Algorithmic Design

Lawrence Snyder
University of Washington, Seattle

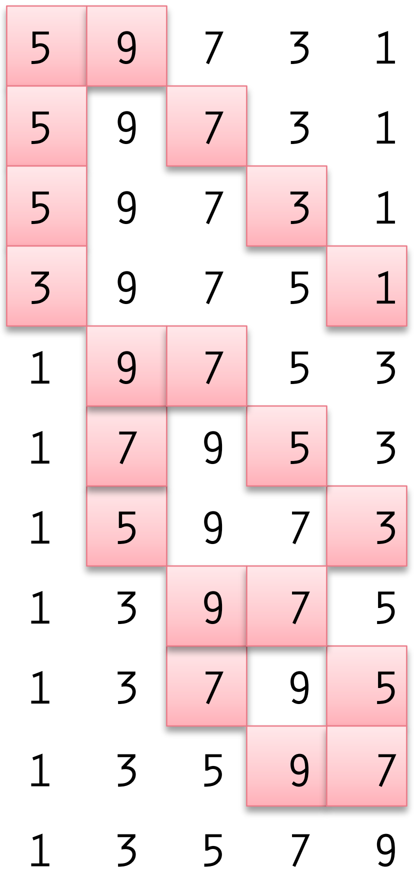
Algorithms

- Def. *An algorithm is a precise, systematic process directing 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

Many Alternative Algorithms …

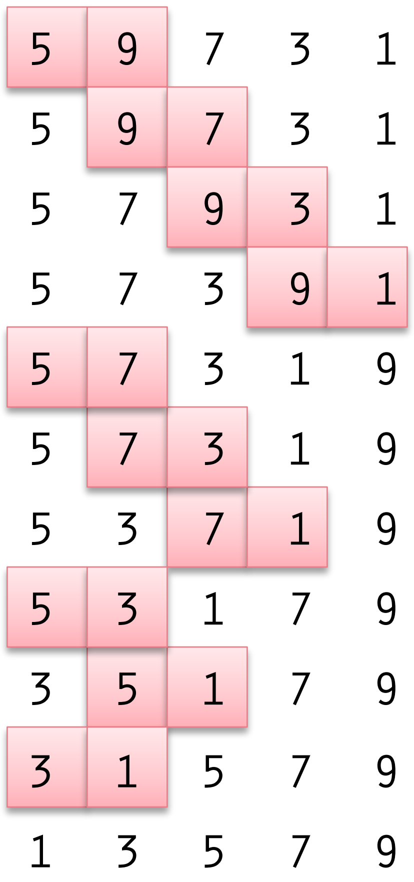
- There are always different algorithms to pick
- Consider sorting …
  - Put items in order
  - Exchange Sort: visit all items, starting with first; for each item, compare it with all following, exchanging any out of order pairs …

Steadily finds 1st smallest, then 2nd smallest, then …



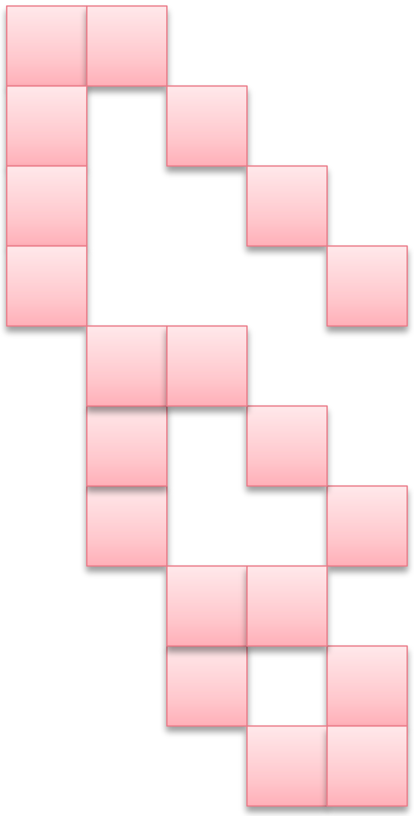
Now Consider Bubble Sort…

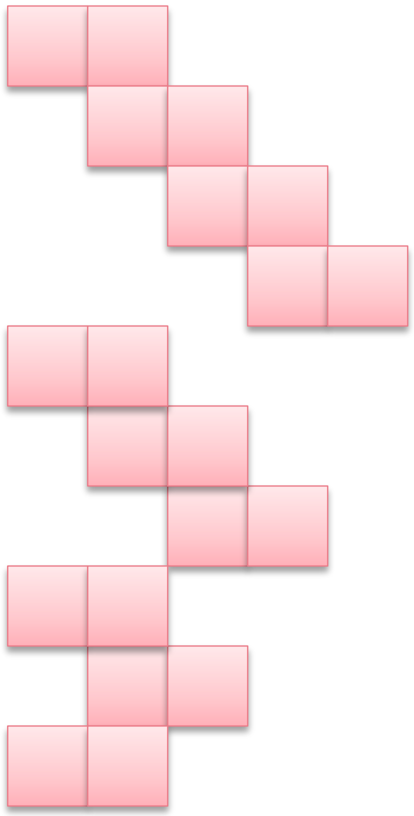
- Bubble Sort: go through list in pairs, correcting out of order pairs; can progressively stop earlier and earlier …
- Pushes largest to the end, then pushes second largest to "next to end' position, then pushes …



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





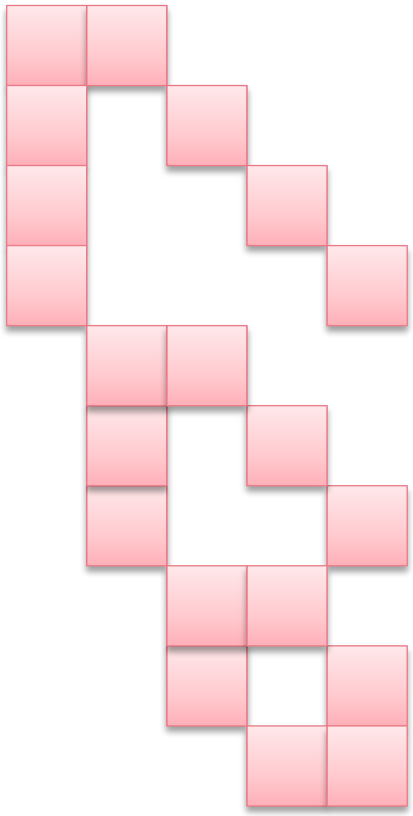
Key Question for Today & Always

- 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

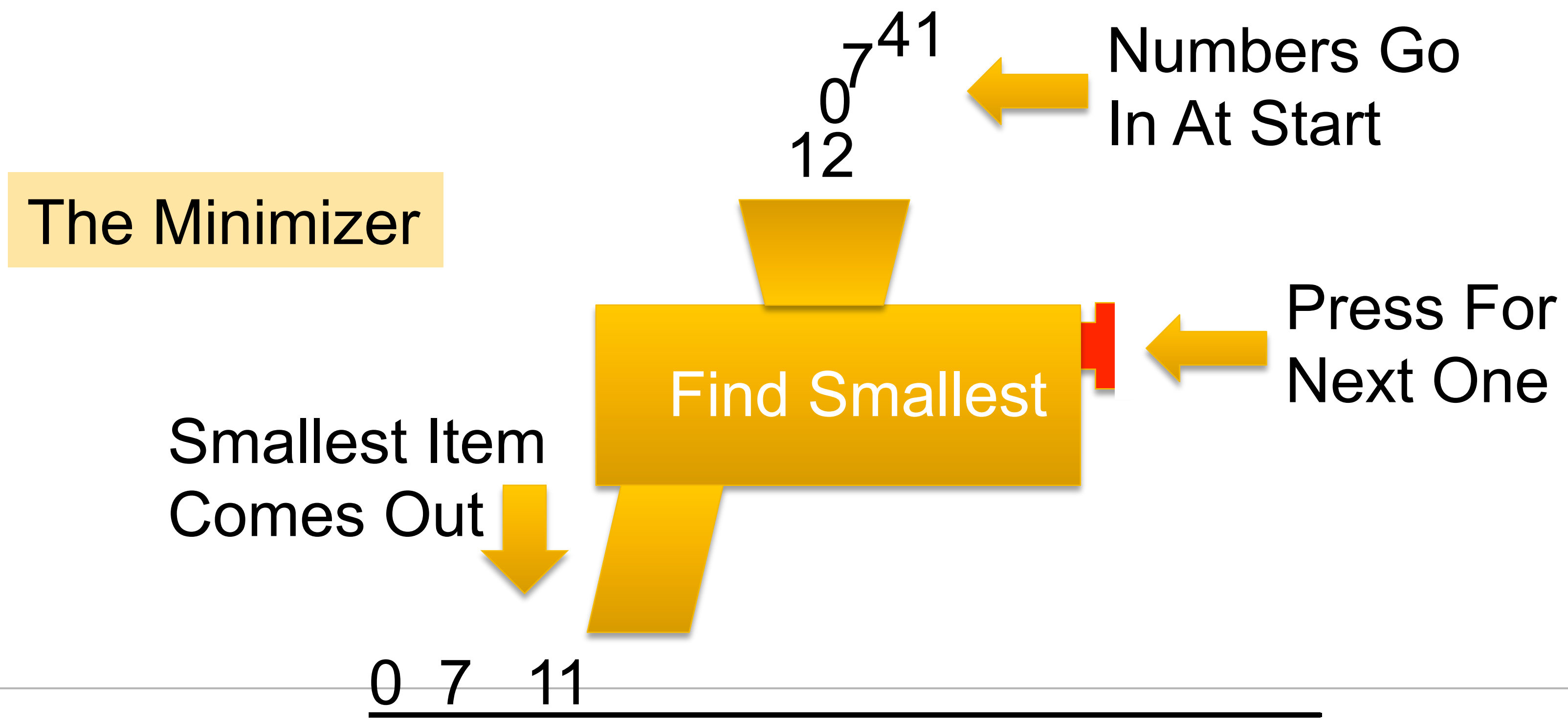
Explaining Why Algorithm Works

- Say What You're Claiming: "Exchange Sort Puts Numbers or Words in Ascending Order"
- It's not automatically obvious



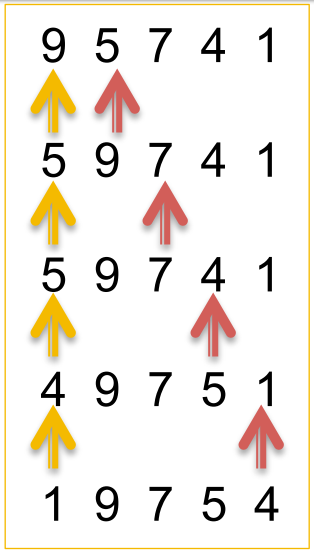
Explaining Why Algorithm Works

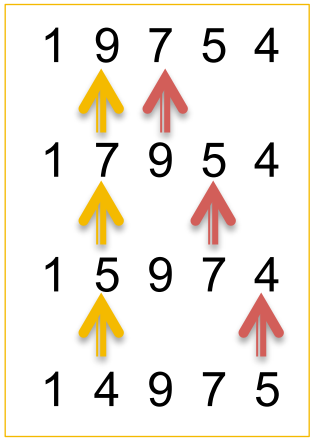
- Formulate a way to see why it does work
- Explain how "big picture-wise" … use analogy
- Cookie "Spritzer" or Power Tool

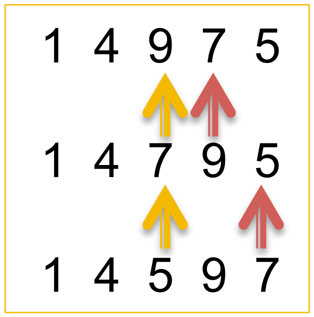


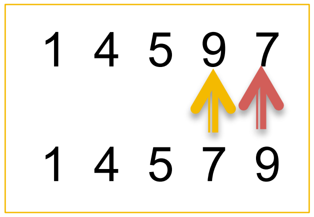
Explain Why Code Works That Way

- If Minimizer Sorts, So Does Exchange Sort
- The Operation of Exchange Sort Works Like The Minimizer … each pass "emits" the smallest item



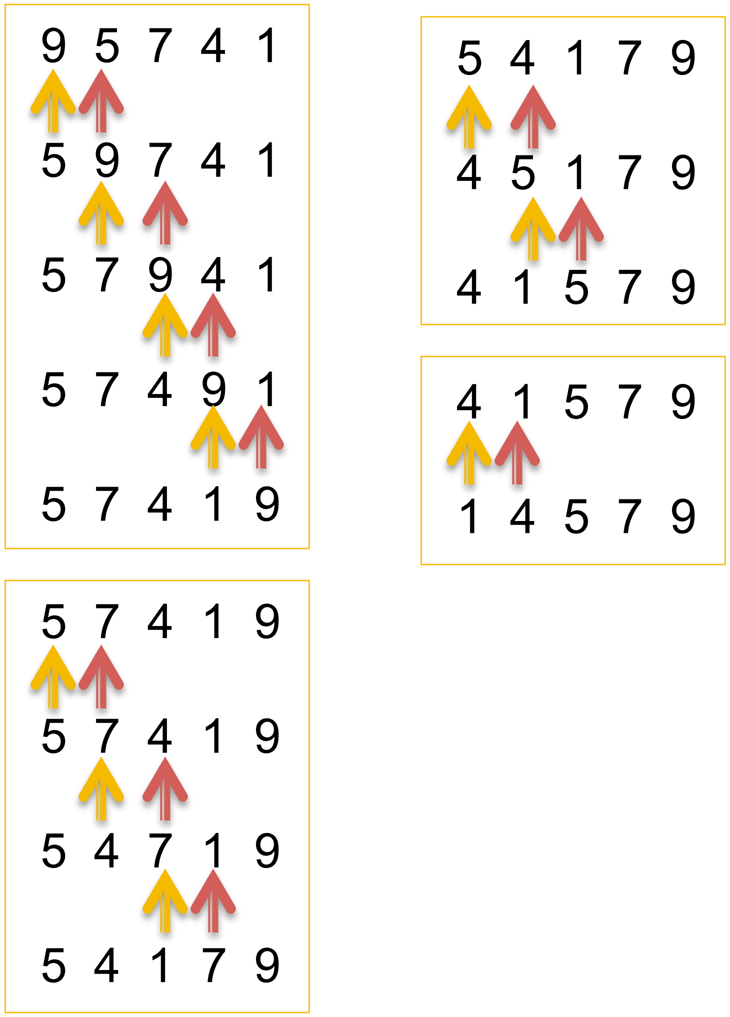






How About Bubble Sort?

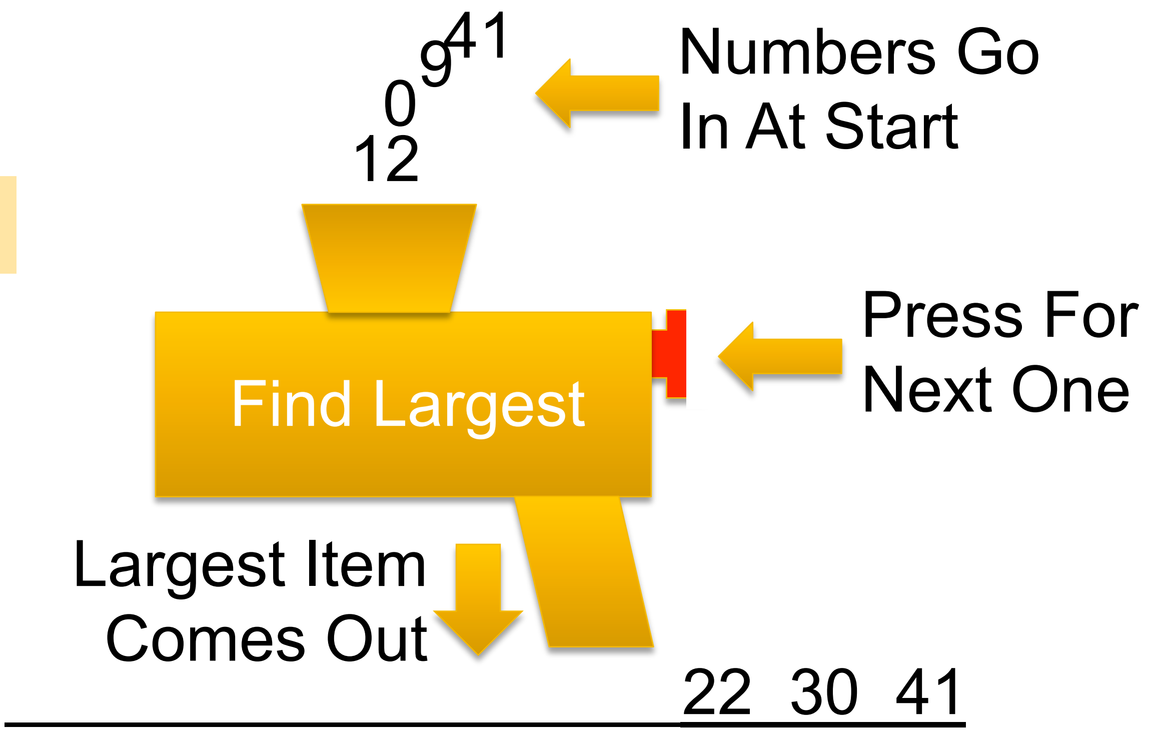
- Recall, Bubble Sort "pushes" the largest as far as possible to the right



Explaining Why Algorithm Works

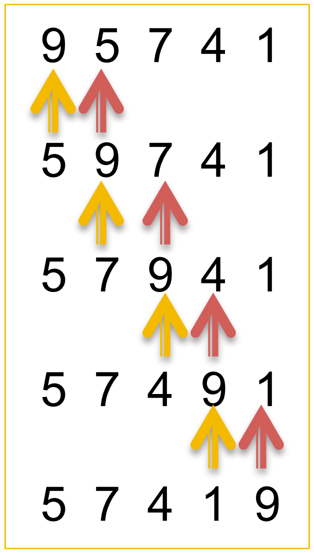
- Say What You're Claiming: "Bubble Sort Puts Numbers or Words in Ascending Order"
- Explain how to do it "big picture-wise" … use an analogy:

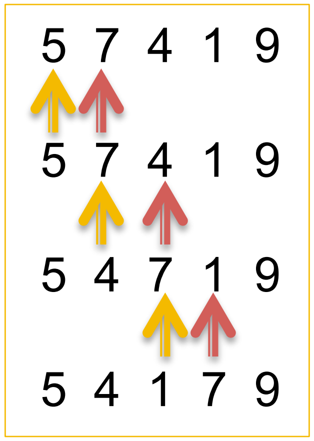
The Maximizer

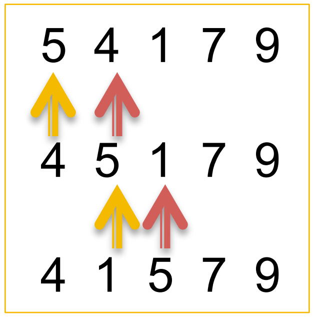


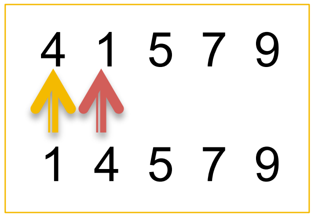
How About Bubble Sort?

- If The Maximizer Sorts then so does Bubble sort … the operation works like the maxi-mizer









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

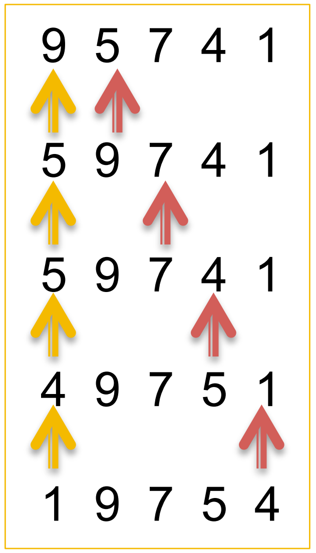
4 + 3 + 2 + 1 = 10 compares for 5 items

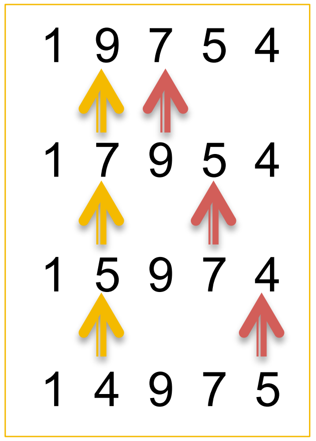
- Generally for n items

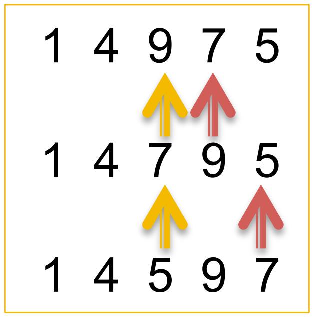
$(n-1) + (n-2) + \ldots + 2 + 1$

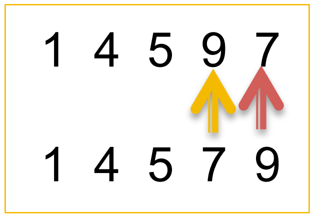
$= n(n-1)/2$

$= 1/2\ (n^2 - n)$





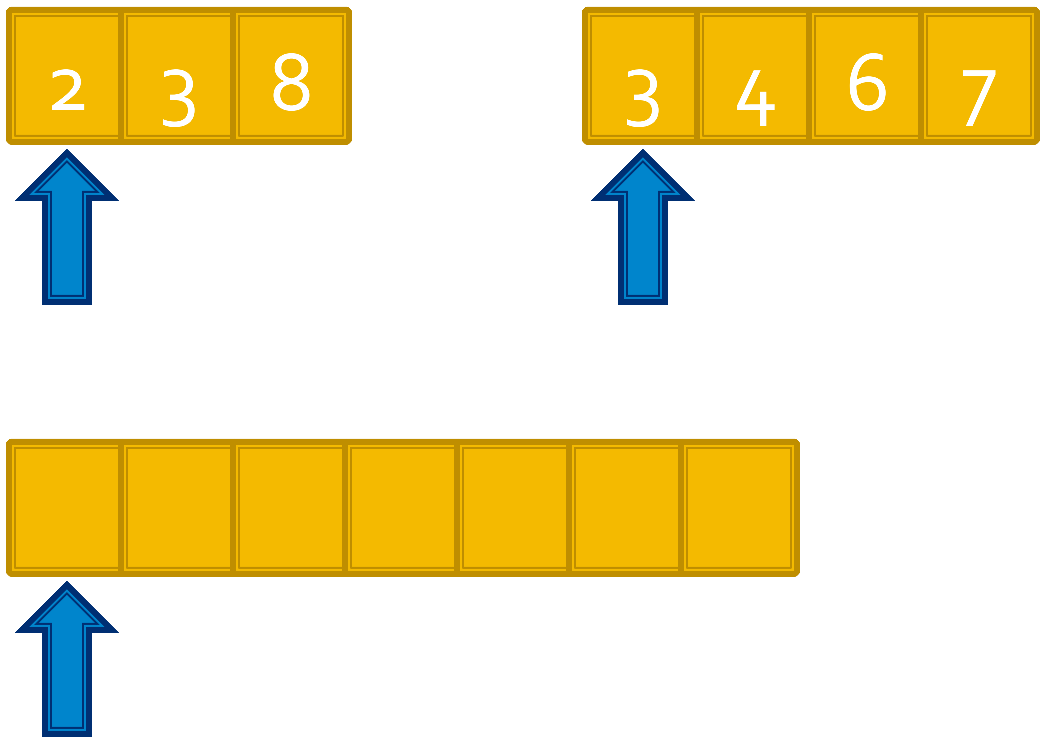




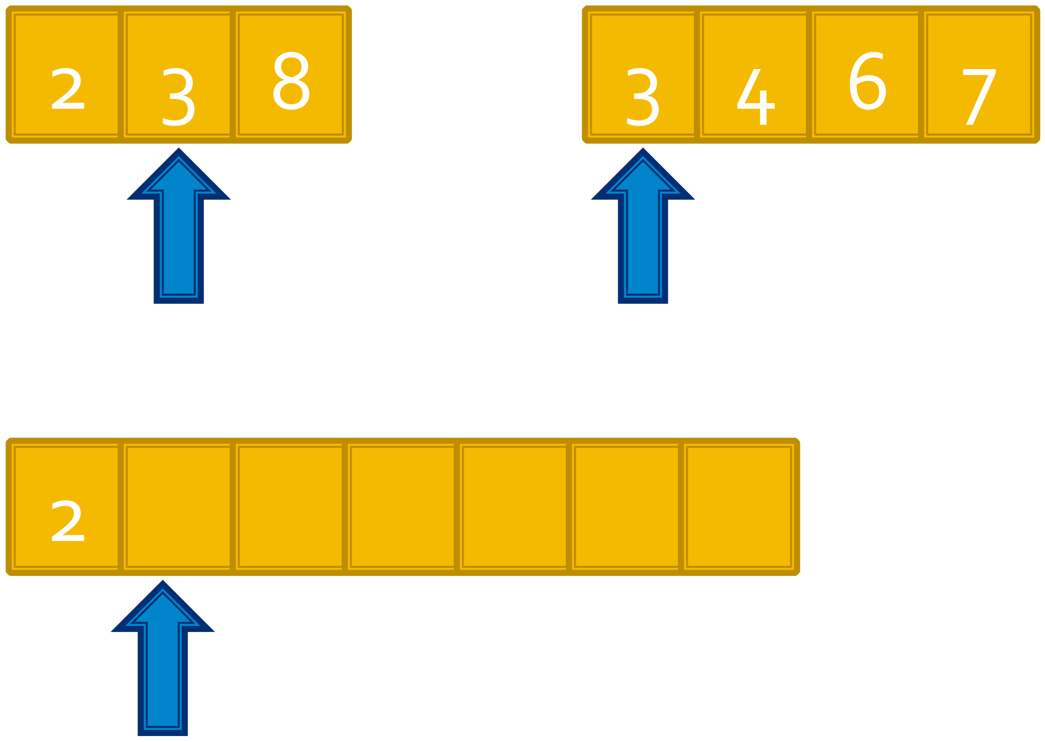
Merging

- Consider another way to sort
- Merging two sorted arrays into a single sorted array is straight forward

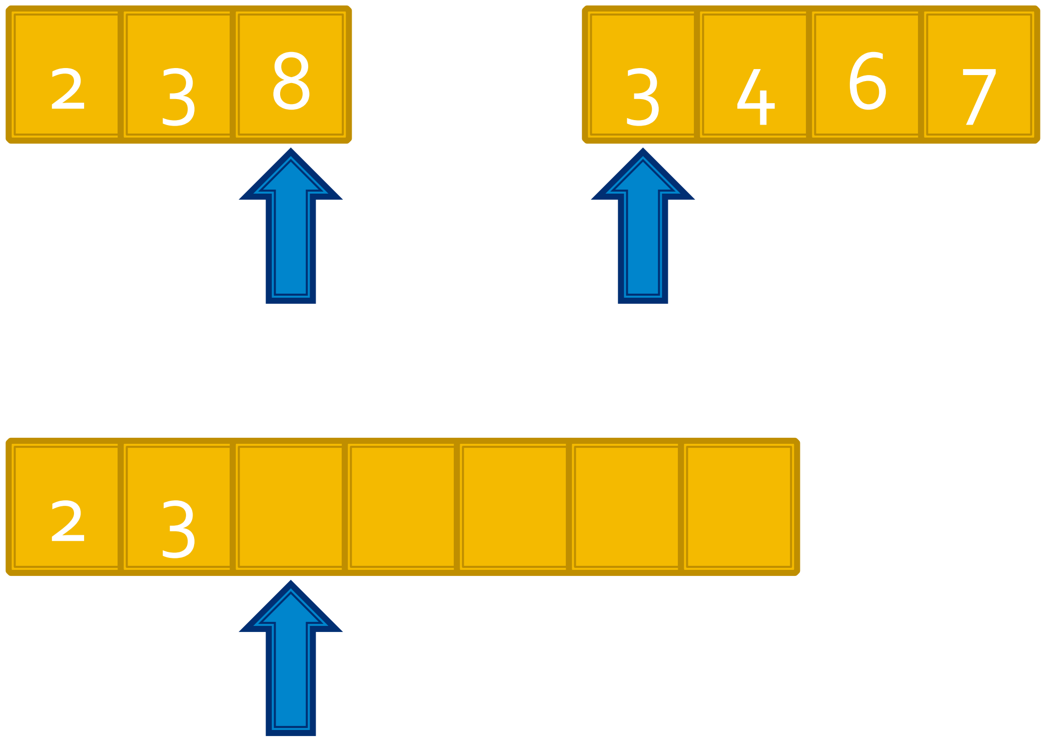
Merging



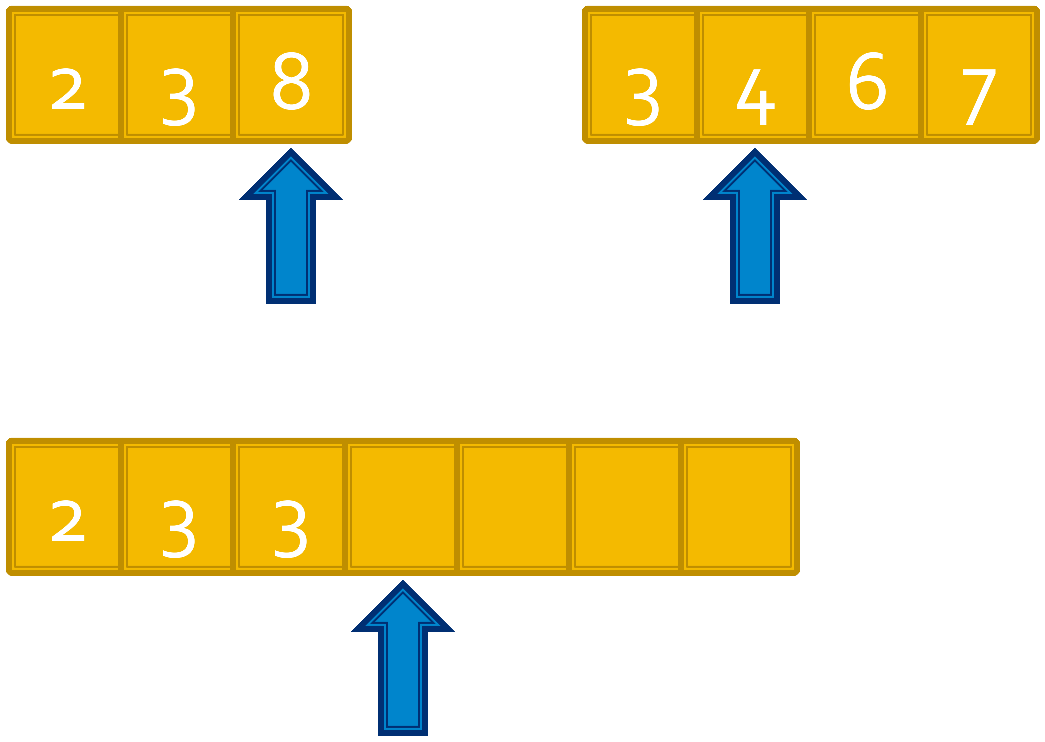
Merging



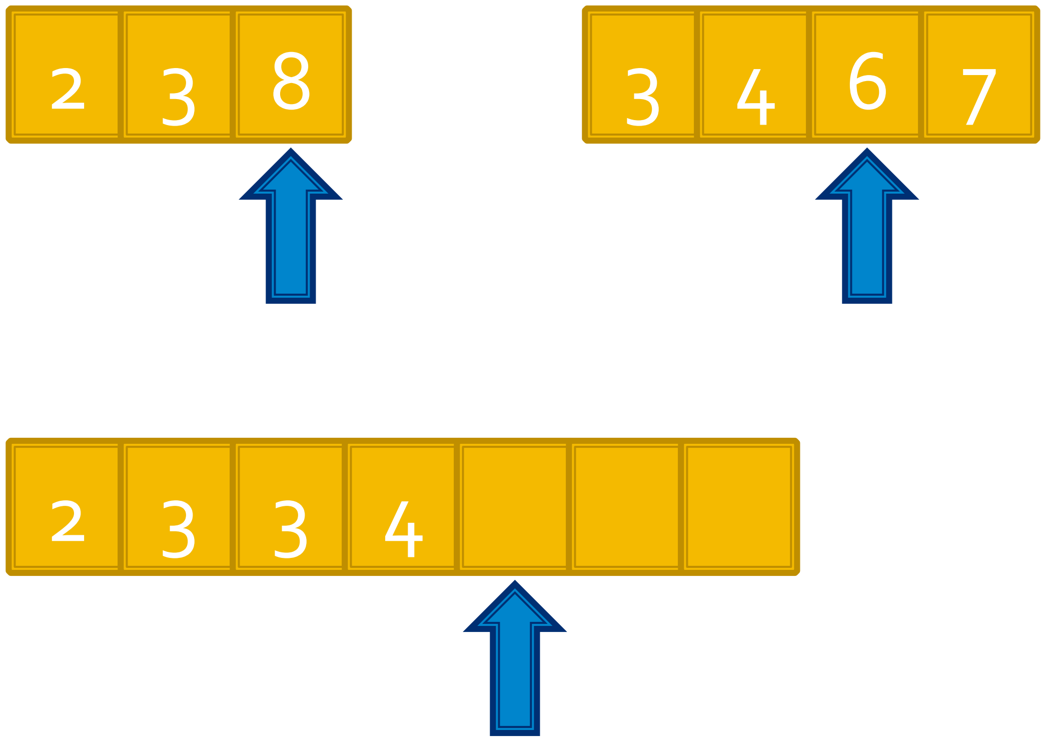
Merging



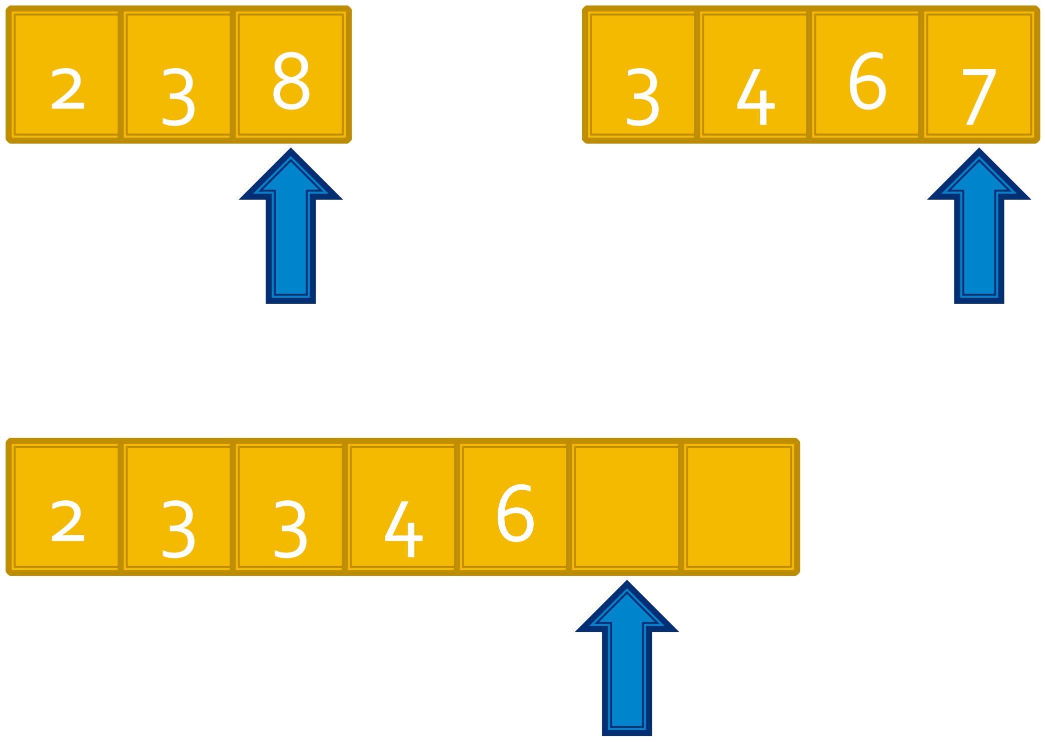
Merging



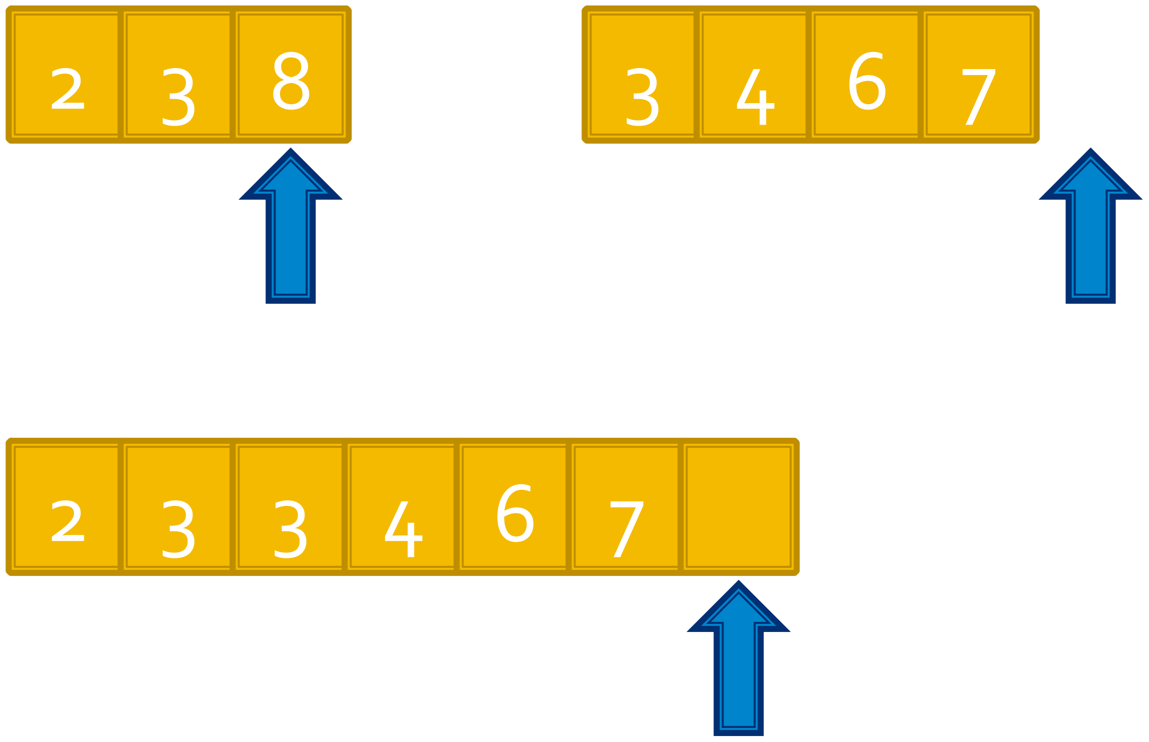
Merging



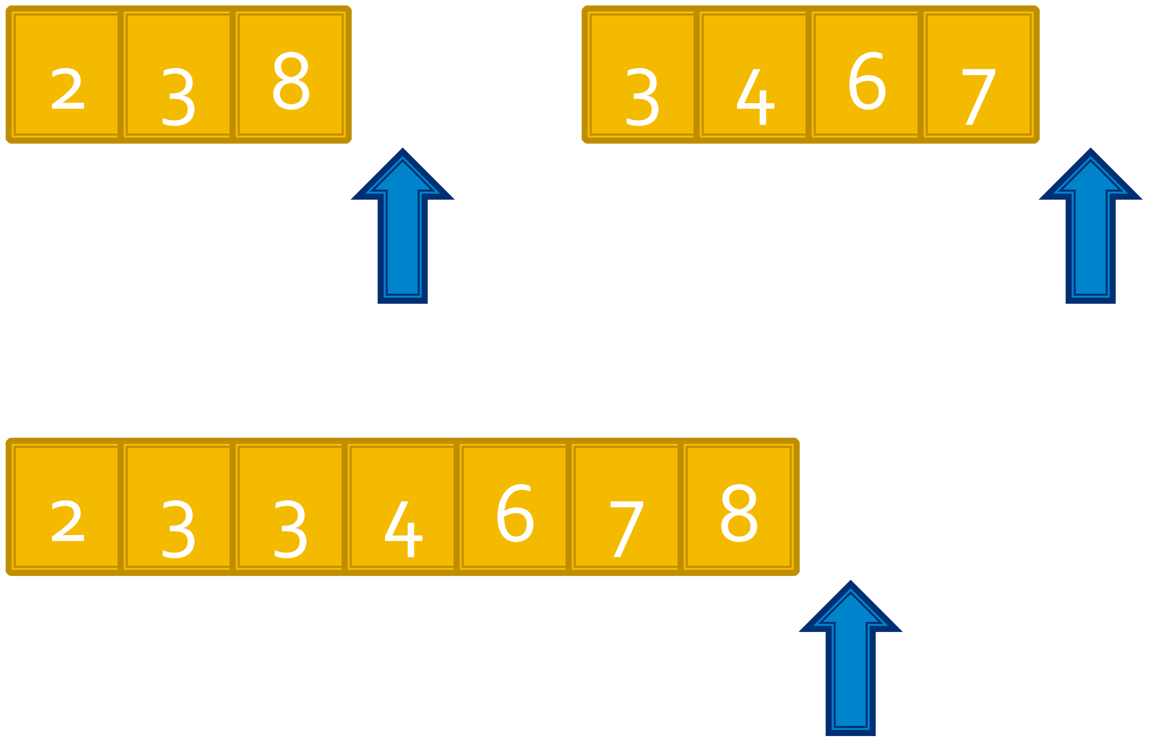
Merging



Merging



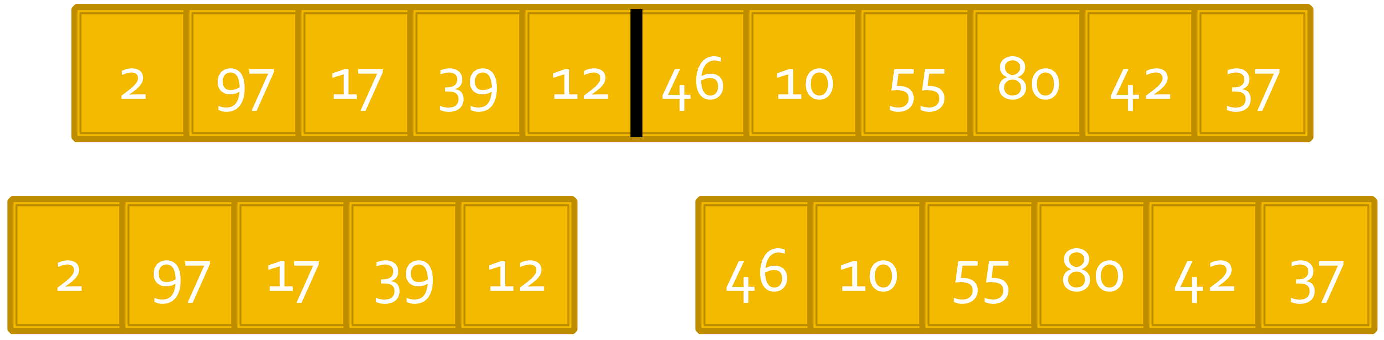
Merging



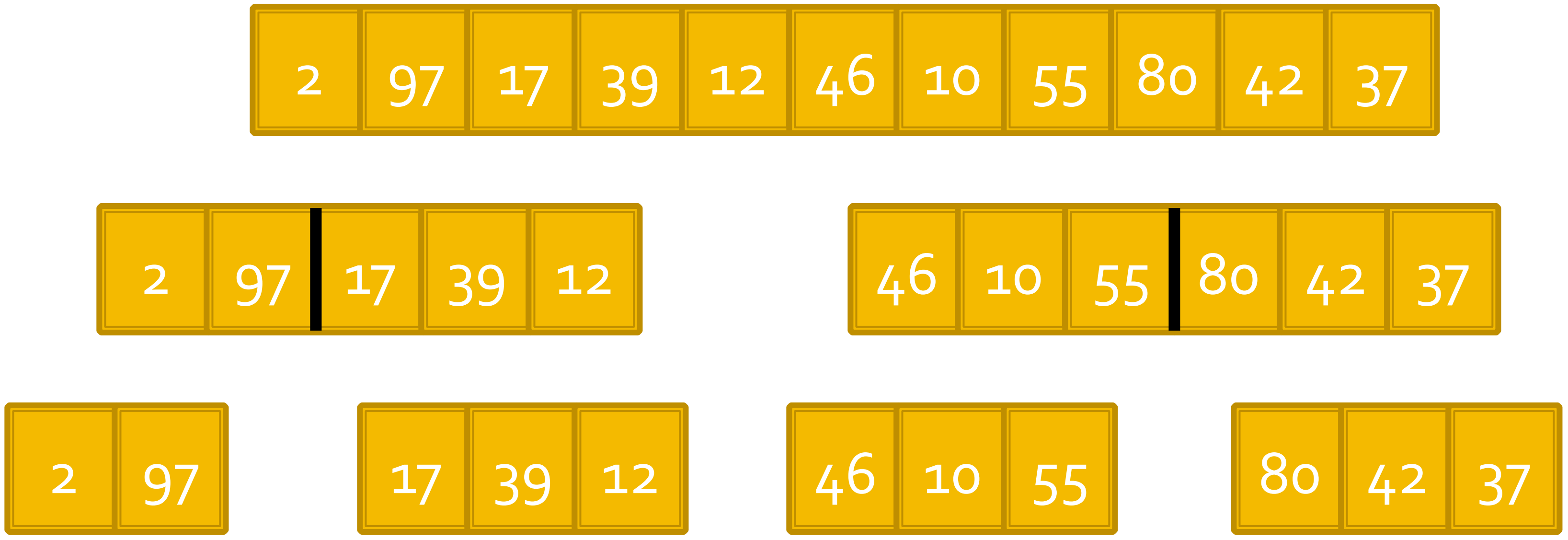
Merge Sort

2	97	17	39	12	46	10	55	80	42	37

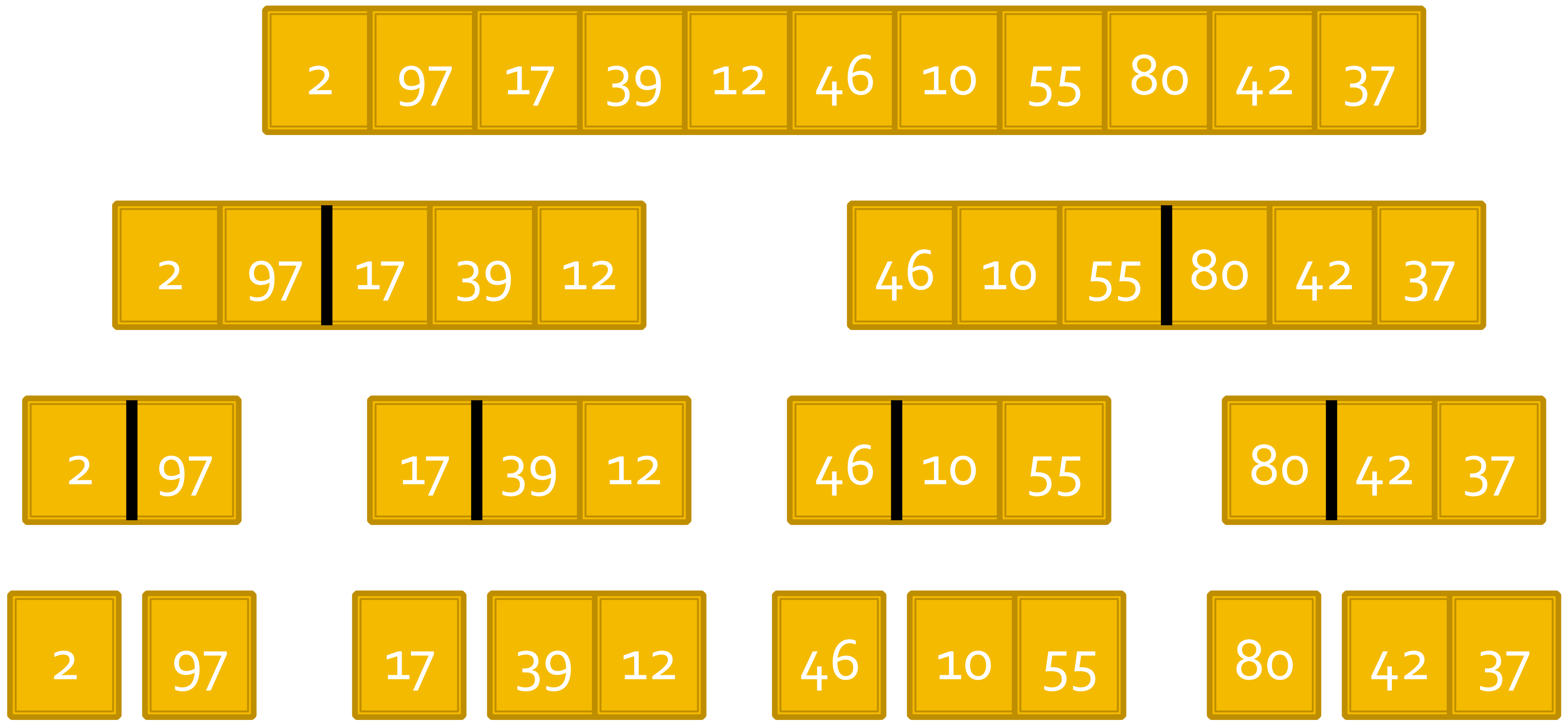
Merge Sort



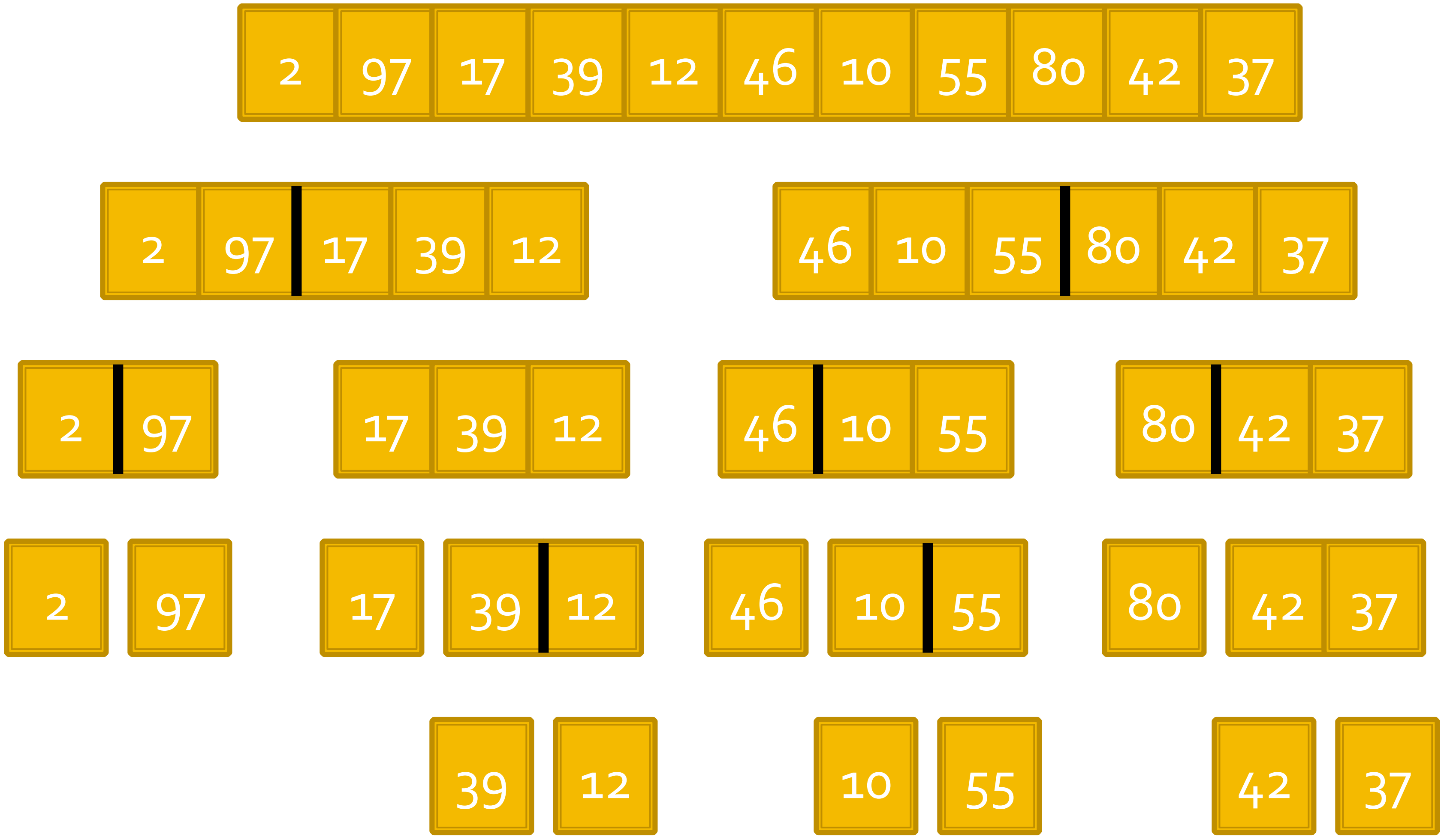
Merge Sort


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

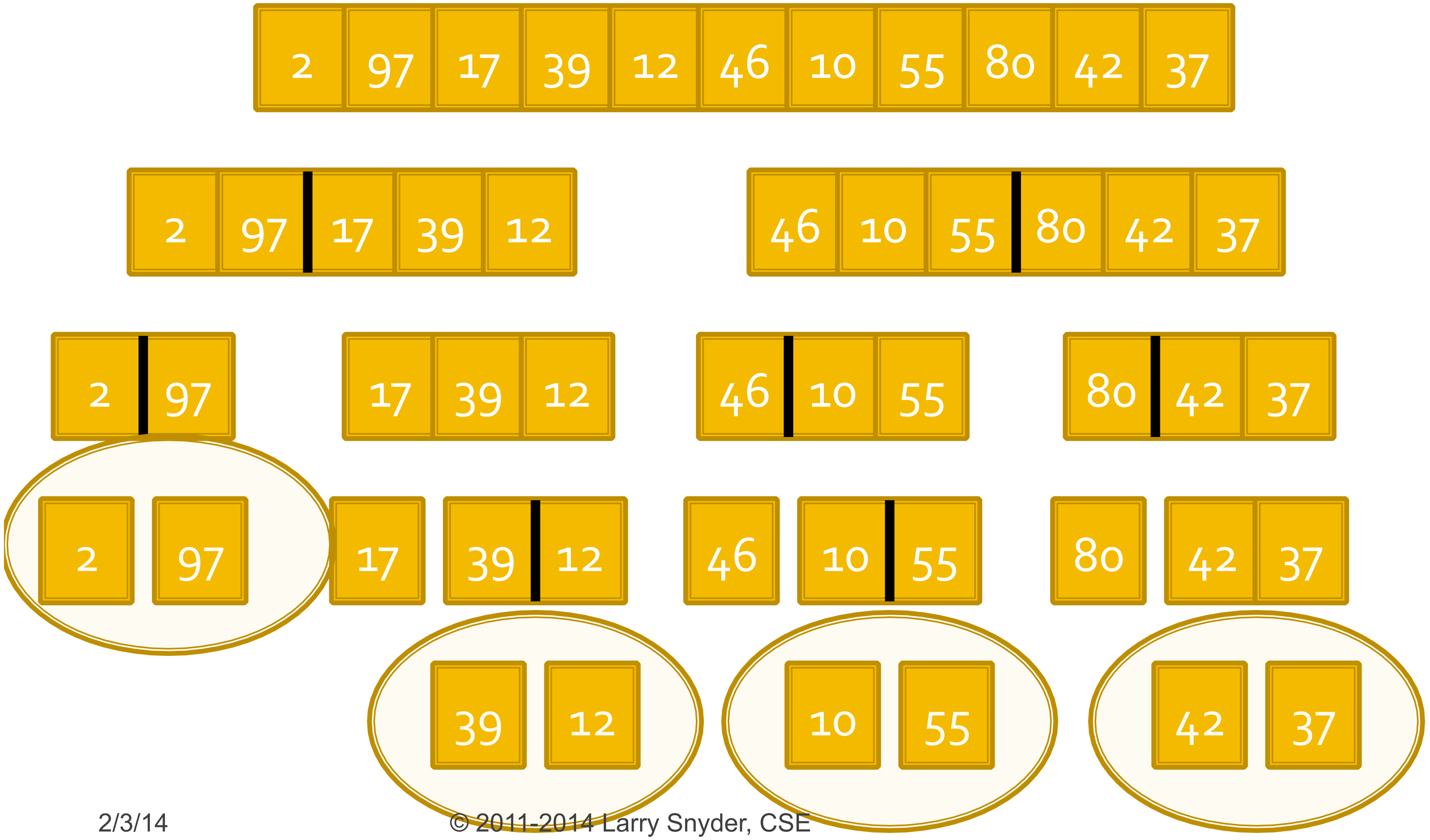
Merge Sort


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

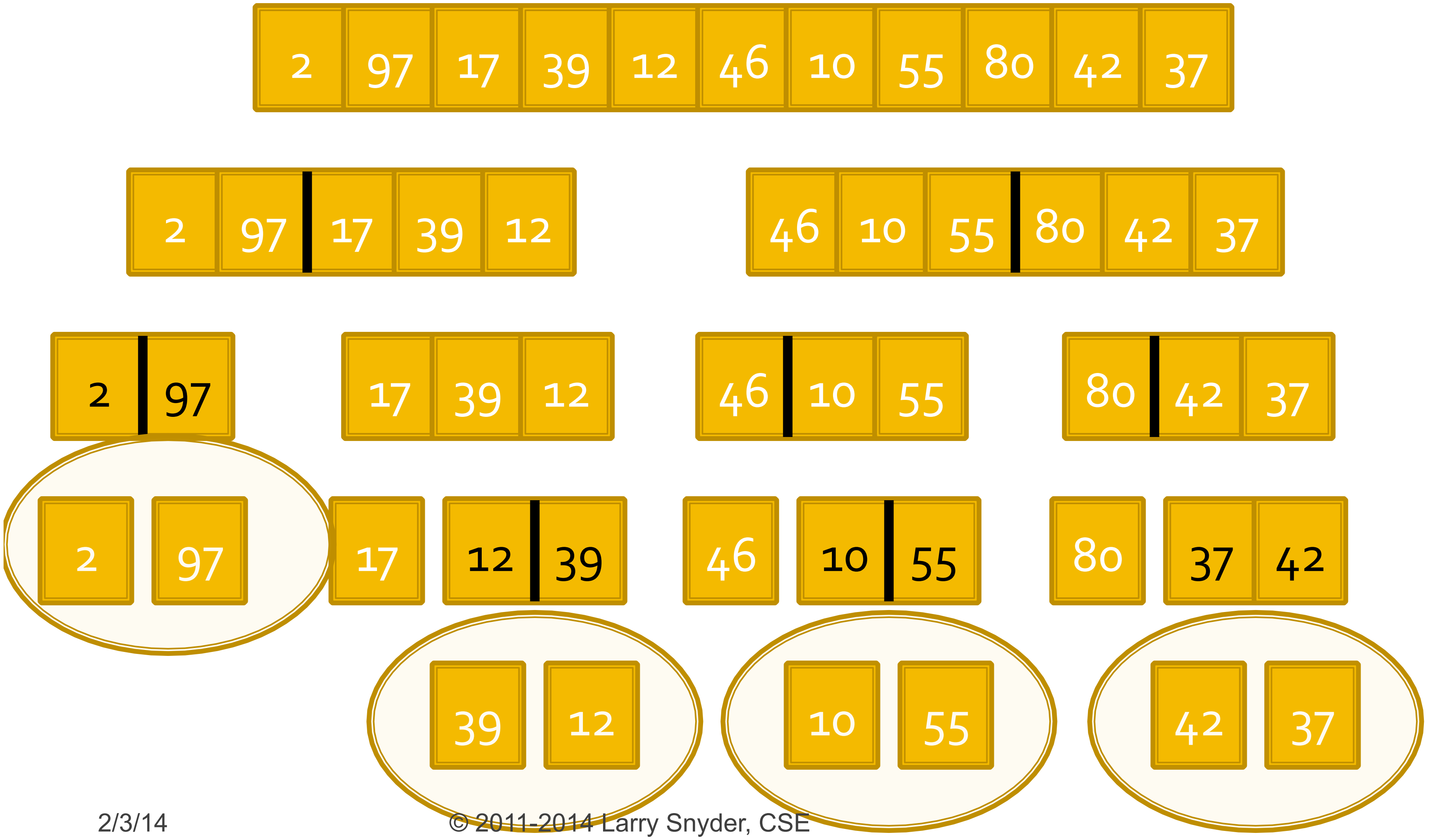

2 97 17 39 12 46 10 55 80 42 37
2 97 17 39 12
46 10 55 80 42 37
2 97
17 39 12
46 10 55
80 42 37
2
97
17
39 12
46
10 55
80
42 37
39
12
10
55
42
37

Merge Sort

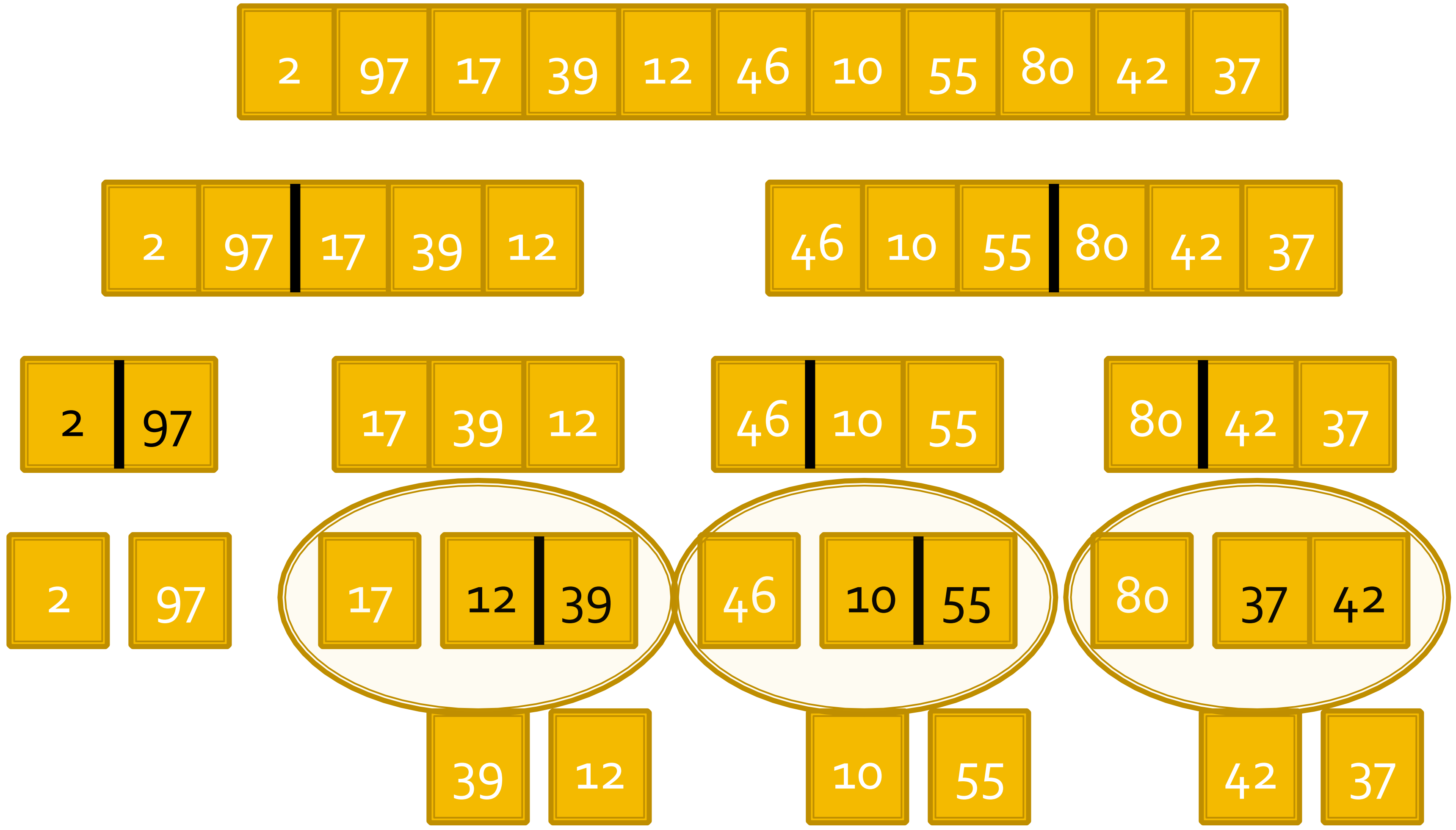


2 97 17 39 12 46 10 55 80 42 37
2 97 17 39 12
46 10 55 80 42 37
2 97
17 39 12
46 10 55
80 42 37
2
97
17
39 12
46
10 55
80
42 37
39
12
10
55
42
37

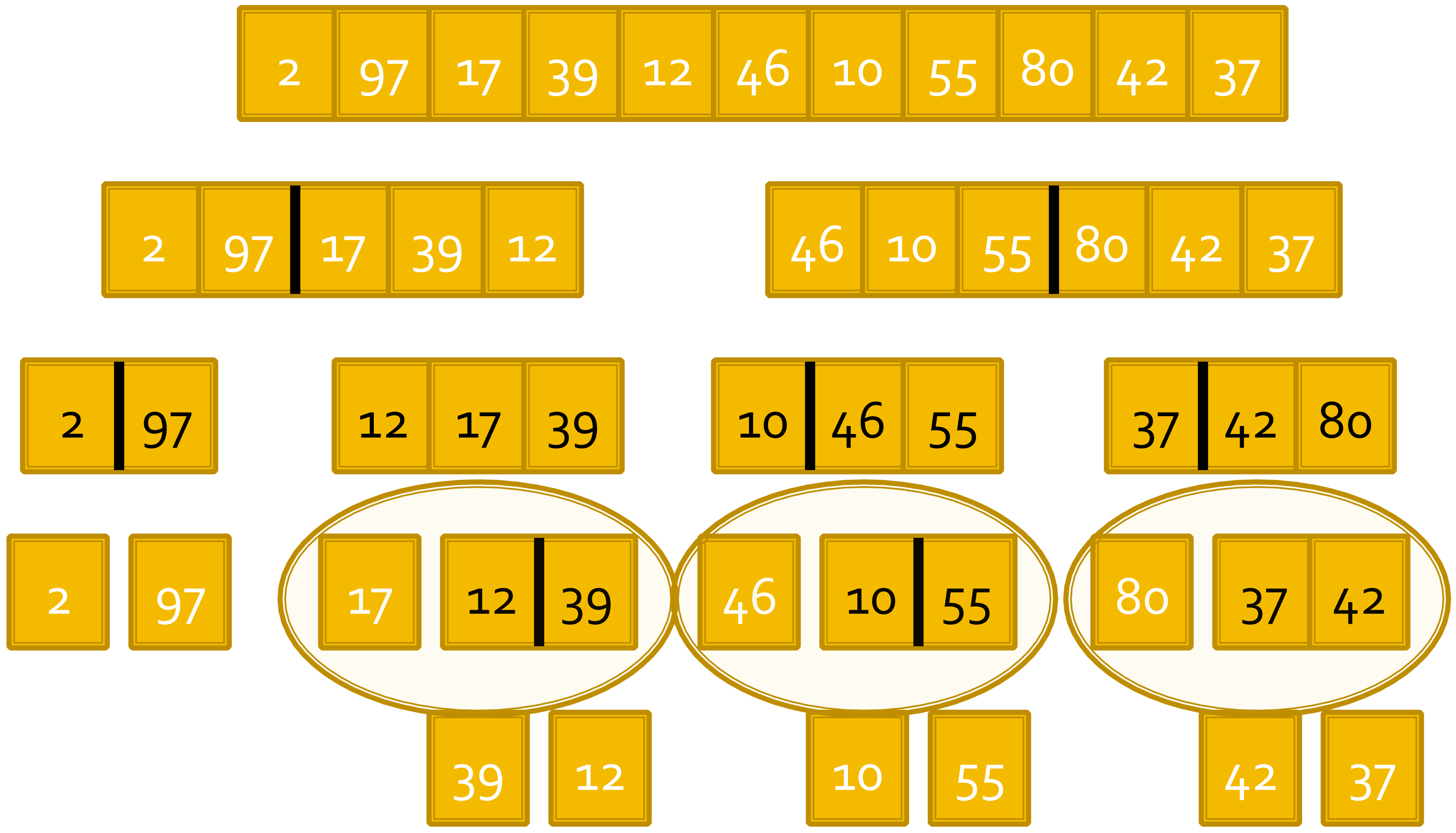
Merge Sort


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

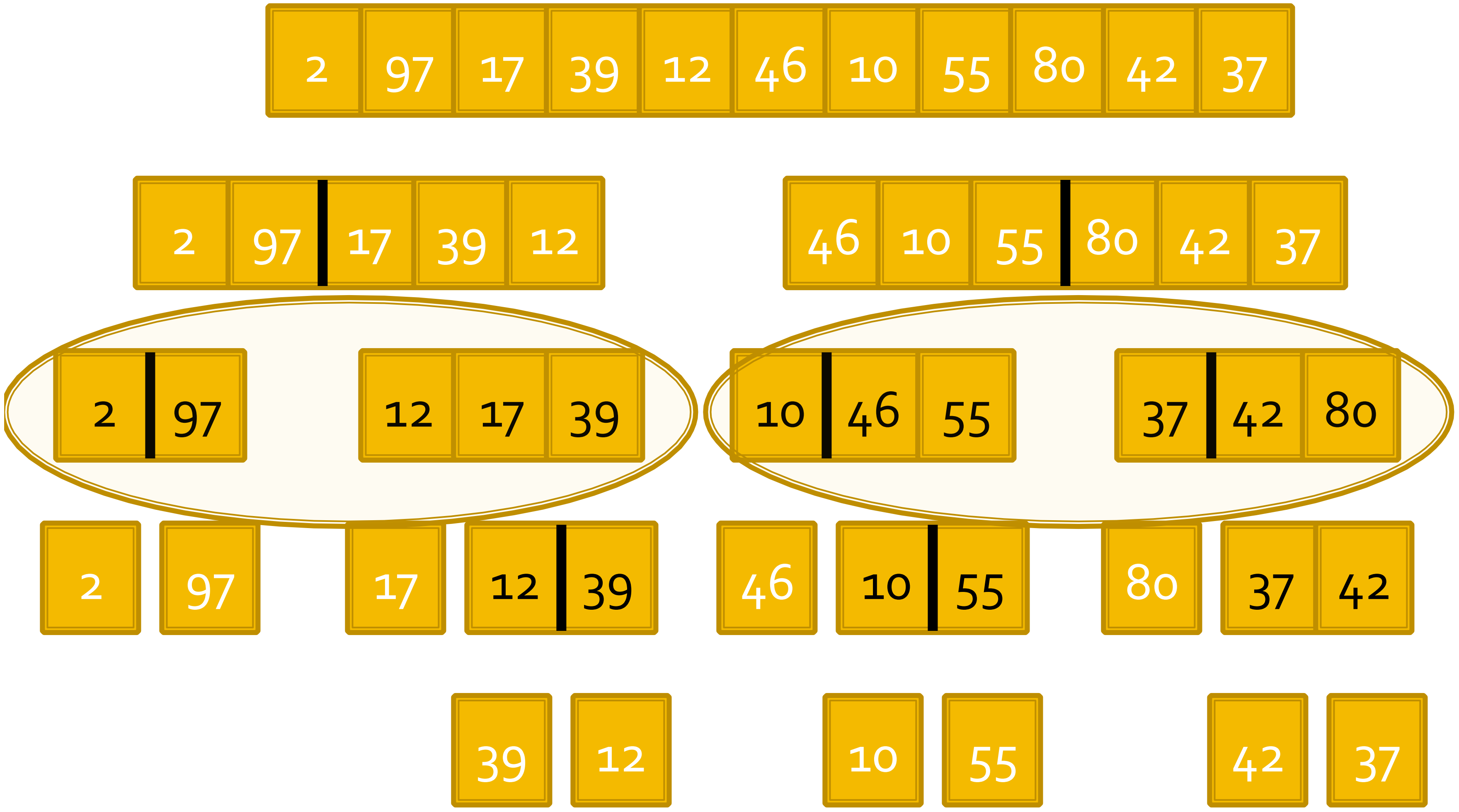
Merge Sort


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

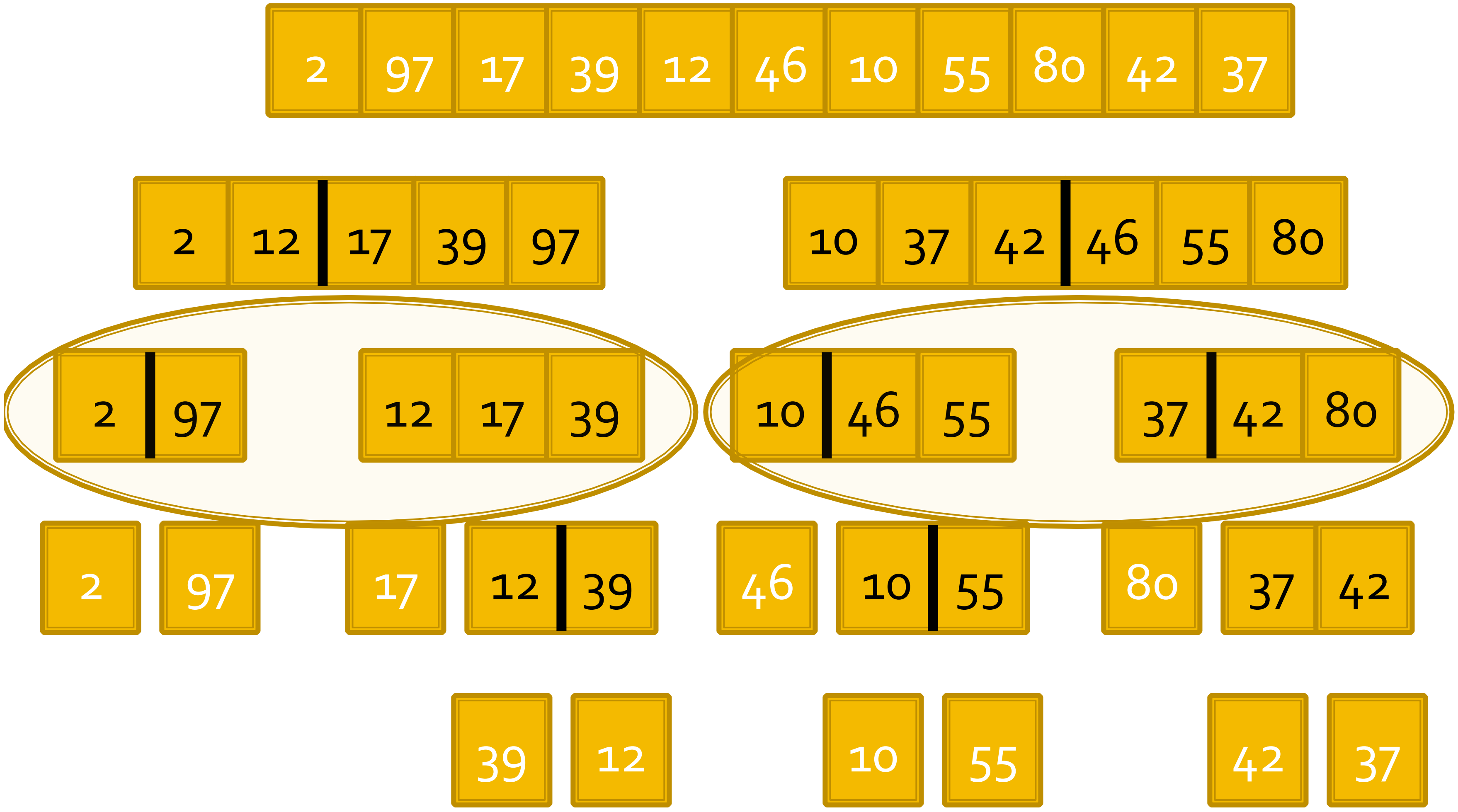
Merge Sort


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

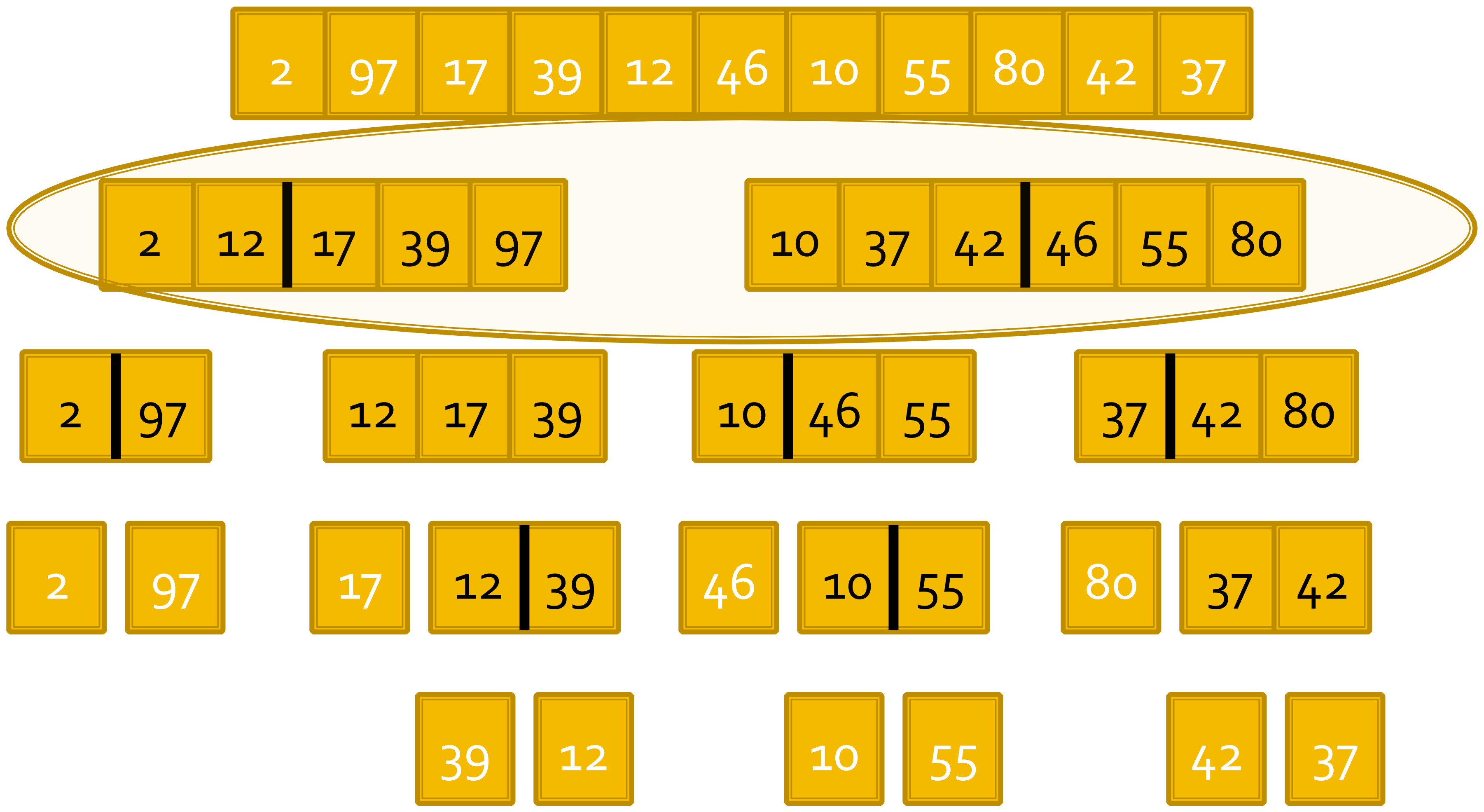
Merge Sort


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

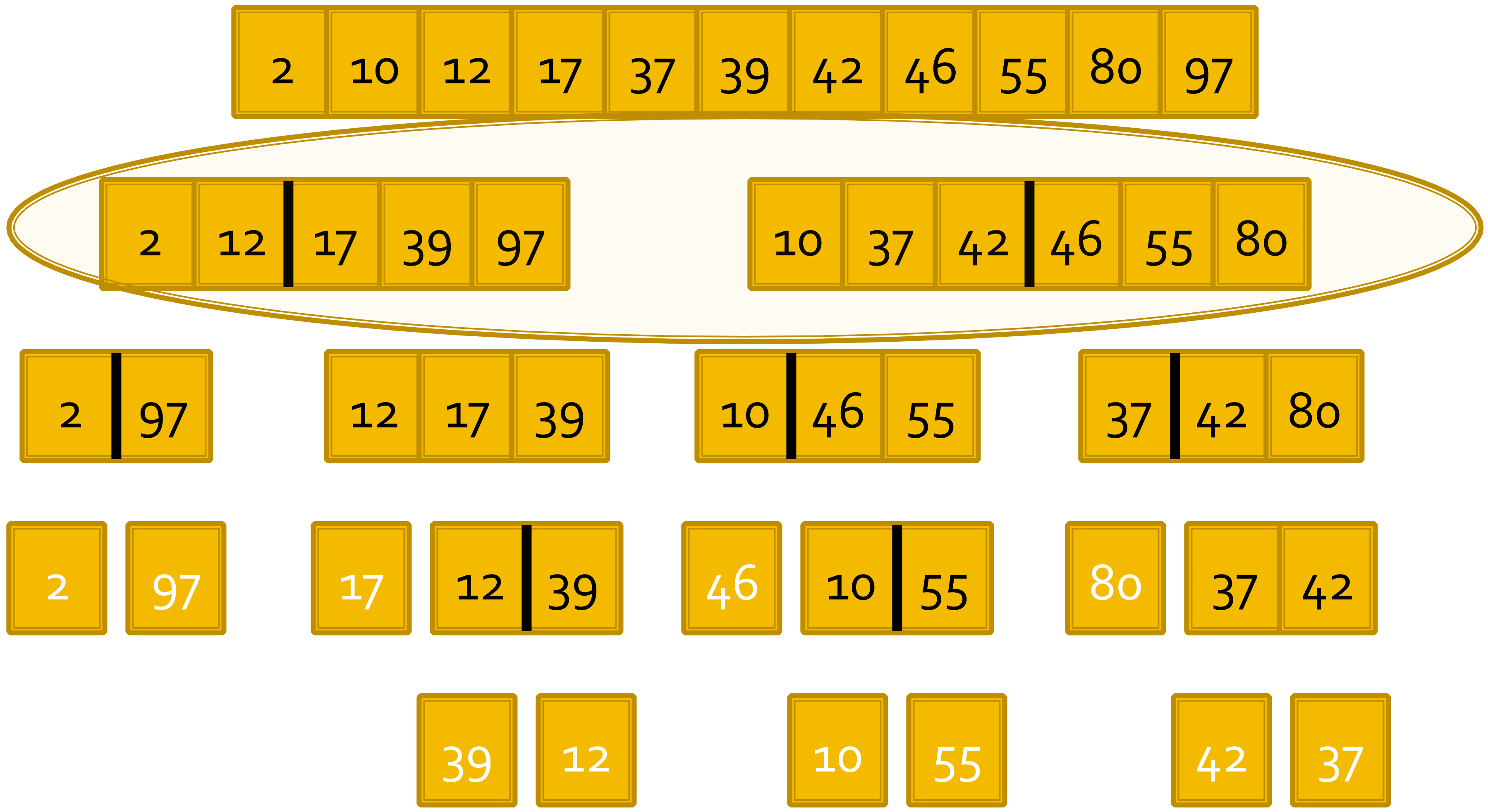
Merge Sort


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

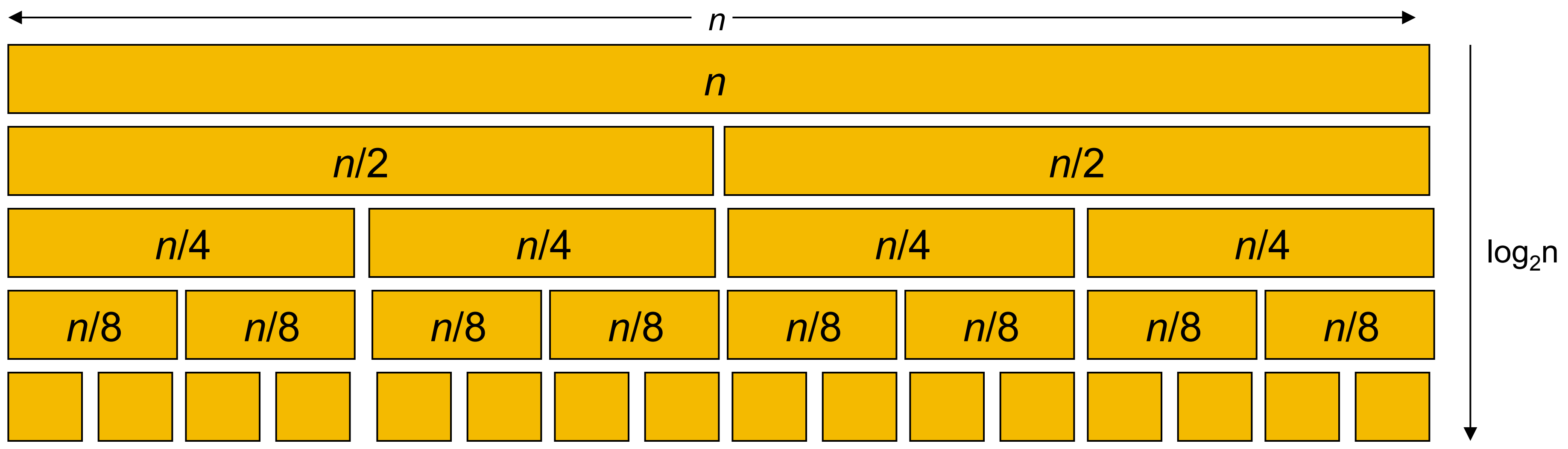
Merge Sort


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



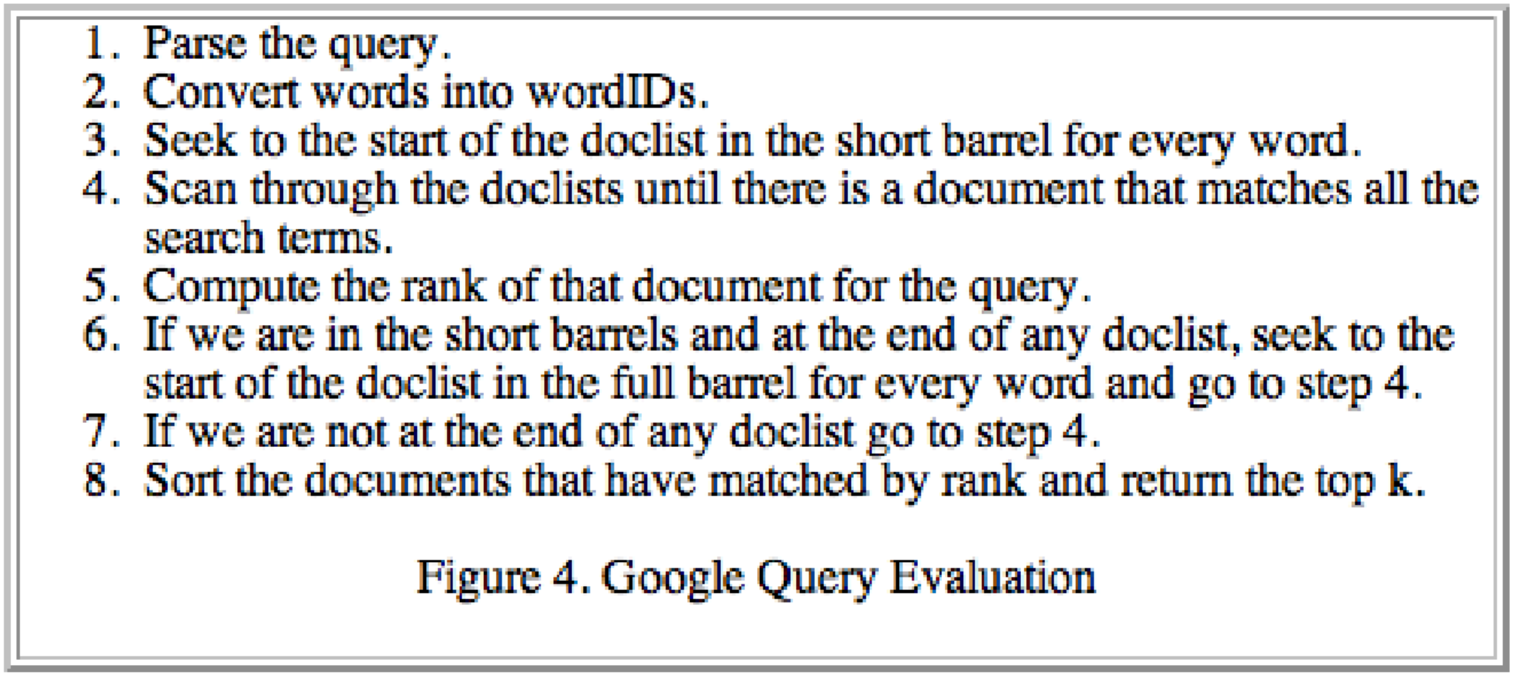
Complexity → $n\log_2 n$

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



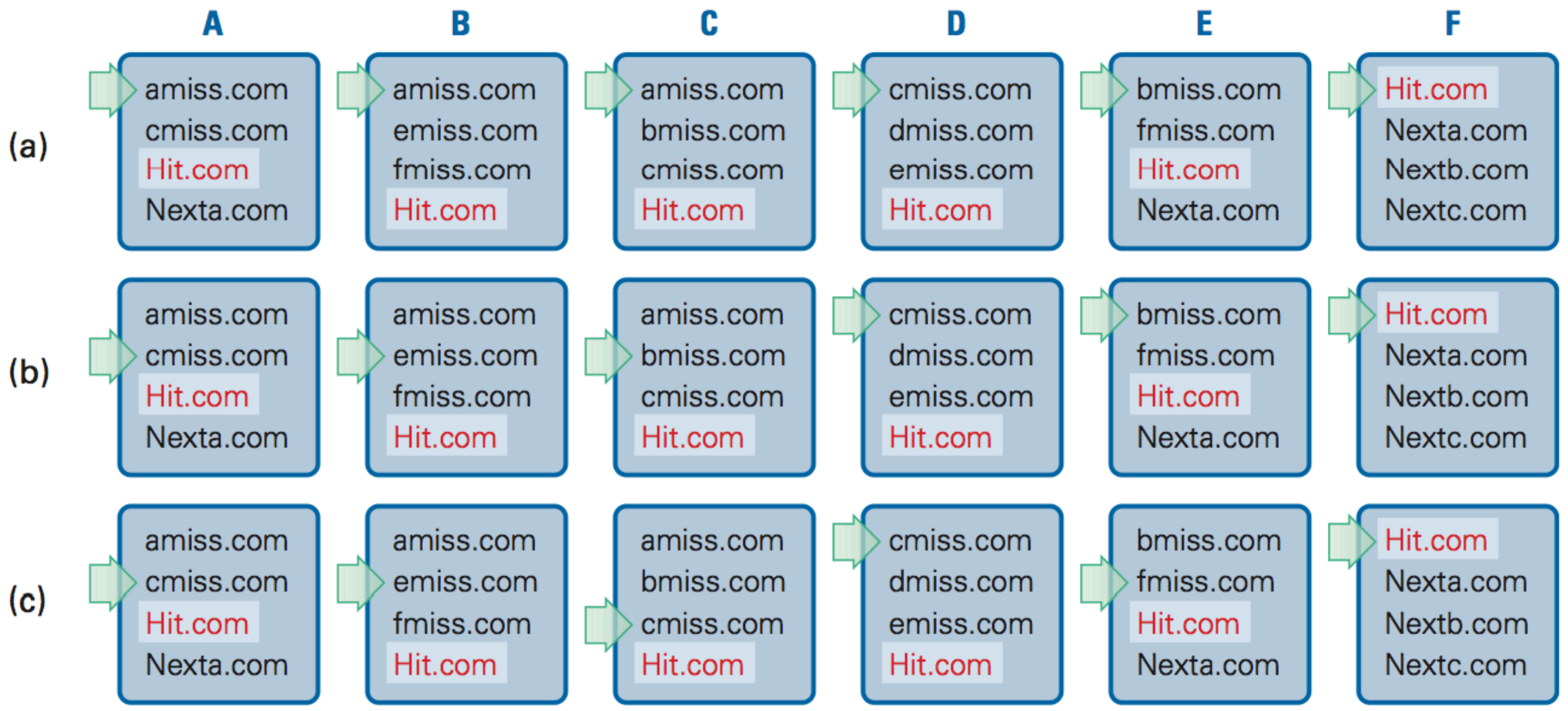
1. Parse the query.
2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and 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

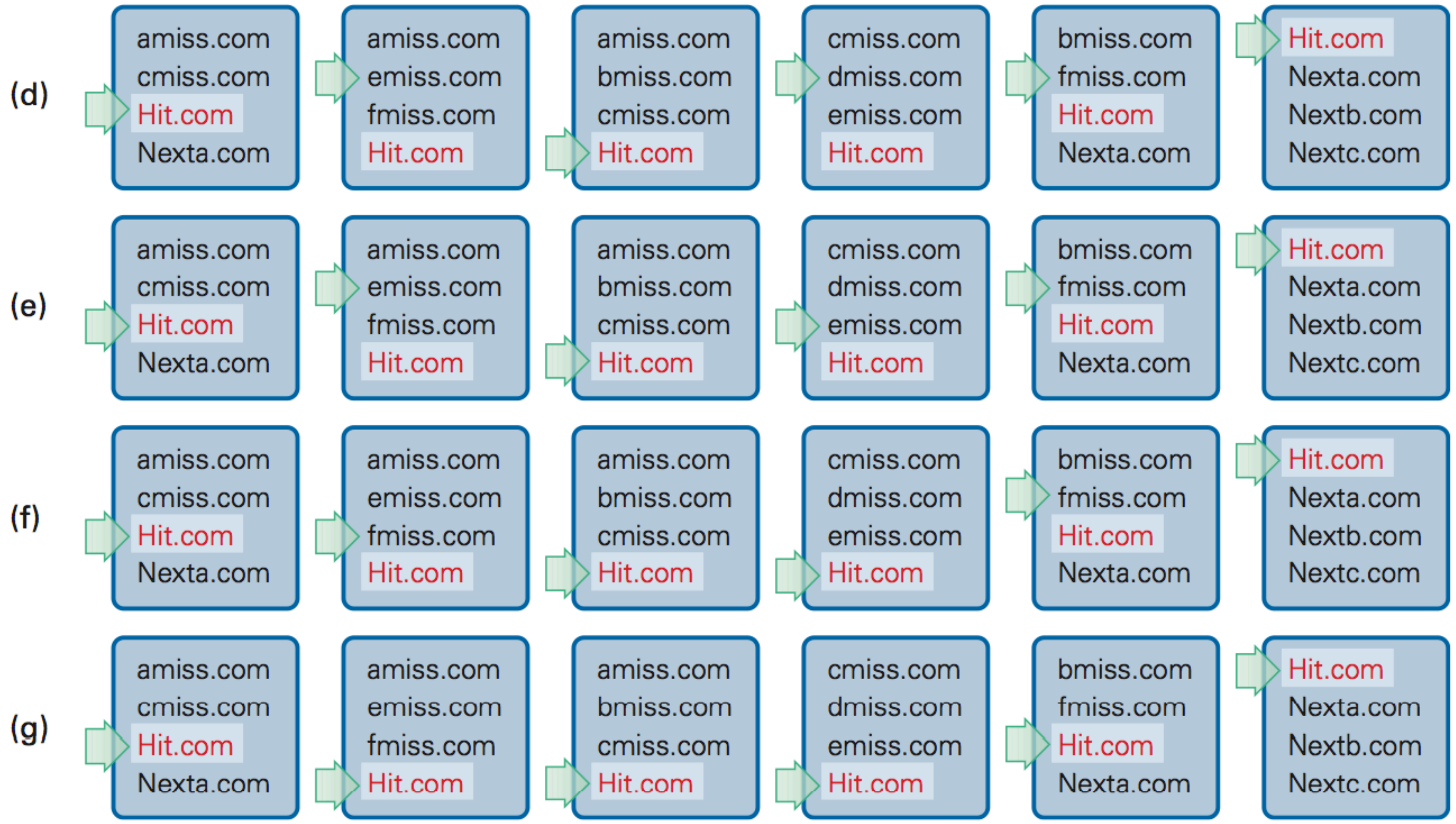
- Algorithms are "given" at many levels of detail – it depends on what agent needs

Intersect Alphabetical Lists

1. Place a marker at head of every list
2. If all markers point to same URL, record a hit
3. Advance marker on alphabetically earliest list(s)
4. If any list is finished stop; otherwise go to step 2

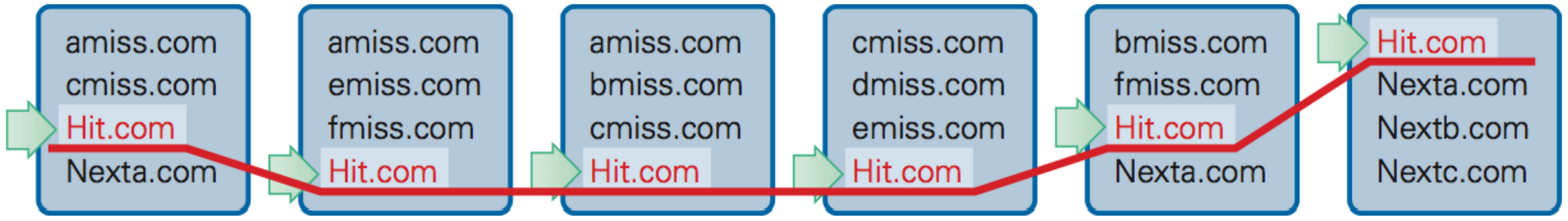


Intersect Alphabetical Lists


(d)
amiss.com cmiss.com Hit.com Nexta.com
amiss.com emiss.com fmiss.com Hit.com
amiss.com bmiss.com cmiss.com Hit.com
cmiss.com dmiss.com emiss.com Hit.com
bmiss.com fmiss.com Hit.com Nexta.com
Hit.com Nexta.com Nextb.com Nextc.com
(e)
amiss.com cmiss.com Hit.com Nexta.com
amiss.com emiss.com fmiss.com Hit.com
amiss.com bmiss.com cmiss.com Hit.com
cmiss.com dmiss.com emiss.com Hit.com
bmiss.com fmiss.com Hit.com Nexta.com
Hit.com Nexta.com Nextb.com Nextc.com
(f)
amiss.com cmiss.com Hit.com Nexta.com
amiss.com emiss.com fmiss.com Hit.com
amiss.com bmiss.com cmiss.com Hit.com
cmiss.com dmiss.com emiss.com Hit.com
bmiss.com fmiss.com Hit.com Nexta.com
Hit.com Nexta.com Nextb.com Nextc.com
(g)
amiss.com cmiss.com Hit.com Nexta.com
amiss.com emiss.com fmiss.com Hit.com
amiss.com bmiss.com cmiss.com Hit.com
cmiss.com dmiss.com emiss.com Hit.com
bmiss.com fmiss.com Hit.com Nexta.com
Hit.com Nexta.com Nextb.com Nextc.com

Does IAL Work?

- Think about what makes the algorithm work?
  - A barrier is an imaginary ragged line across all lists marking the position of a URL that is a hit:



  - At Step 2 all markers are at a barrier

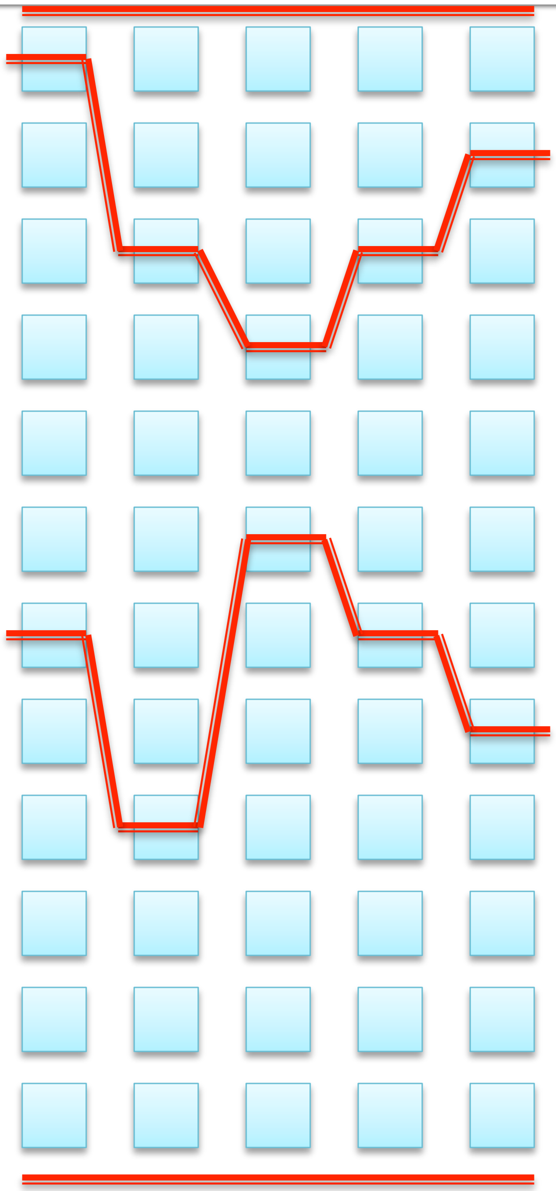
1. Place a marker at head of every list
2. If all markers point to same URL, record a hit
3. Advance marker on alphabetically earliest list(s)
4. If any list is finished stop; otherwise go to step 2

Operating Between 2 Barriers

- Points to notice:
  - IAL starts at a barrier (1)
  - All markers ‘step across’

barrier together

  - No marker crosses w/o others



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!