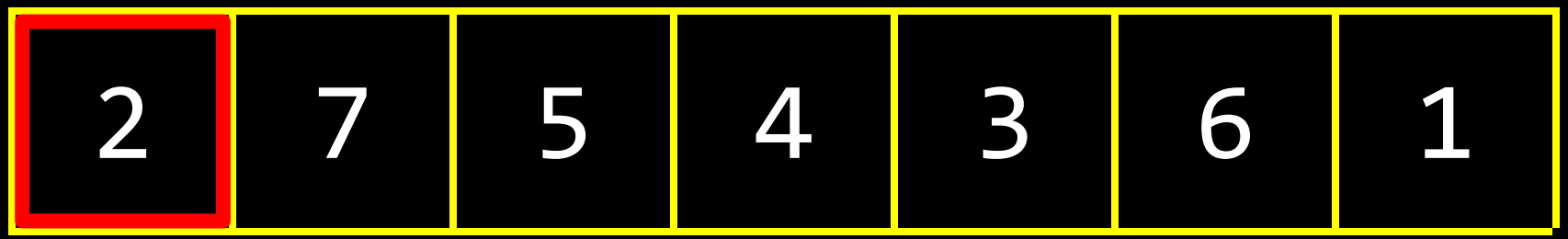


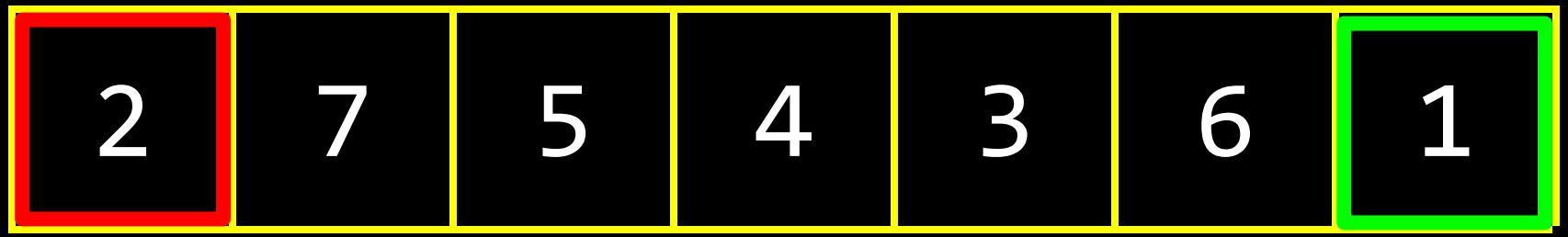

Check: smallest number in rest of the array …

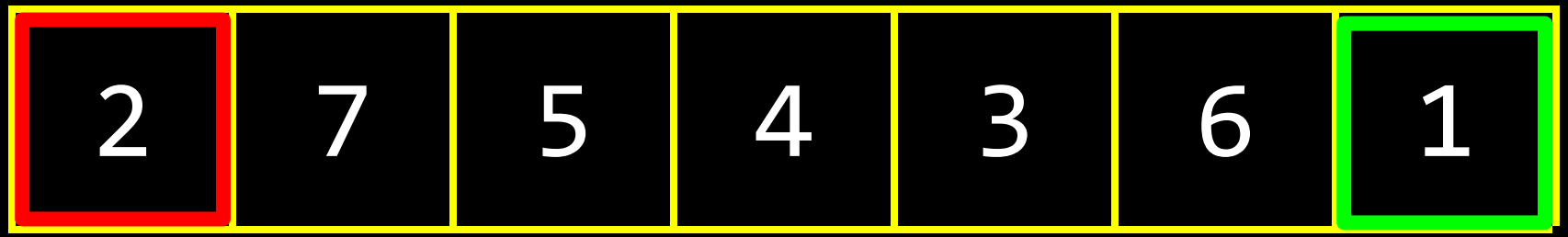

2
7
5
4
3
6
1


2
7
5
4
3
6
1

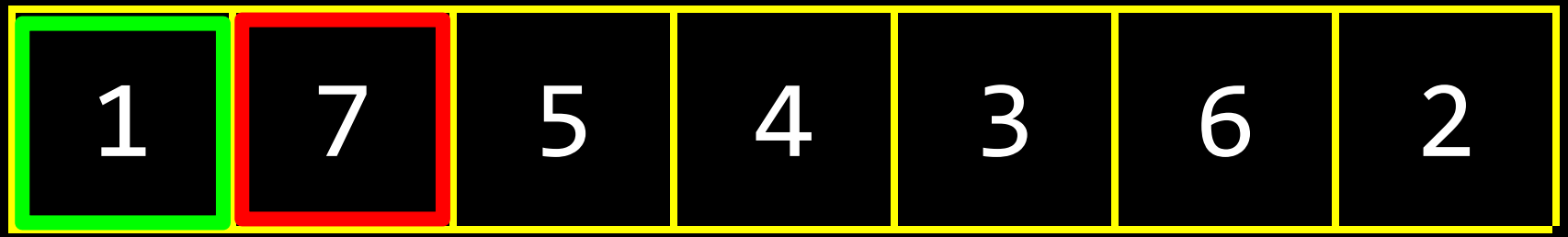

2
7
5
4
3
6
1


2
7
5
4
3
6
1

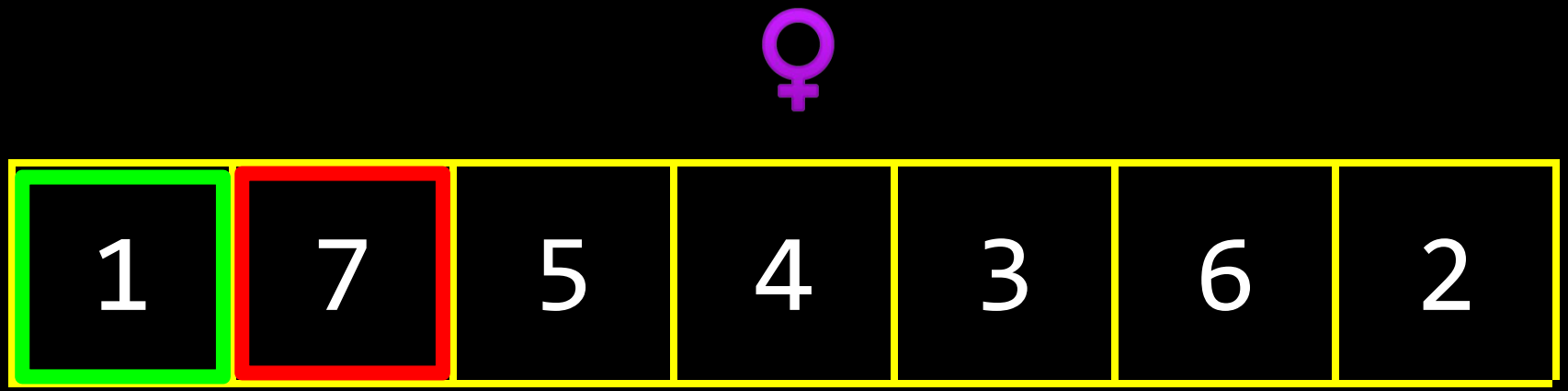

2
7
5
4
3
6
1



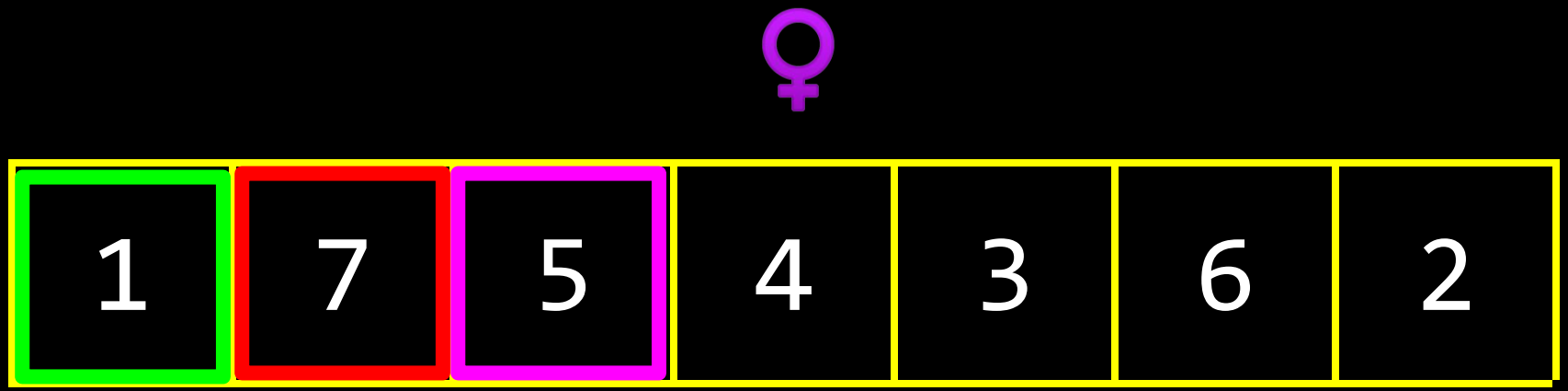

SWAP!



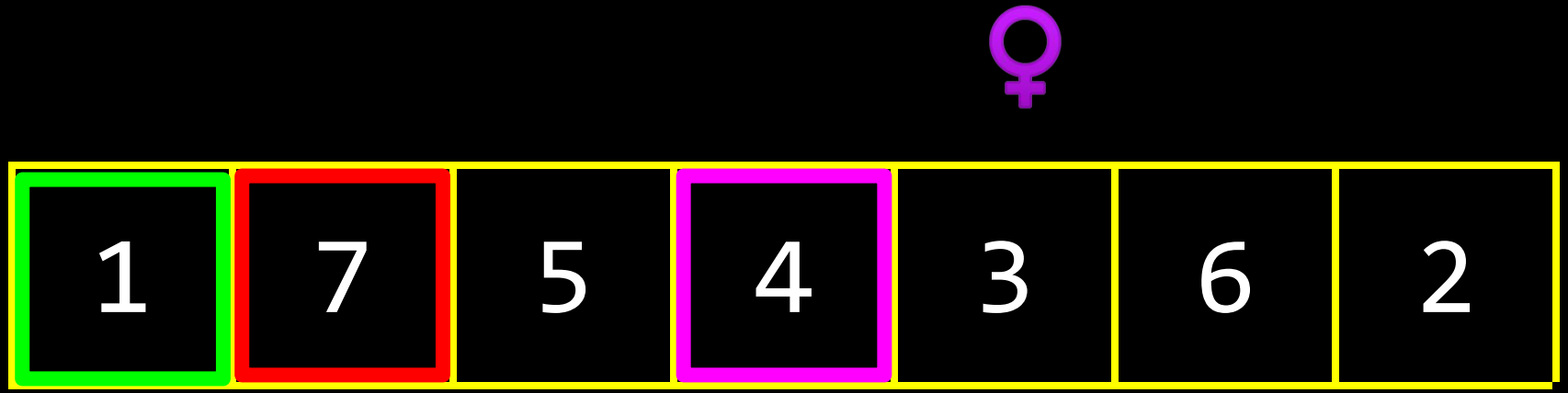

Start again…



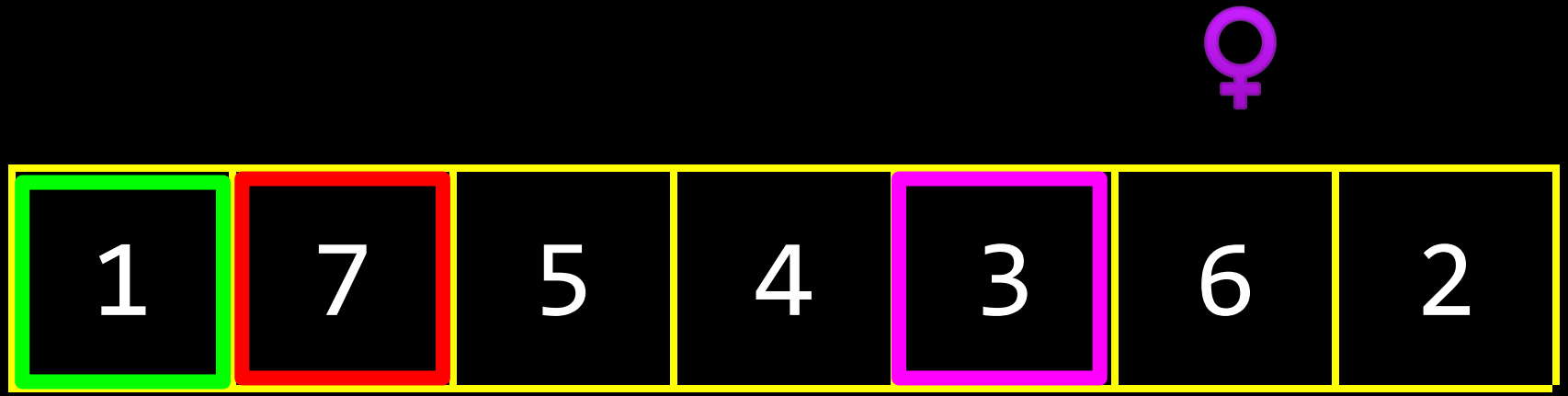

Check: smallest number in rest of the array …



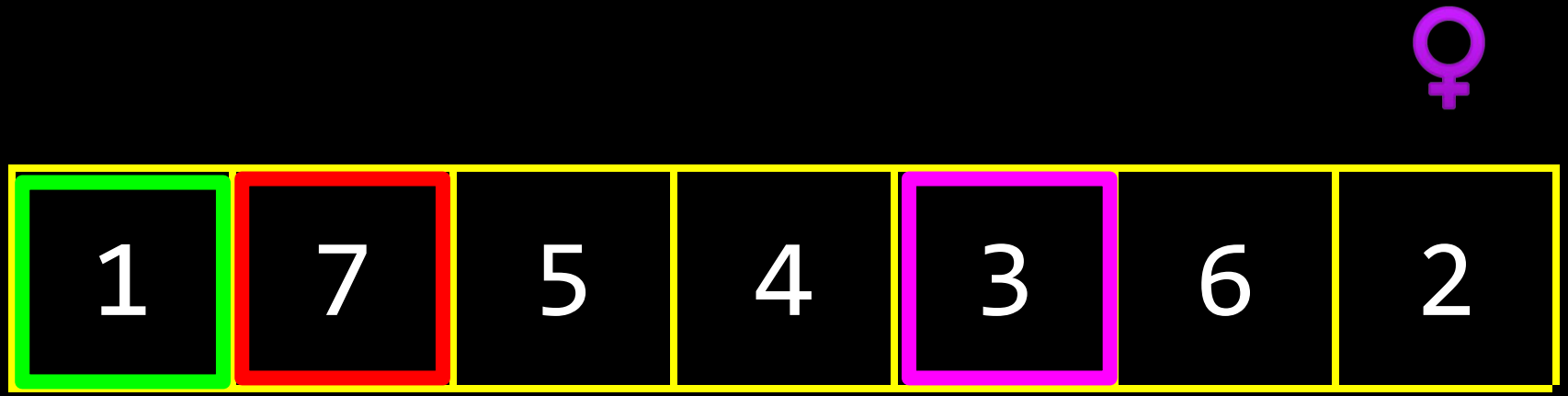

SO FAR: 5



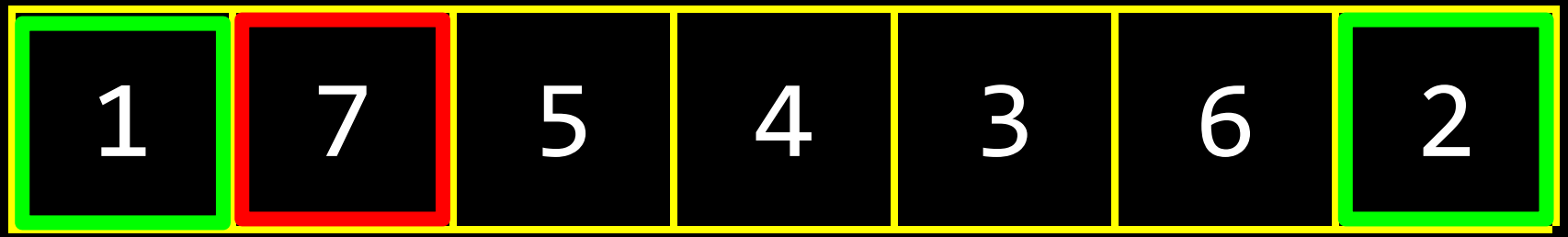

SO FAR: 4



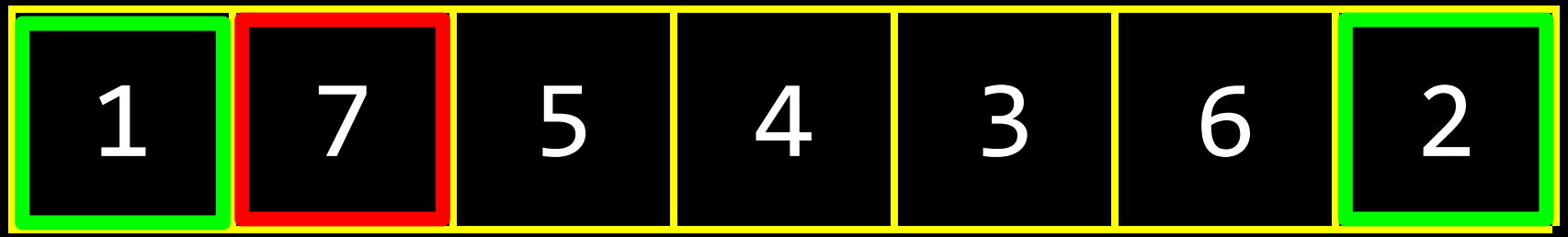

SO FAR: 3




SO FAR: 3



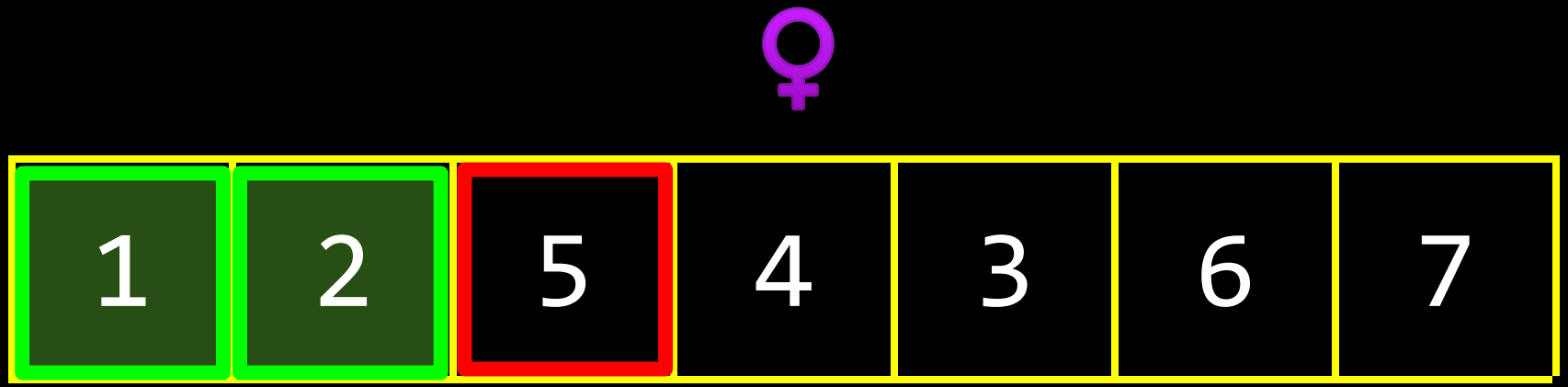

SMALLEST: 2



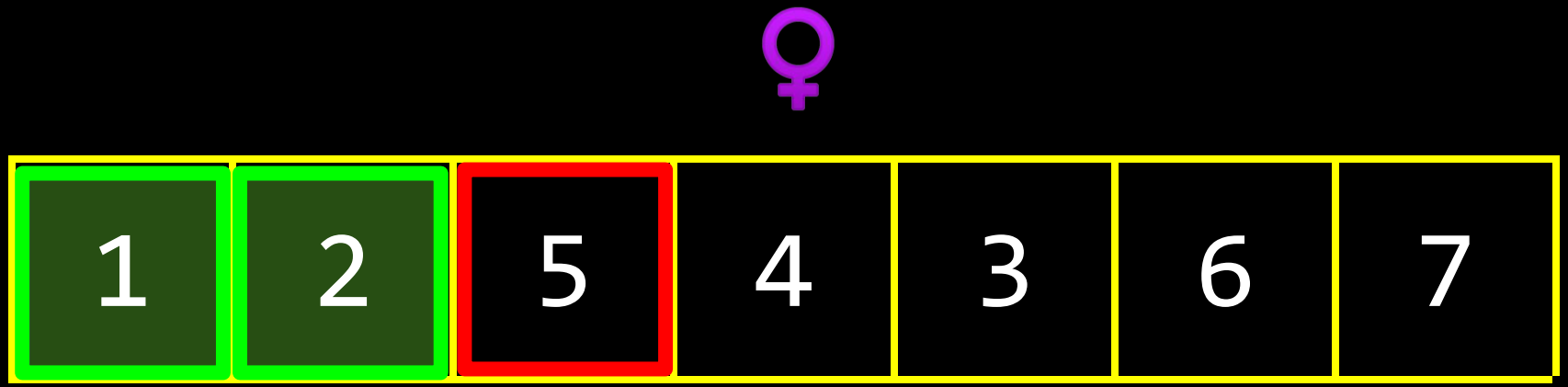

SWAP!


1
2
5
4
3
6
7

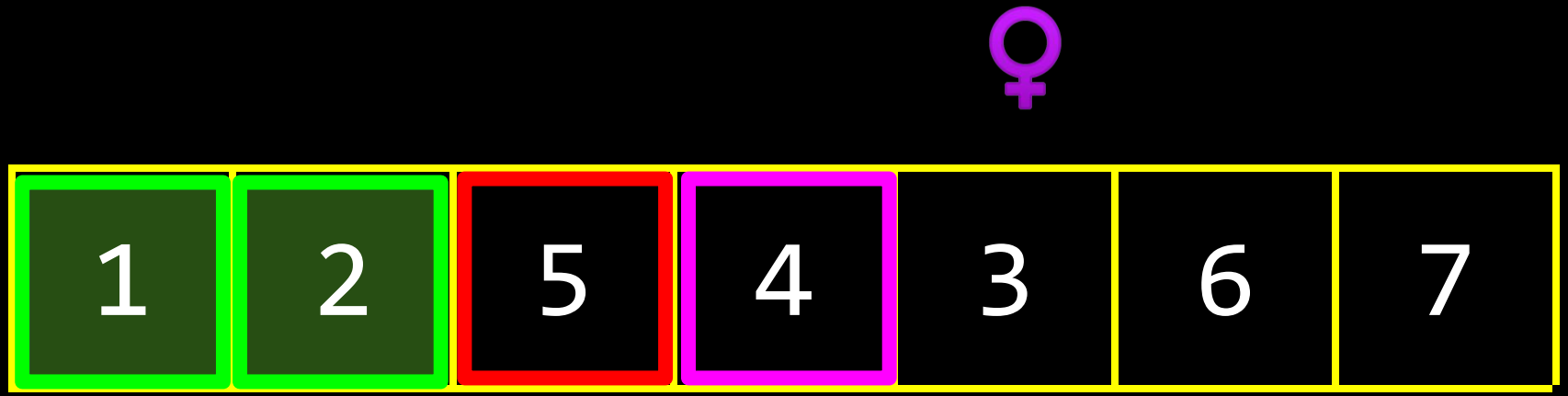

Start again…



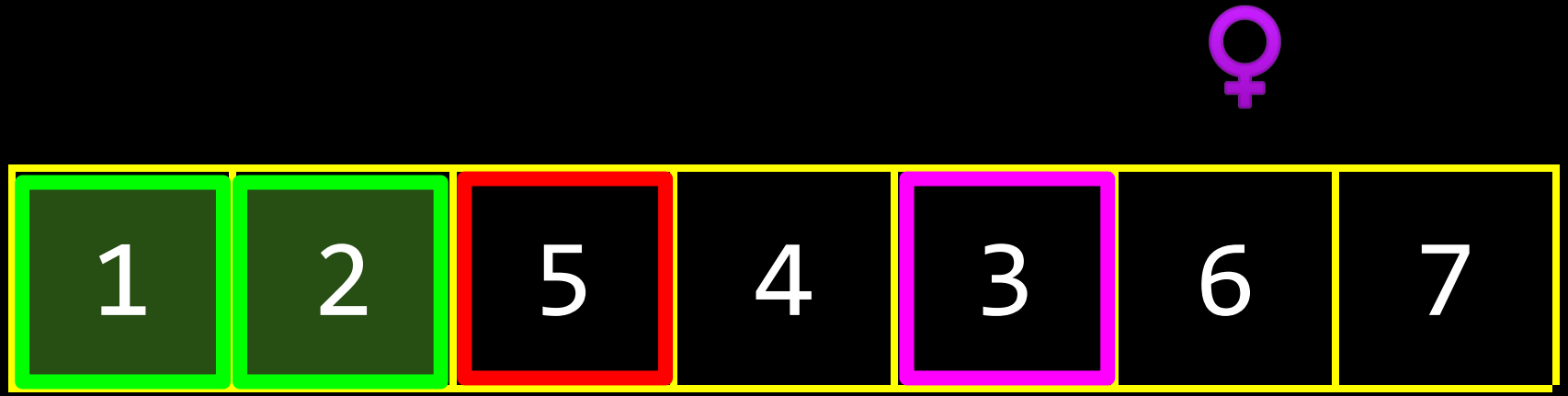

Check: smallest number in rest of the array …



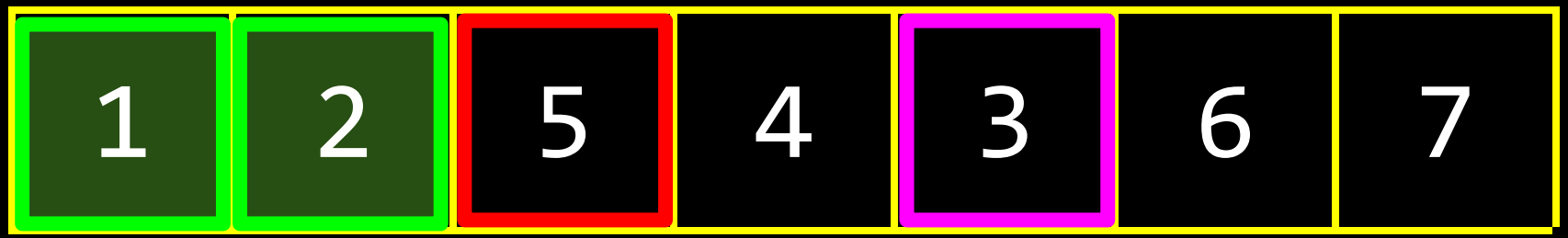

SO FAR: 5




SO FAR: 4




SO FAR: 3



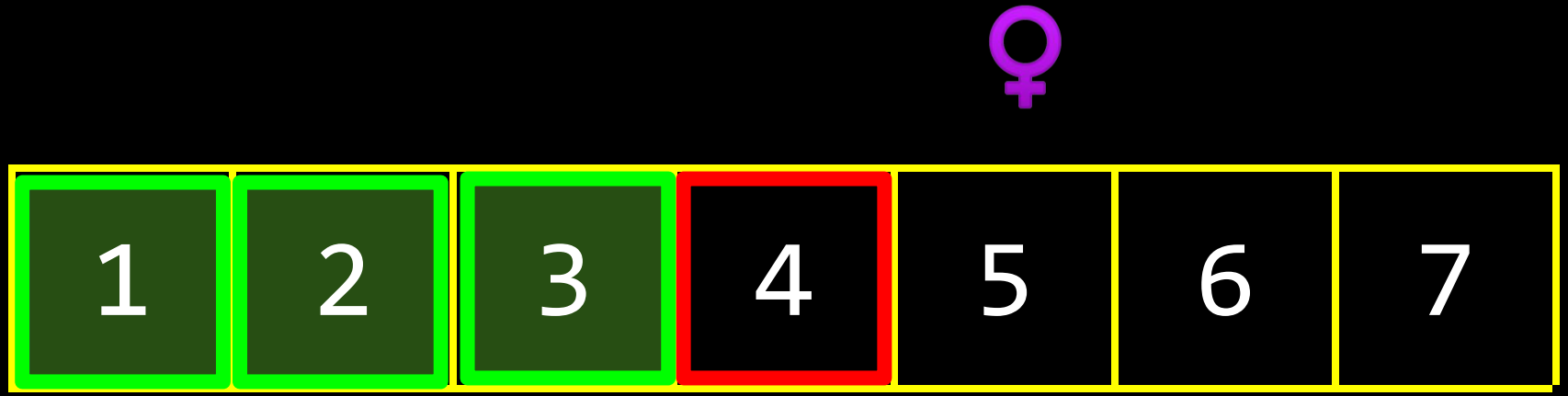

SO FAR: 3


SO FAR: 3

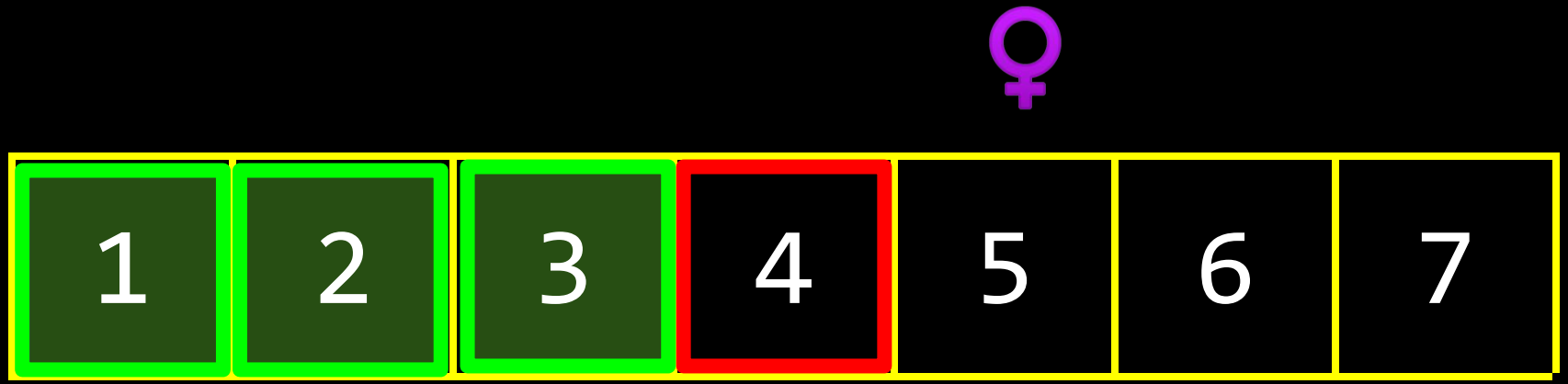

SMALLEST: 3


SWAP!

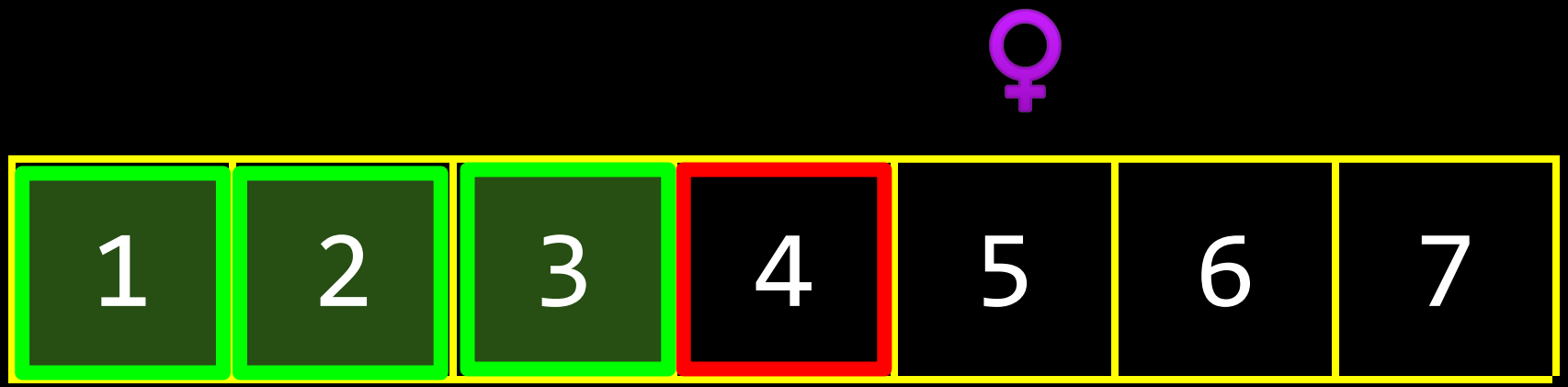

1
2
3
4
5
6
7



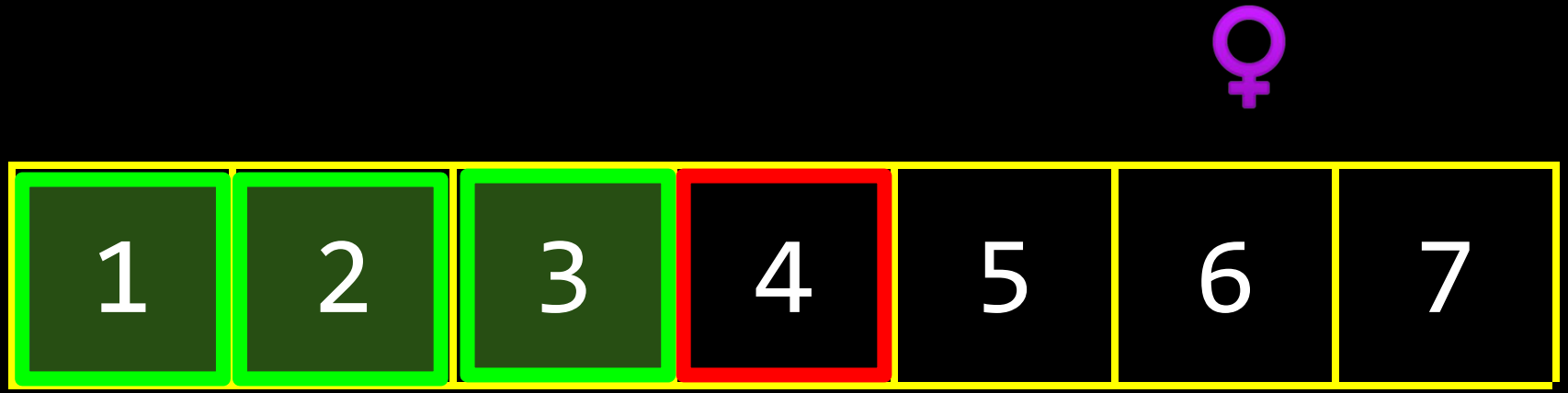

Start again…



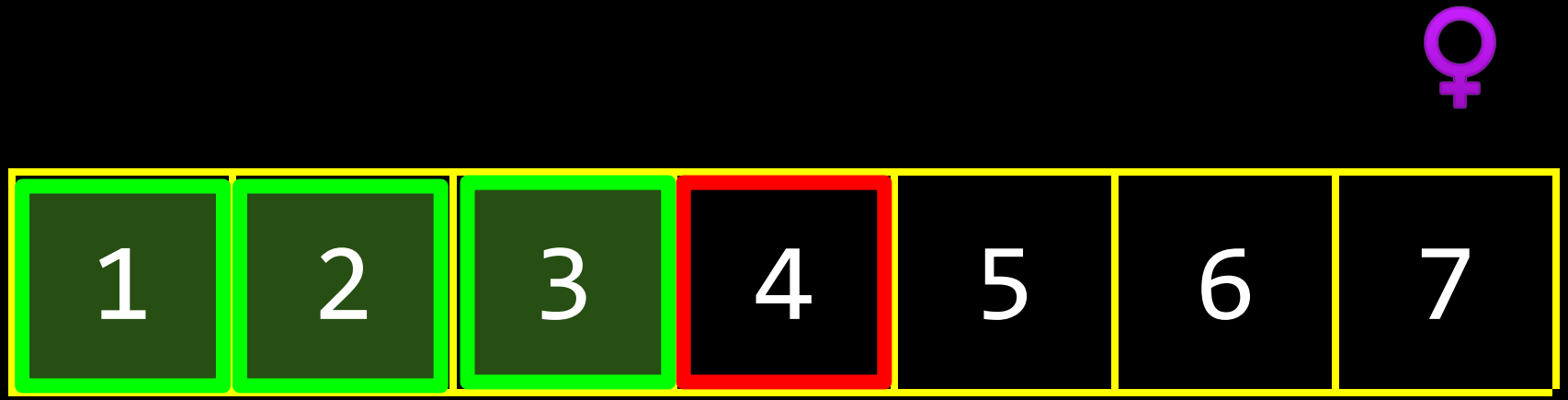

Check: smallest number in rest of the array …




SO FAR: 4

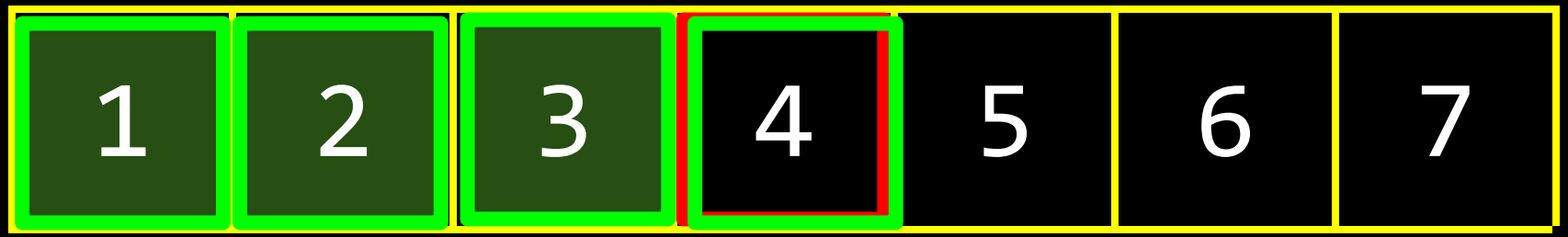



SO FAR: 4



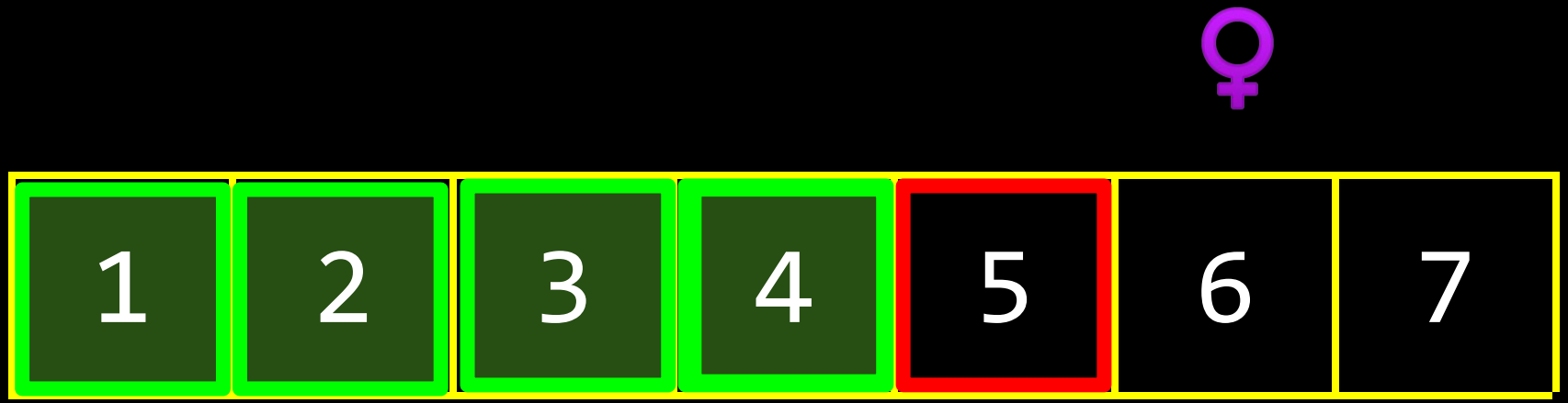

SO FAR: 4


SO FAR: 4

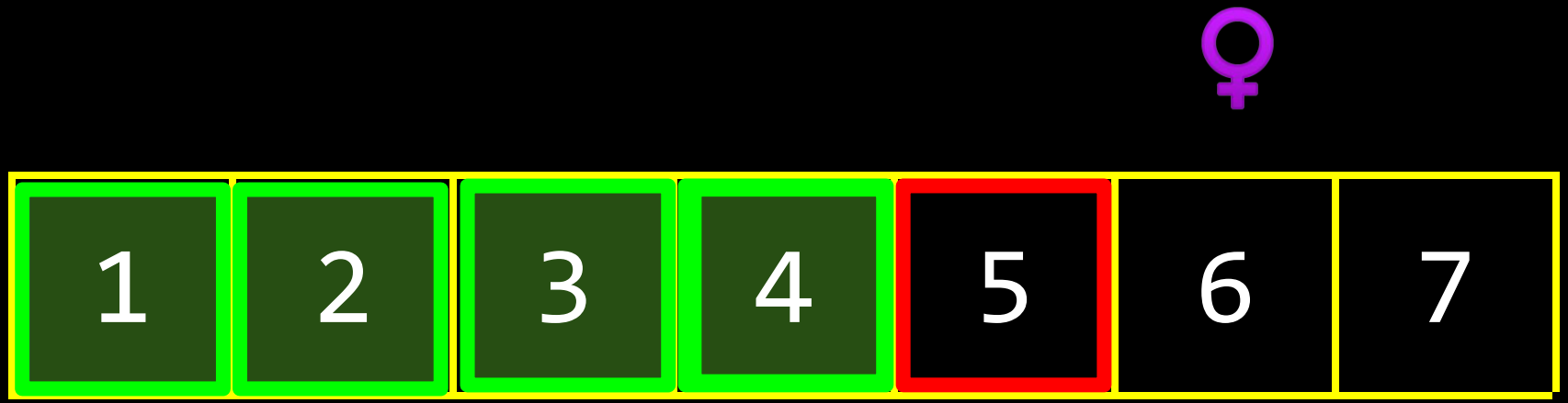



SMALLEST: 4

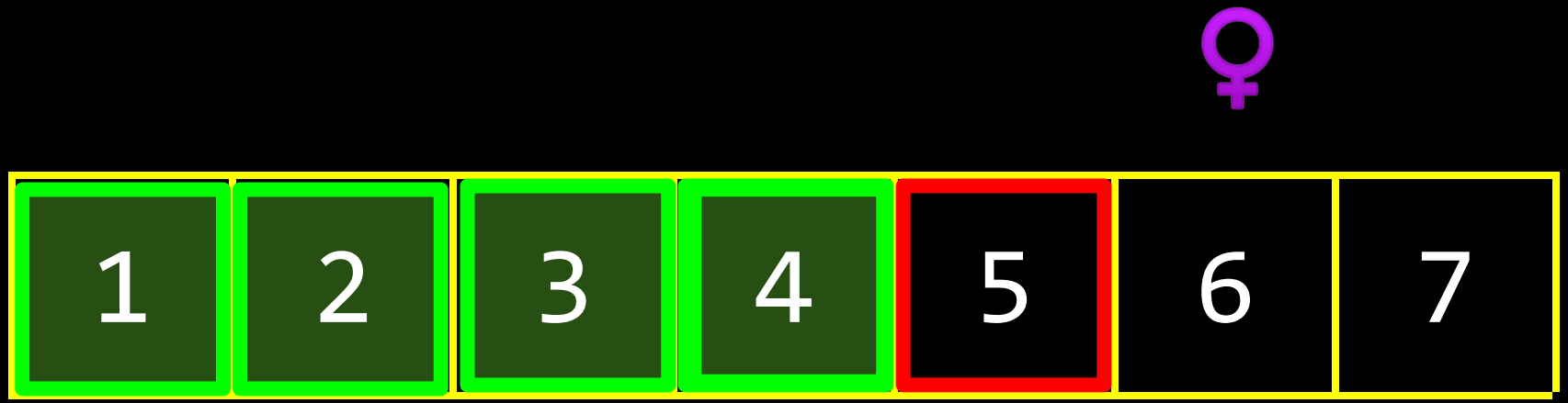

KEEP! (SWAP ITSELF)




Start again…




Check: smallest number in rest of the array …




SO FAR: 5


SO FAR: 5


SMALLEST: 5


SMALLEST: 5


**SMALLEST: 6**


SMALLEST: 6


SMALLEST: 7


SORTED. ✅

# SELECTION SORT

- Create a “**sorted**” and “**unsorted**” part of array
- Search and find smallest item of “**unsorted**” array
- Add this smallest item to “**sorted**” part by swapping with the beginning of “**unsorted**” part
- Continue throughout entire array

$O(n^2)$ selection sort

$O(n \log n)$

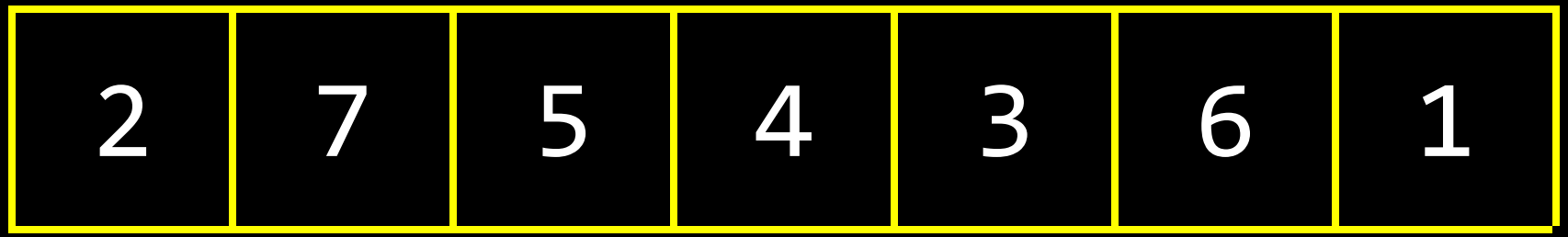
$O(n)$

$O(\log n)$

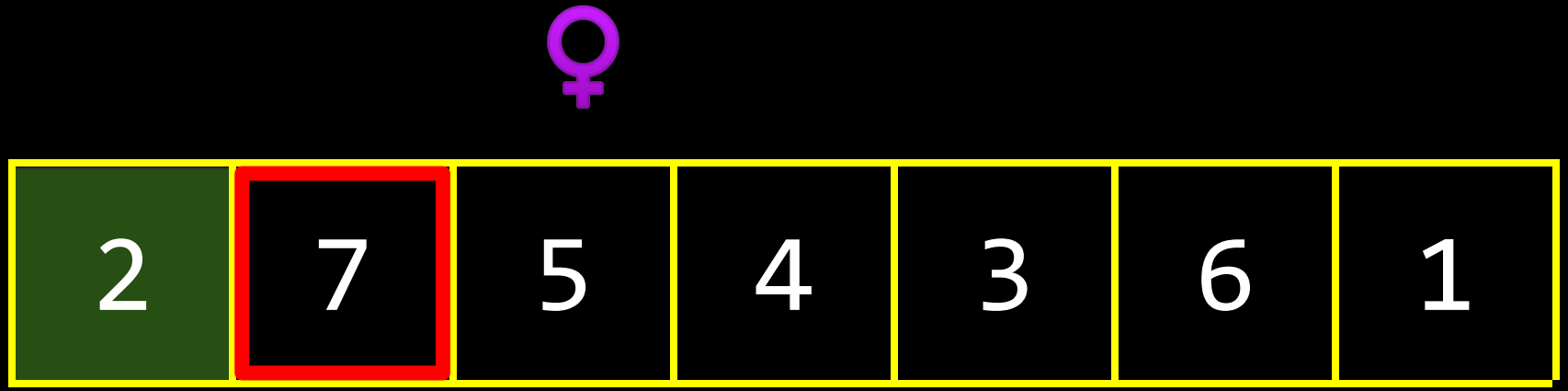
$O(1)$

from Harvard CS50

# INSERTION SORT

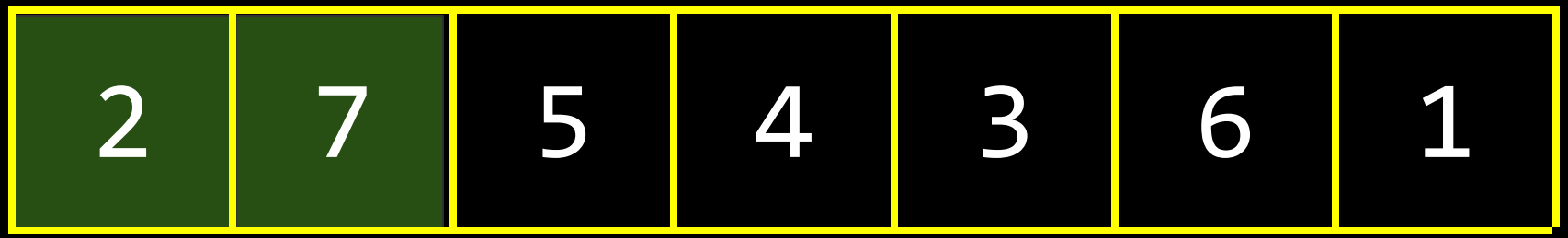


# INSERTION SORT

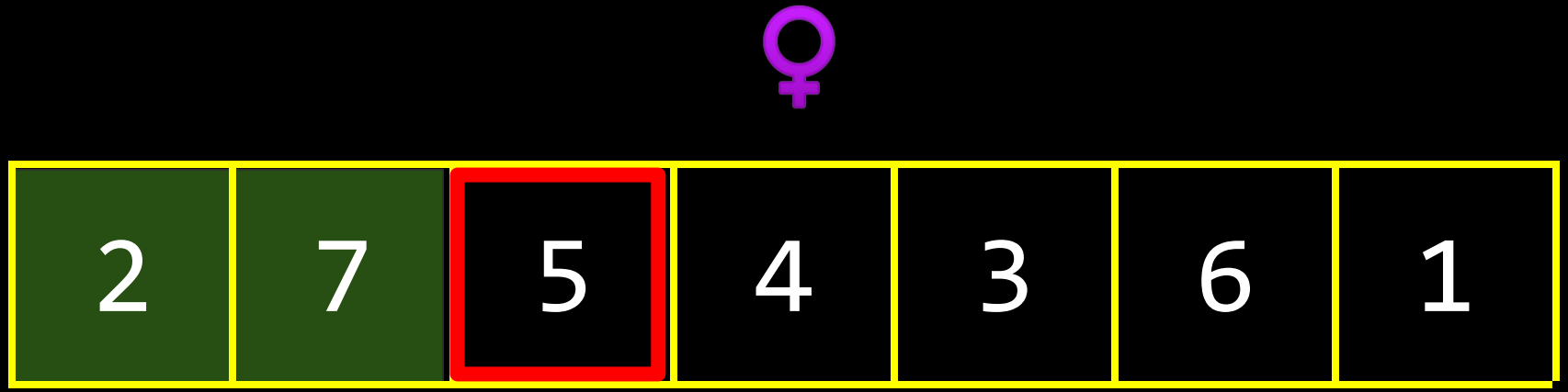



Check to the LEFT: is the one on the left smaller?

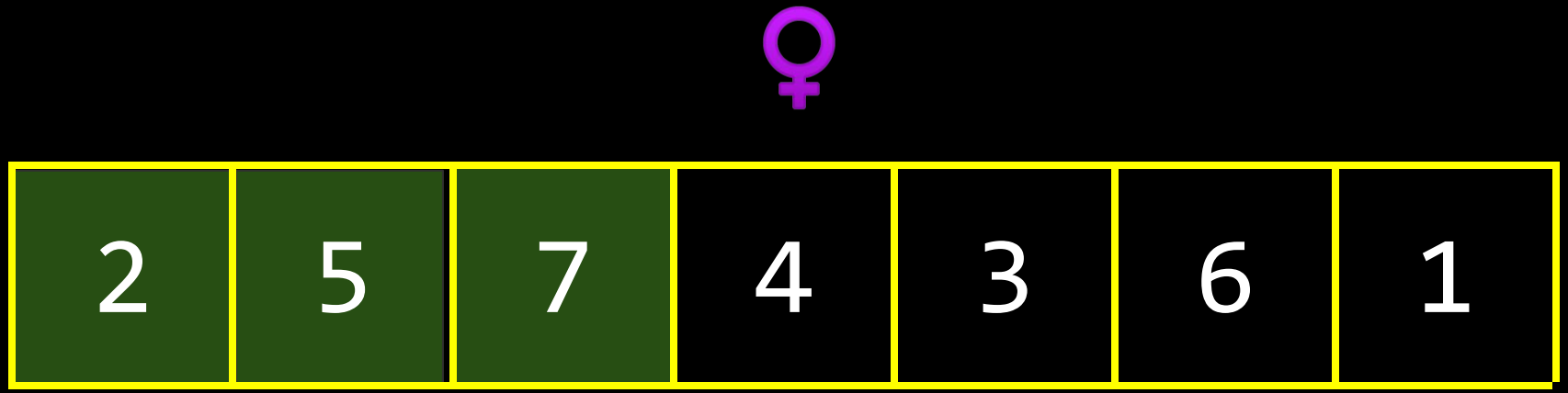
# INSERTION SORT



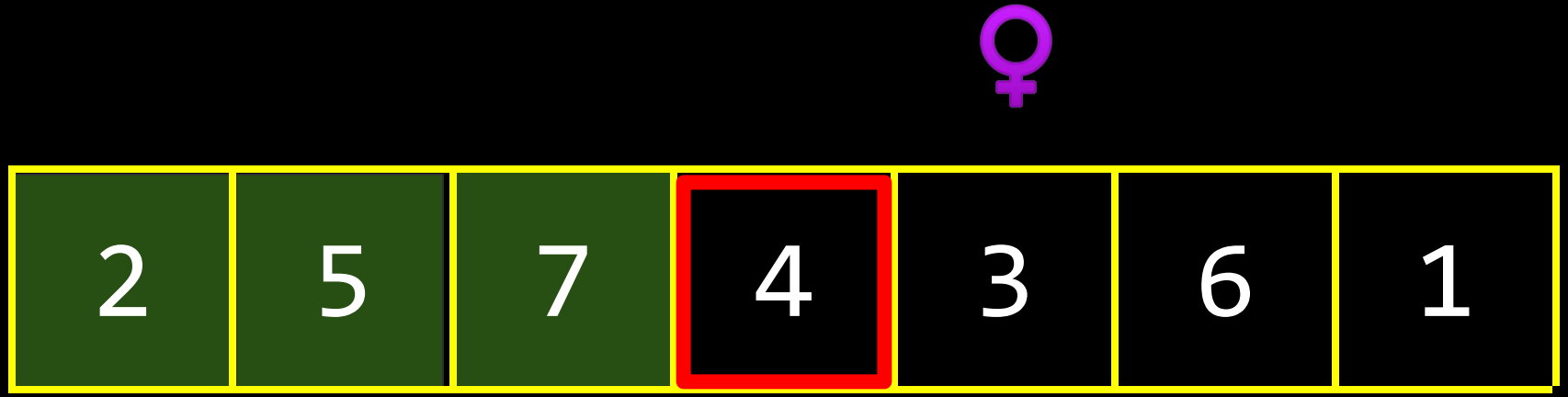

IF YES, move on …



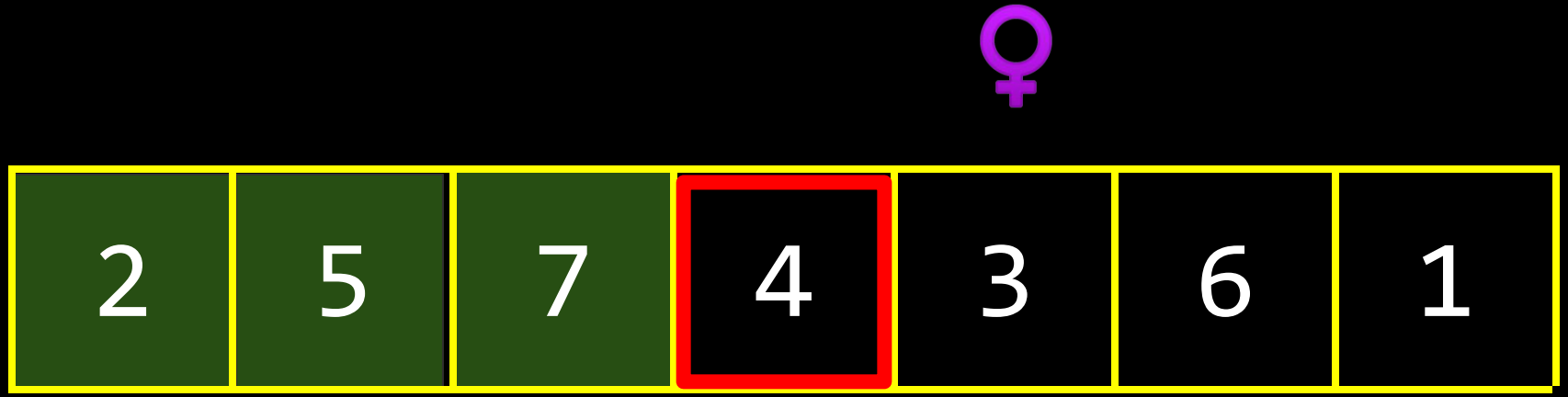

Check to the LEFT: is the one on the left smaller?



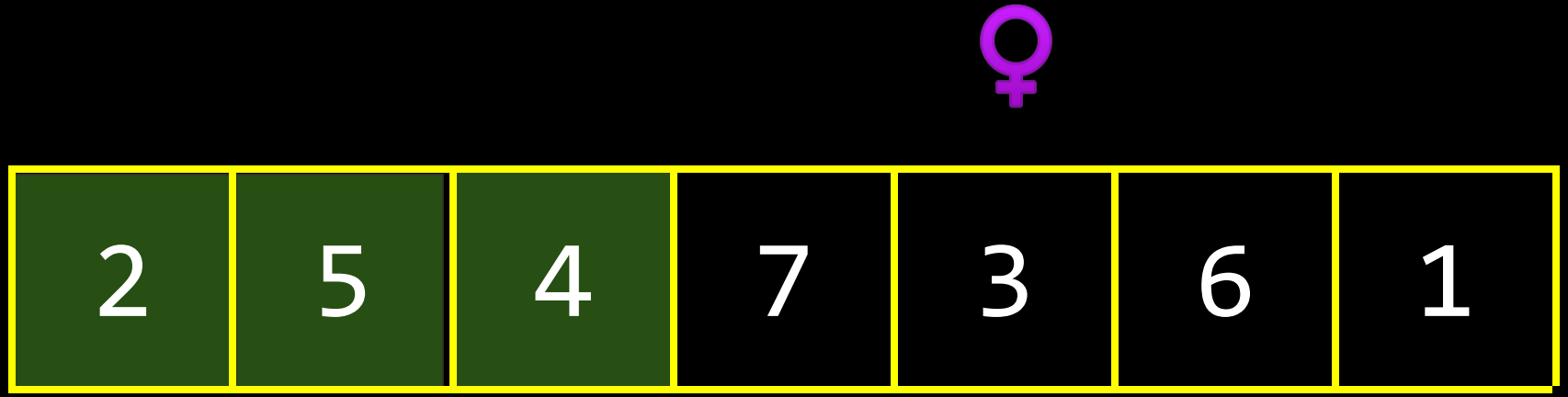

INSERT in correct spot!
(SWAP until the left is smaller)



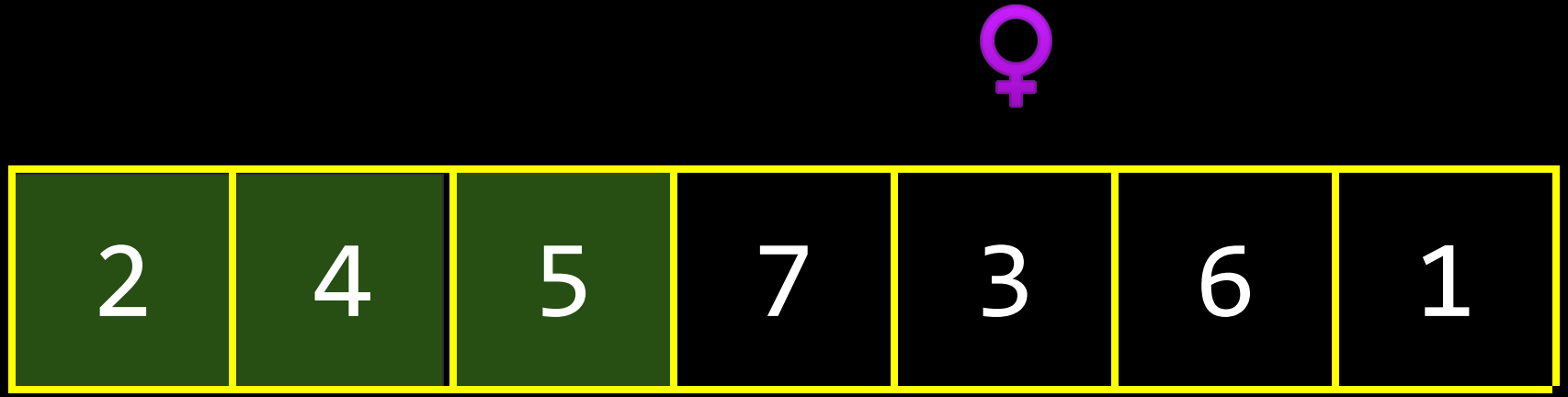

Check to the LEFT: is the one on the left smaller?



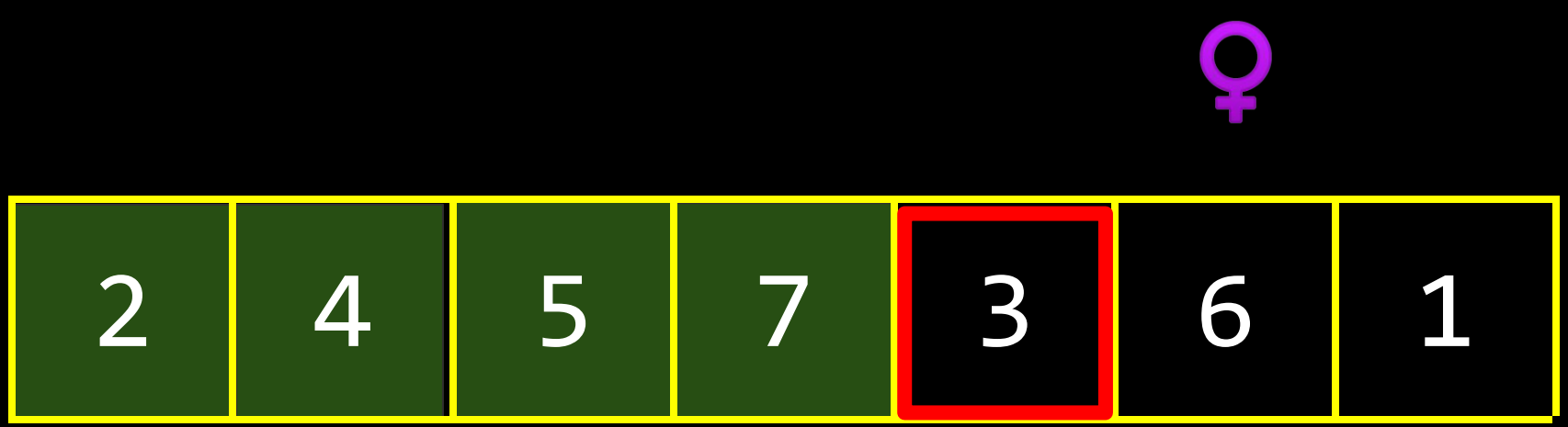

INSERT in correct spot!



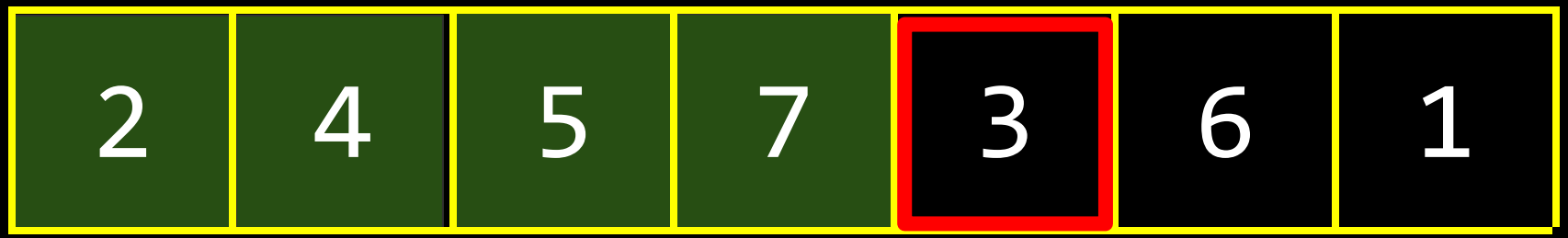

INSERT in correct spot!
(swap until in order)




INSERT in correct spot!
(swap until in order)

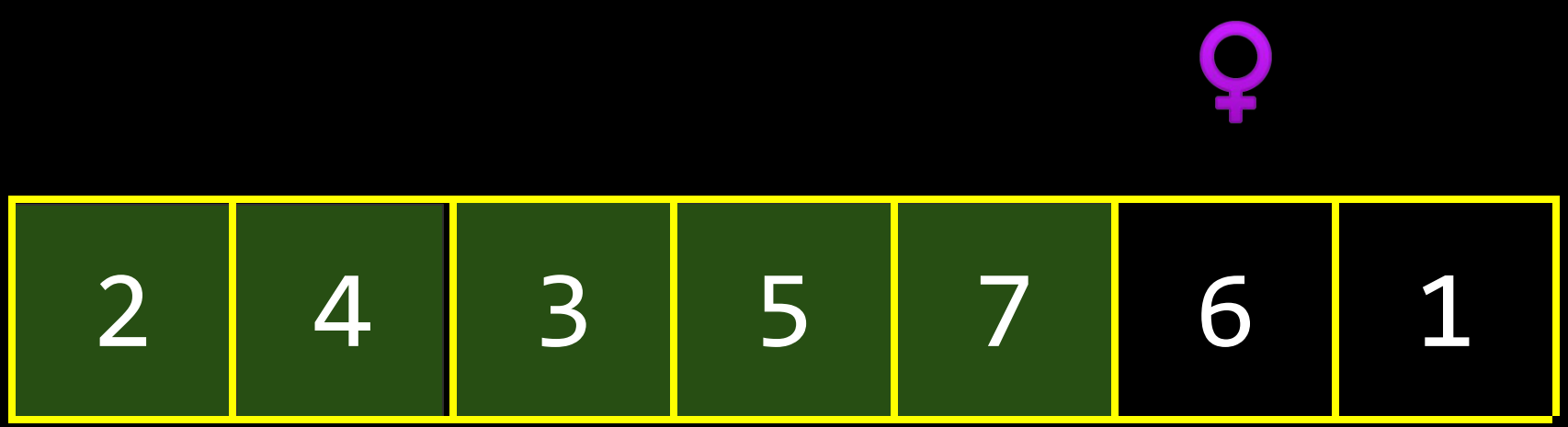



Check to the LEFT: is the one on the left smaller?

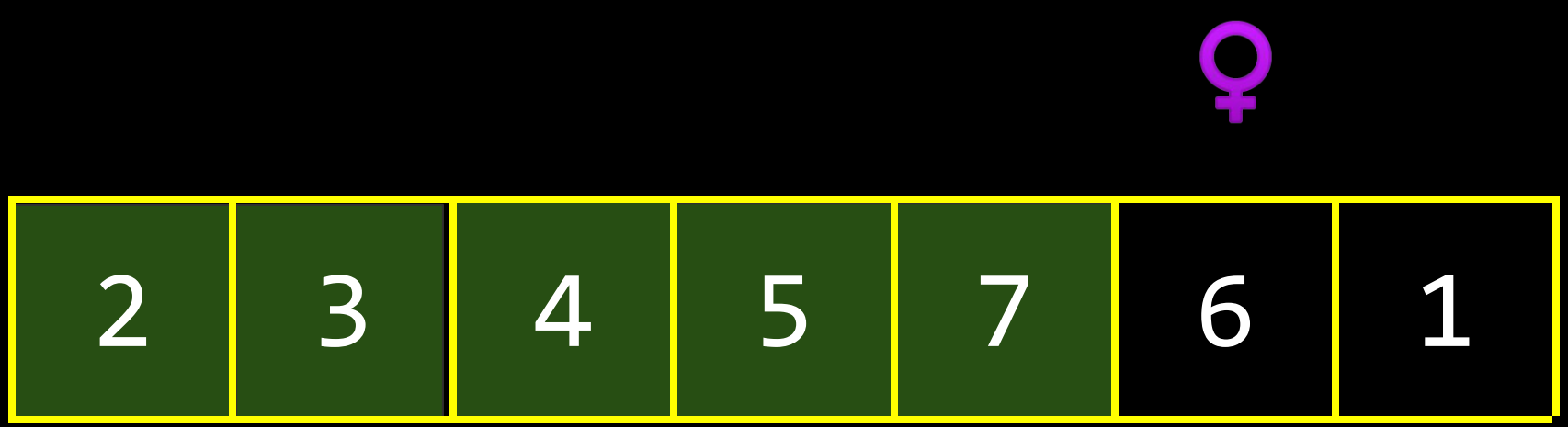



INSERT in correct spot!

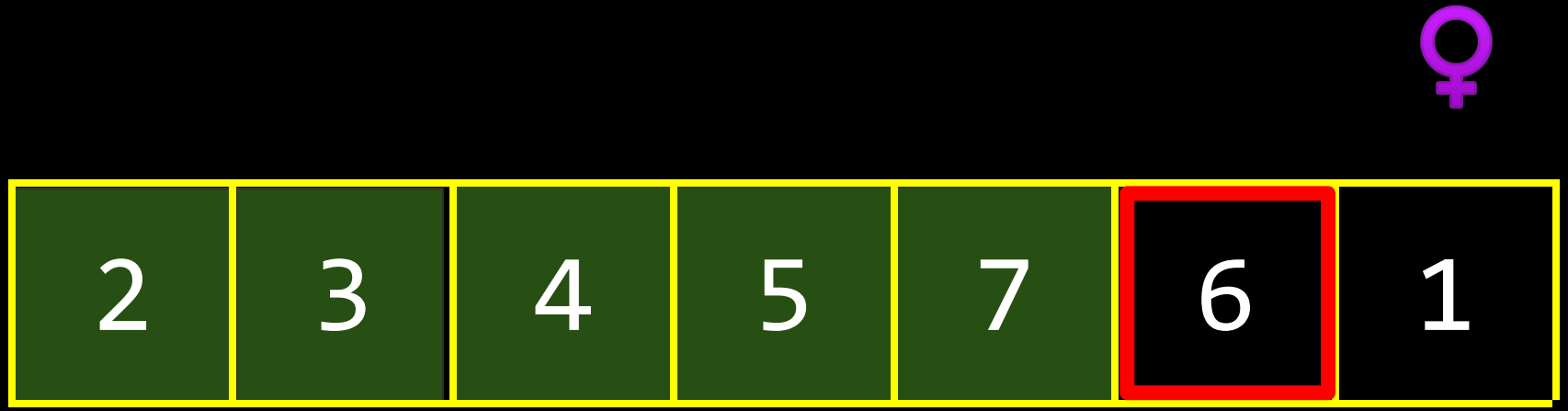

INSERT in correct spot!
(swap until in order)




INSERT in correct spot!
(swap until in order)




INSERT in correct spot!
(swap until in order)



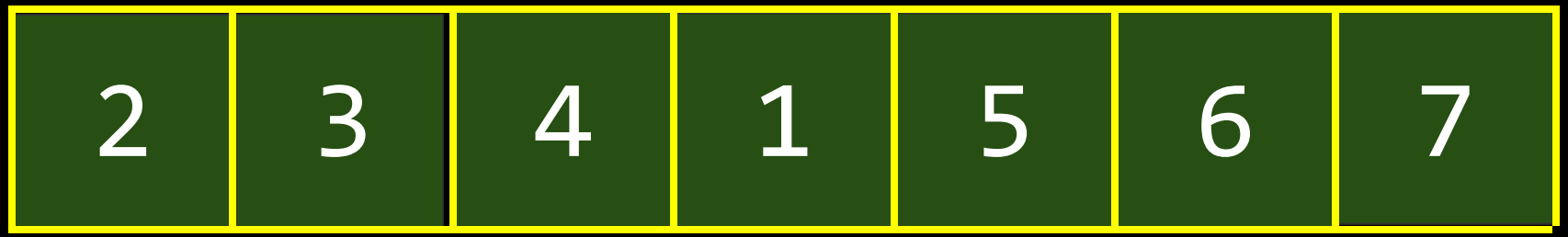

Check to the LEFT: is the one on the left smaller?


INSERT in correct spot!
(swap until in order)


Check to the LEFT: is the one on the left smaller?


INSERT in correct spot!
(swap until in order)


INSERT in correct spot!
(swap until in order)




INSERT in correct spot!
(swap until in order)


INSERT in correct spot!
(swap until in order)


INSERT in correct spot!
(swap until in order)


SORTED. ✅

# INSERTION SORT

- ♠ Like sorting playing cards in your hand
- Create a "**sorted**" and "**unsorted**" part of array
- Compare the start of the "**unsorted**" part of the array on the right with the end of the "**sorted**" on the left - are they in the correct order?
- If not, swap those 2 values
- Continue swapping until the new value is in the correct order in the "**sorted**" array

$O(n^2)$ selection sort, insertion sort

$O(n \log n)$

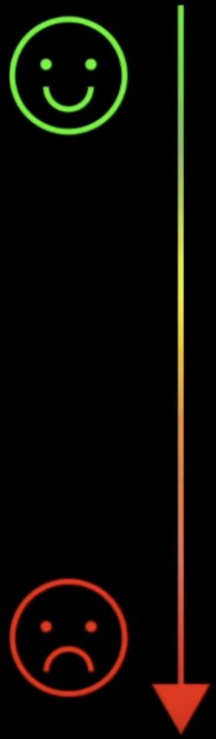
$O(n)$

$O(\log n)$

$O(1)$

from Harvard CS50

# Classes of Running Times



Constant - $O(1)$

Logarithmic - $O(\log N)$

Linear - $O(N)$

Linearithmic - $O(N \log N)$

Polynomial - $O(N^2), O(N^3), O(N^4)$, etc.

Exponential - $O(2^N), O(3^N), O(4^N)$, etc.

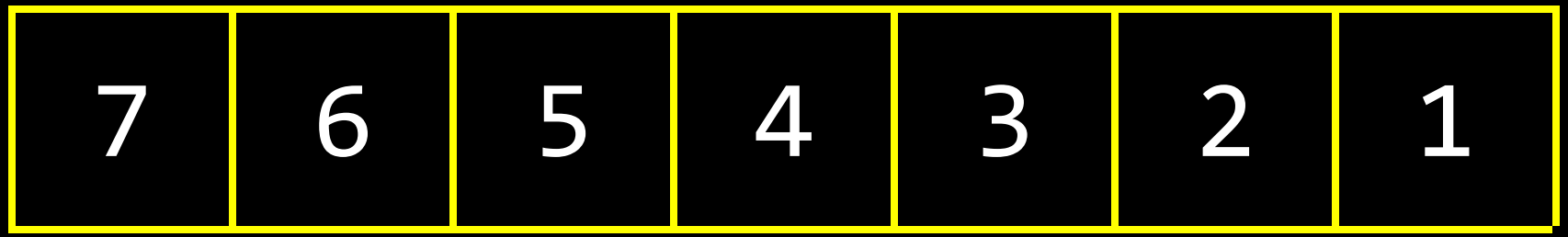
# SELECTION VS. INSERTION SORT

- Both OK for small lists!
- Both use nested loops - $O(n^2)$
- Insertion better for "**nearly sorted**" lists
- Ω (best case) is O(n) for **insertion sort**
- Ω (best case) is still $O(n^2)$ for **selection sort** - we still have to go through all elements in the array left to right to check if sorted

# SELECTION VS. INSERTION SORT

- Ω (best case) for **insertion sort?**
- **It's already sorted! O(n)**
- O (worst case) for **insertion sort?**
- **It's sorted in reverse $O(n^2)$**

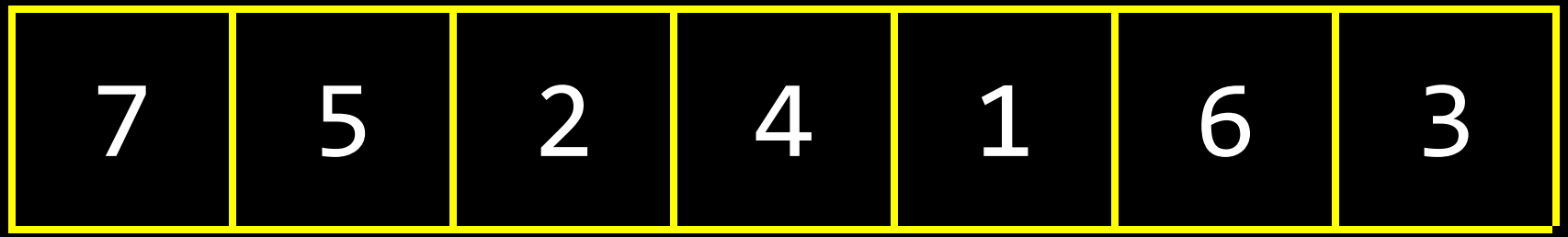
# INSERTION SORT



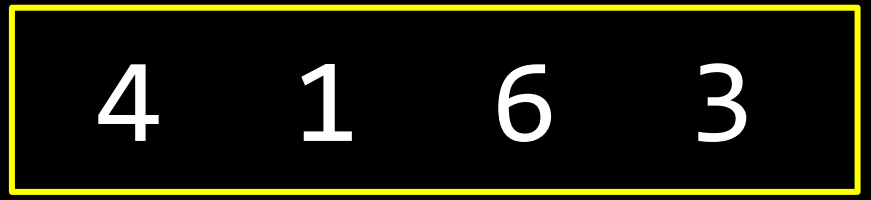
# ✨ RECURSION + MERGE SORT

- If list has only 1 number in it, SORTED ✅
- Sort the left half of list
- Sort the right half of list
- **Merge** the 2 sorted halves

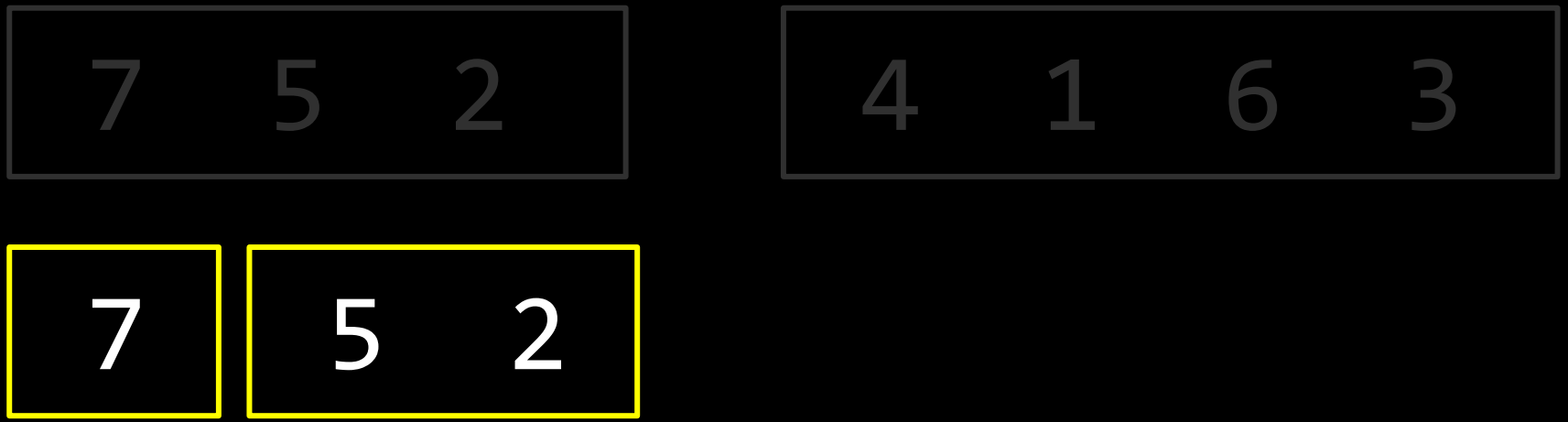
# MERGE SORT!



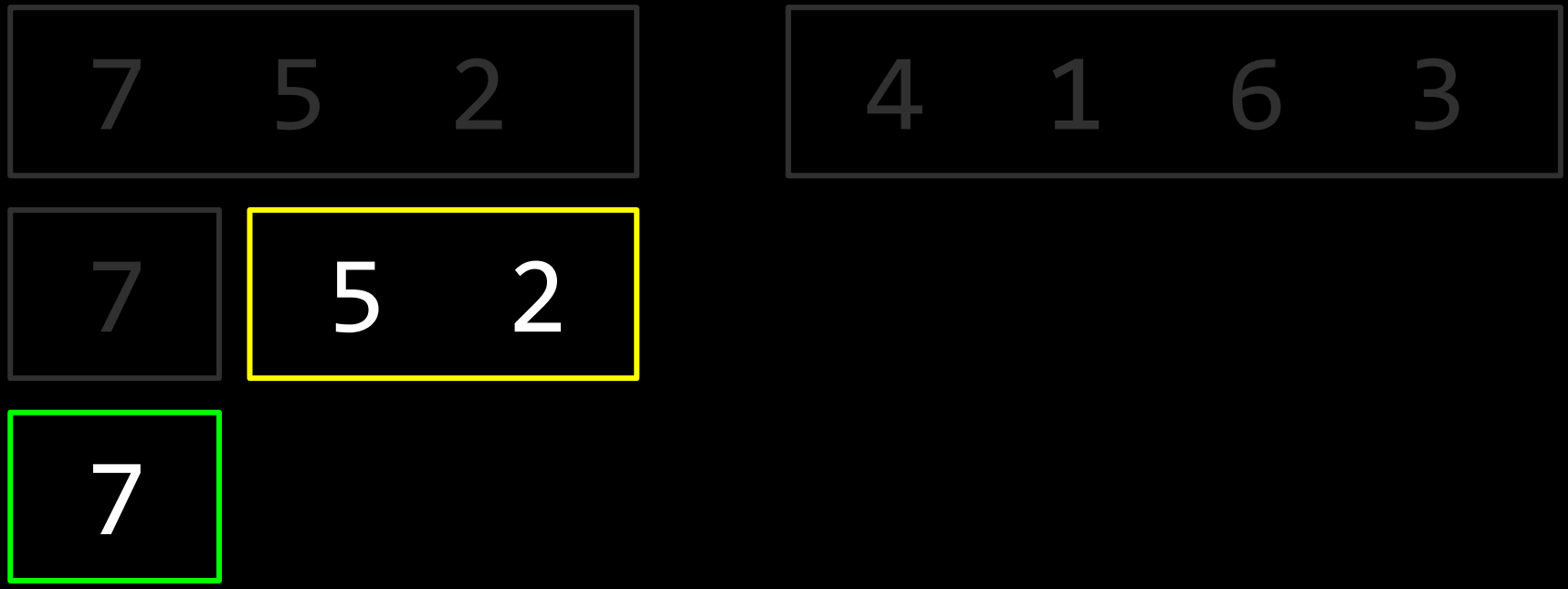
# MERGE SORT!

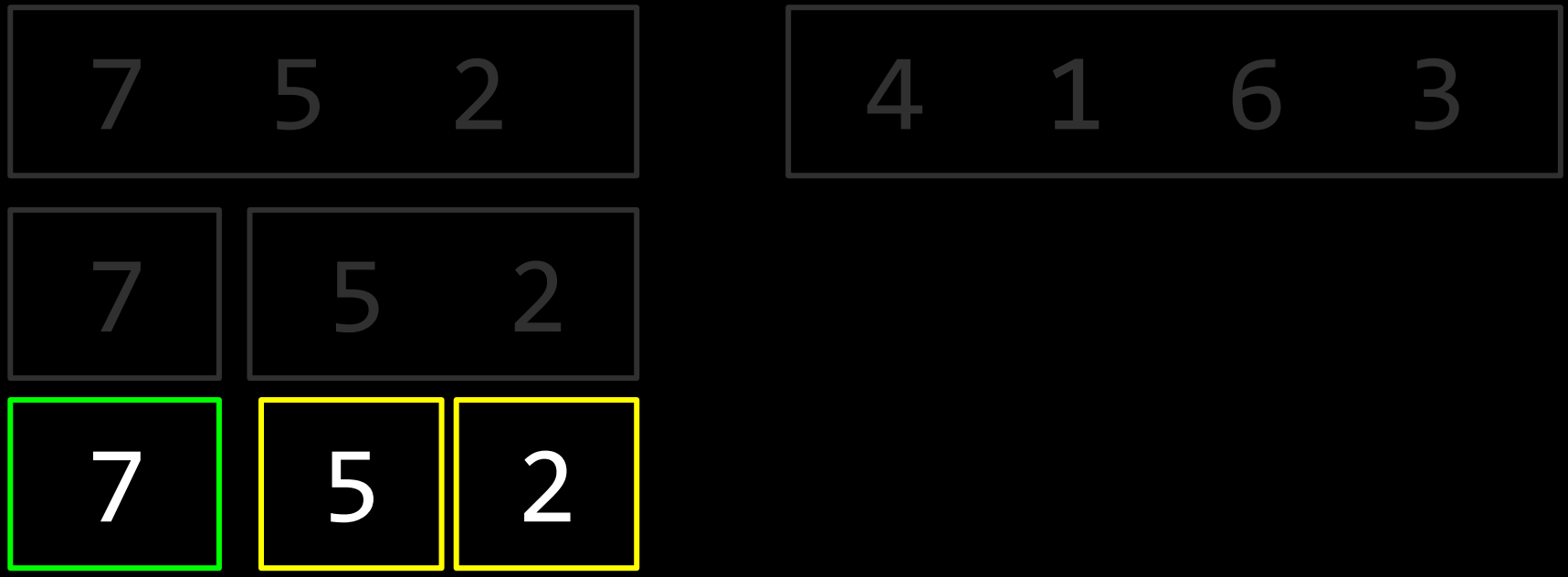
# MERGE SORT!



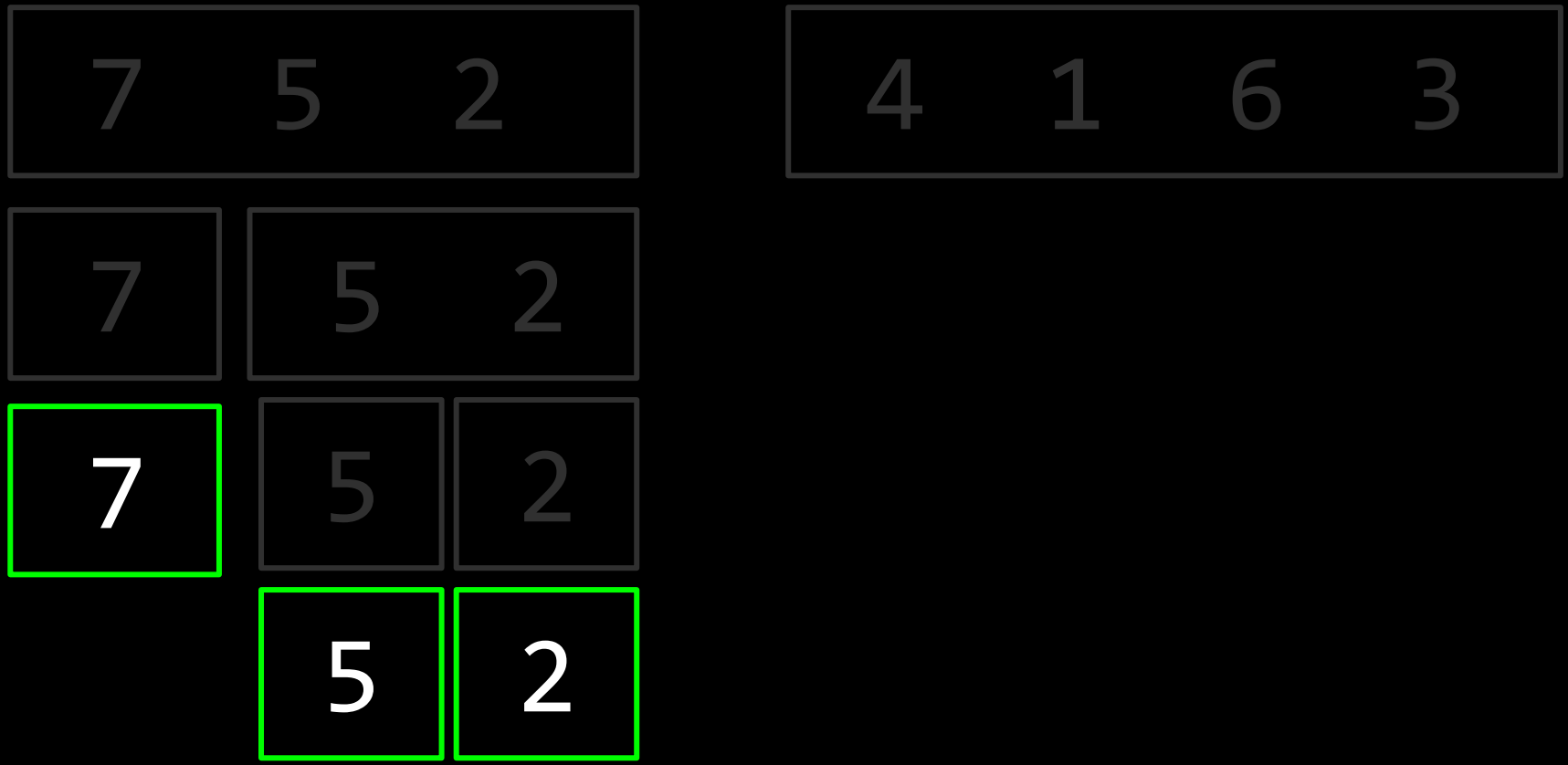
# MERGE SORT!



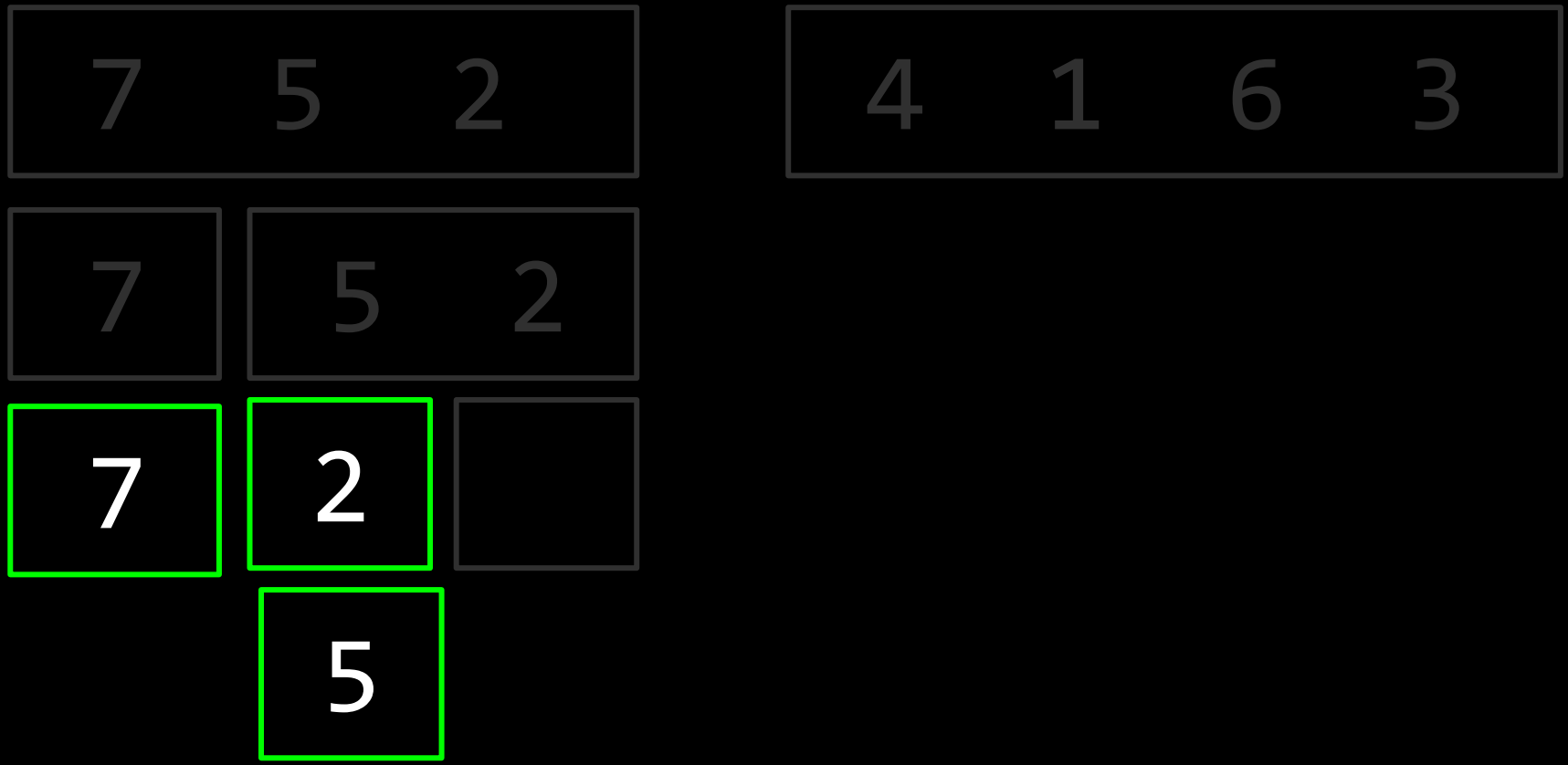
# MERGE SORT!



# MERGE SORT!



# MERGE SORT!



# MERGE SORT!

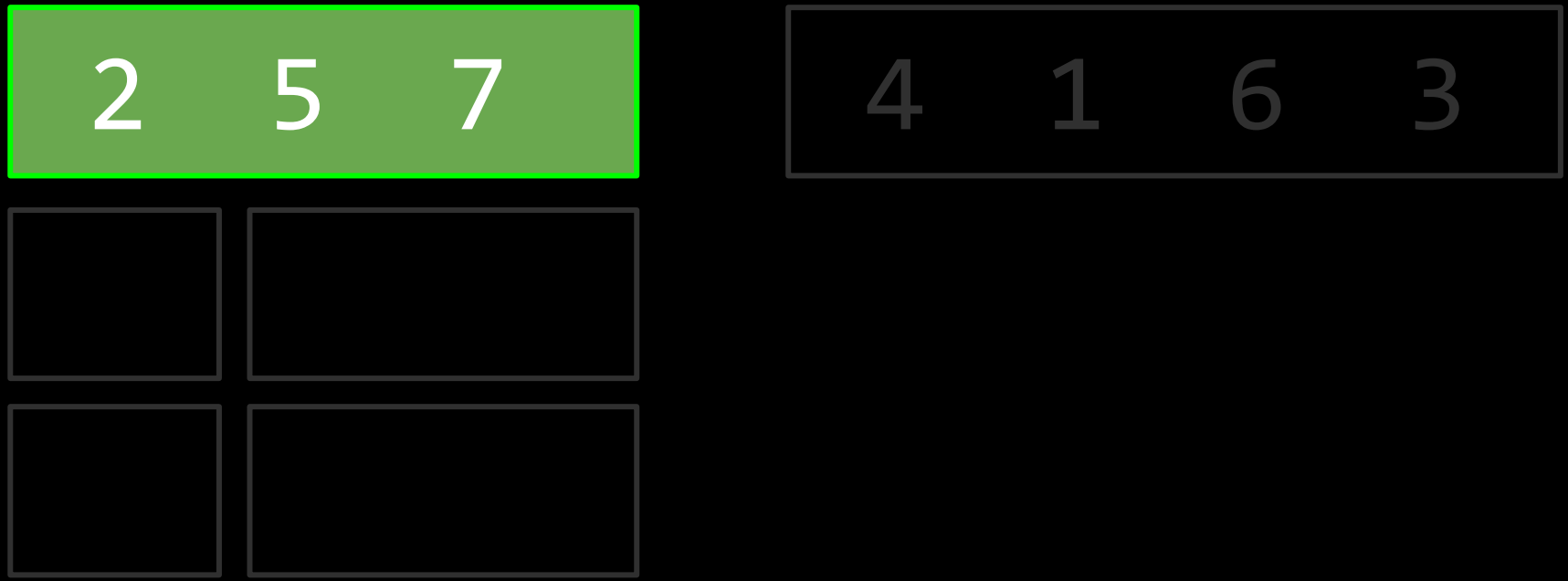
# MERGE SORT!

# MERGE SORT!

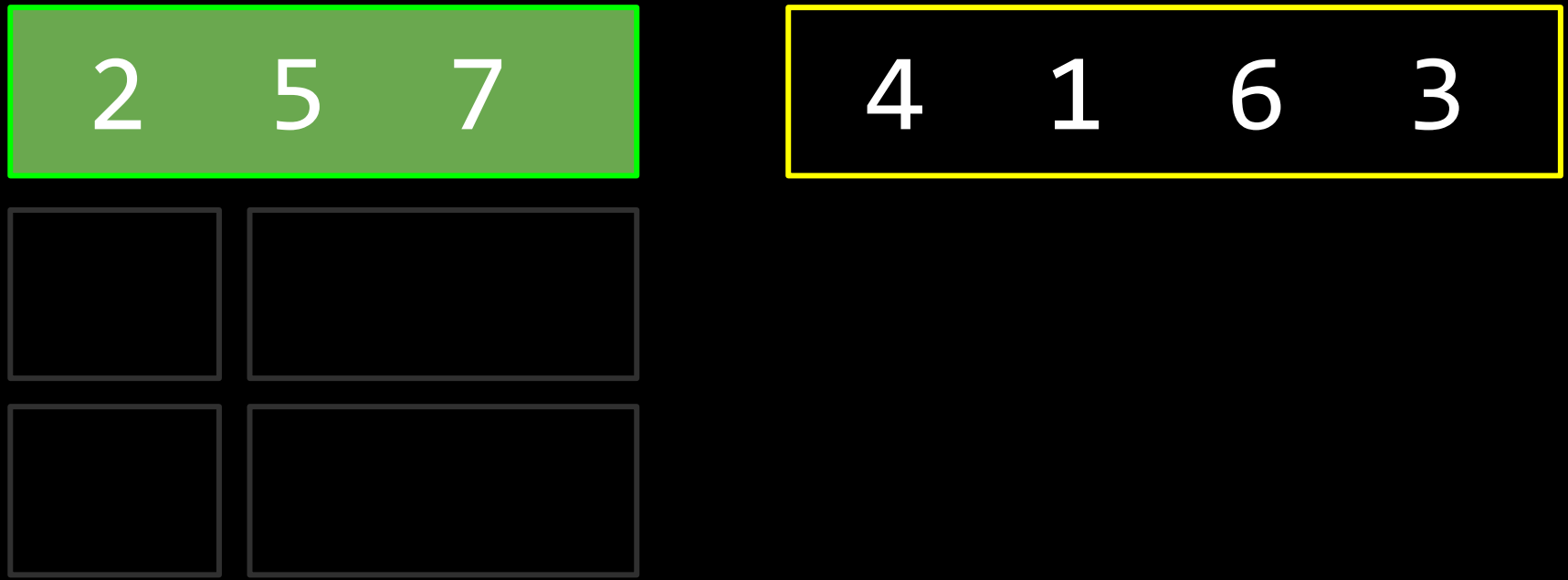
# MERGE SORT!

# MERGE SORT!

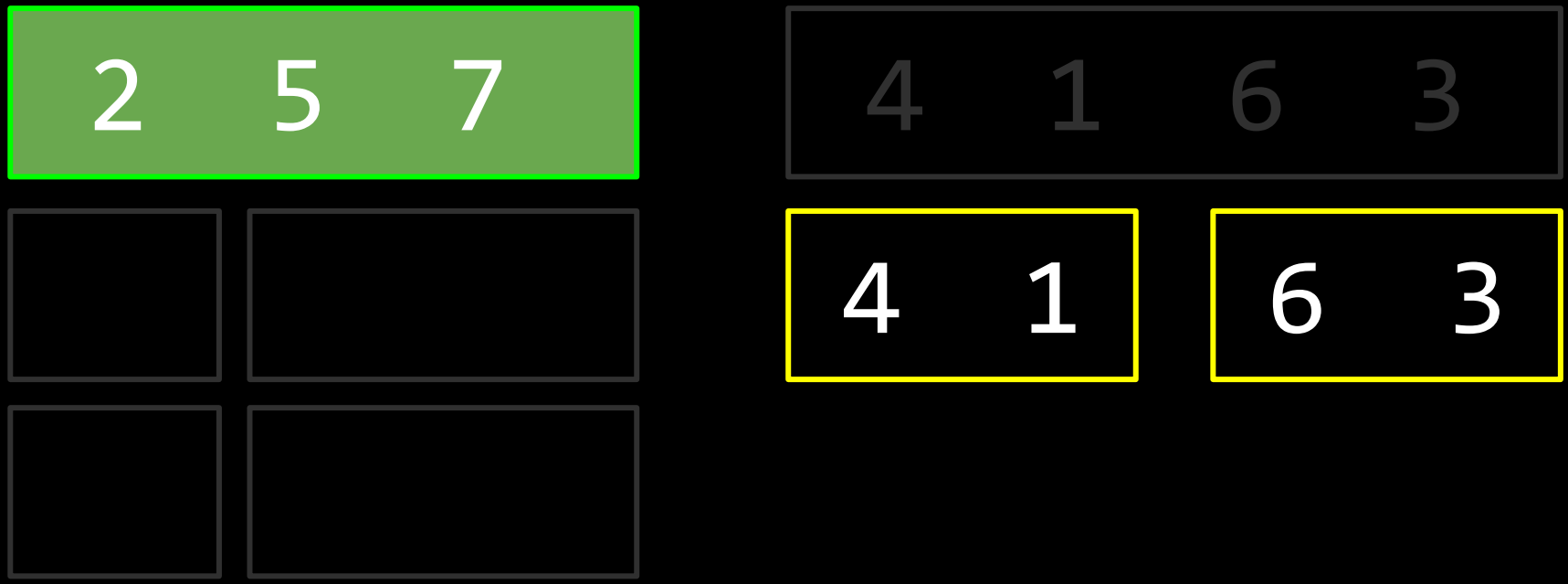
# MERGE SORT!



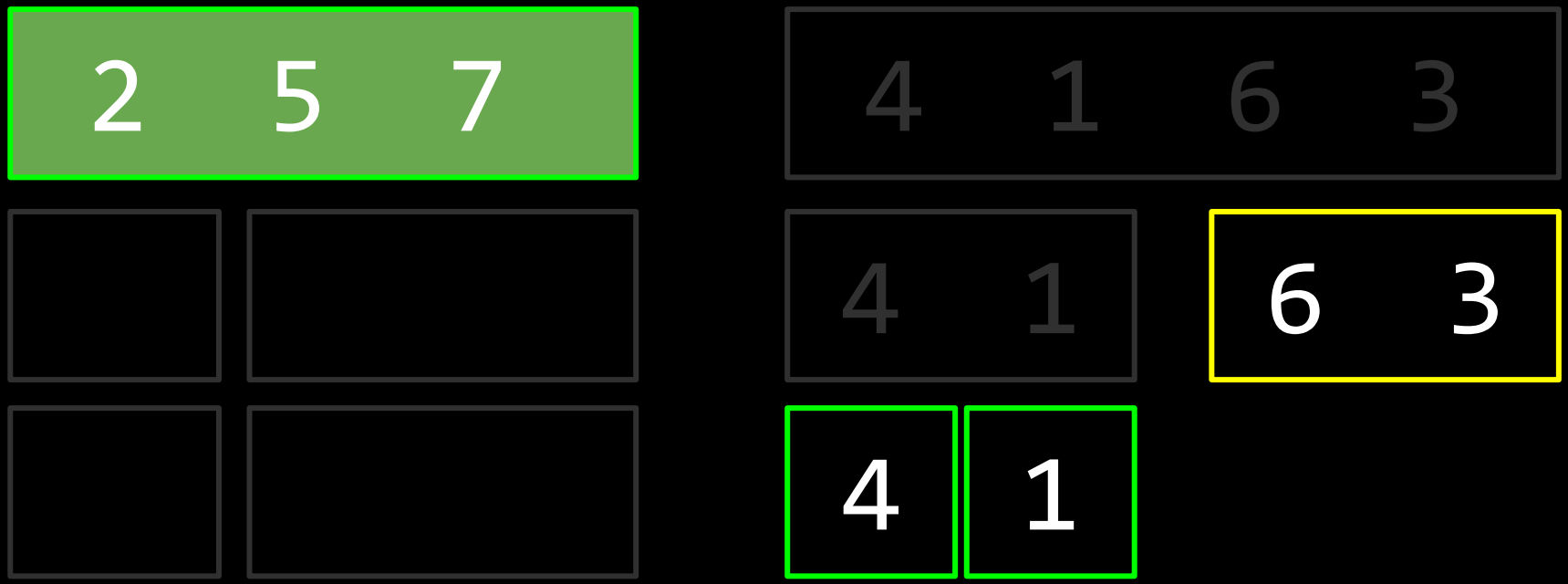
# MERGE SORT!



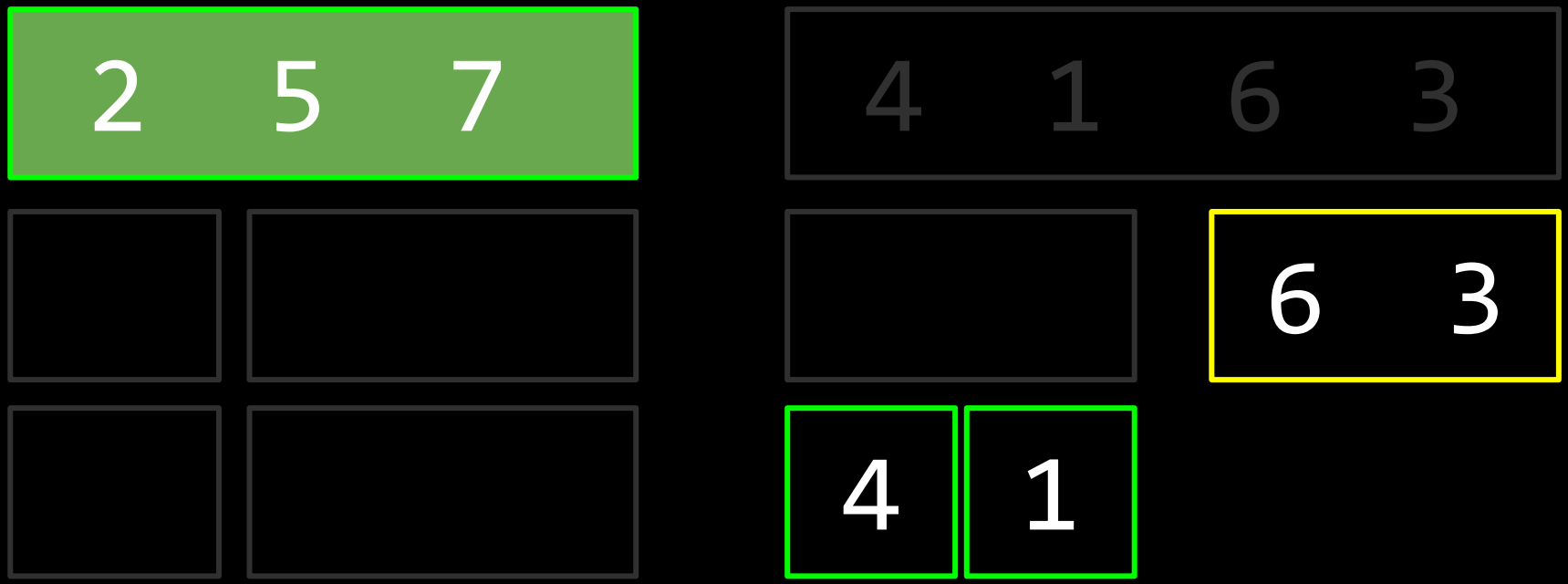
# MERGE SORT!



# MERGE SORT!

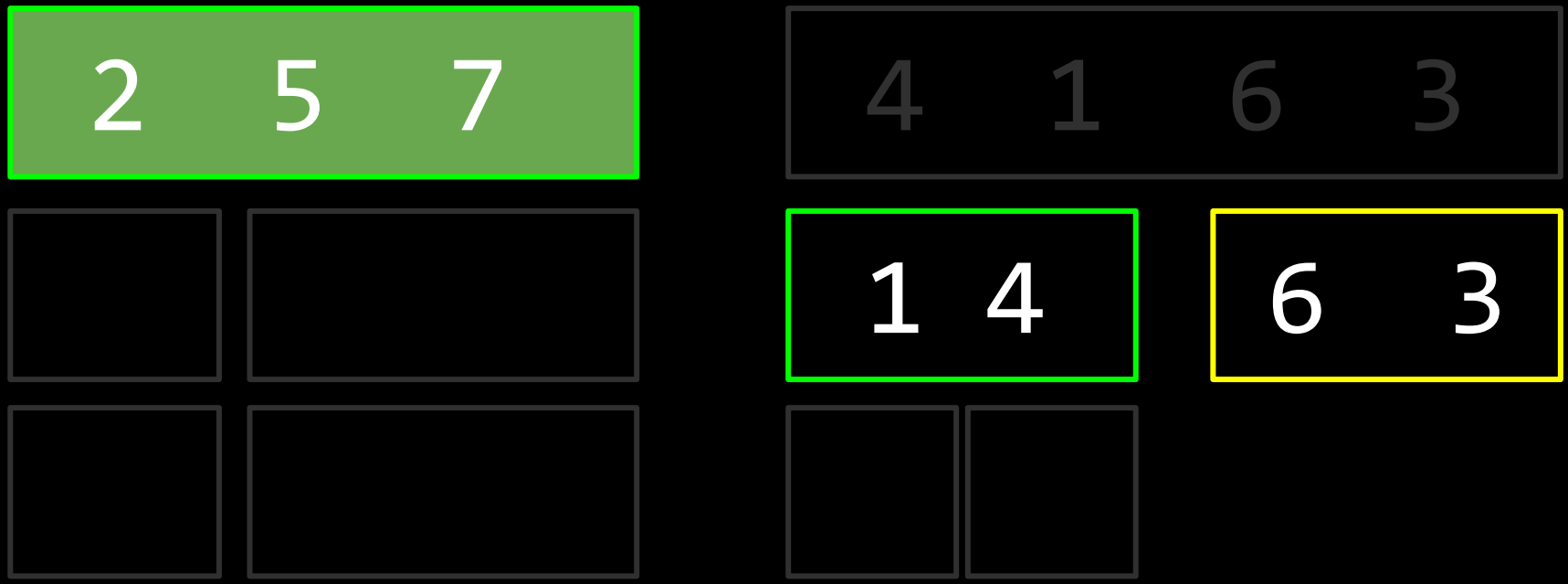


# MERGE SORT!

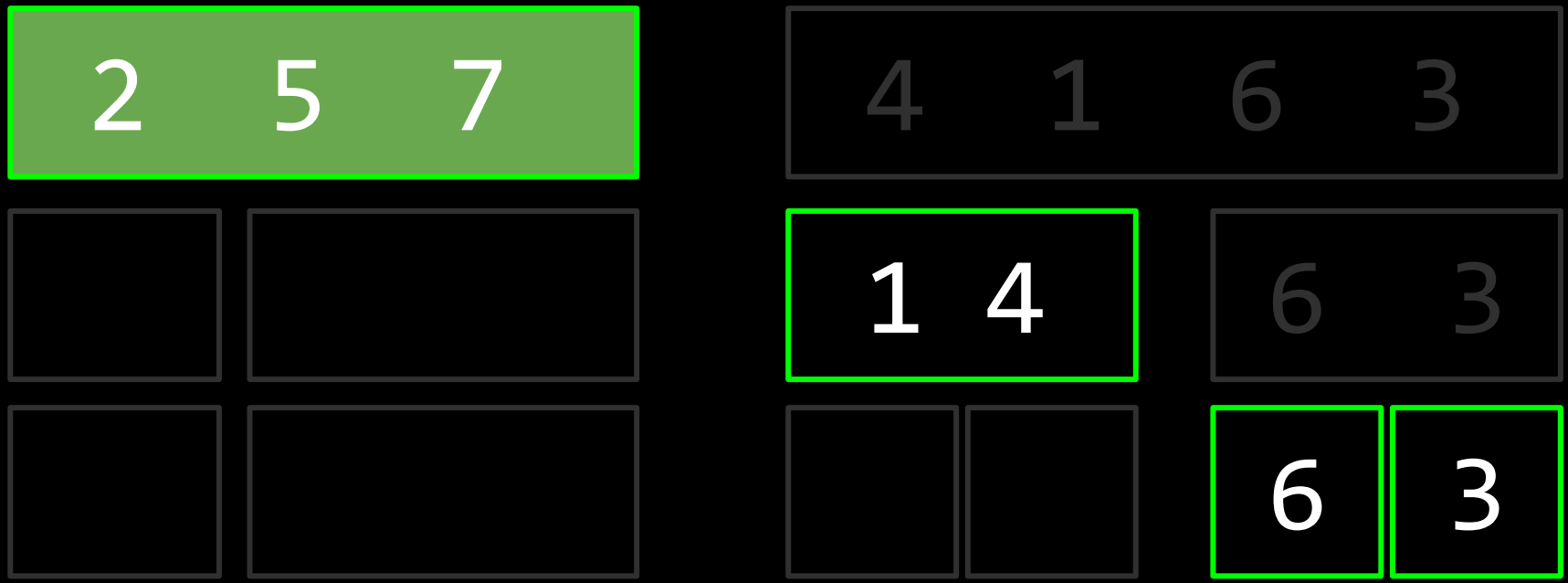


# MERGE SORT!

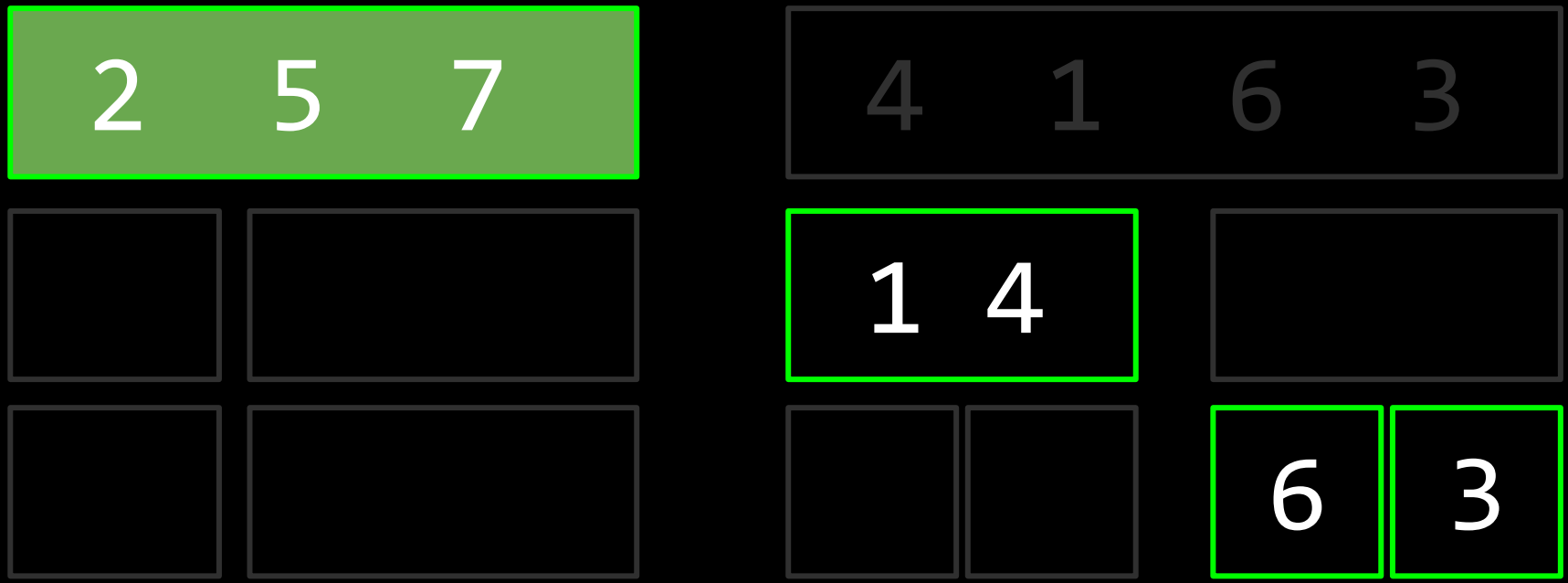
# MERGE SORT!



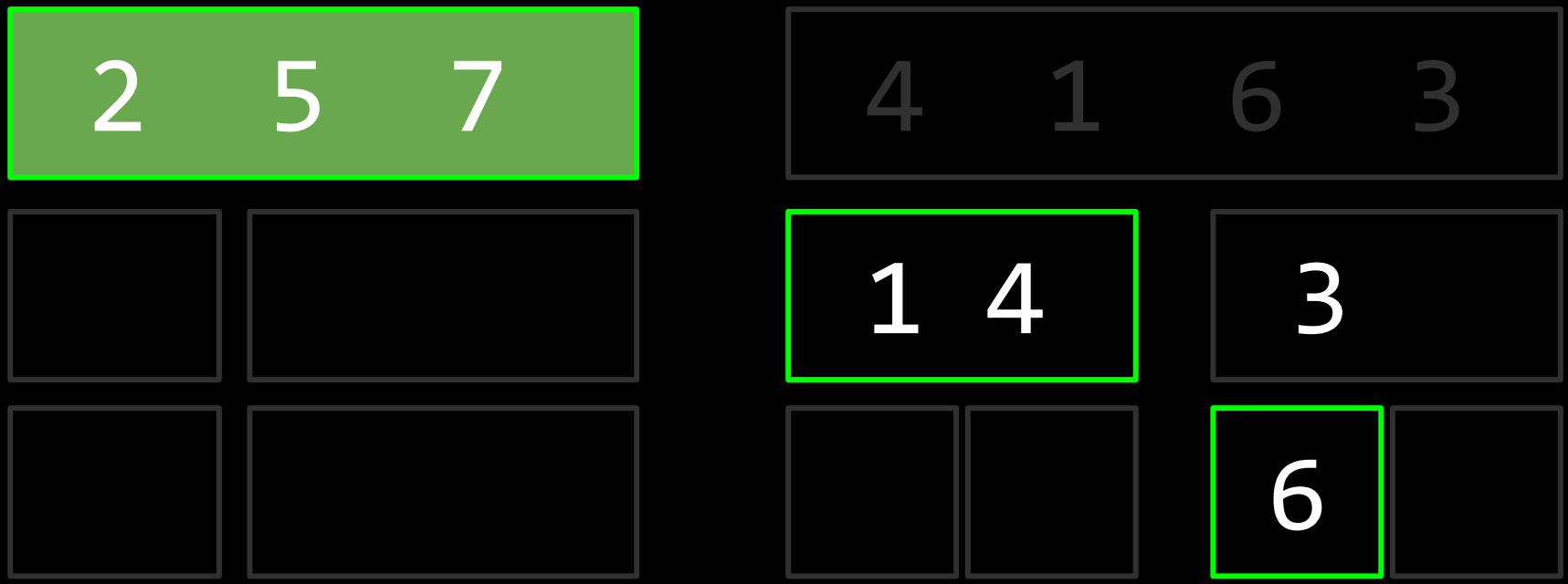
# MERGE SORT!



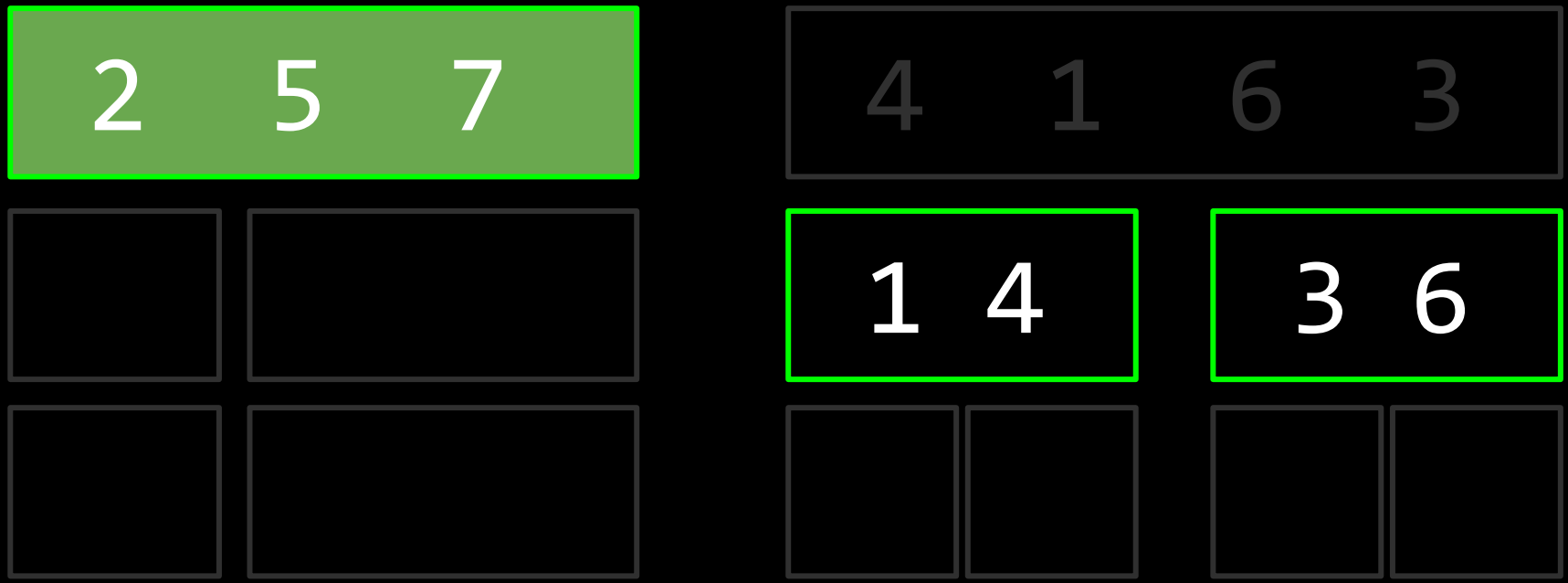
# MERGE SORT!



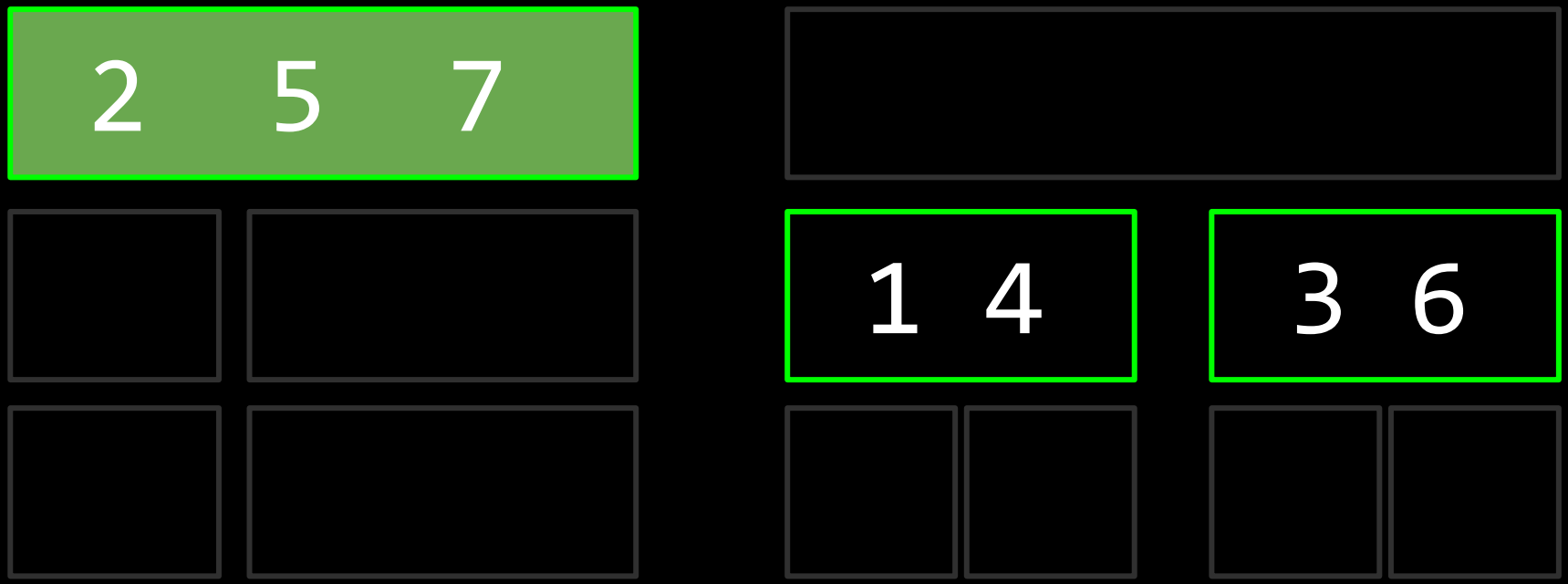
# MERGE SORT!



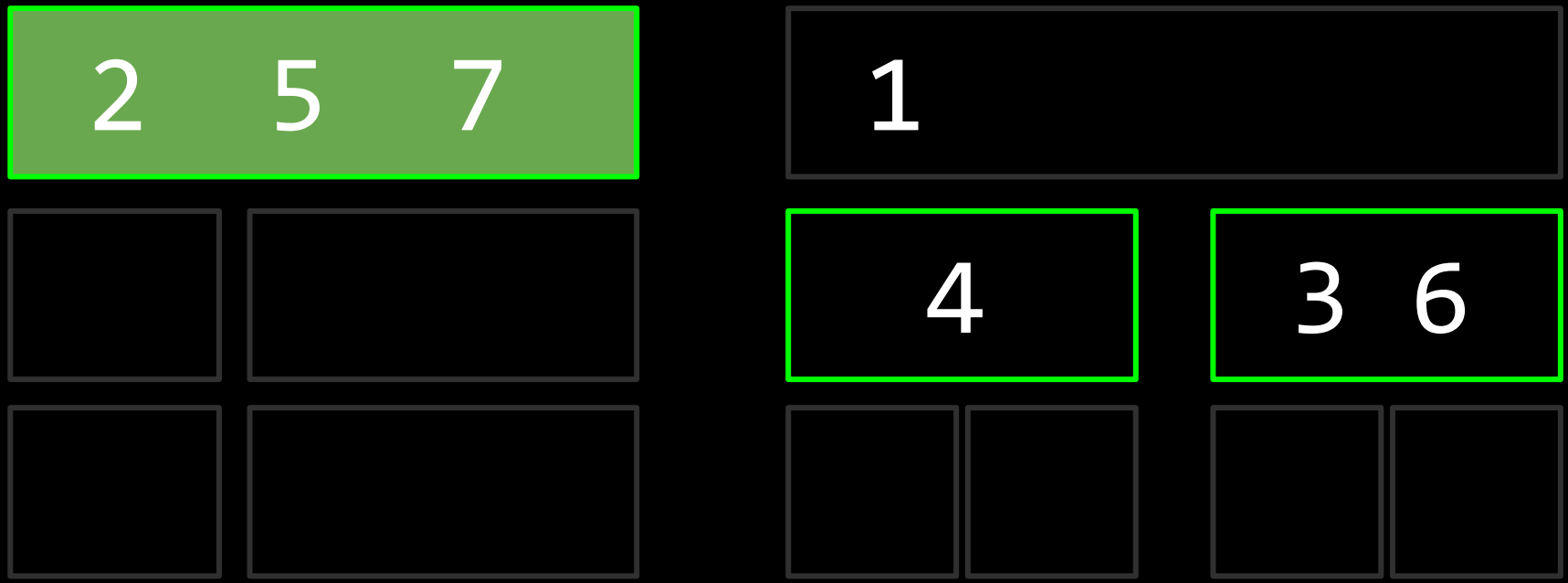
# MERGE SORT!



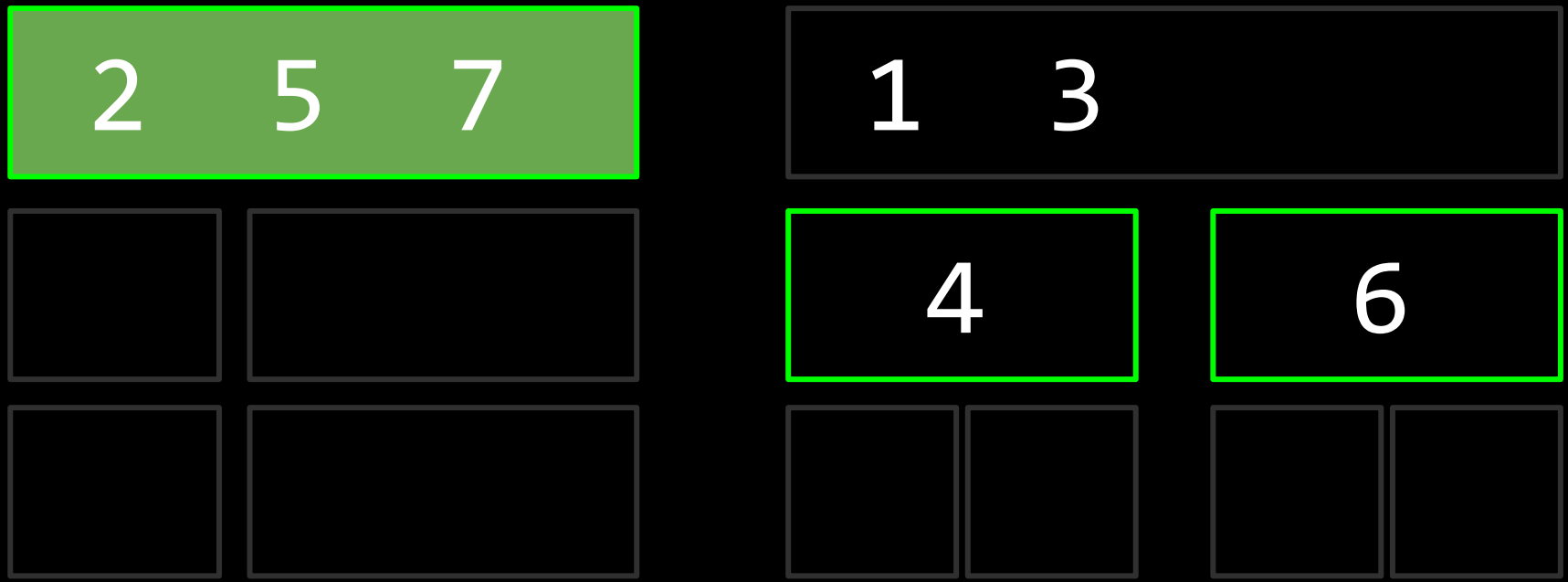
# MERGE SORT!



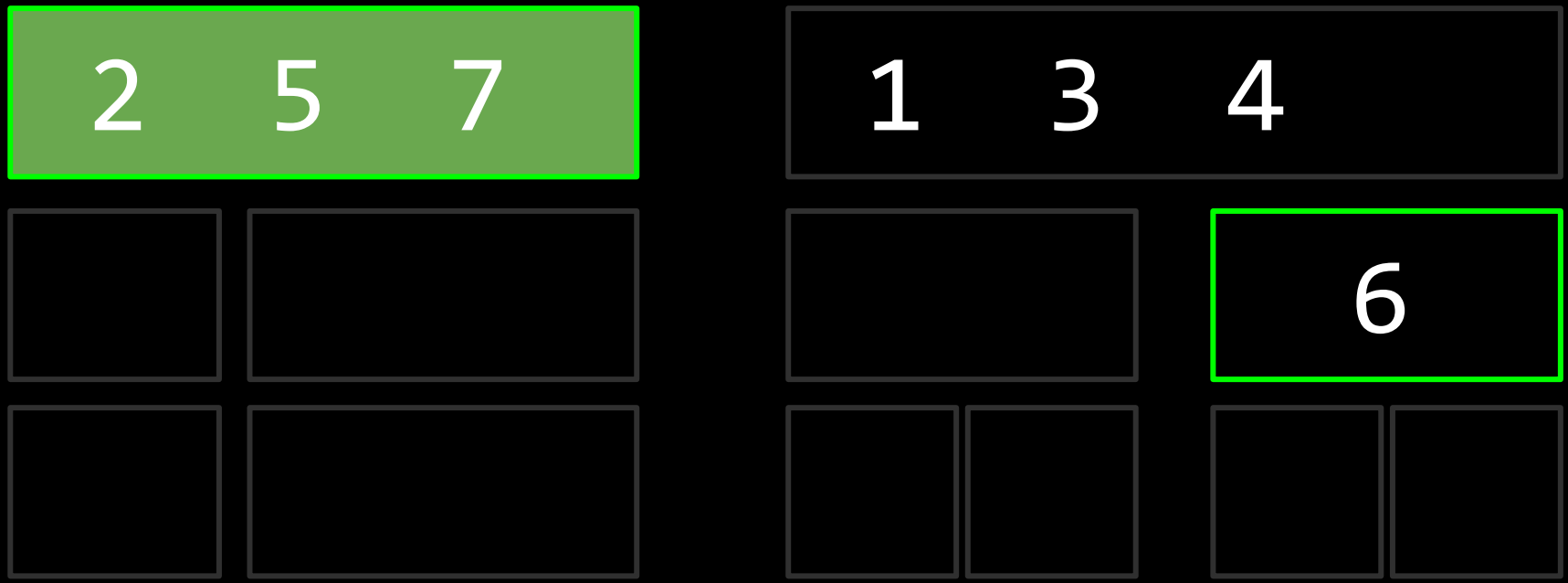
# MERGE SORT!



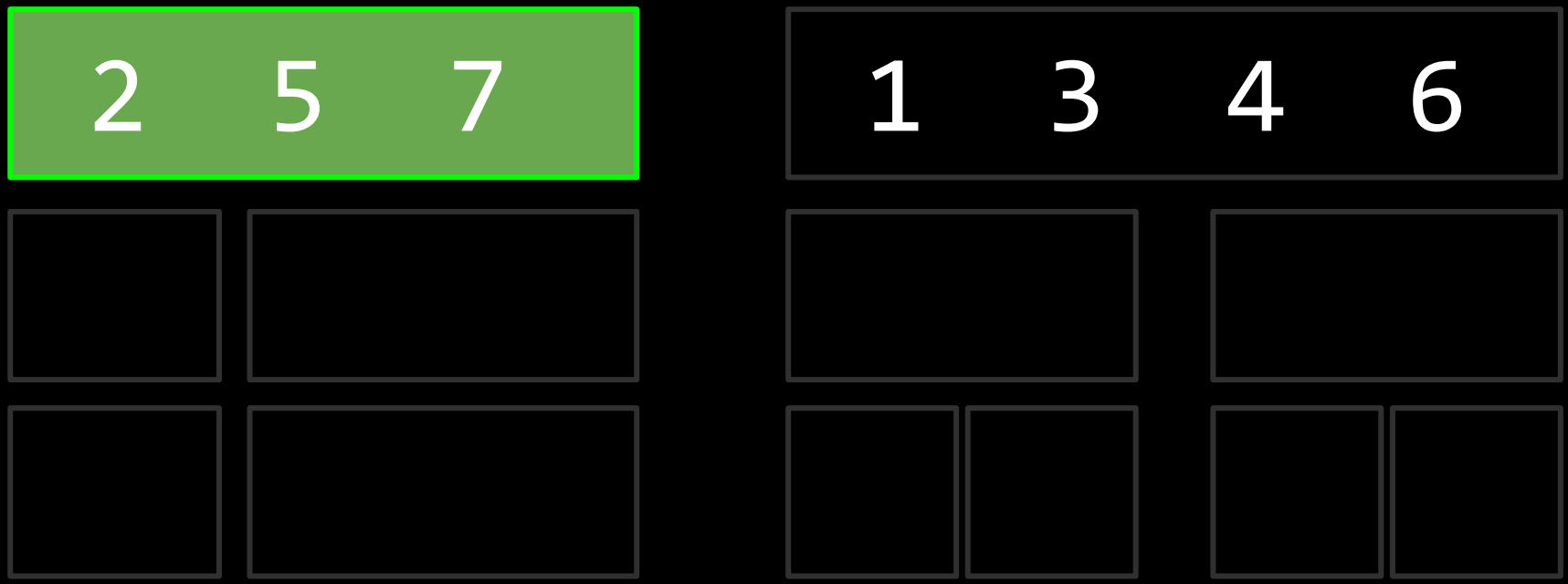
# MERGE SORT!



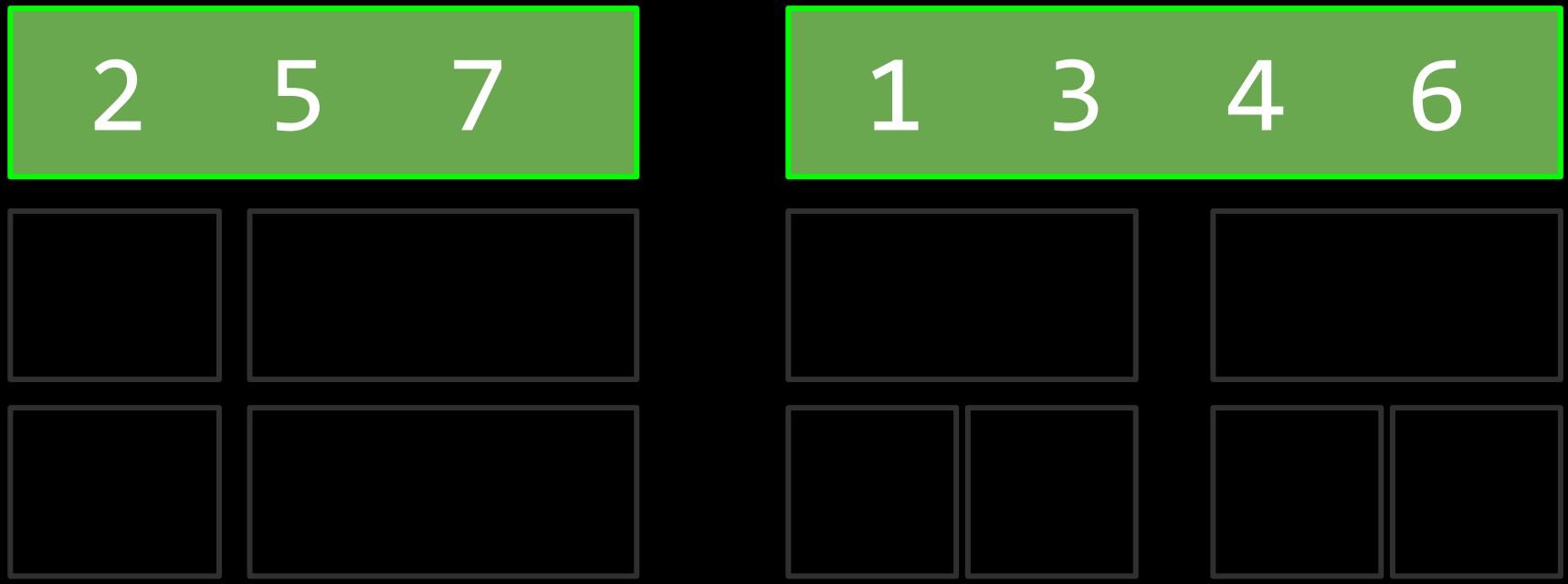
# MERGE SORT!



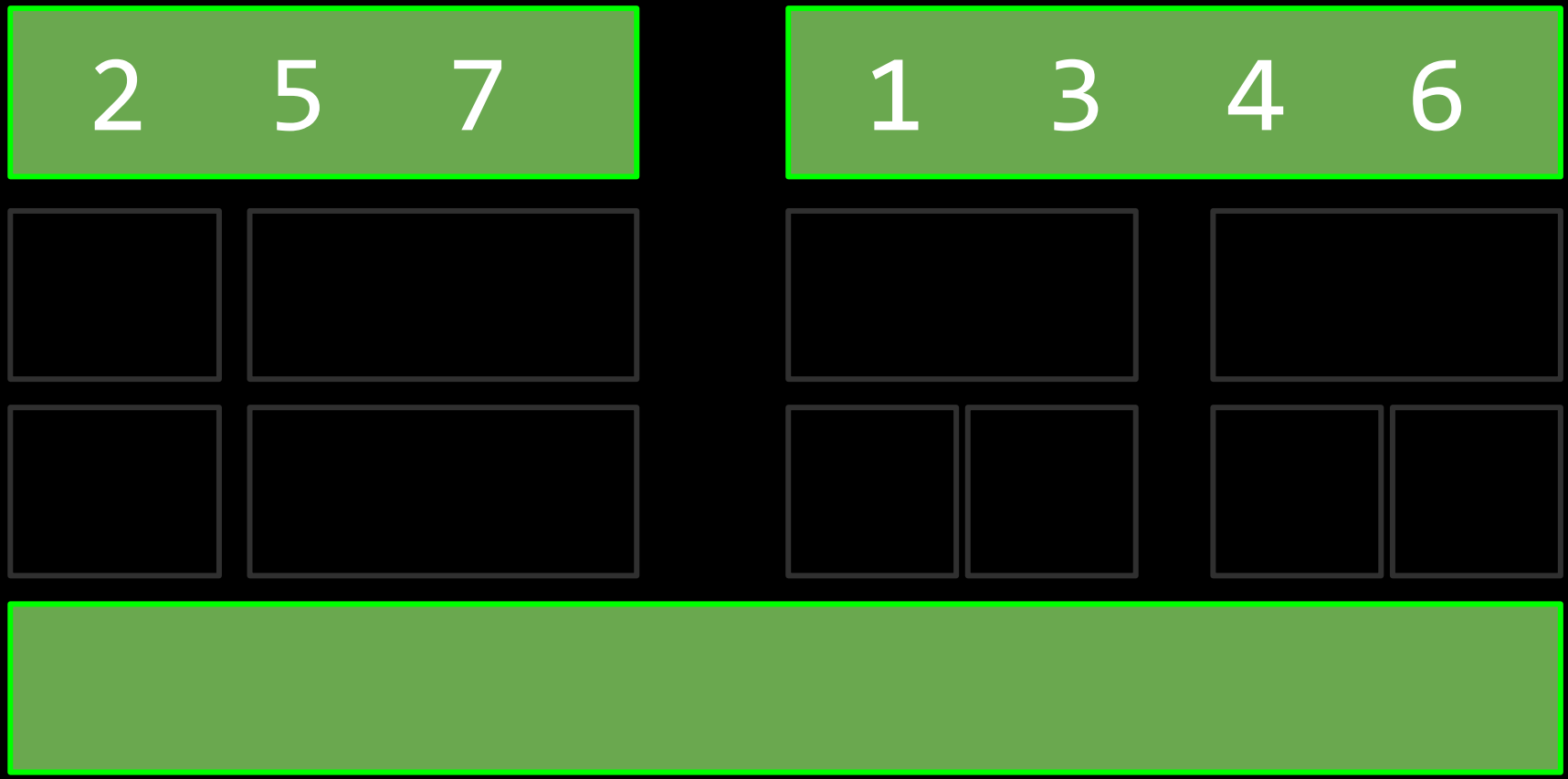
# MERGE SORT!



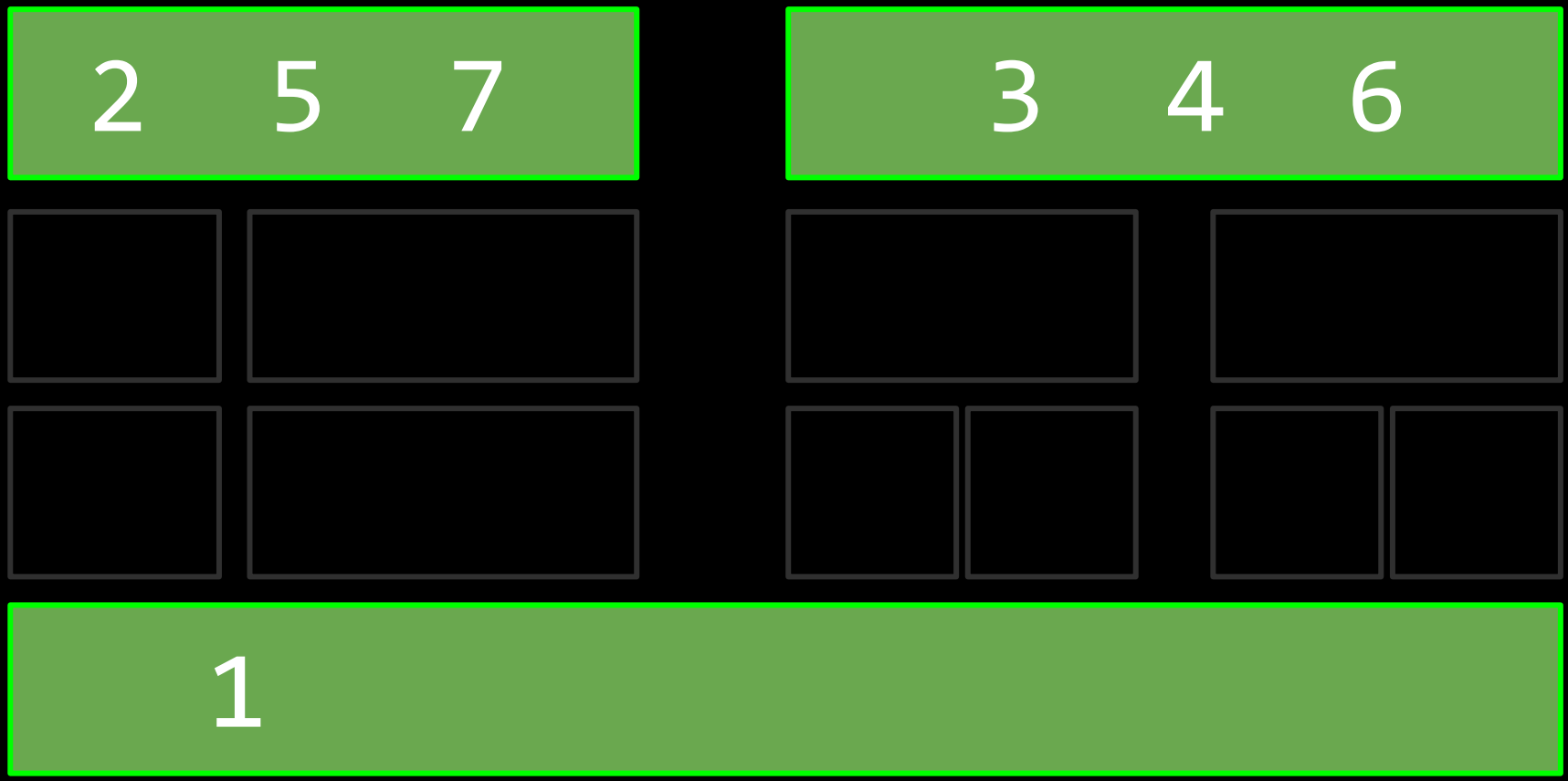
# MERGE SORT!



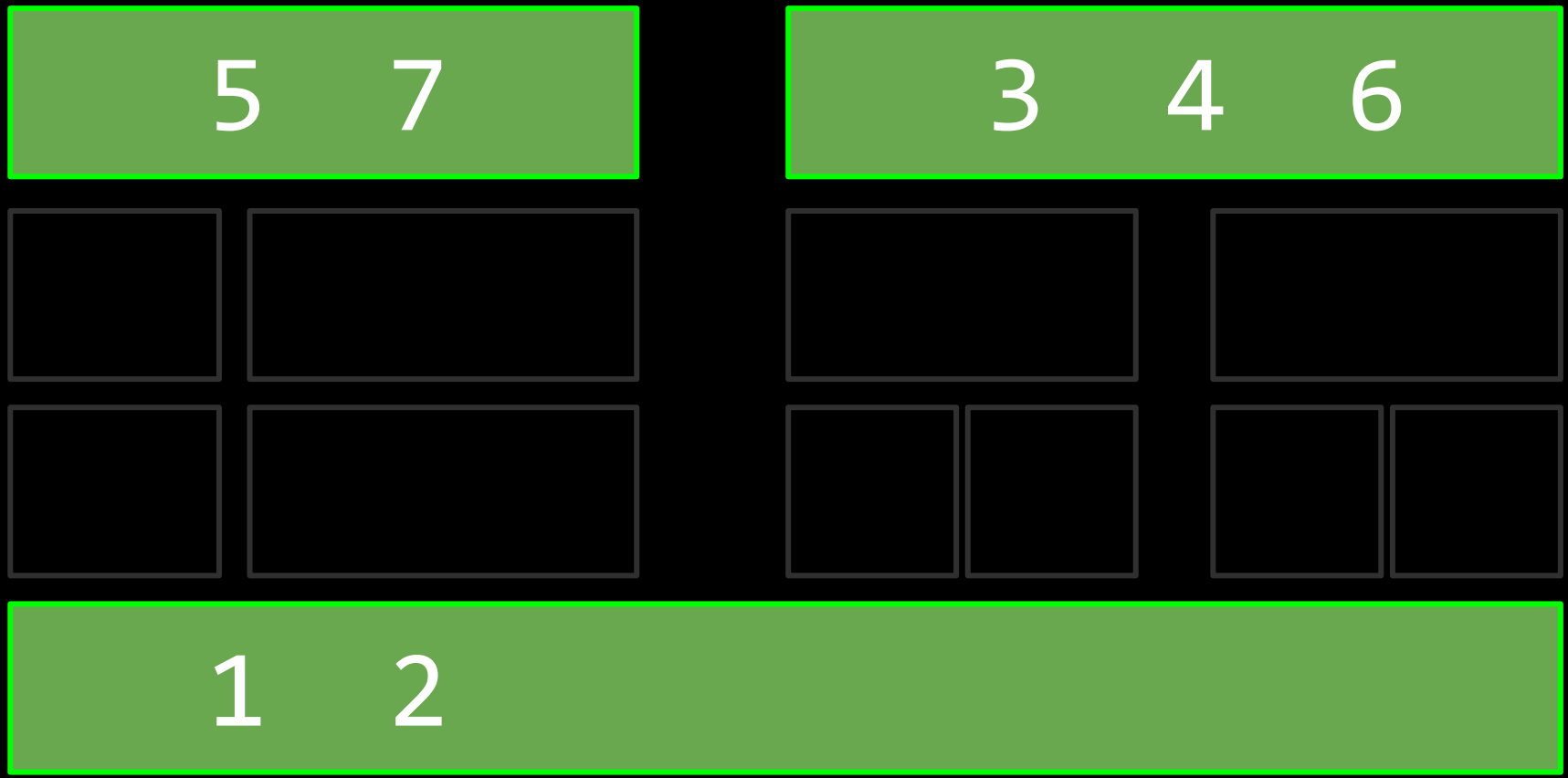
# MERGE SORT!



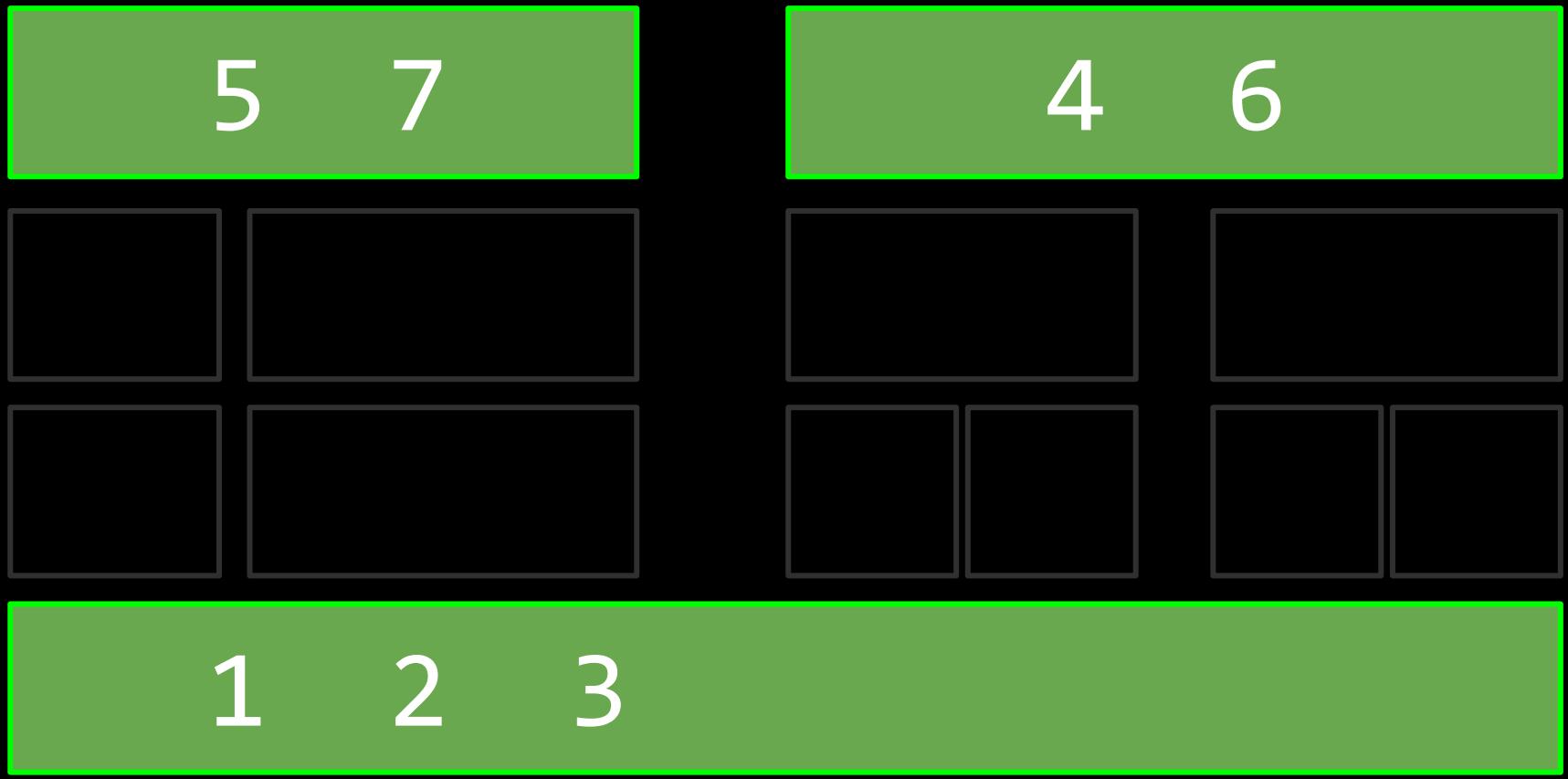
# MERGE SORT!



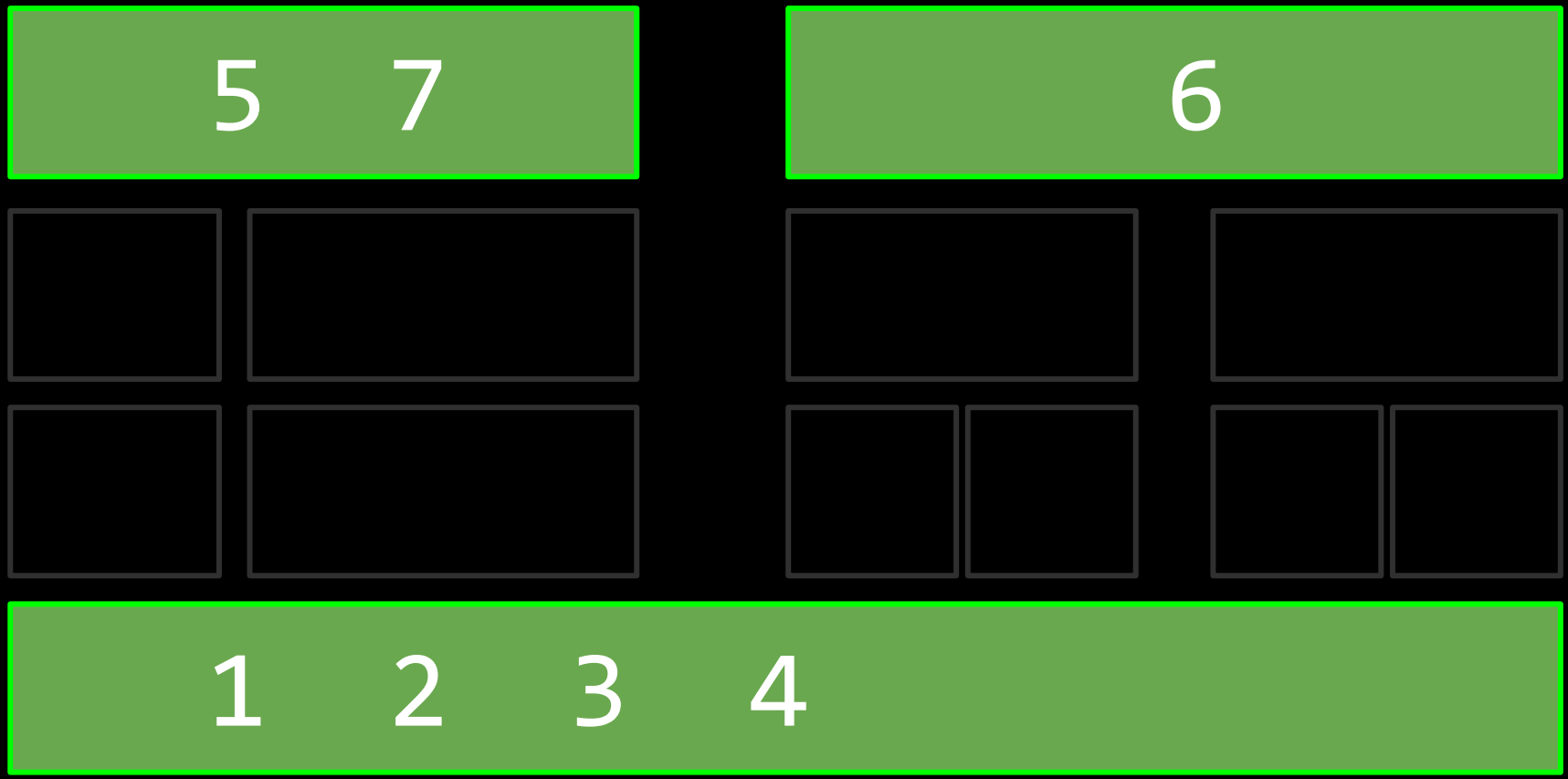
# MERGE SORT!



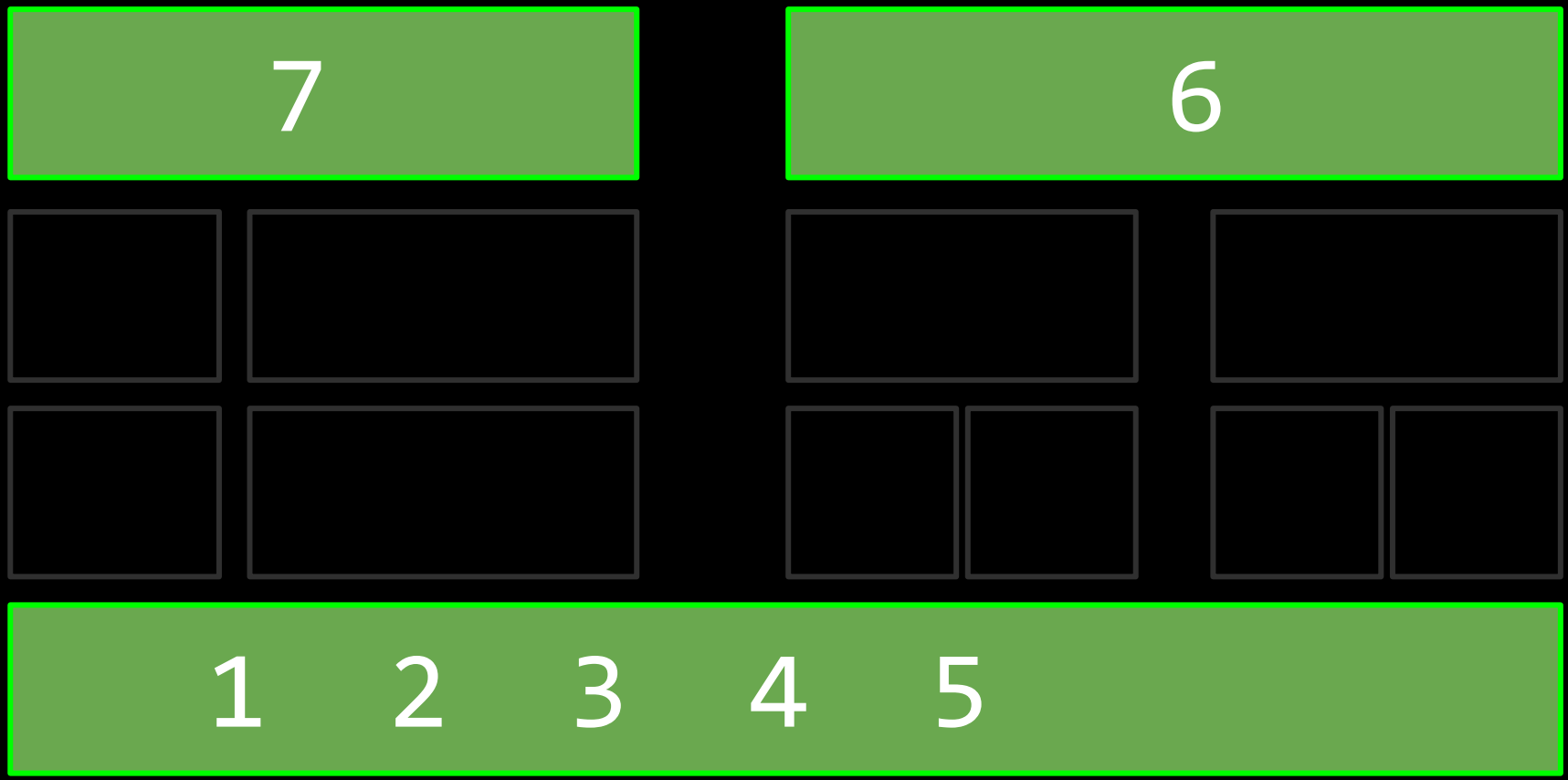
# MERGE SORT!



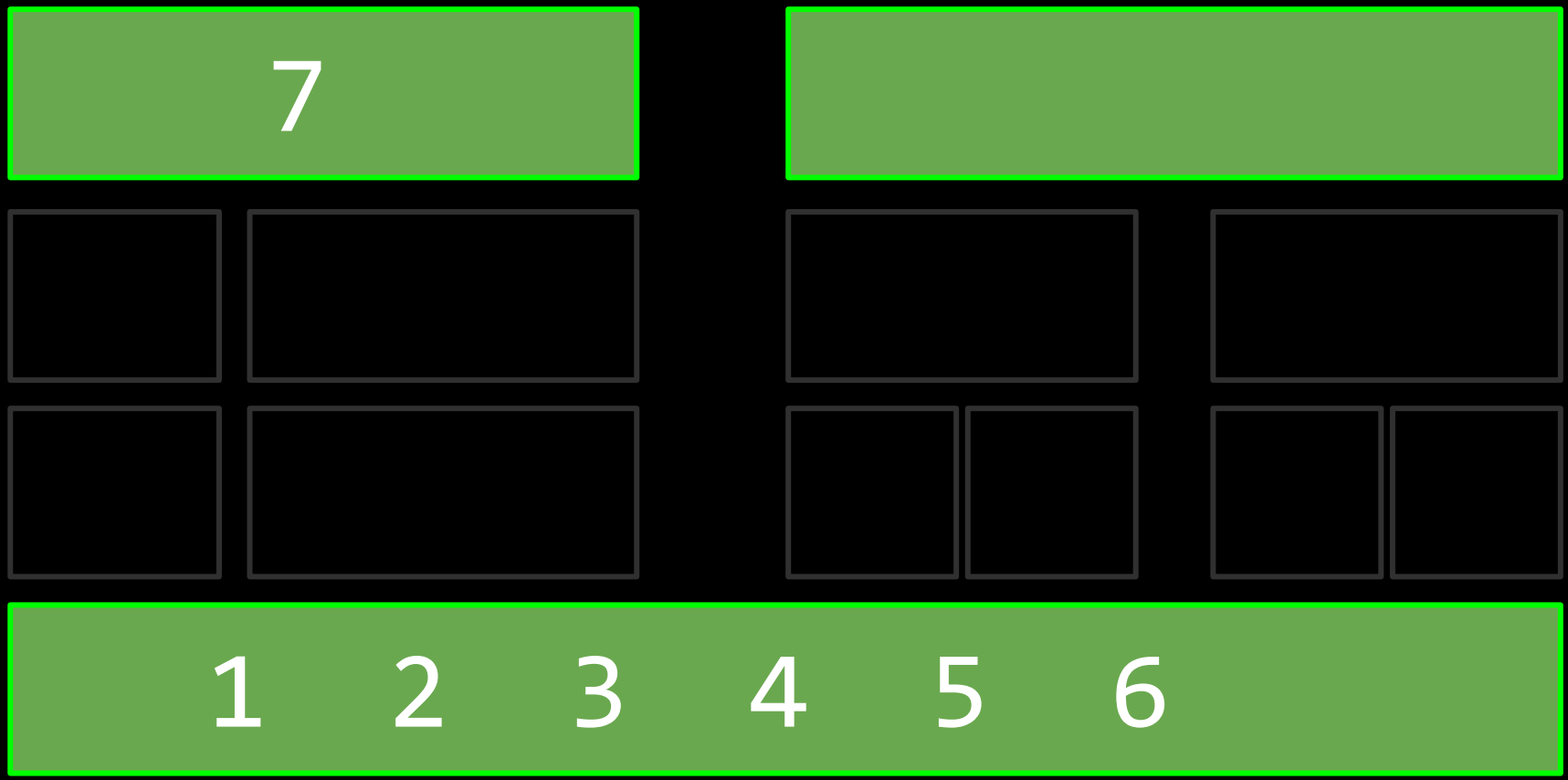
# MERGE SORT!



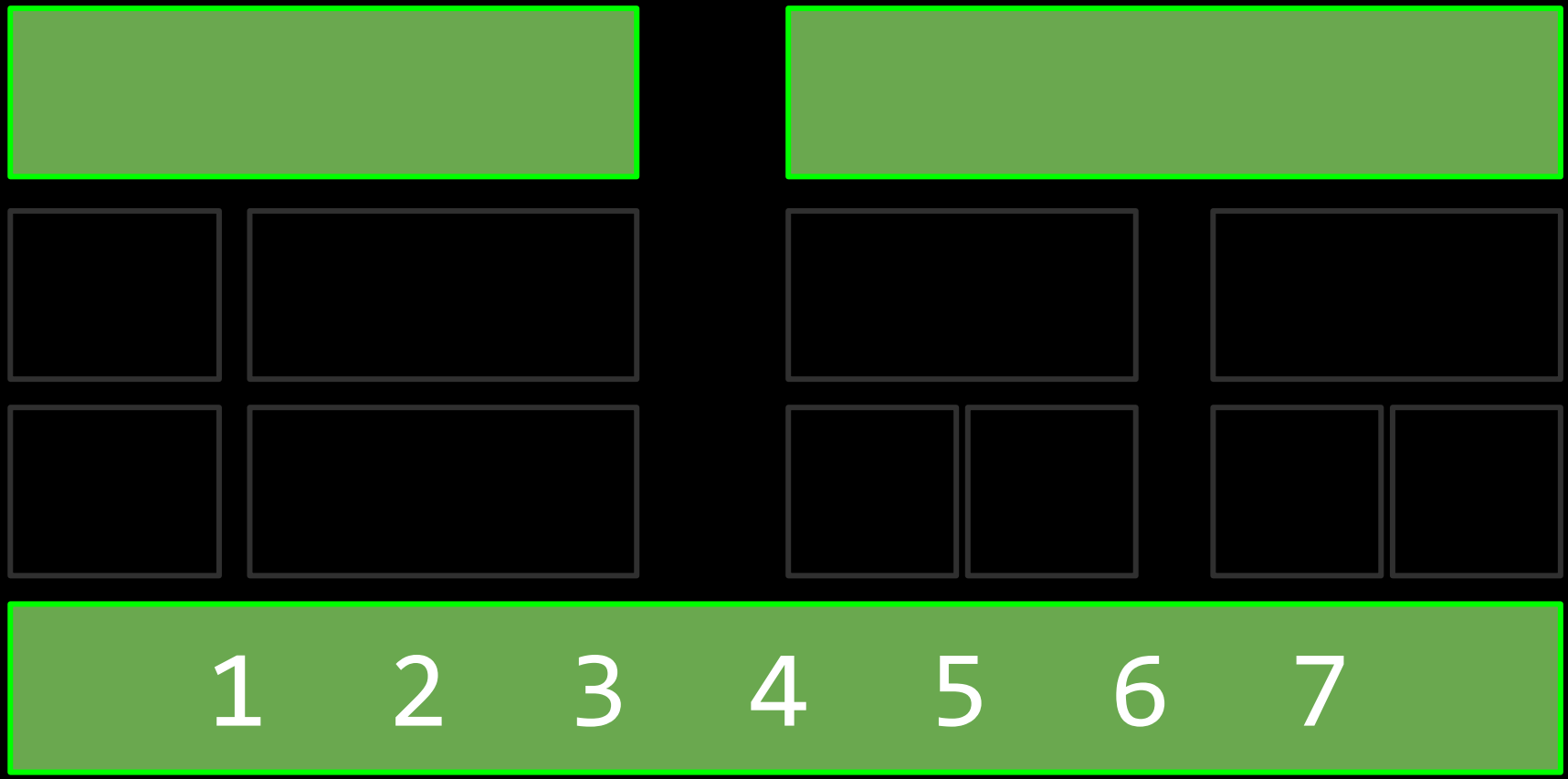
# MERGE SORT!



# MERGE SORT!



# MERGE SORT!



# ✨ RECURSION + MERGE SORT

- If list has only 1 number in it, SORTED ✅
- Sort the left half of list
- Sort the right half of list
- **Merge** the 2 sorted halves
- Takes more **memory**, because of the copying + new arrays

$O(n^2)$

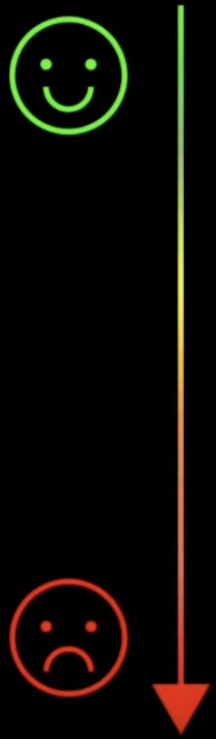
$O(n \log n)$ merge sort

$O(n)$

$O(\log n)$

$O(1)$

# Classes of Running Times



Constant - $O(1)$

Logarithmic - $O(\log N)$

Linear - $O(N)$

Linearithmic - $O(N \log N)$

Polynomial - $O(N^2), O(N^3), O(N^4)$, etc.

Exponential - $O(2^N), O(3^N), O(4^N)$, etc.

# MERGE SORT!