# Algorithmic Design

*Lawrence Snyder*
*University of Washington, Seattle*

# Algorithms

- Def. *An algorithm is a precise, systematic process for an agent to produce a specified result*
- Five properties characterize algorithms
  - **Input specified** – tell form and amount of input required
  - **Output specified** – tell form and amount of output produced
  - **Definiteness** – say explicitly what to do & in what order
  - **Effectiveness** – operations within agent's abilities
  - **Finiteness** – will stop and give an answer or say "none"
  - Programs are algorithms

# Algorithms At Many Levels of Detail

- The binary code computers execute are algorithms
- Software developers create algorithms all the time
  - Using languages like C, Processing, JavaScript, etc.
  - A compiler (it's a translator) converts to binary code
- These cases specify computation in complete detail because computers are clueless
- But at other levels the agent is a person …

# Google Query Algorithm

- In their paper Larry Page and Sergey Brin gave their algorithm for processing a Google query

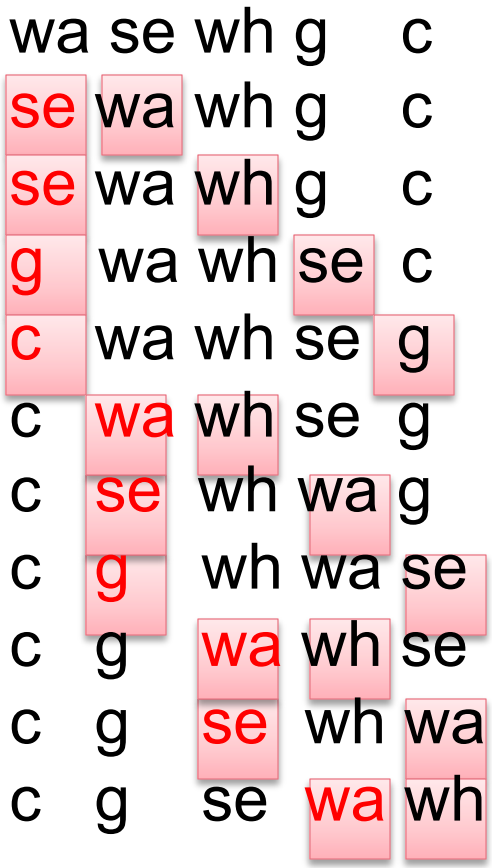
1. Parse the query.
2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7. If we are not at the end of any doclist go to step 4.
8. Sort the documents that have matched by rank and return the top k.

Figure 4. Google Query Evaluation

- Algorithms are "given" at many levels of detail – it depends on what agent needs

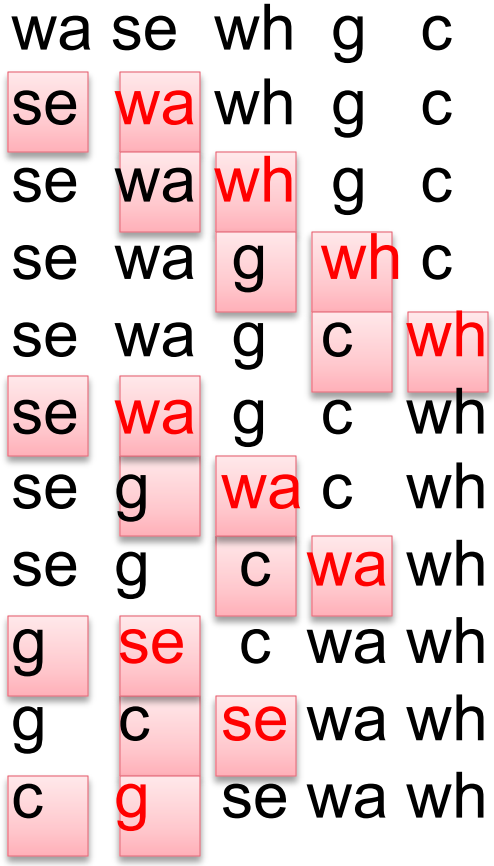
# Many Alternative Algorithms …

- There are always different algorithms to pick
- Consider sorting …
  - Put items in order
  - Exchange Sort for
    - walrus
    - seal
    - whale
    - gull
    - clam
  - Compare with every value following



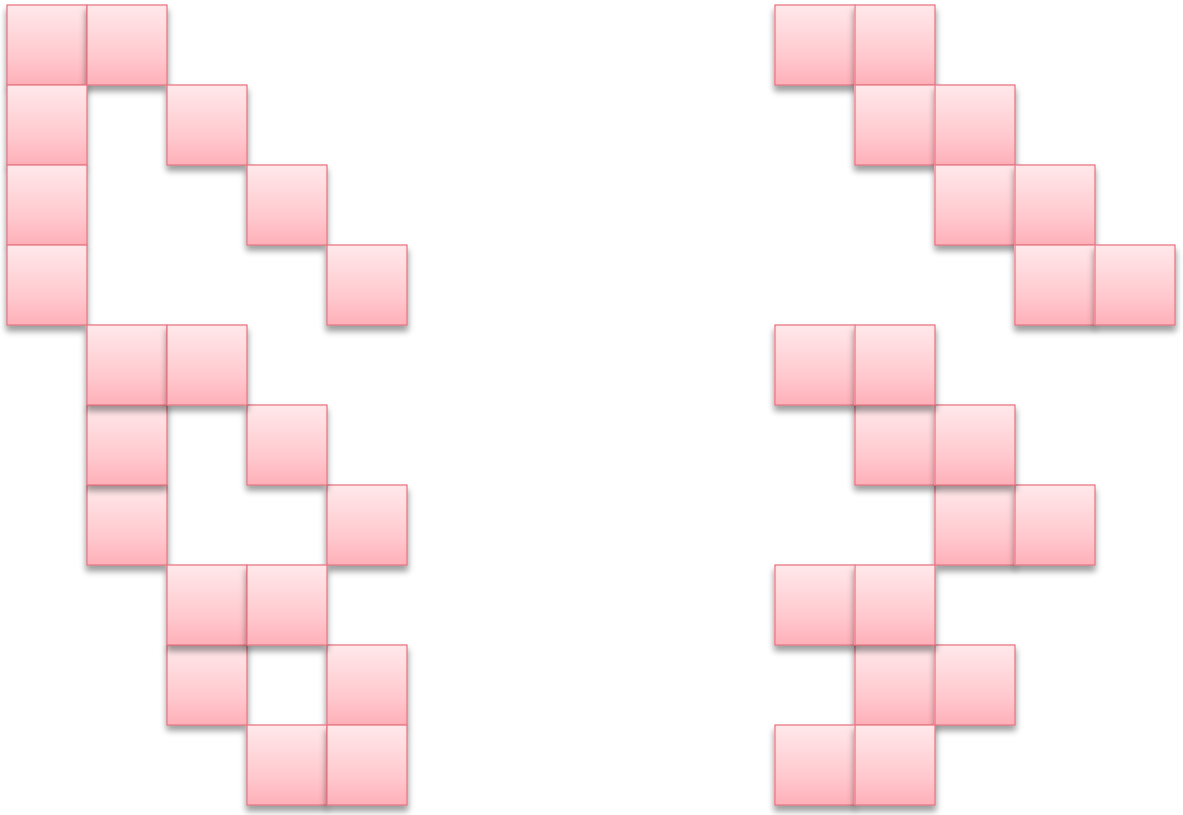
# Now Consider Bubble Sort…

- Bubble Sort for
  - walrus
  - seal
  - whale
  - gull
  - clam
- Compare in pairs, letting larger values "bubble up"



# The Algorithms Are Different

- The two algorithms take the same amount of time, but they are different as we see from the patterns of their comparisons

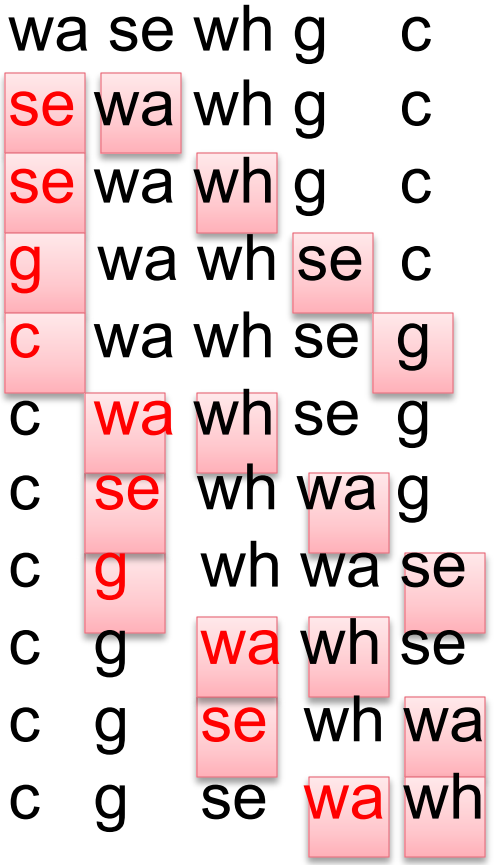


# Key Question for Today & Always

- How do we know that the algorithms work?
  - Developing algorithms is not just thinking them up
  - It is also reasoning through why they work … you need to know *why* explicitly enough to tell someone else
- Let's see how to do that

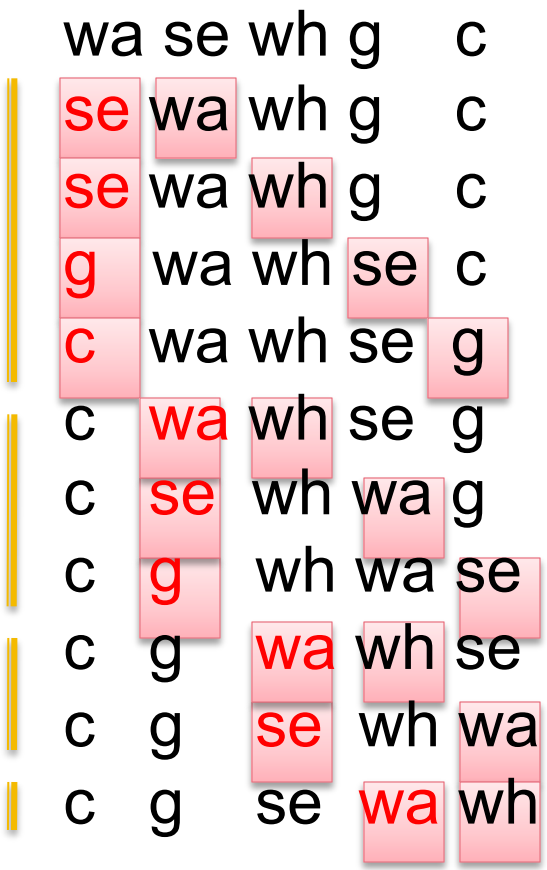
# Why Does Exchange Sort Work

- Why do you think it sorts?



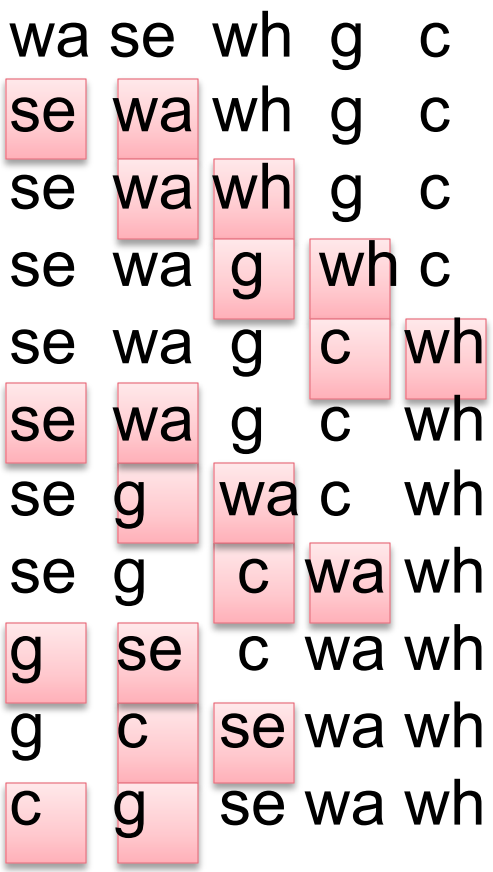
# Why Does Exchange Sort Work?

- Why do you think it sorts?
  - There are several passes through the data with *leading item unsorted* fixed (marked with lines)
- Notice this property: After the pass, the *leading item* must be the smallest of all processed on the pass



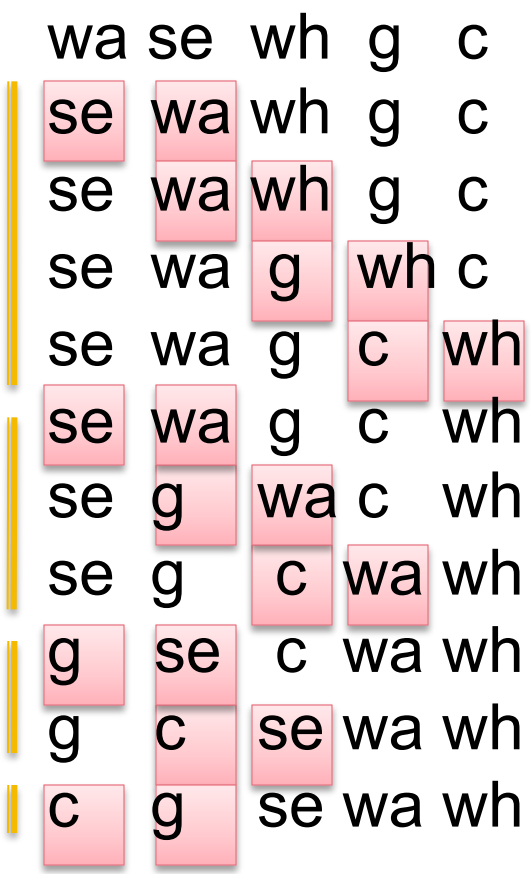
# Why Does Bubble Sort Work?

- Why do you think it sorts?



# Why Does Bubble Sort Work?

- Why do you think it sorts?
  - There are several passes through the data comparing pairs of data (marked w/ lines)




- Notice this property: After the pass, pair compares will locate the next largest item and bubble it into position

# Sorting Analysis

- What is the complexity of the sorting algorithms that we just looked at?
- Often has two parts: get item in order; repeat
  - Exchange – interchange to put smallest earlier
  - Bubble – compare adjacent values, push largest further on
- How many times do we do that
  - Be careful, not exactly how many times but what "order" …

# Exchange Sort Work

- This exchange sort needs

4 + 3 + 2 + 1 = 10 compares for 5 items

- Generally for n items

(n-1) + (n-2) + … + 2 + 1

= n(n-1)/2

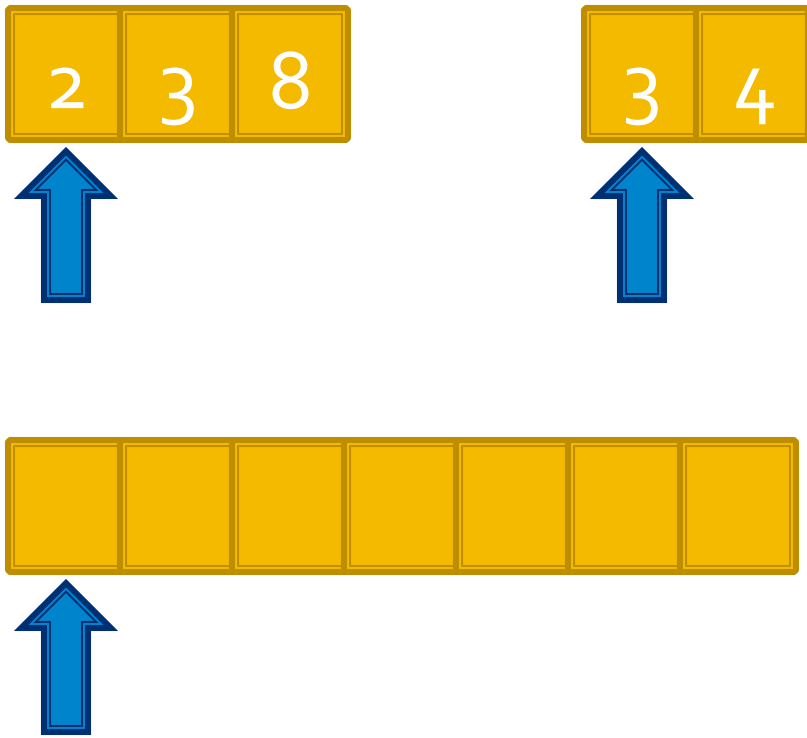
= 1/2 ($n^2$ –n)

| | | | | |
|---|---|---|---|---|
| wa | se | wh | g | c |
| se | wa | wh | g | c |
| se | wa | wh | g | c |
| g | wa | wh | se | c |
| c | wa | wh | se | g |
| c | wa | wh | se | g |
| c | se | wh | wa | g |
| c | g | wh | wa | se |
| c | g | wa | wh | se |
| c | g | se | wh | wa |
| c | g | se | wa | wh |

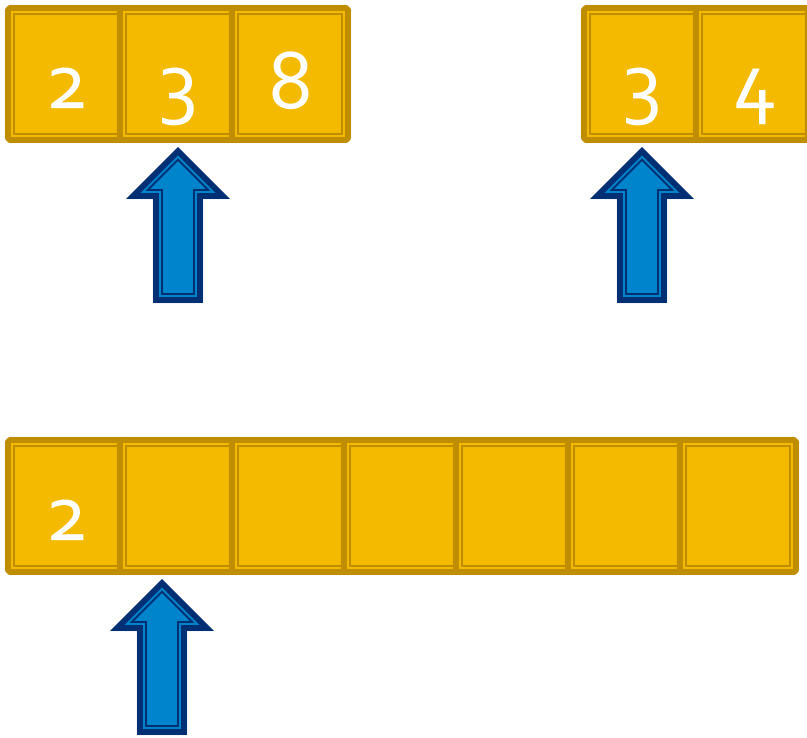
# Merging

- Merging two sorted arrays into a single sorted array is straight forward

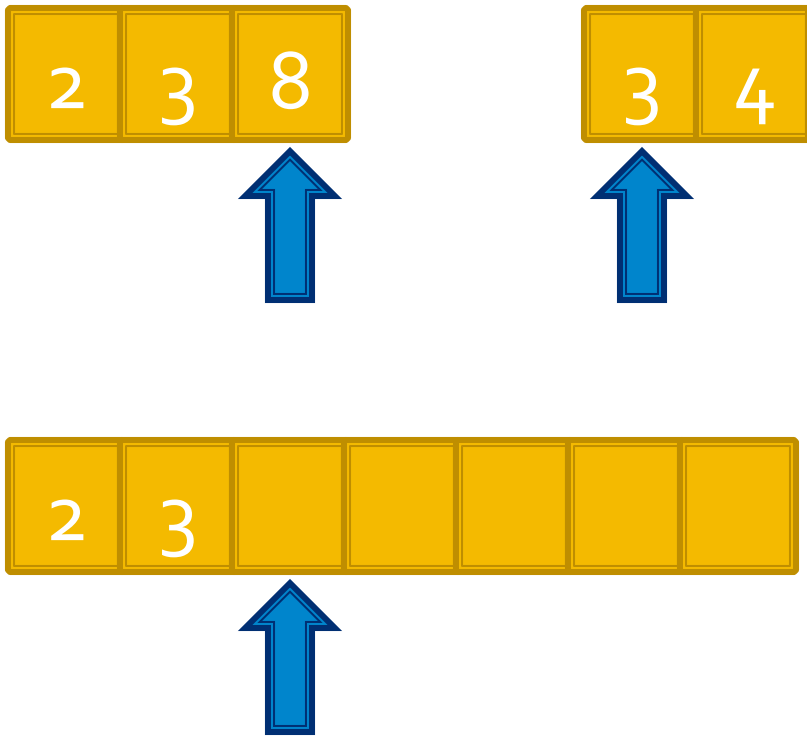
# Merging



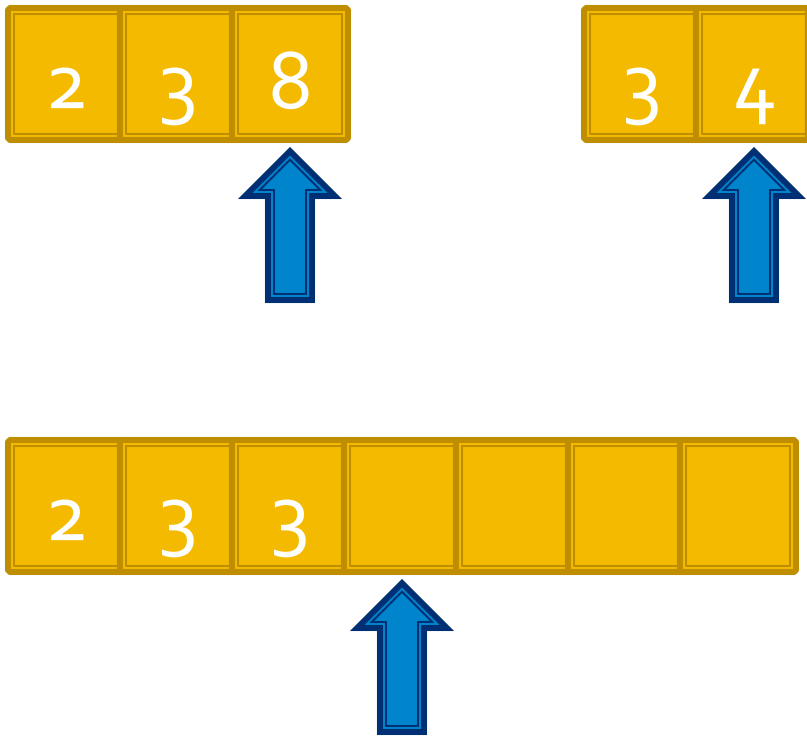
# Merging



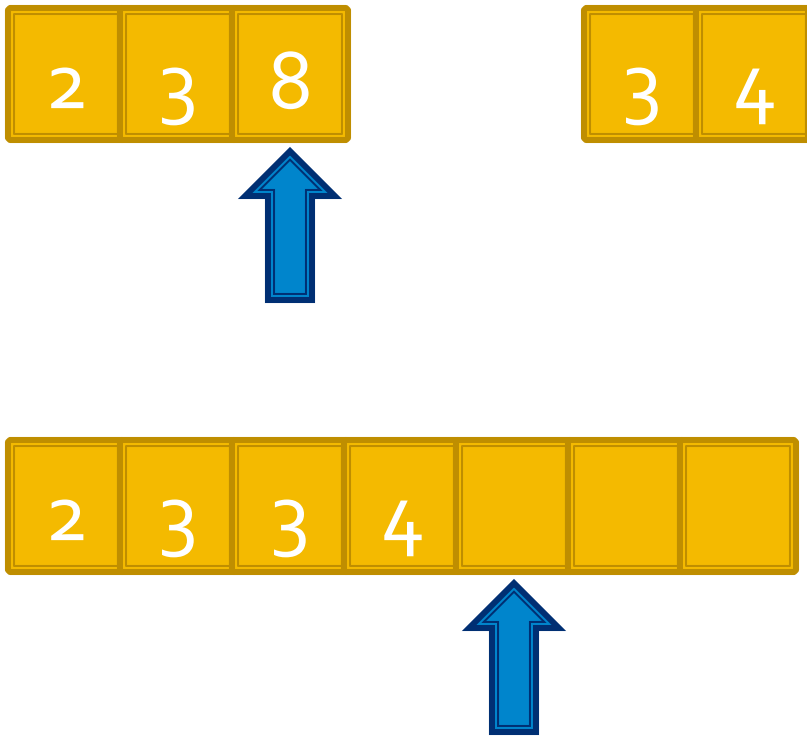
# Merging



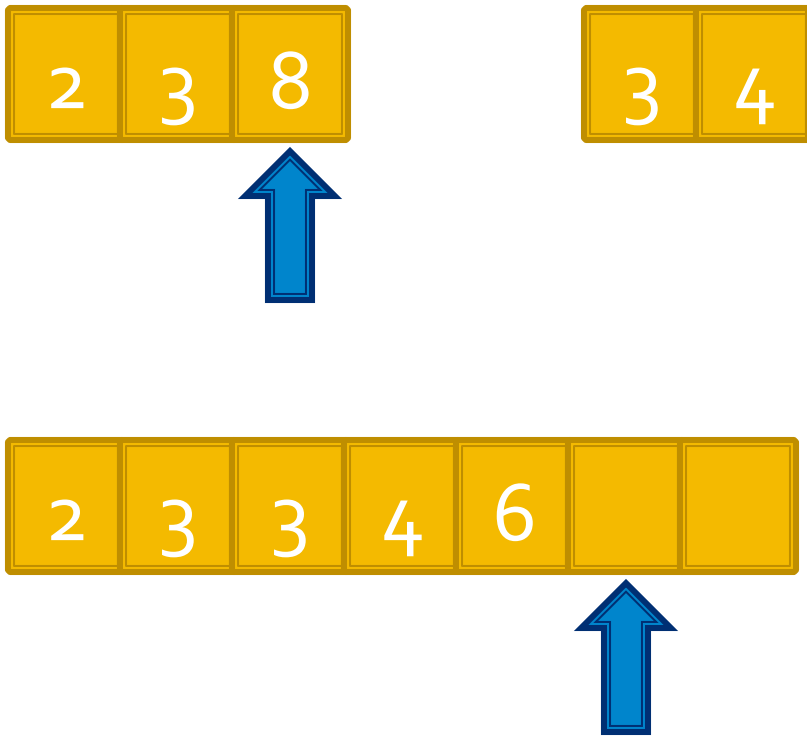
# Merging



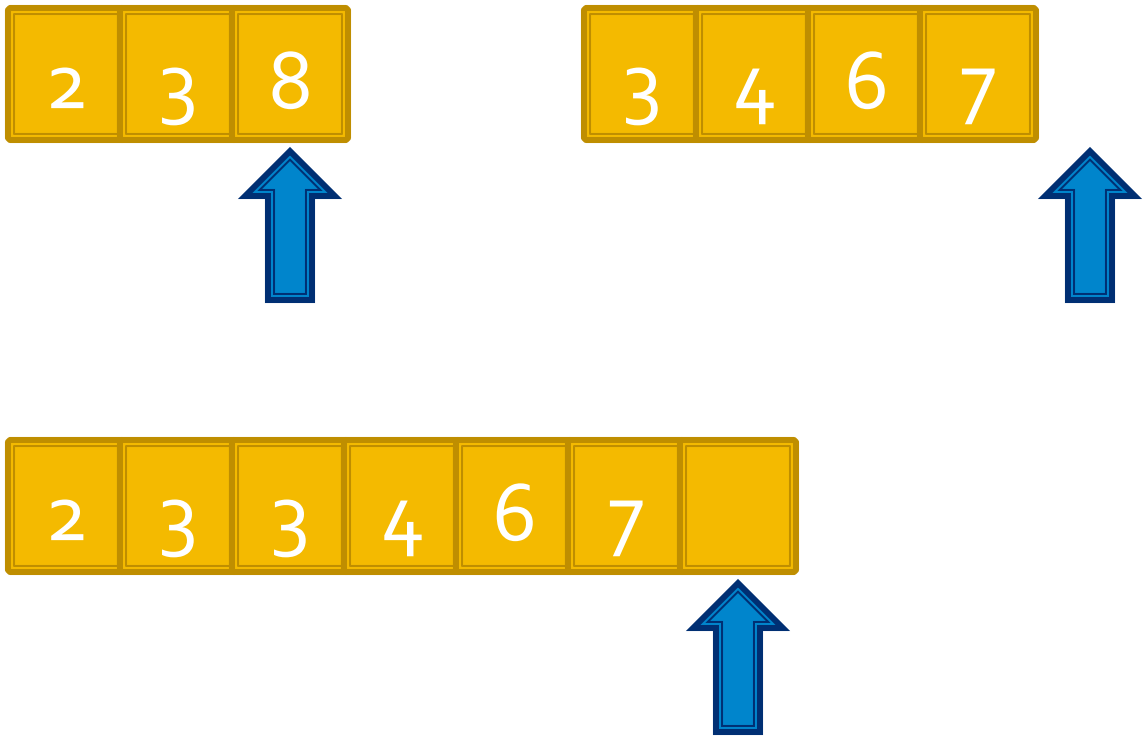
# Merging



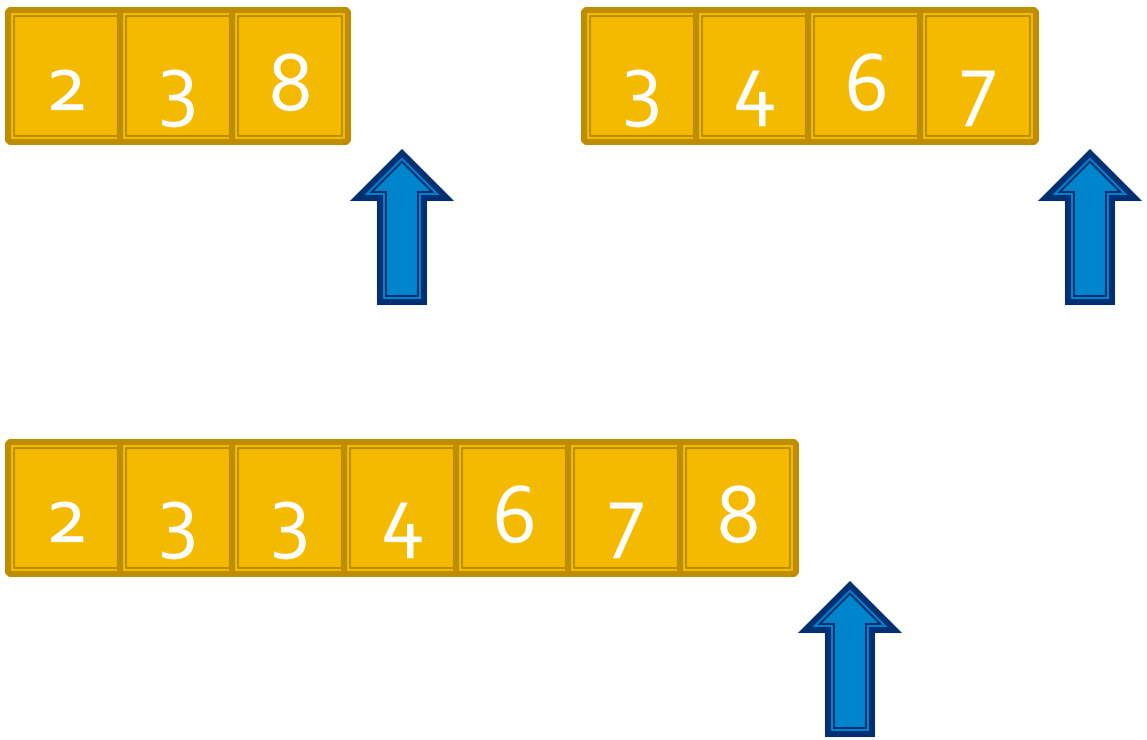
# Merging



# Merging

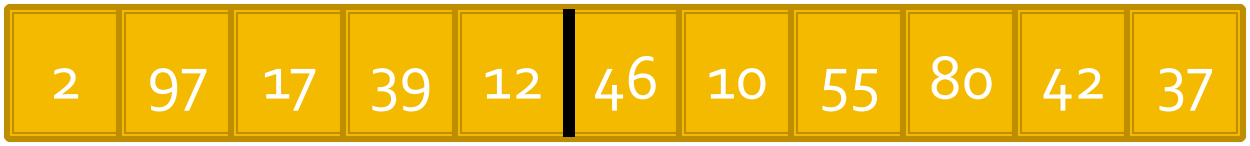


# Merging

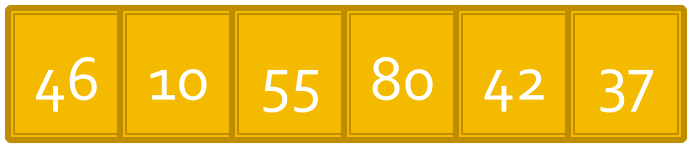


# Merge Sort

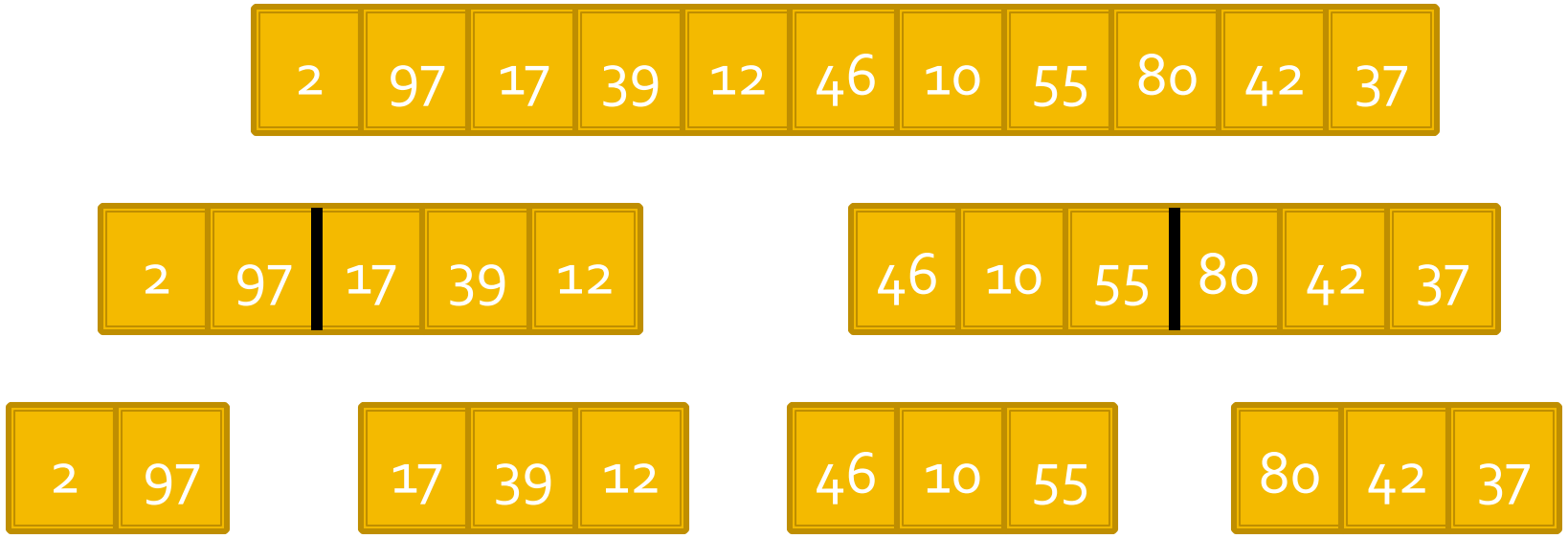
# Merge Sort



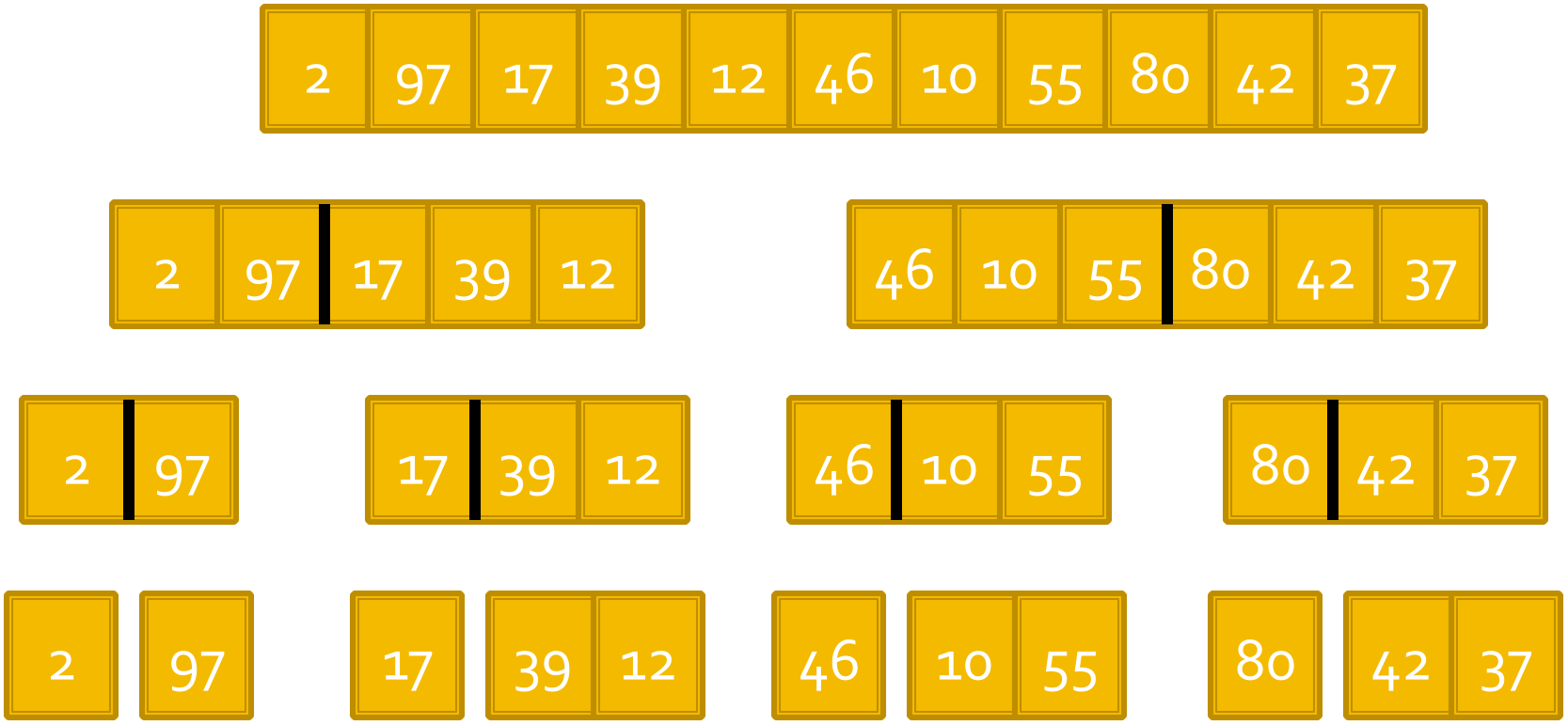




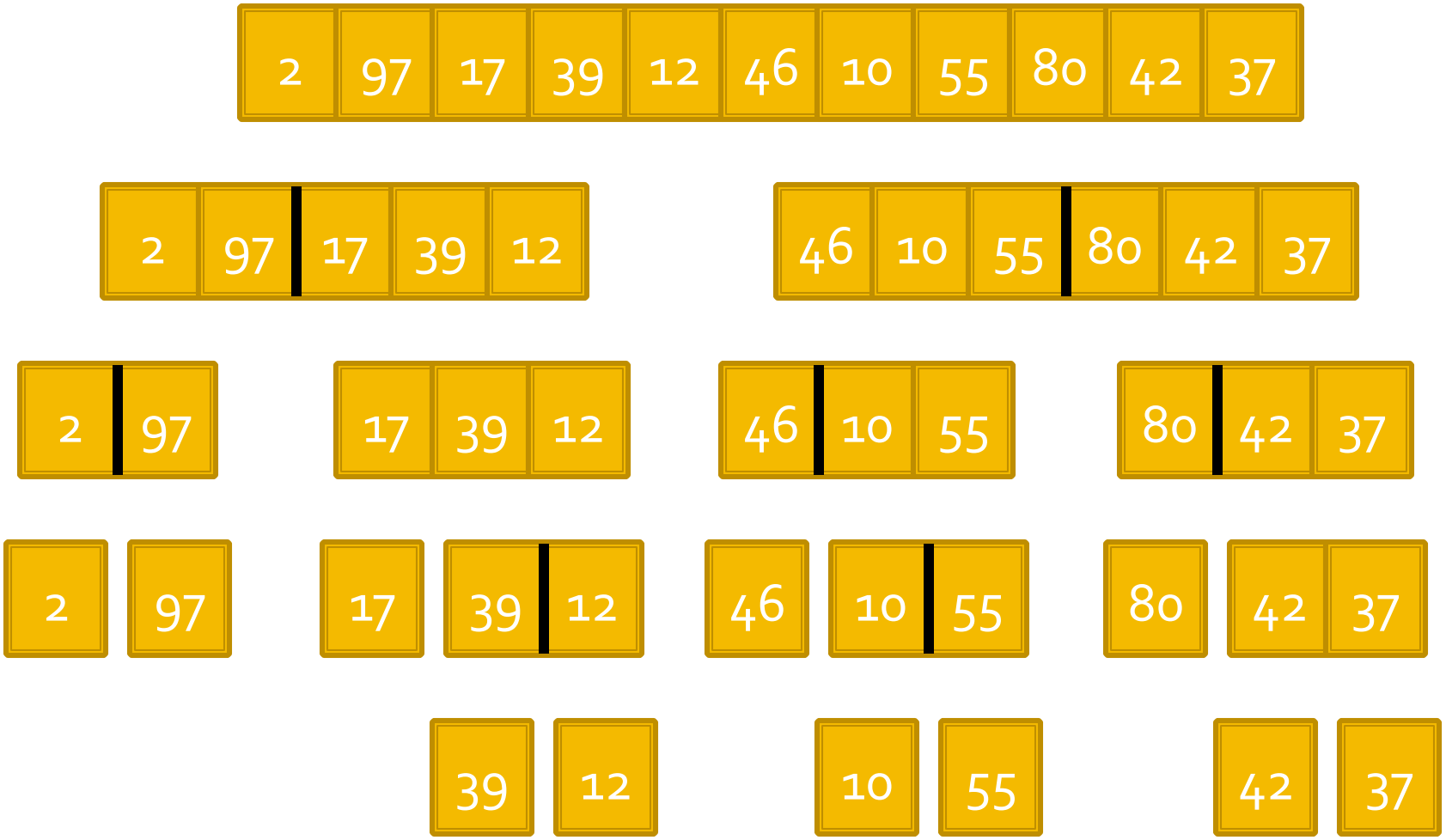
# Merge Sort


2 97 17 39 12 46 10 55 80 42 37
2 97 17 39 12
46 10 55 80 42 37
2 97
17 39 12
46 10 55
80 42 37

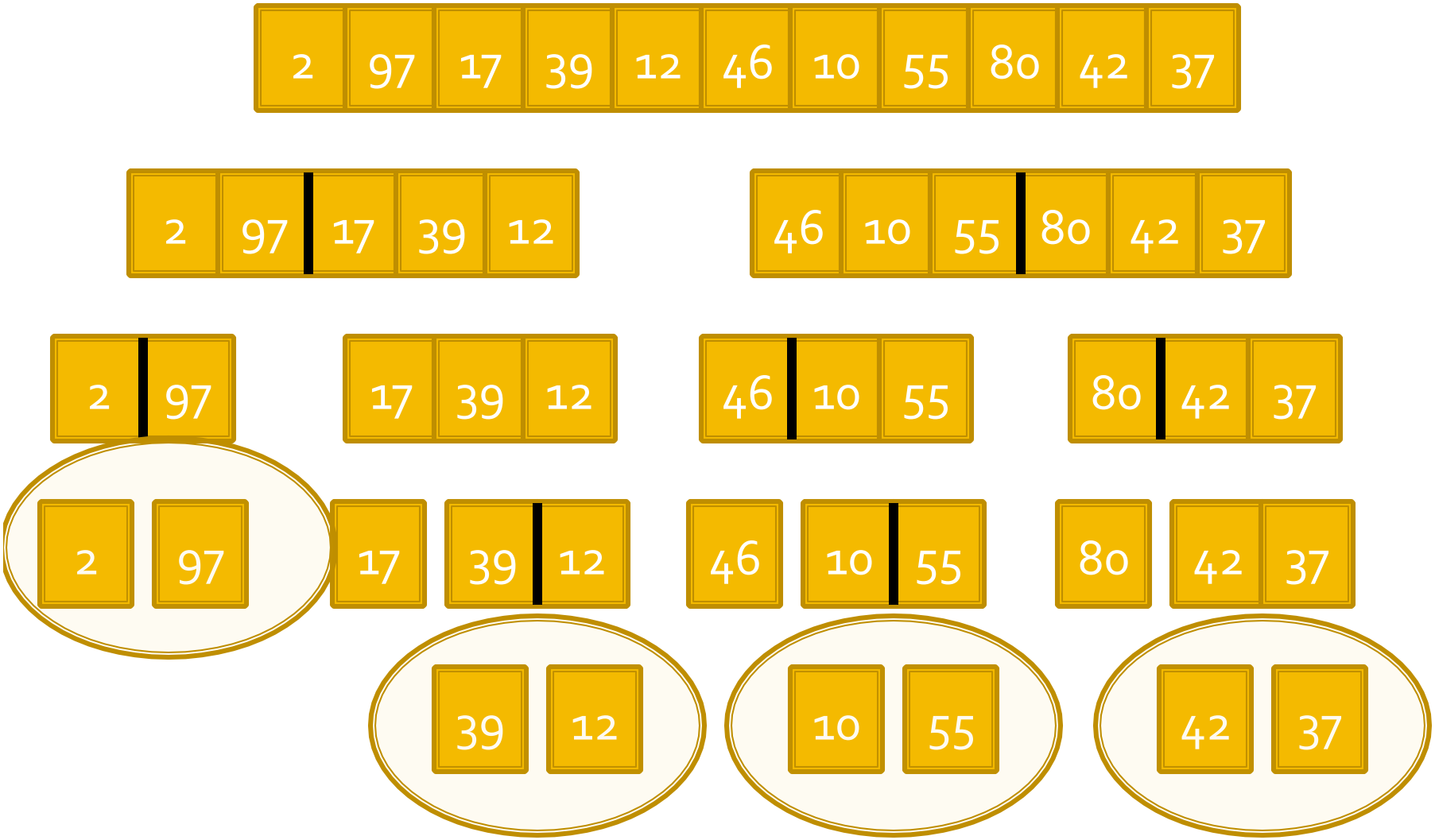
# Merge Sort


2 97 17 39 12 46 10 55 80 42 37
2 97 17 39 12
46 10 55 80 42 37
2 97
17 39 12
46 10 55
80 42 37
2
97
17
39 12
46
10 55
80
42 37

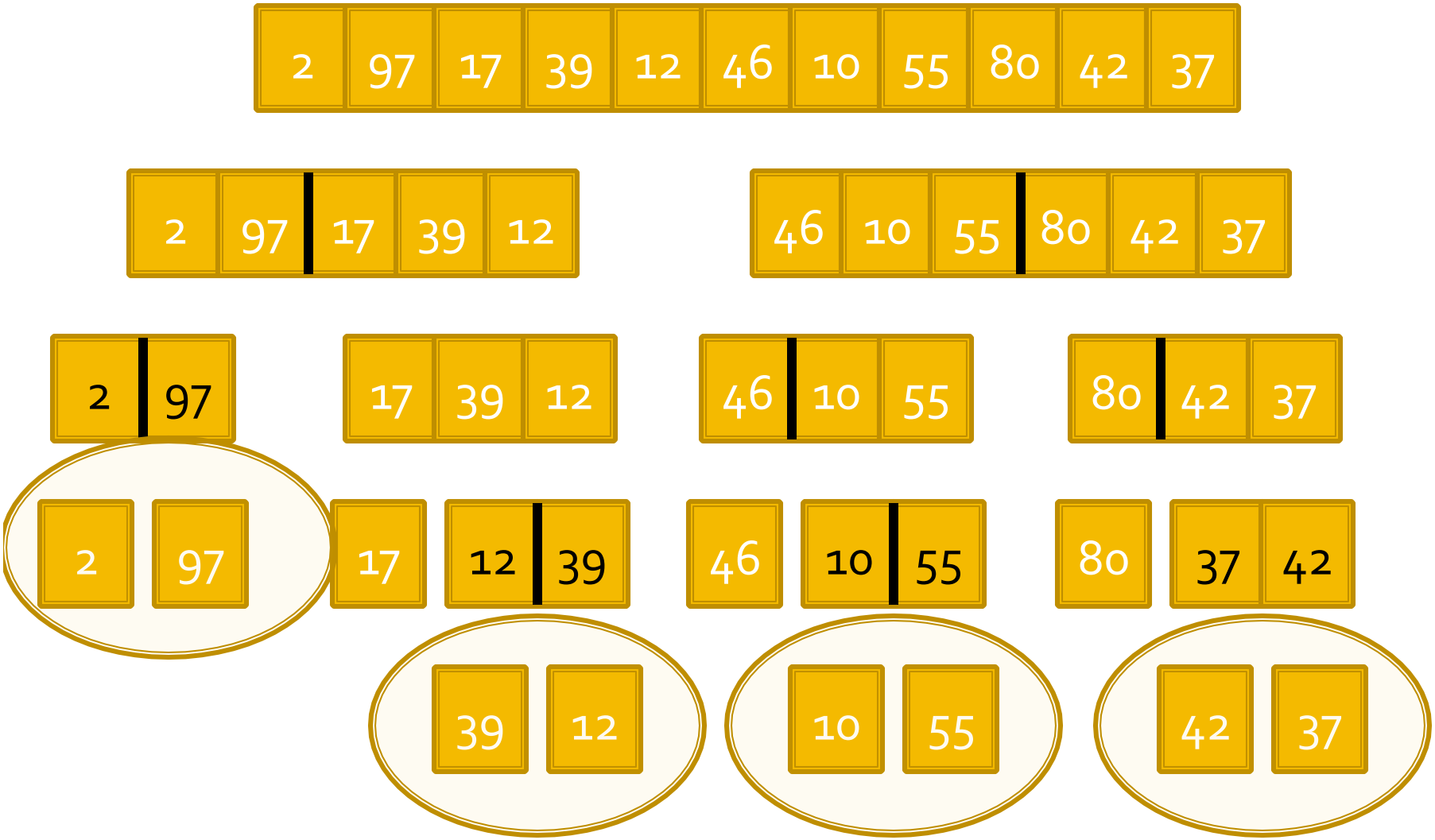
# Merge Sort


2 97 17 39 12 46 10 55 80 42 37
2 97 17 39 12
46 10 55 80 42 37
2 97
17 39 12
46 10 55
80 42 37
2
97
17
39 12
46
10 55
80
42 37
39
12
10
55
42
37

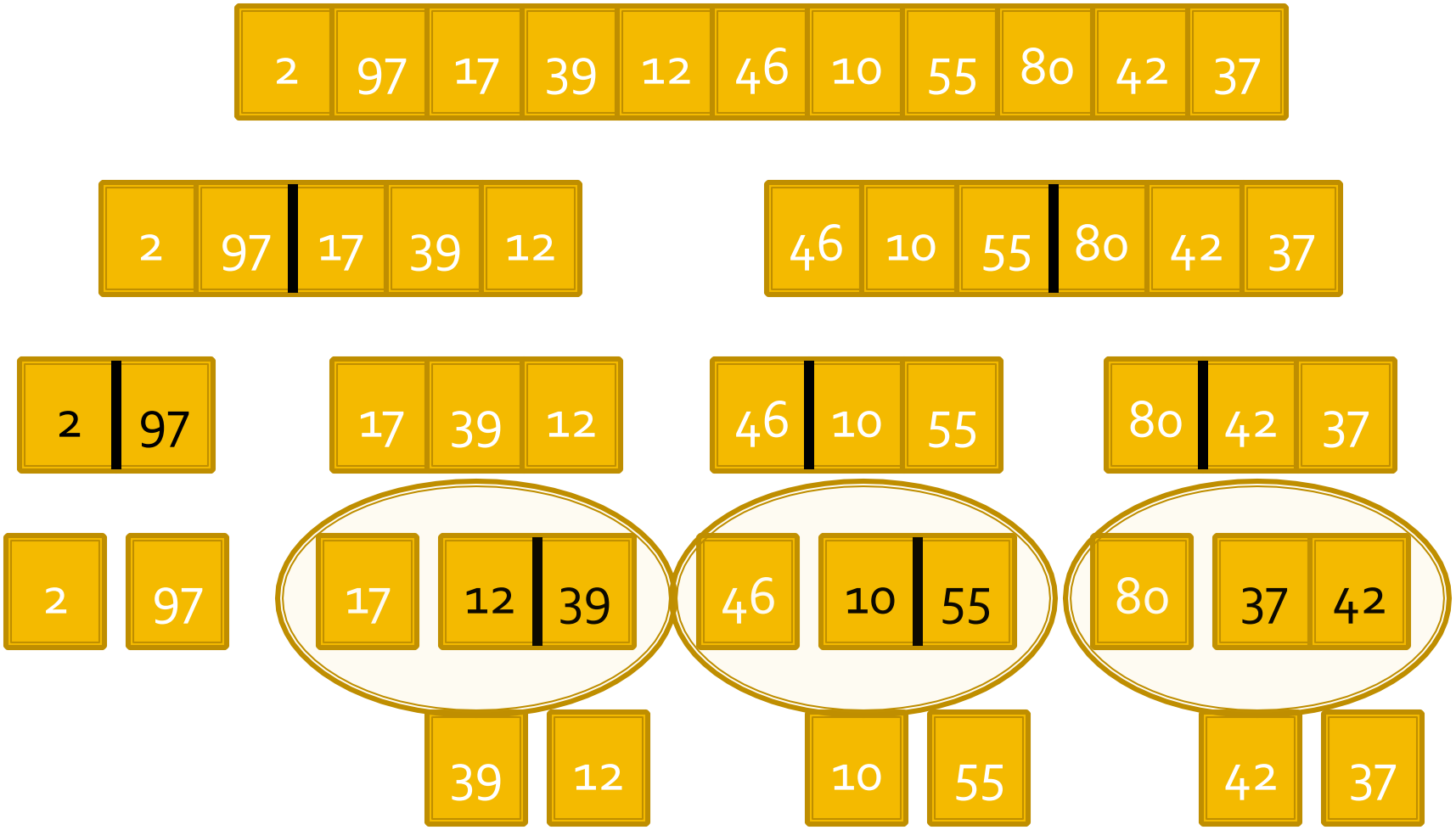
# Merge Sort


2 97 17 39 12 46 10 55 80 42 37
2 97 17 39 12
46 10 55 80 42 37
2 97
17 39 12
46 10 55
80 42 37
2 97
17
39 12
46
10 55
80
42 37
39 12
10 55
42 37

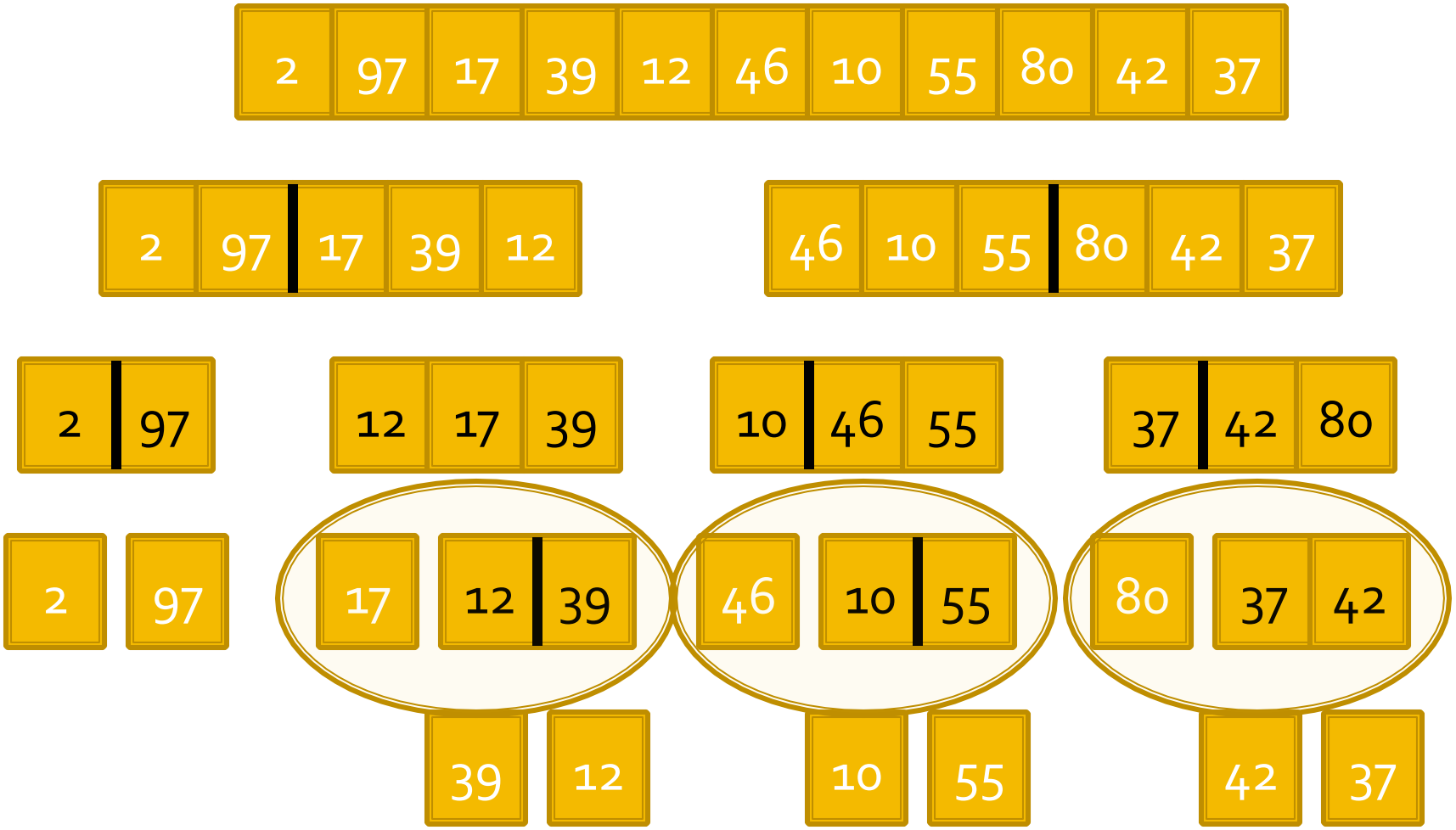
# Merge Sort


2 97 17 39 12 46 10 55 80 42 37
2 97 17 39 12
46 10 55 80 42 37
2 97
17 39 12
46 10 55
80 42 37
2 97
17
12 39
46
10 55
80
37 42
39 12
10 55
42 37

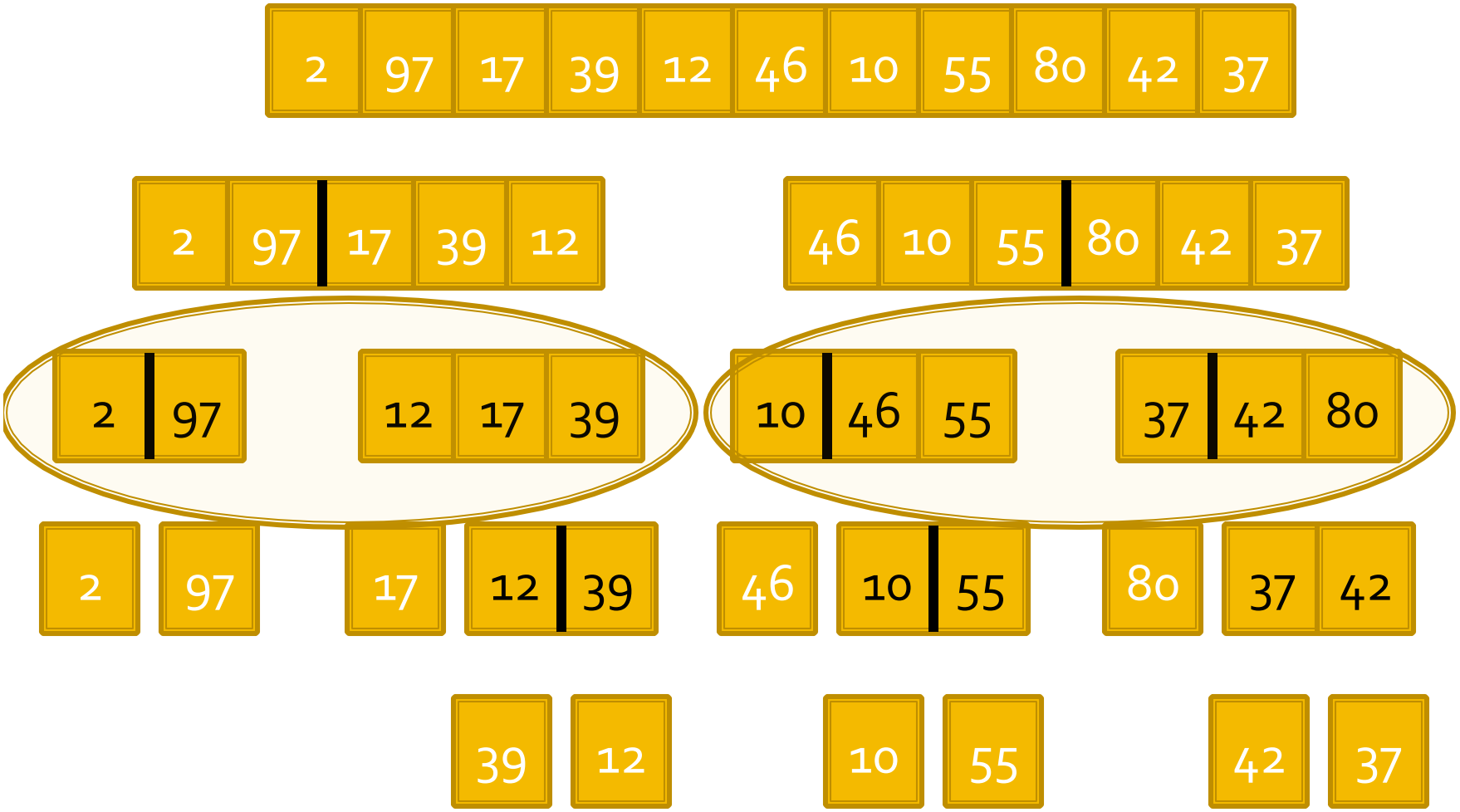
# Merge Sort


2 97 17 39 12 46 10 55 80 42 37
2 97 17 39 12
46 10 55 80 42 37
2 97
17 39 12
46 10 55
80 42 37
2
97
17
12 39
46
10 55
80
37 42
39
12
10
55
42
37

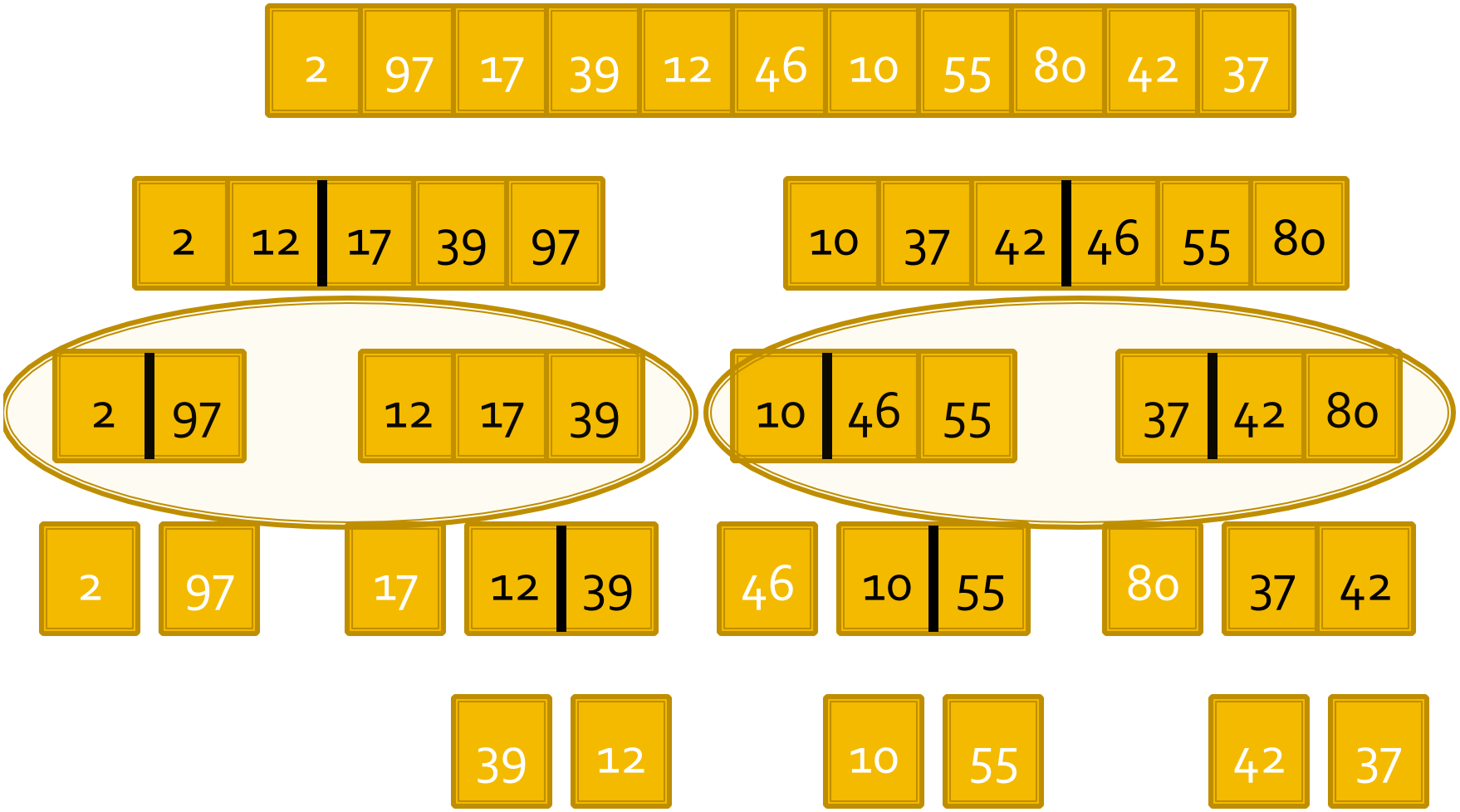
# Merge Sort


2 97 17 39 12 46 10 55 80 42 37
2 97 17 39 12
46 10 55 80 42 37
2 97
12 17 39
10 46 55
37 42 80
2 97
17 12 39
46 10 55
80 37 42
39 12
10 55
42 37

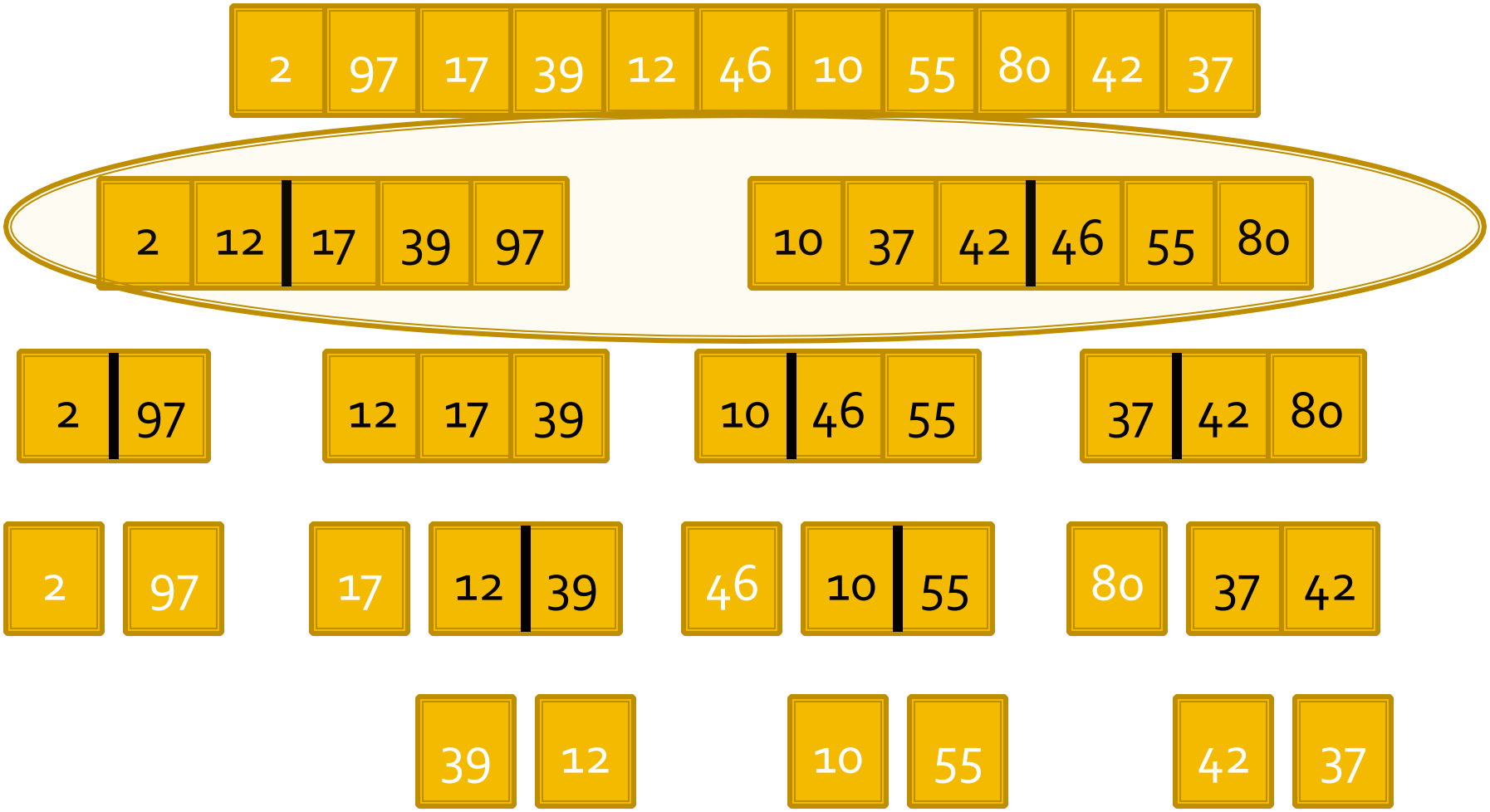
# Merge Sort


2 97 17 39 12 46 10 55 80 42 37
2 97 17 39 12
46 10 55 80 42 37
2 97
12 17 39
10 46 55
37 42 80
2
97
17
12 39
46
10 55
80
37 42
39
12
10
55
42
37

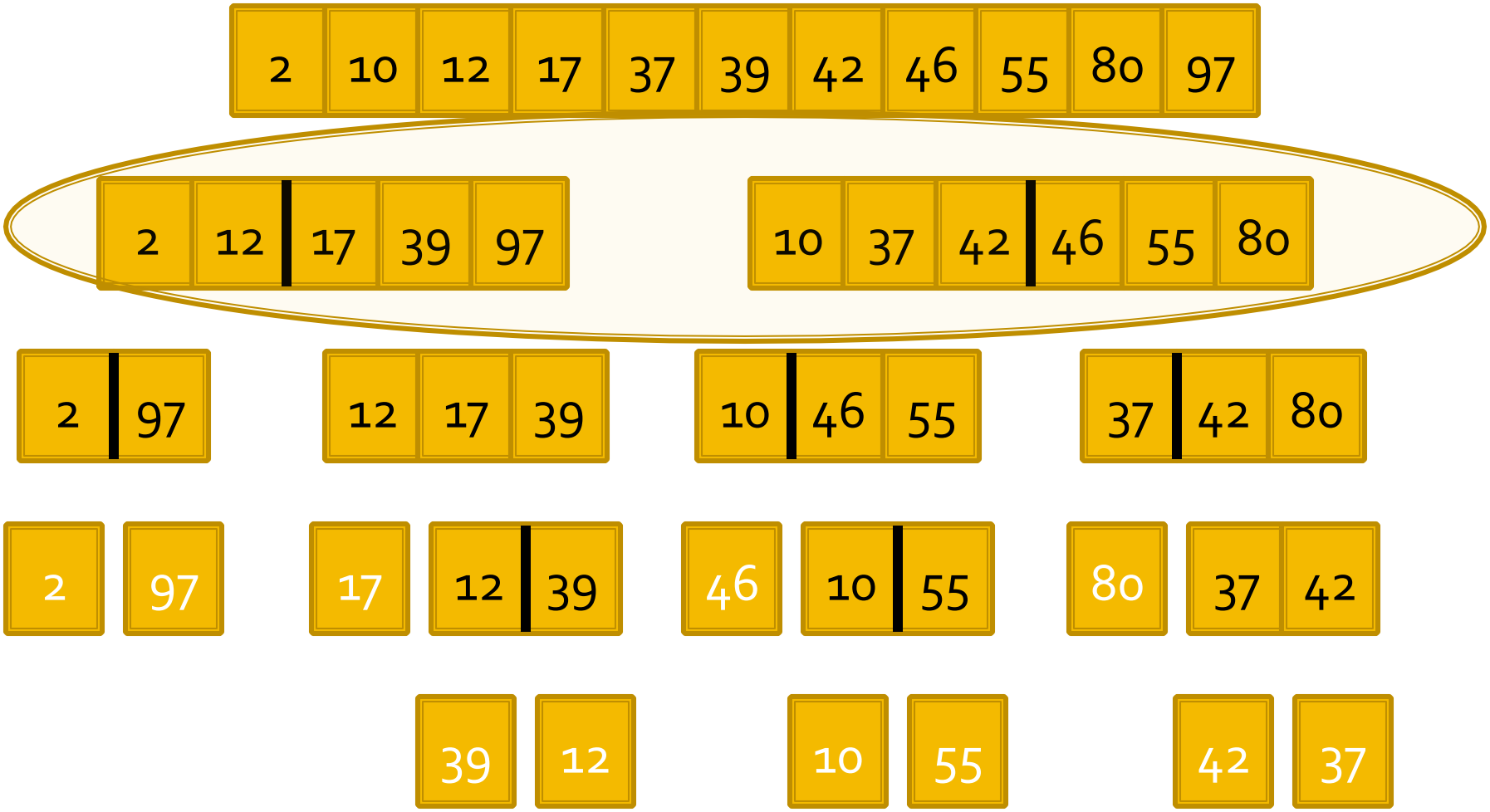
# Merge Sort


2 97 17 39 12 46 10 55 80 42 37
2 12 17 39 97
10 37 42 46 55 80
2 97
12 17 39
10 46 55
37 42 80
2
97
17
12 39
46
10 55
80
37 42
39
12
10
55
42
37

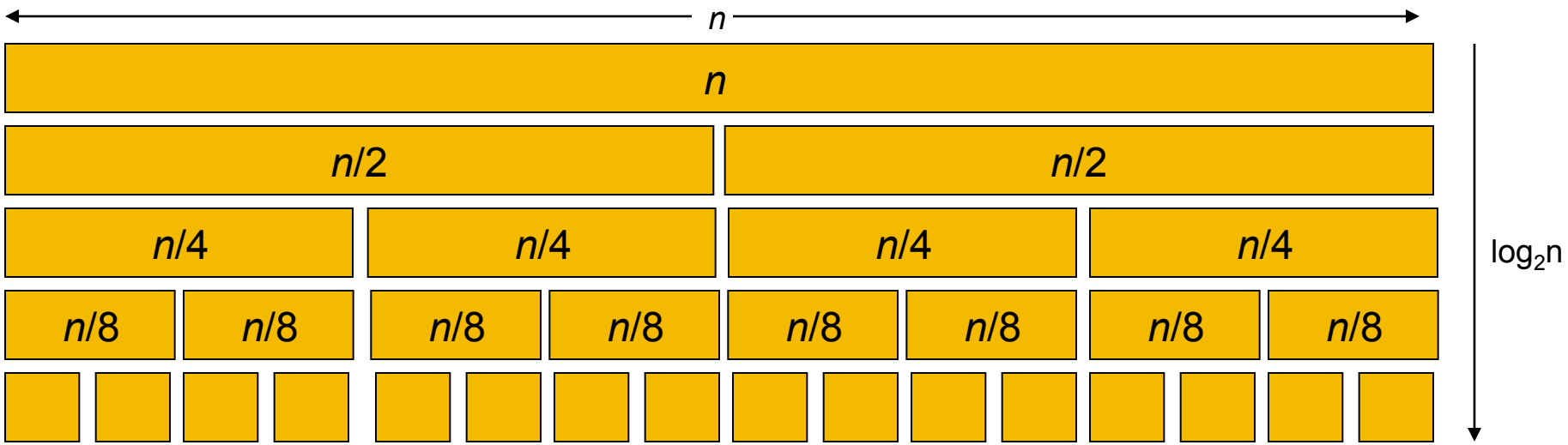
# Merge Sort


2 97 17 39 12 46 10 55 80 42 37
2 12 17 39 97
10 37 42 46 55 80
2 97
12 17 39
10 46 55
37 42 80
2 97
17 12 39
46 10 55
80 37 42
39 12
10 55
42 37

# Merge Sort


2 10 12 17 37 39 42 46 55 80 97
2 12 17 39 97
10 37 42 46 55 80
2 97
12 17 39
10 46 55
37 42 80
2
97
17
12 39
46
10 55
80
37 42
39
12
10
55
42
37

# Analysis




Complexity → $n\log_2 n$

# Summary

- It is not sufficient to think up a clever algorithm … **you need to know why it works**
- It's usually not tough, because the logic of your method typically translates into an explanation of why it works.

But you must think about it!!

- Once you know it works – CS people figure out how fast it is!