WEB INTELLIGENCE

Lecture 4: Text Processing II

Russa Biswas

September 23, 2025

AALBORG UNIVERSITY

(Based on slides of Dan Jurafsky, Chris Manning, Mausam, Manfred Jaeger)

Binary Term-Document Incidence Matrix

documents

terms	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t, and is 0 otherwise

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Julius Caesar: 1111000

Binary Term-Document Incidence Matrix

documents

terms	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t, and is 0 otherwise

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Brutus: 110100
Caesar: 110111

Limitations with Bigger Collection

	#no. of documents	#unique words	#size of a Term-Document Incidence Matrix
William Shakespear's plays	38	20,000 – 30,000	38 x 30k
English Wikipedia (as of 29th September, 2024)	6.89 million	> 1.7 million	6.89 x 1.7

A term incidence matrix with V terms and D documents has **O(V x D)** entries, i.e., size of the Term-Document Incidence Matrix

Storage:

6.89 million × 1.7 million = 11.713 Trillion elements

For 32-bit int (4 bytes per element) = 46.85 TB
For 8-bit byte (1 byte per element) = 11.71 TB

Matrix is extremely sparse

https://www.shakespeare.org.uk/explore-shakespeare/shakespedia/shakespeares-words/

Inverted Index: Motivation

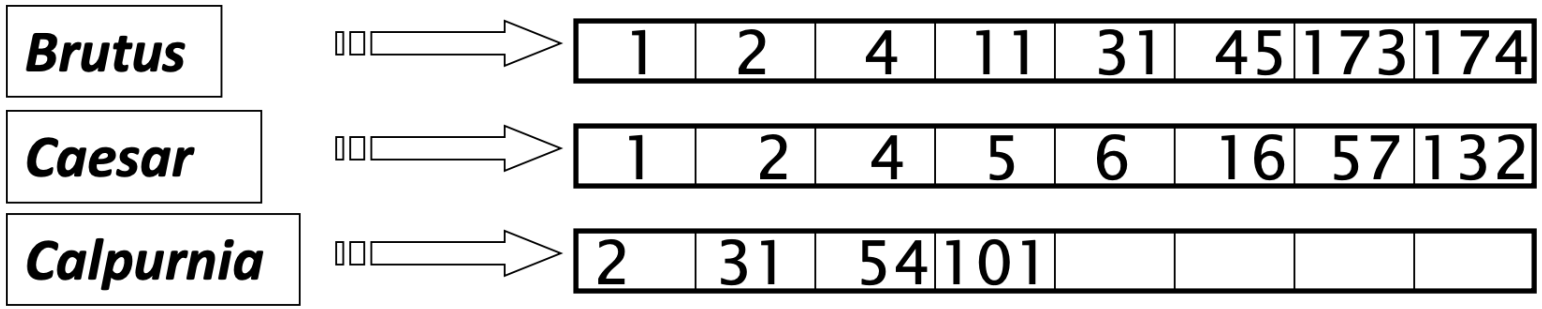
- A reasonably-sized index of the web contains many billions of documents and has a massive vocabulary.
- Search engines run roughly 10^5 queries per second over that collection.
- We need fine-tuned data structures and algorithms to provide search results in much less than a second per query. O(n) and even O(log n) algorithms are often not nearly fast enough.
- The solution to this challenge is to run an inverted index on a massive distributed system

Inverted Indexes

- Two insights allow us to reduce this to a manageable size:
  - The matrix is sparse – any document uses a tiny fraction of the vocabulary.
  - A query only uses a handful of words, so we don't need the rest. E.g., *BRUTUS and CAESAR*
- What's a better representation?
  - We only record the 1 positions
- We use an inverted index instead of using a term incidence matrix directly.

Inverted Index

- For each term t, we must store a list of all documents that contain t
  - Identify each doc by a **docID**, a document serial number

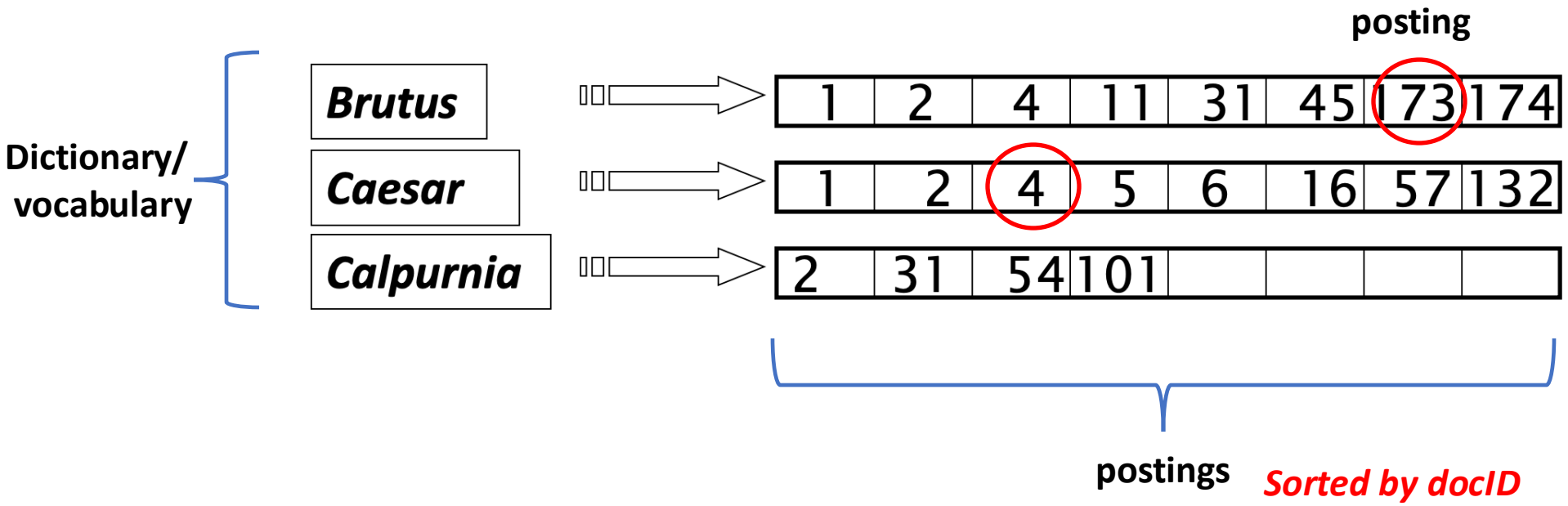


Can we use fixed-size arrays for this?

What happens if the word Caesar is added to document 14?

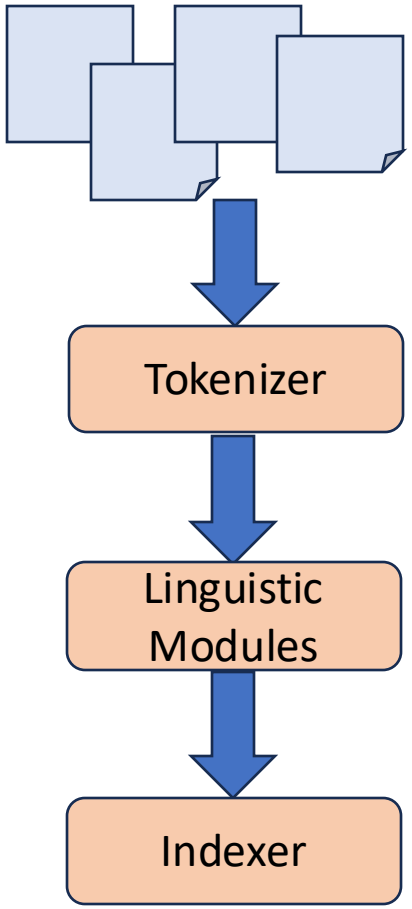
Inverted Index

- We need variable-size postings lists
  - On disk, a continuous run of postings is normal and best
  - In memory, can use linked lists or variable length arrays



Inverted Index Construction

Documents to be indexed



Token stream

Modified tokens

Inverted index

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Julius	Caesar	killed	ambitious

julius	caesar	kill	ambitious

Text Preprocessing - RECAP

- Tokenization
  - Cut character sequence into word tokens
    - Deal with ***"John's"***, ***a state-of-the-art solution***
- Normalization
  - Map text and query term to same form
    - You want ***U.S.A.*** and ***USA*** to match
- Stemming
  - We may wish different forms of a root to match
    - ***authorize***, ***authorization***
- Stop words
  - We may omit very common words (or not)
    - ***the, a, to, of***

Inverted Index

Indexer steps: Token sequence

- Sequence of (Modified token, Document ID) pairs.

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

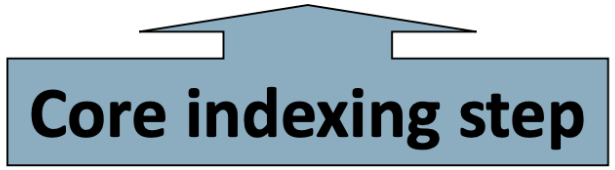
Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Inverted Index- Indexer Step : Sort

- Sort by terms
  - And then docID



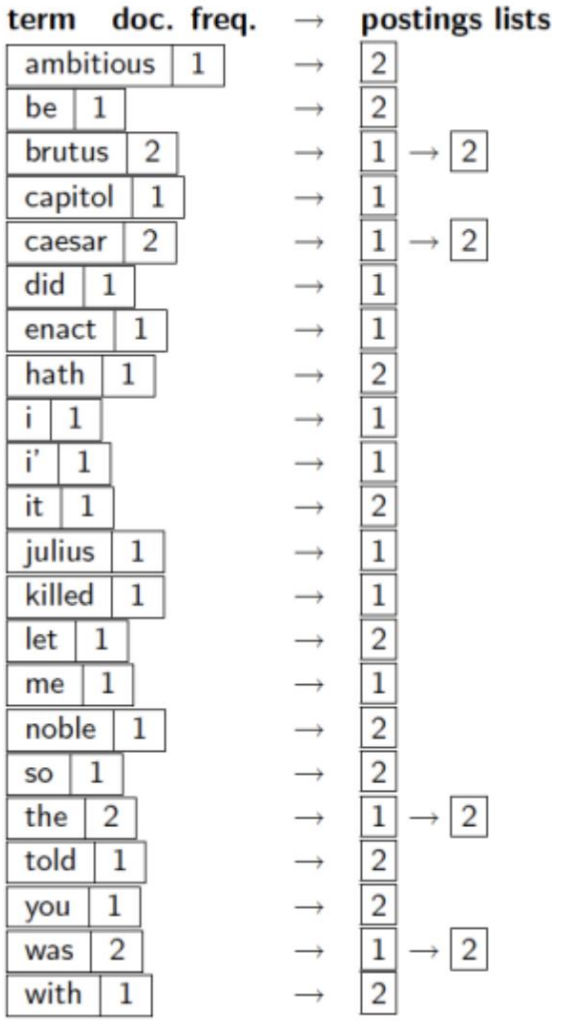
Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Inverted Index

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

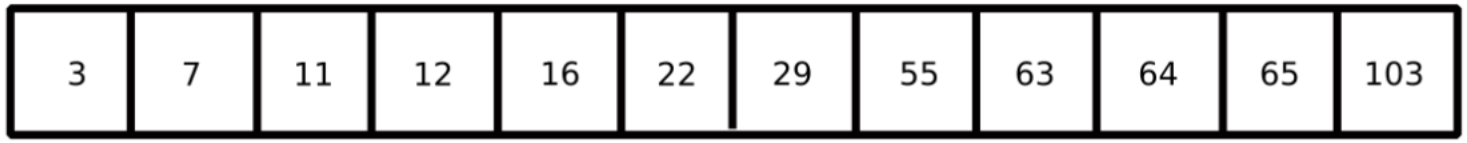
Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

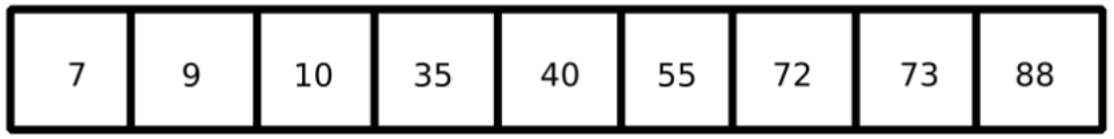


Query Processing

Consider processing the query:

- Brutus AND Caesar
  - Locate Brutus in the Dictionary;
    - Retrieve its postings.
  - Locate Caesar in the Dictionary;
    - Retrieve its postings
  - “Merge” the two postings



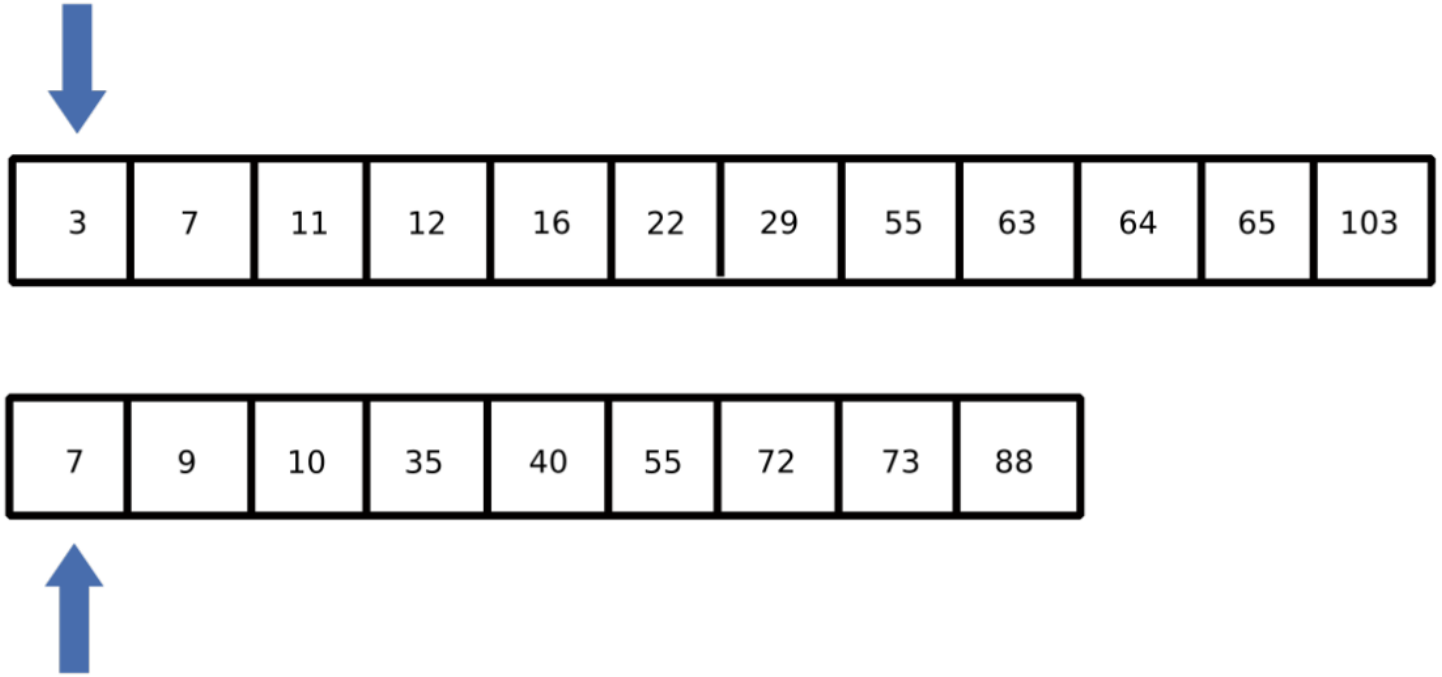


Merge Algorithm

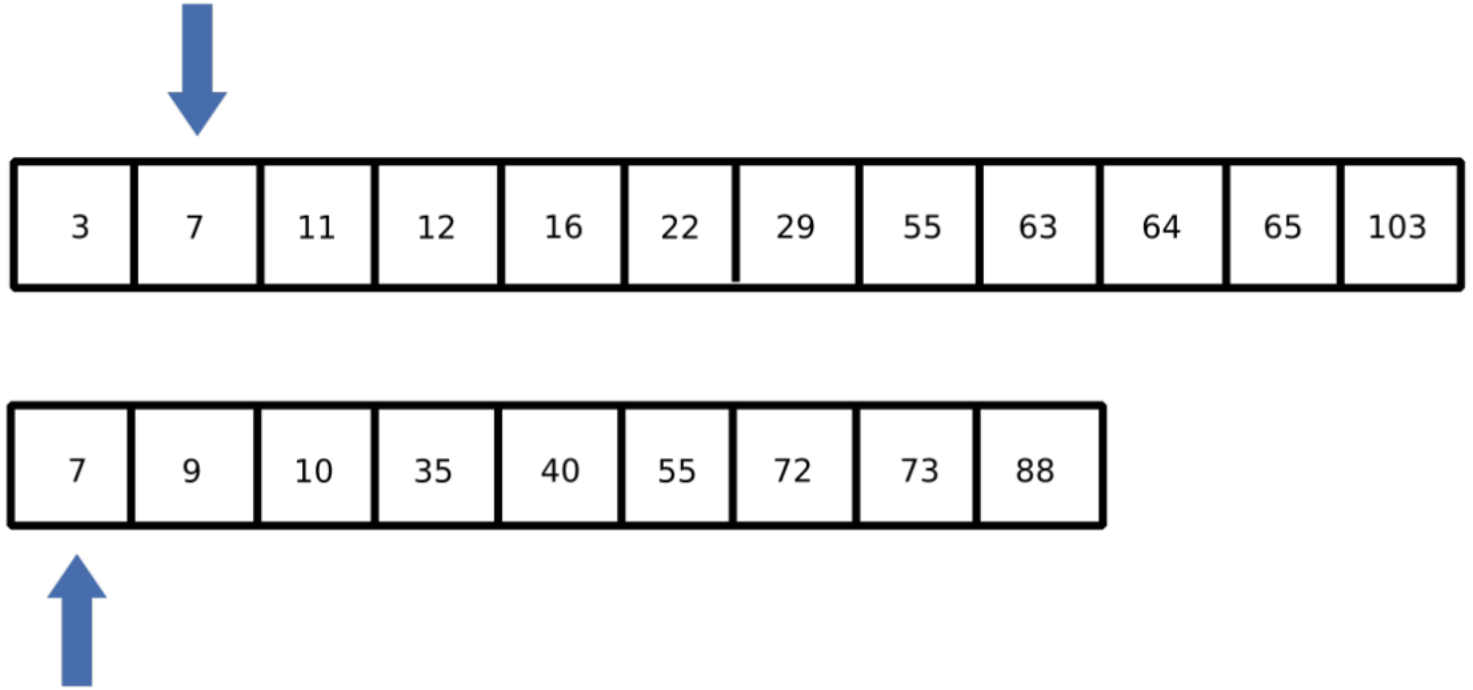
Intersecting two postings lists
(a "merge" algorithm)

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

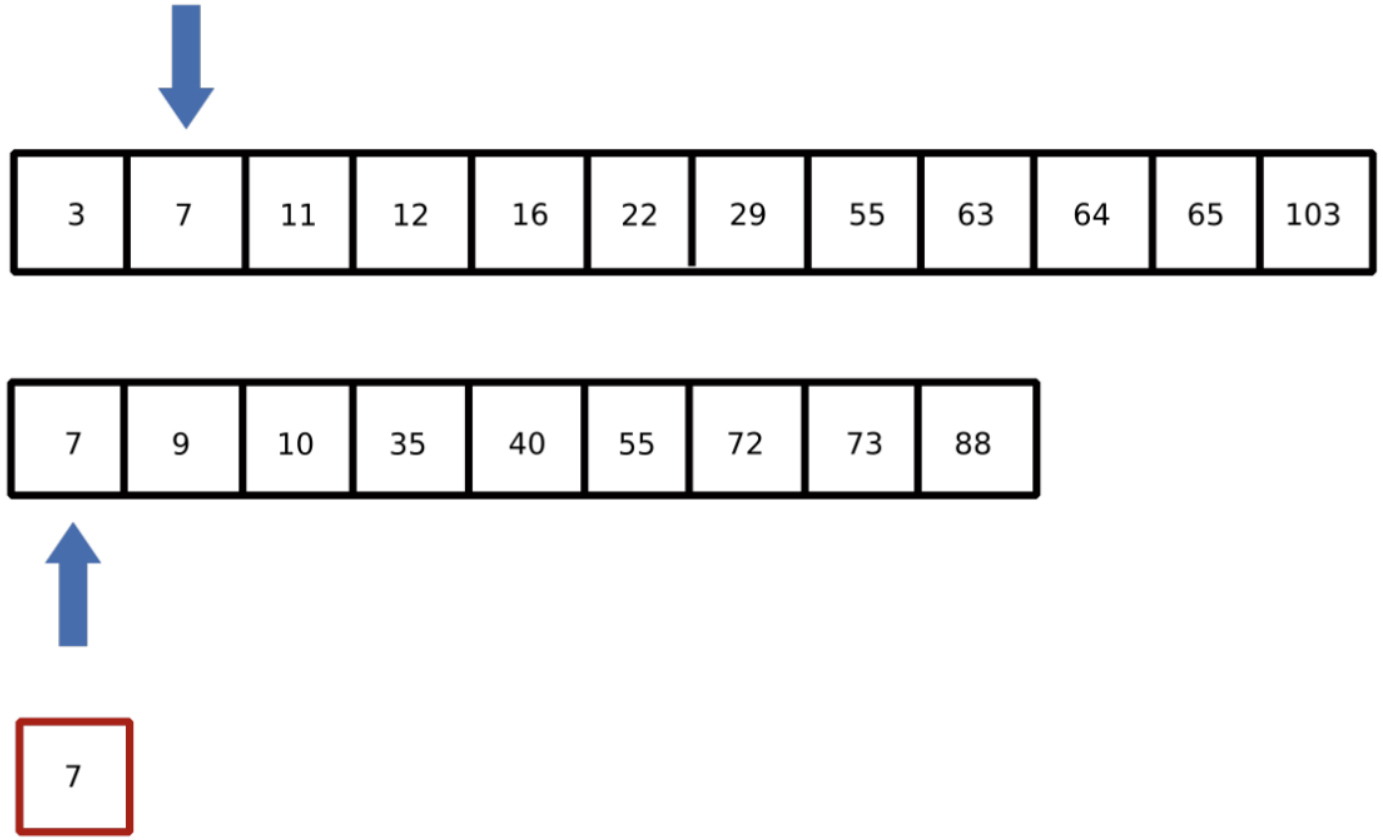
Merge Algorithm



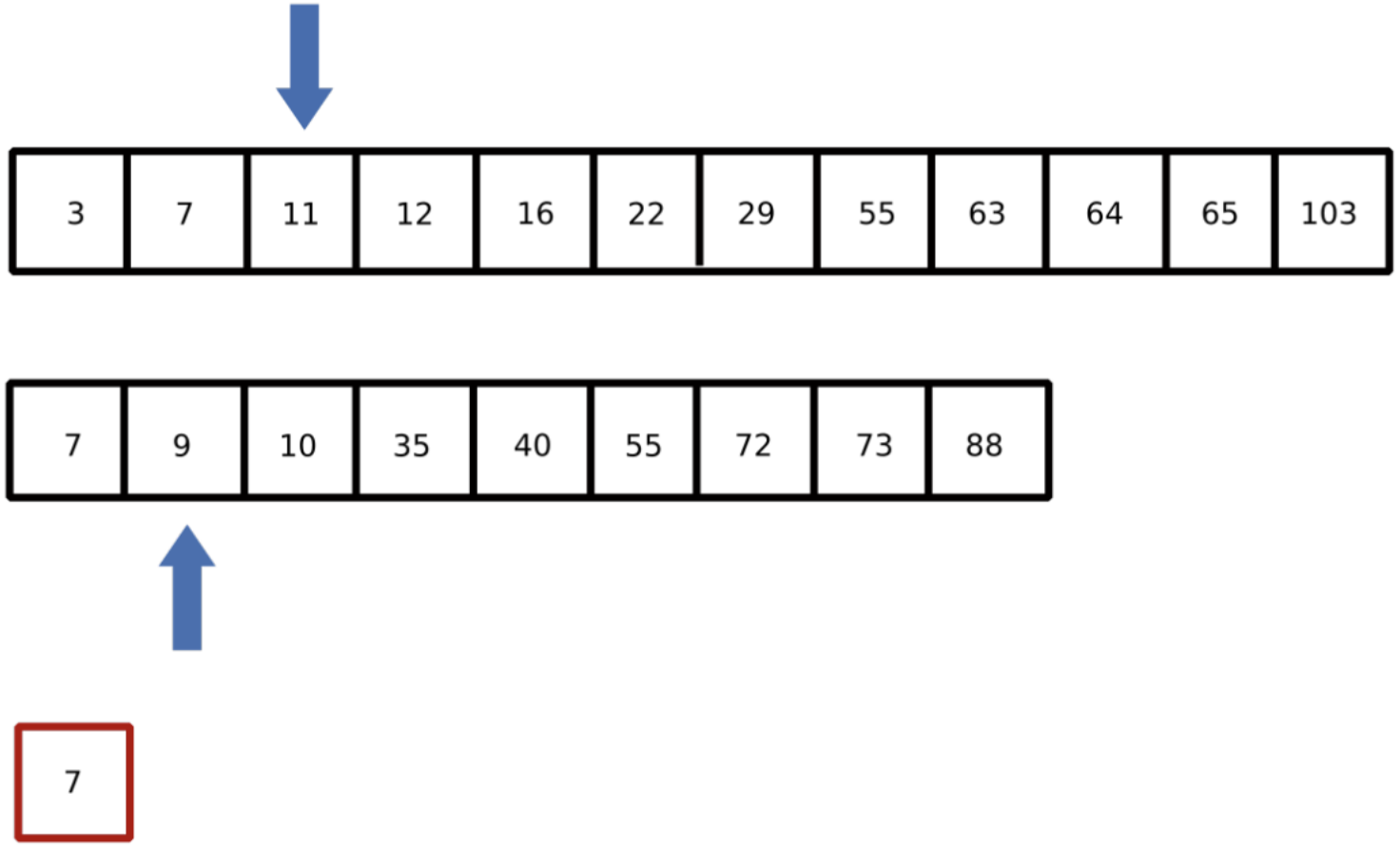
Merge Algorithm



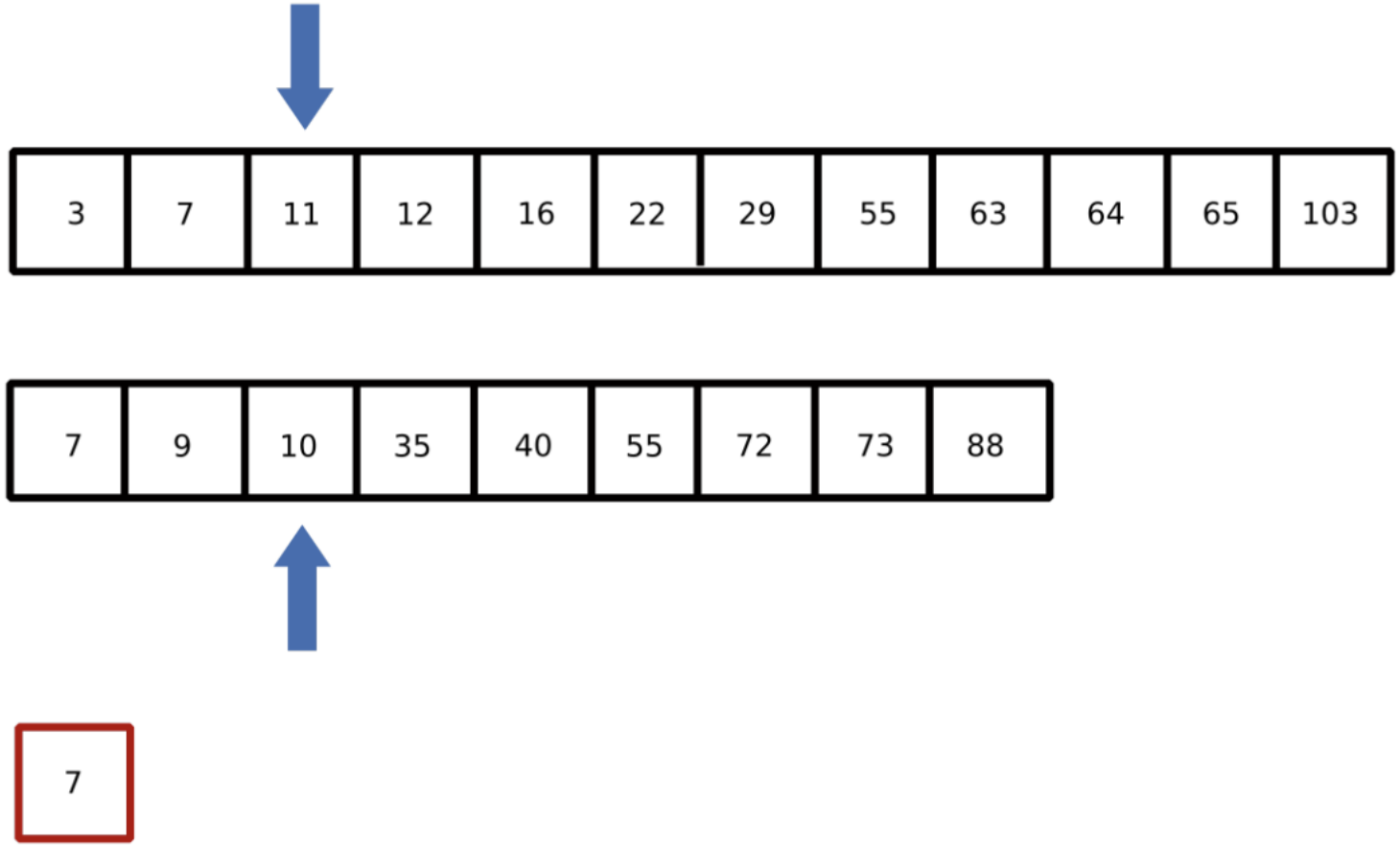
Merge Algorithm



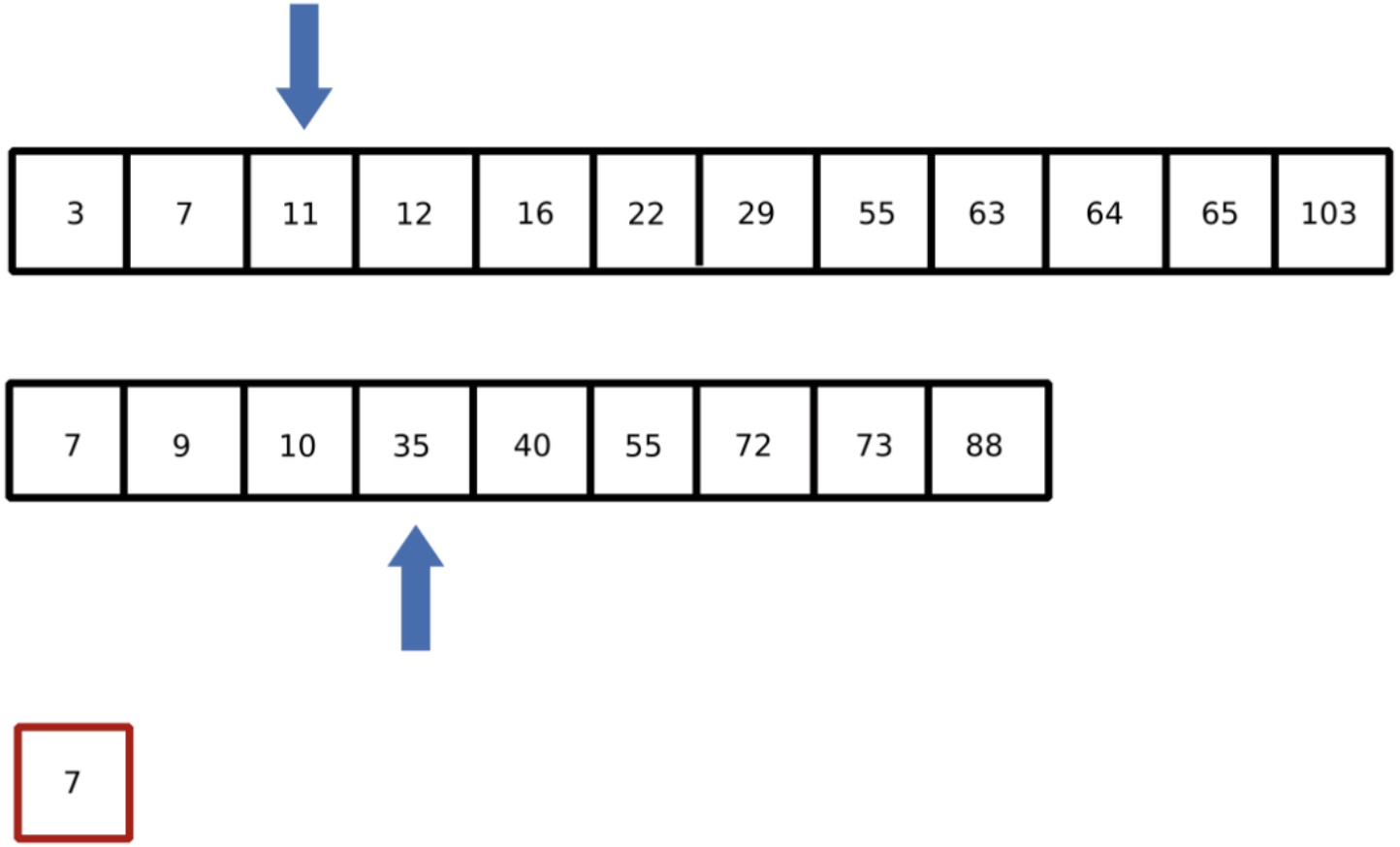
Merge Algorithm



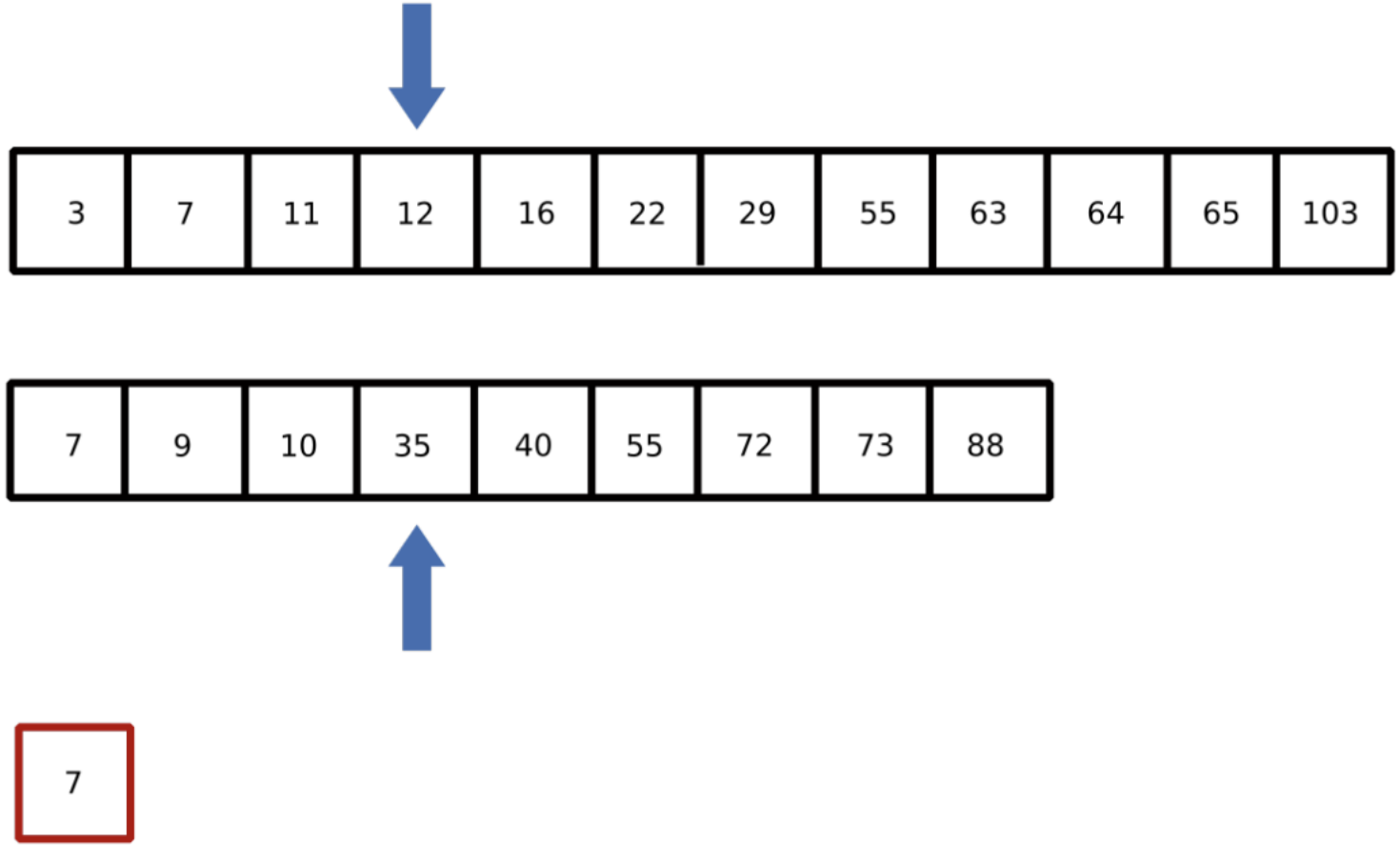
Merge Algorithm



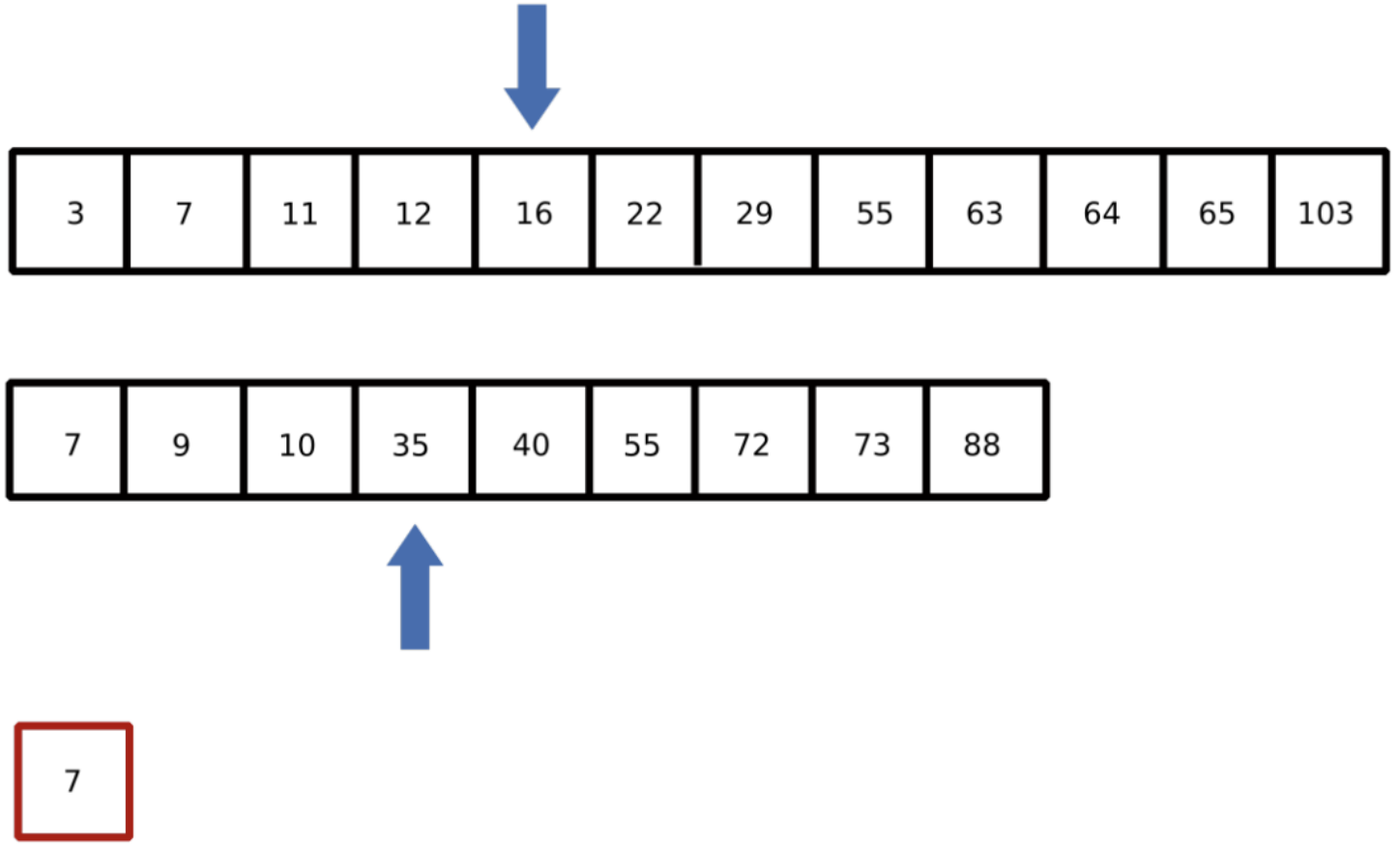
Merge Algorithm



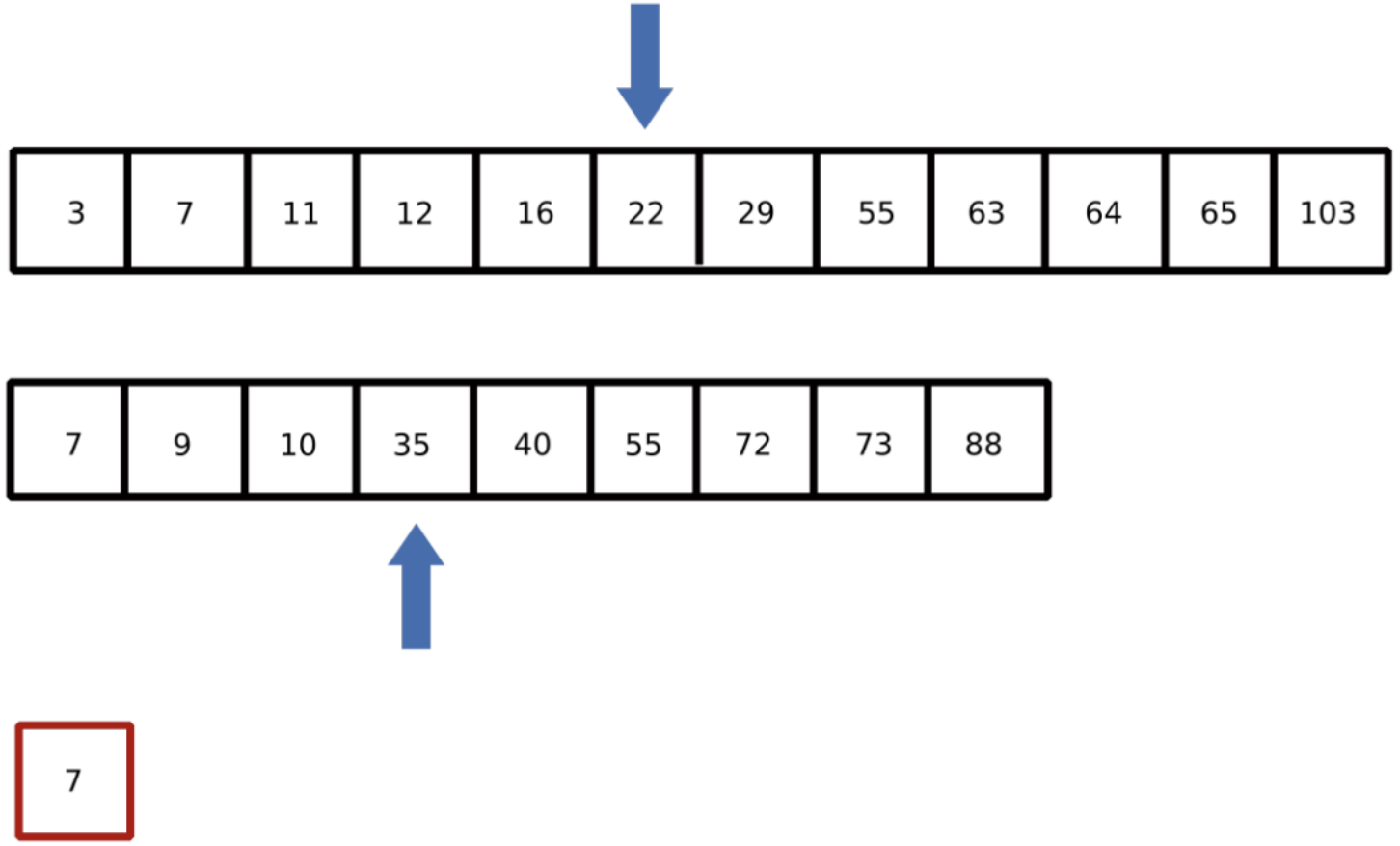
Merge Algorithm



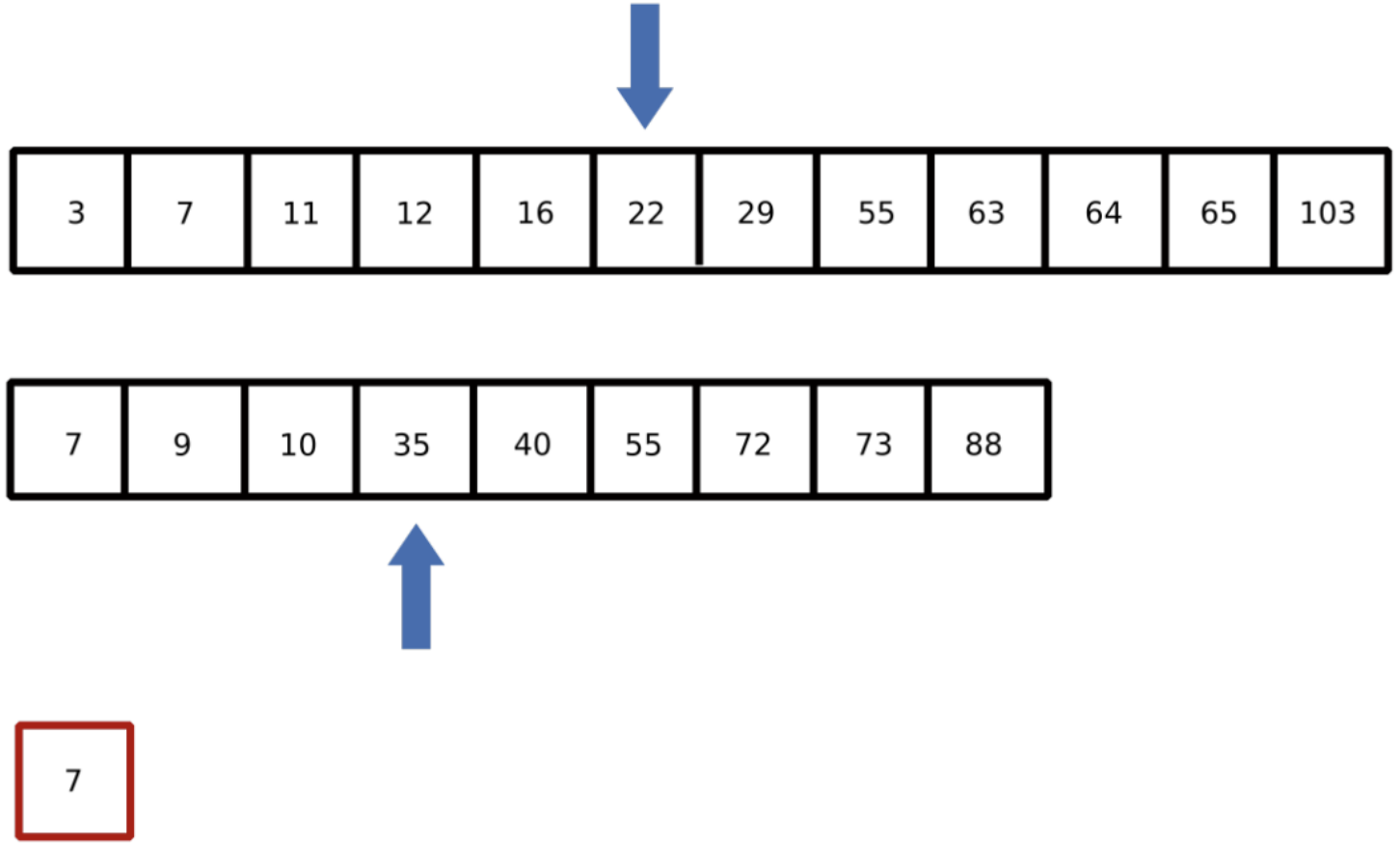
Merge Algorithm



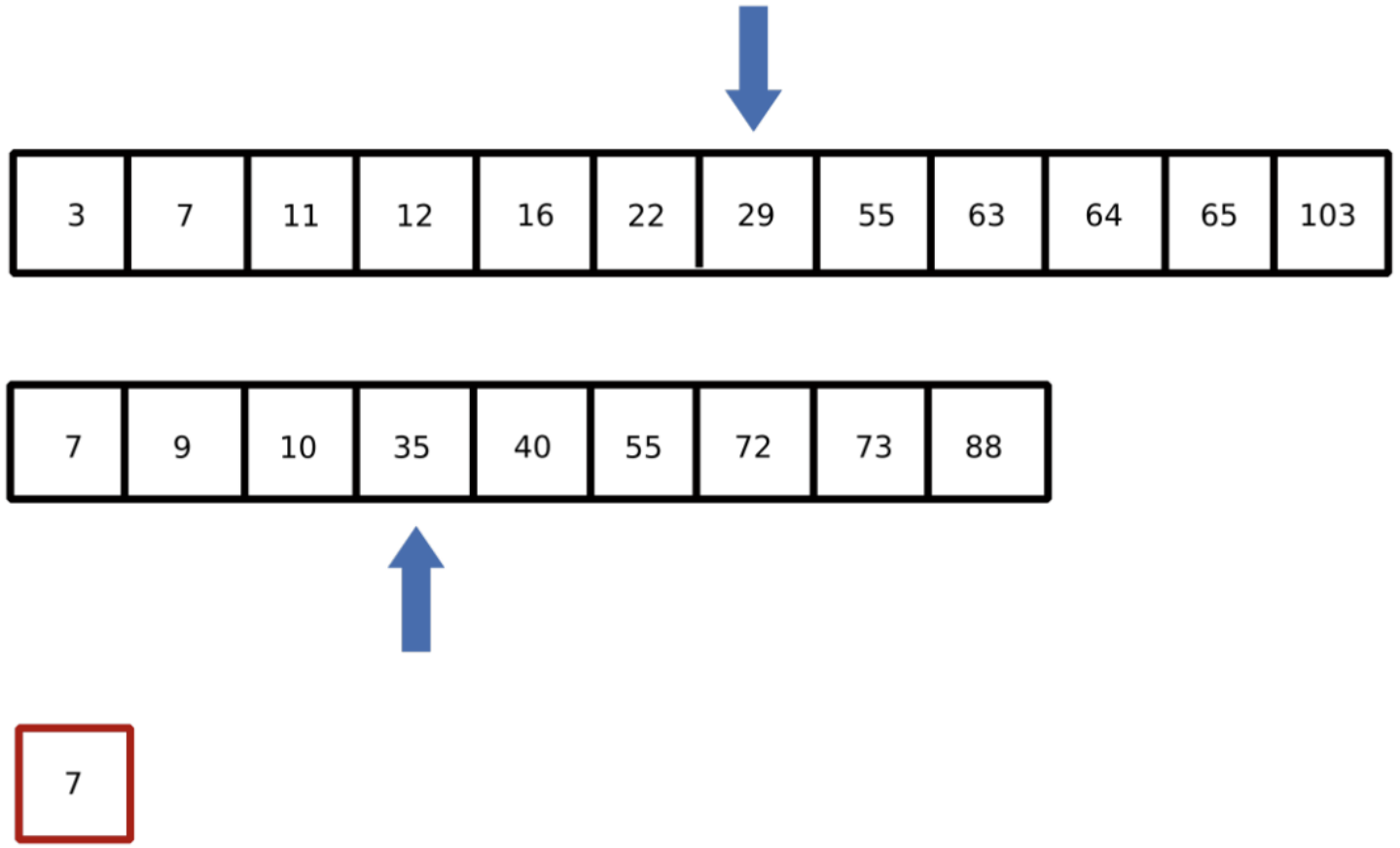
Merge Algorithm



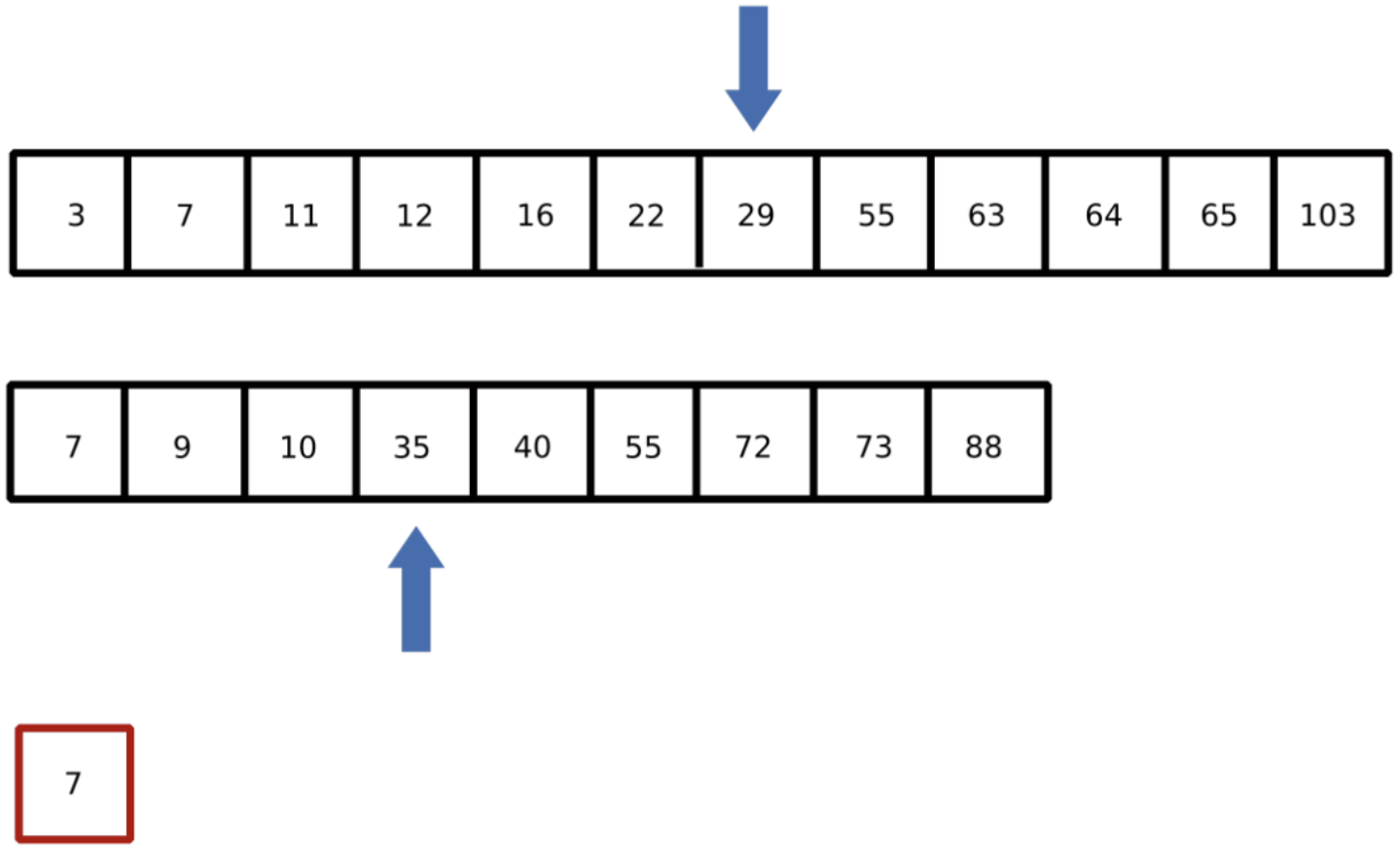
Merge Algorithm



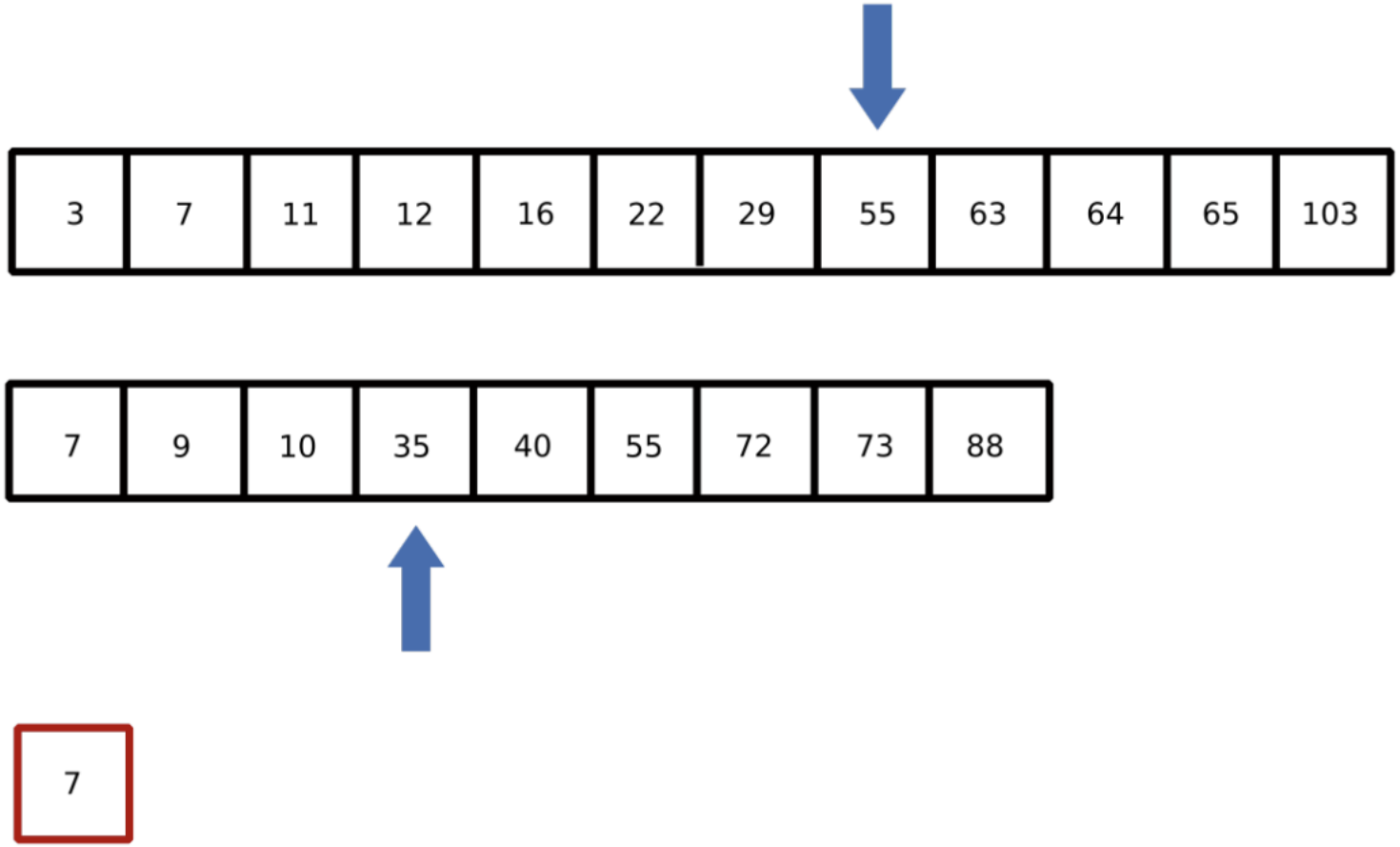
Merge Algorithm



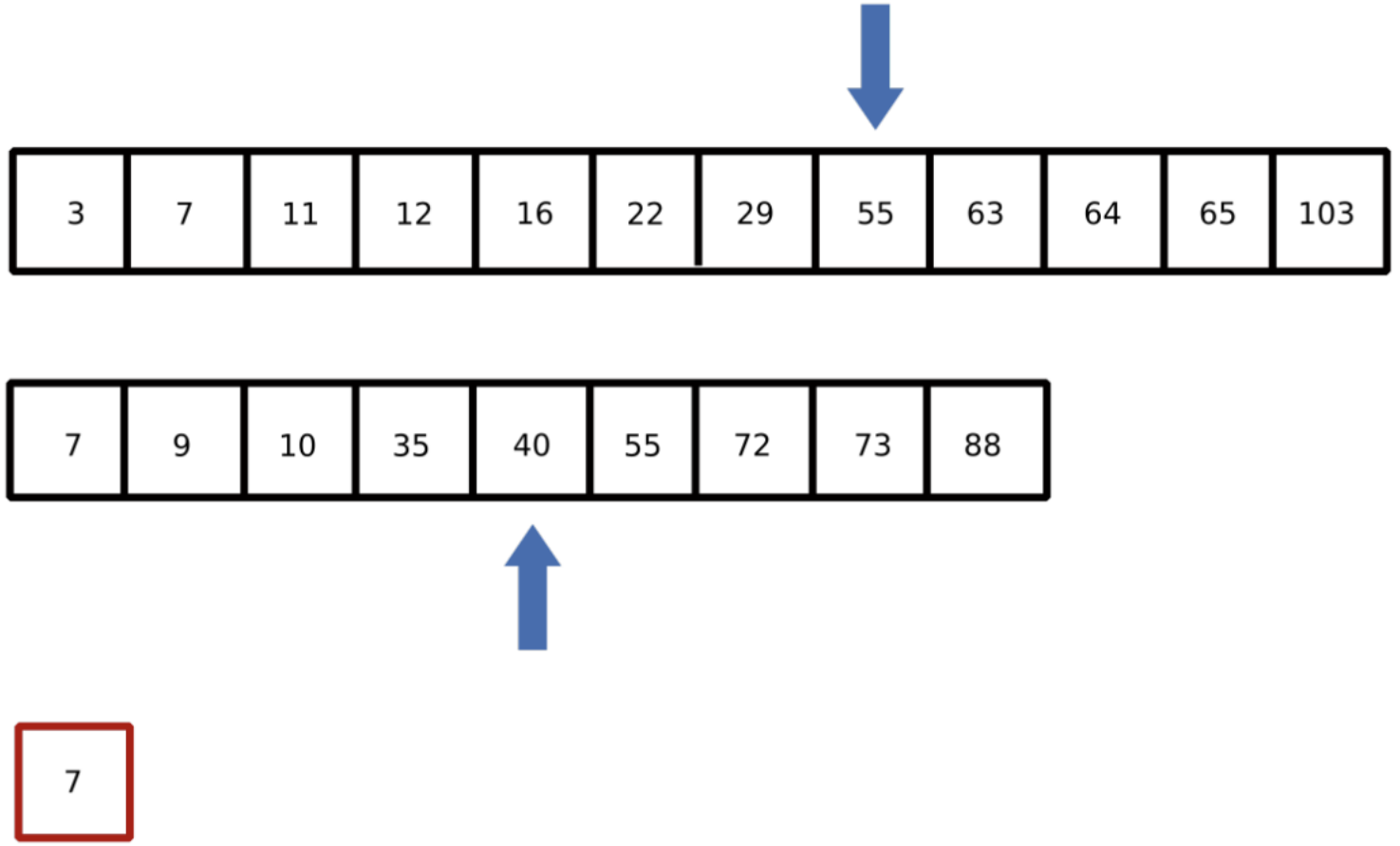
Merge Algorithm



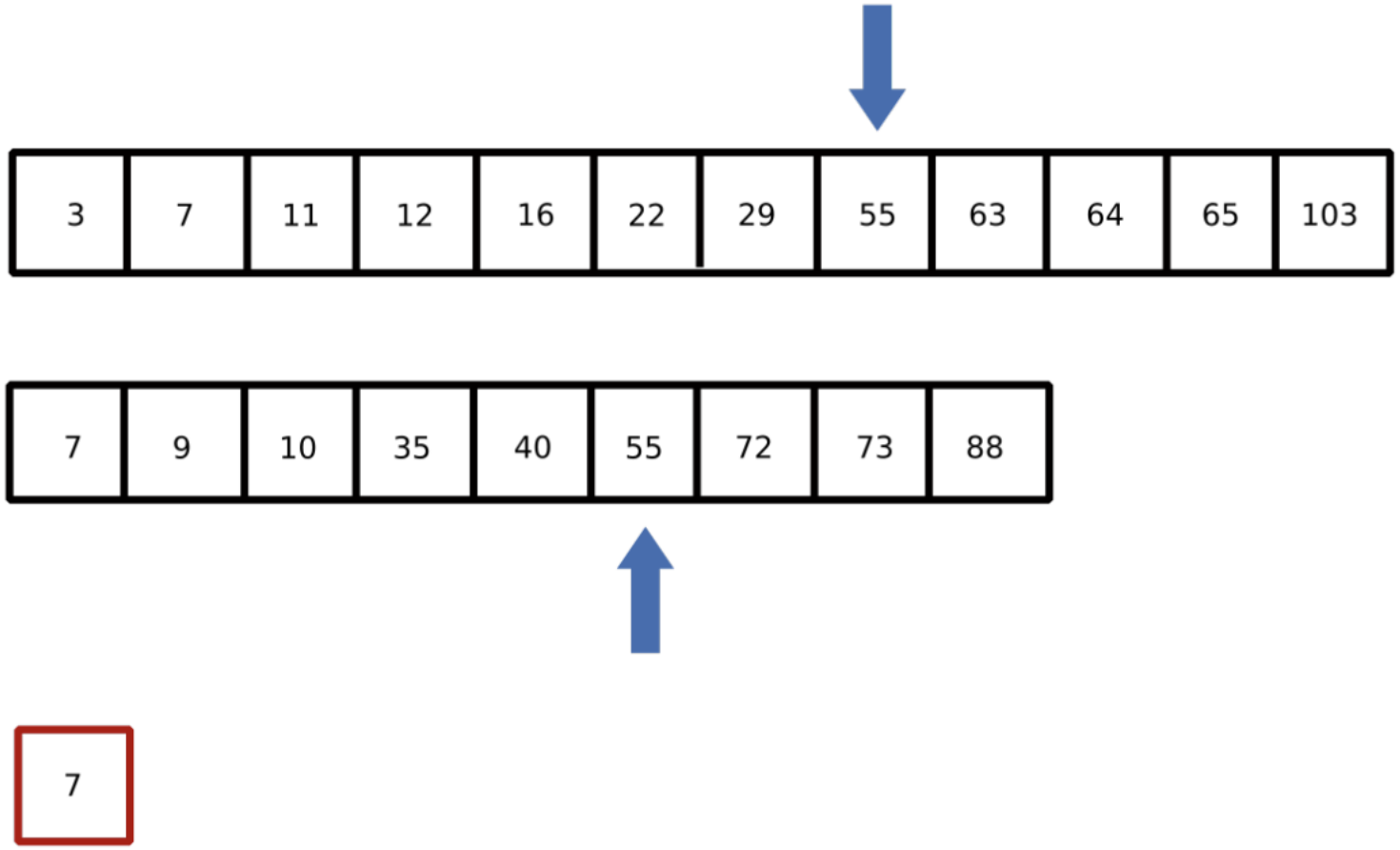
Merge Algorithm



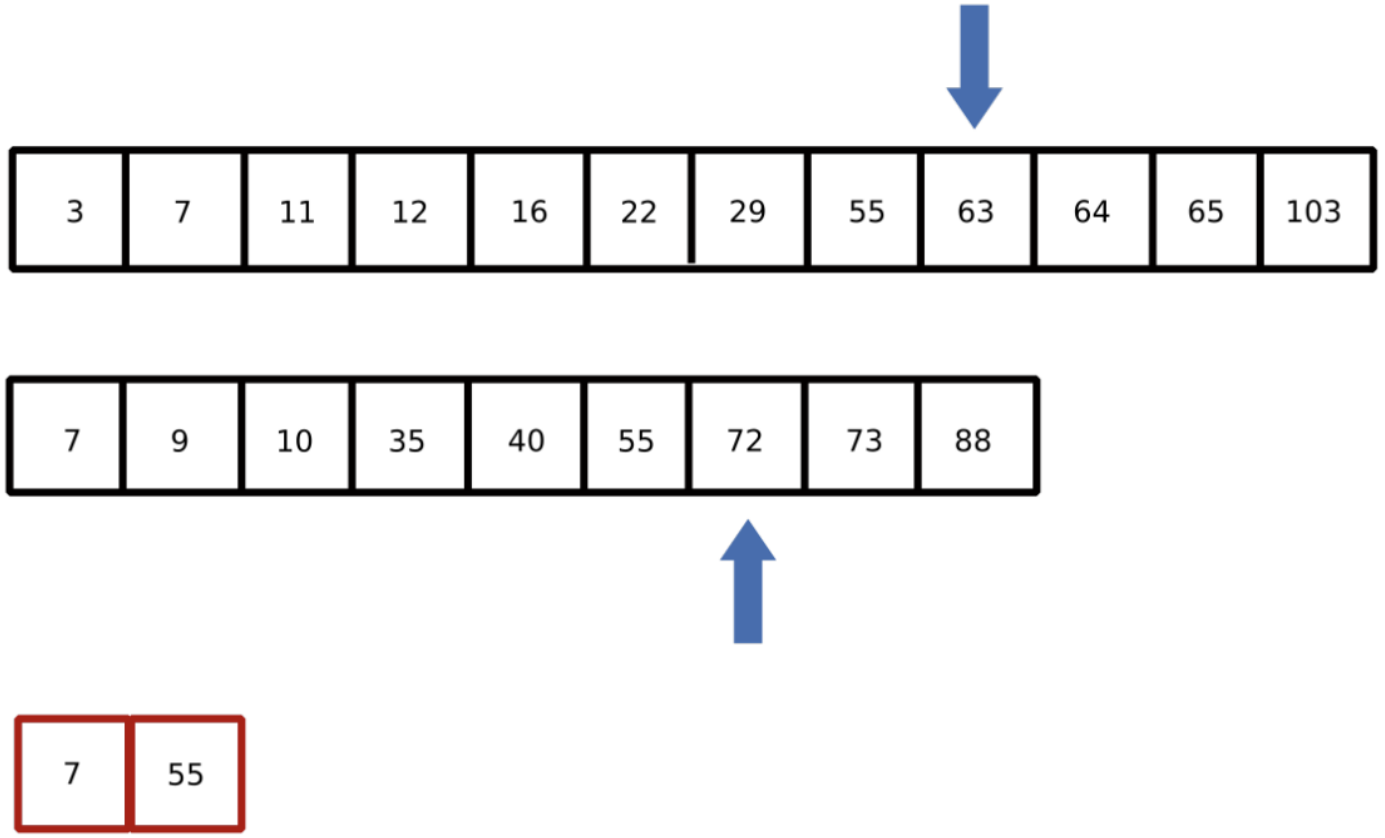
Merge Algorithm



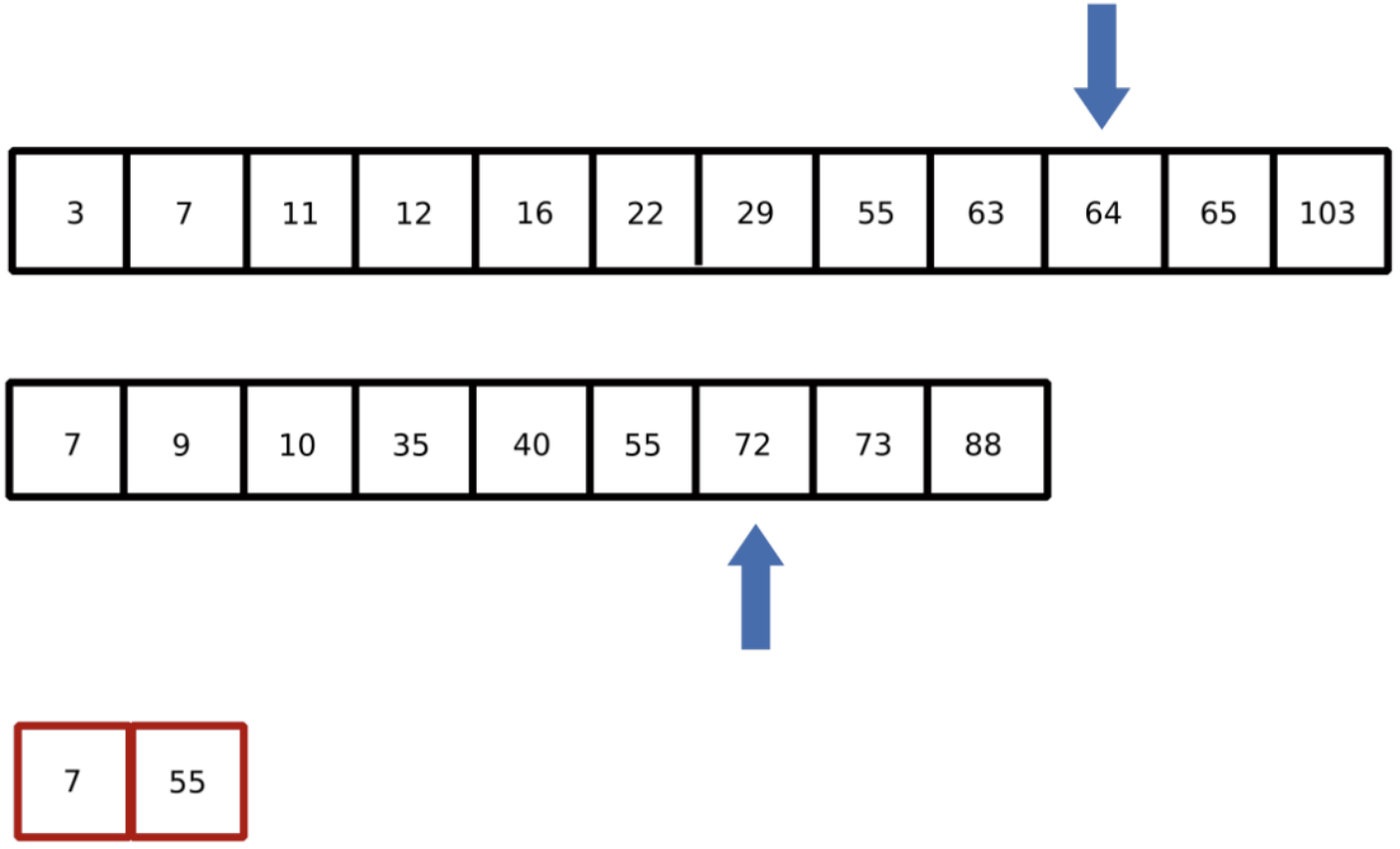
Merge Algorithm



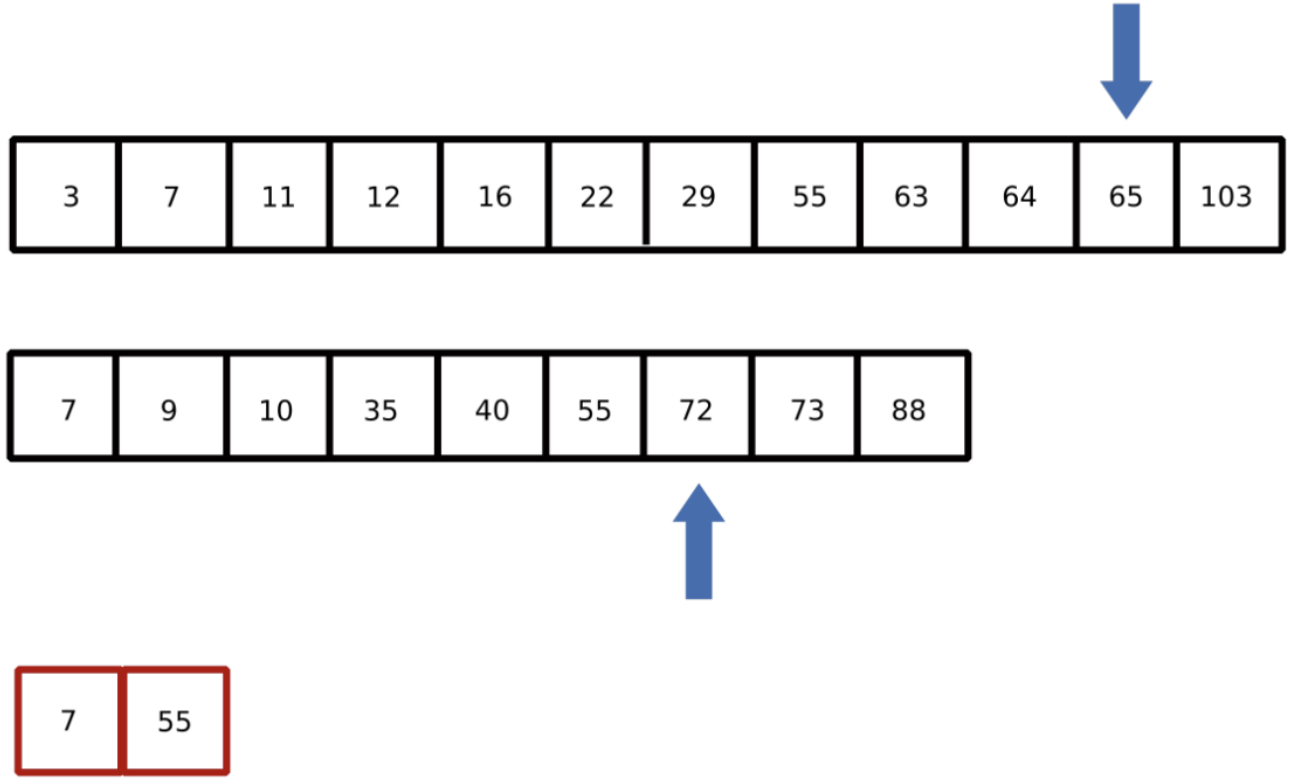
Merge Algorithm



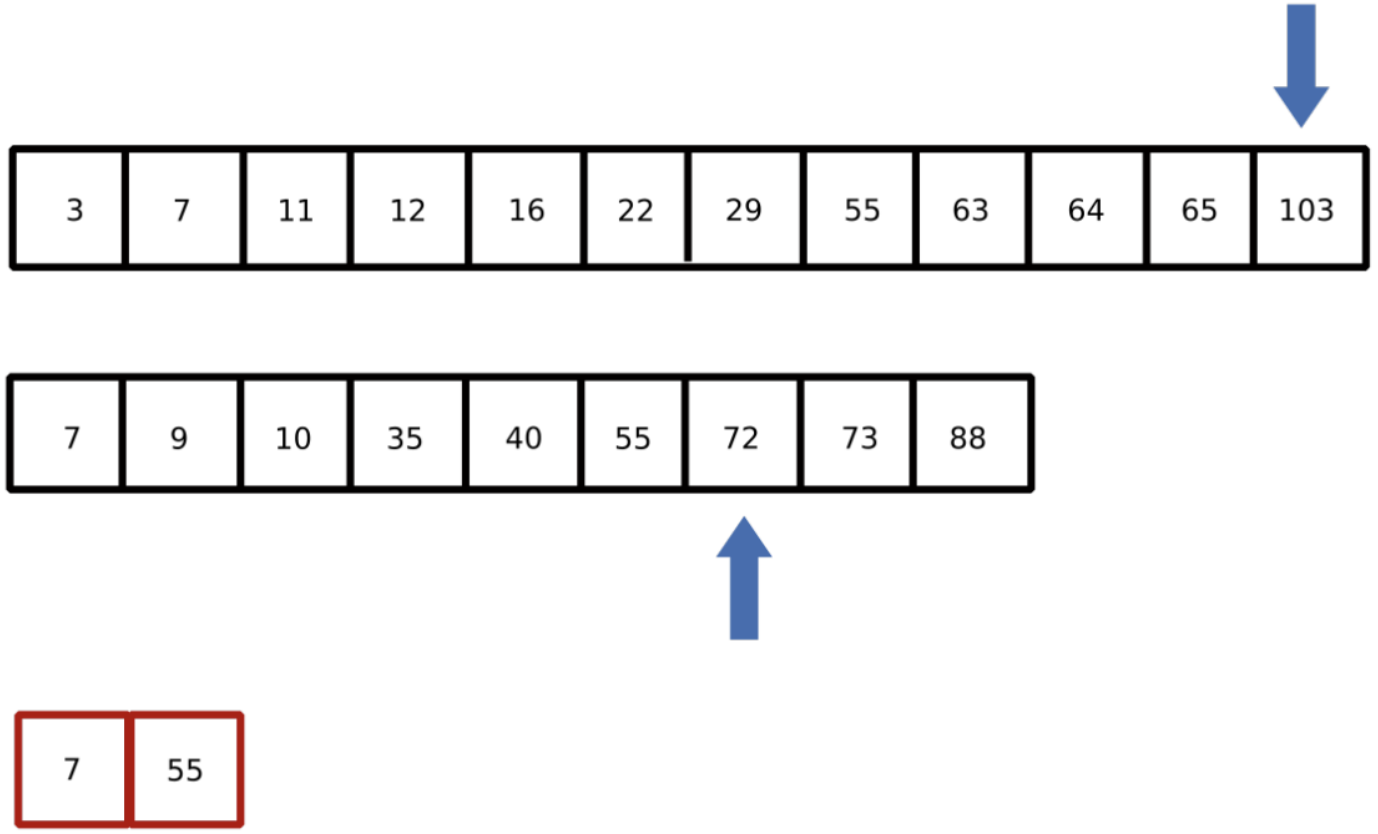
Merge Algorithm



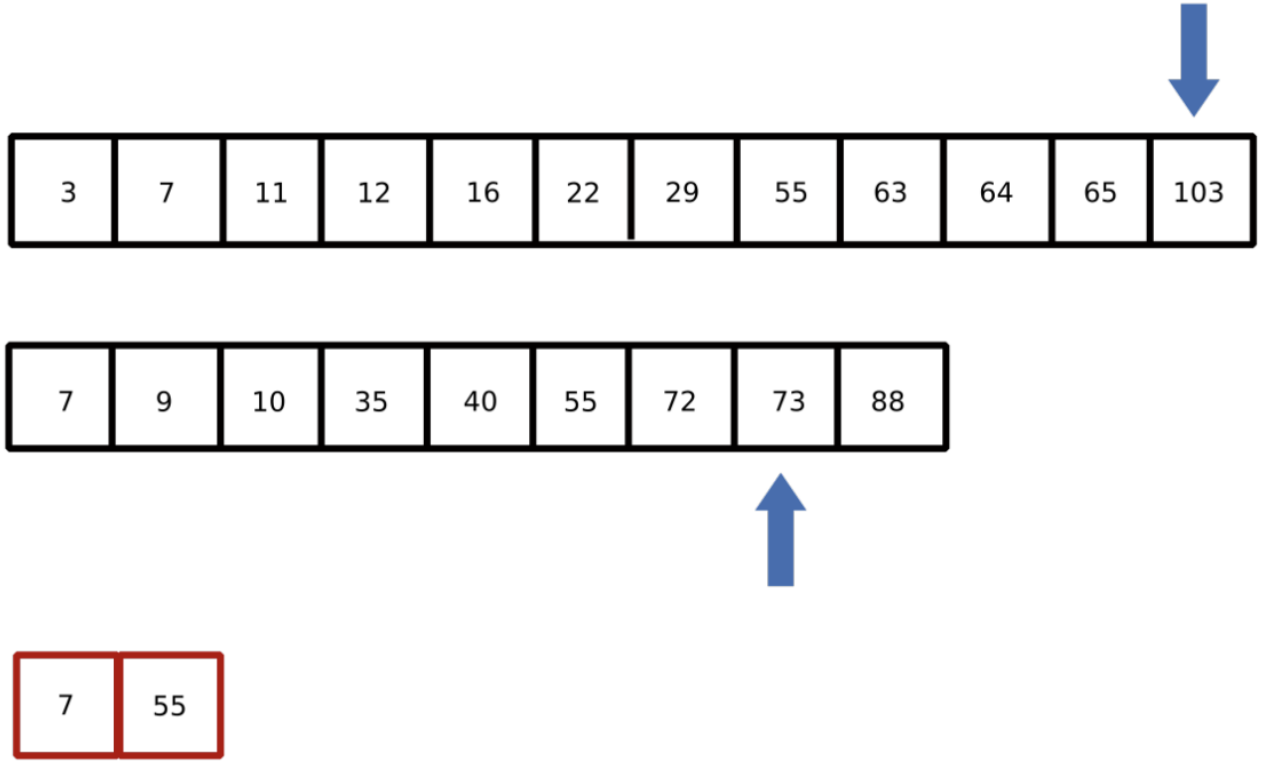
Merge Algorithm



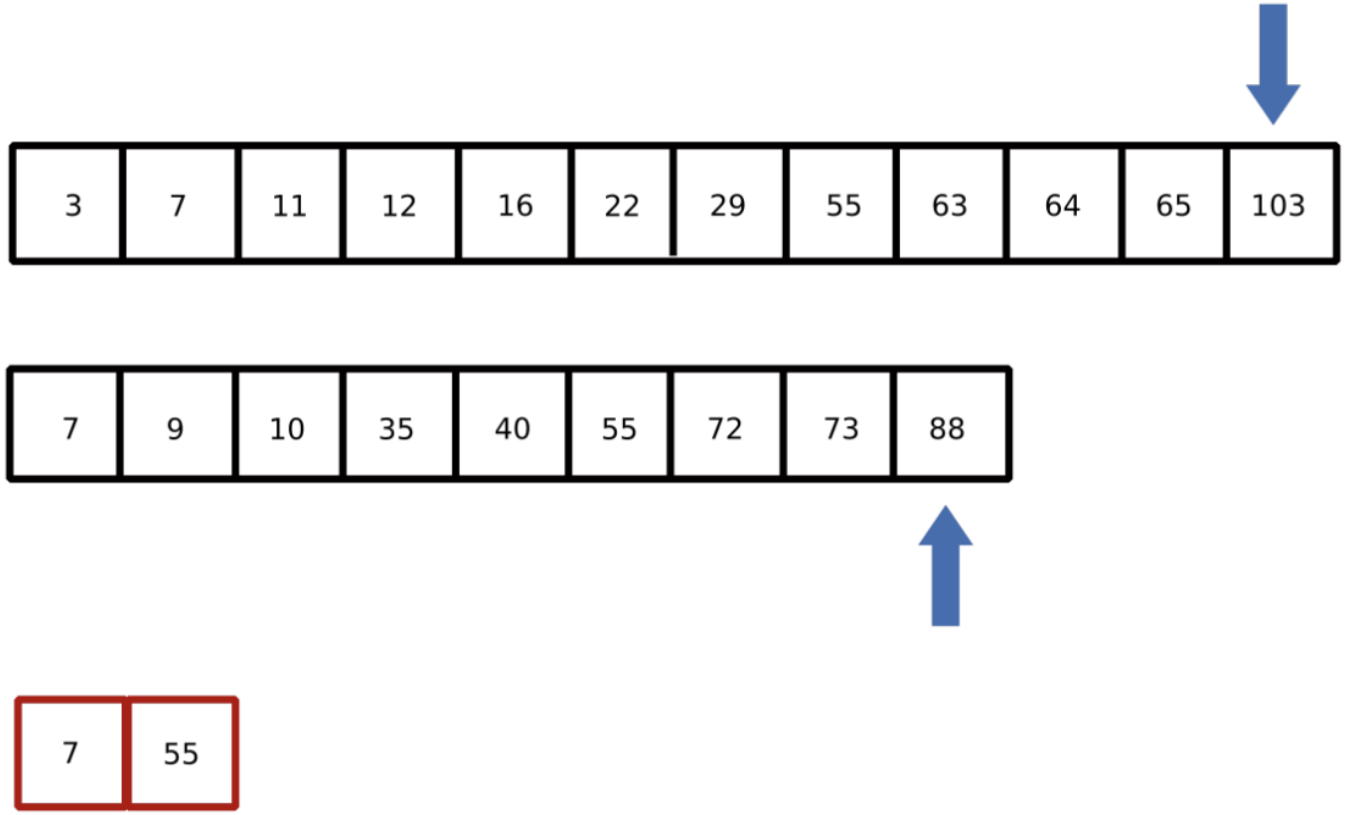
Merge Algorithm



Merge Algorithm



Merge Algorithm



Merge Algorithm

- Walk through the two postings simultaneously, in time linear in the total number of postings entries
- If the list lengths are x and y, the merge takes **O(x+y)** operations.

Crucial: postings sorted by docID

More on Inverted Indexing

- An index can be updated from a new batch of documents by merging the posting lists from the new documents. However, this is inefficient for small updates.
- Instead, we can run a search against both old and new indexes and merge the result lists at search time. Once enough changes have accumulated, we can merge the old and new indexes in a large batch.
- In order to handle deleted documents, we also need to maintain a delete list of docids to ignore from the old index. At search time, we simply ignore postings from the old index for any docid in the delete list.
- If a document is modified, we place its docid into the delete list and place the new version in the new index.
- If each term's inverted list is stored in a separate file, updating the index is straightforward: we simply merge the postings from the old and new index.
- However, most filesystems can't handle very large numbers of files, so several inverted lists are generally stored together in larger files. This complicates merging, especially if the index is still being used for query processing.
- There are ways to update live indexes efficiently, but it's often simpler to simply write a new index, then redirect queries to the new index and delete the old one.

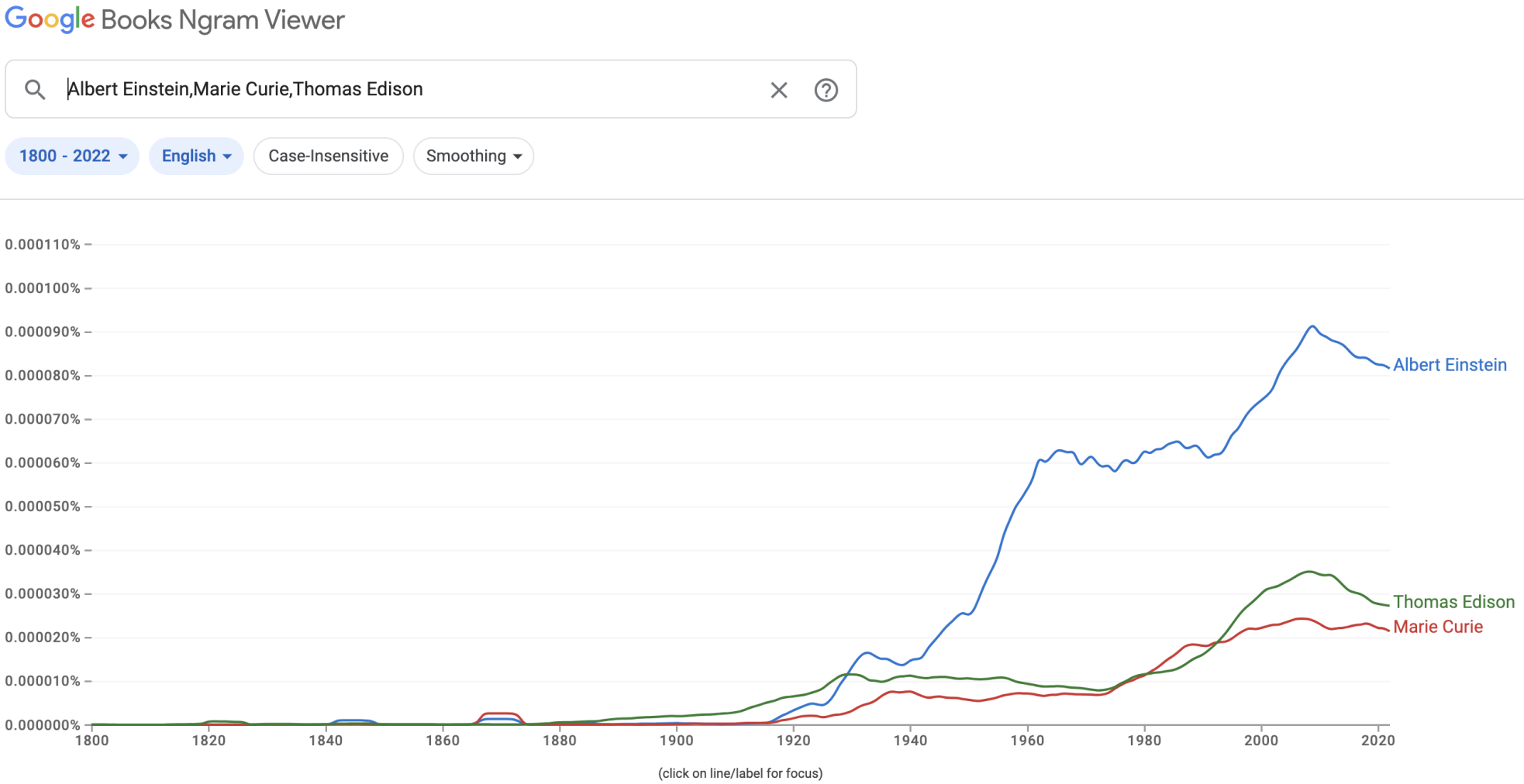
N-grams

N-grams are continuous sequence of n-items (words, characters, phonemes) from a given text

- For example, "Natural language processing involves the interaction between computers and human languages."
- **n-gram Examples:**

- **Unigrams (n = 1)**: Single words in sequence.
    - ["Natural", "language", "processing", "involves", "the", "interaction", "between", "computers", "and", "human", "languages"]
- **Bigrams (n = 2)**: Two-word sequences.
    - ["Natural language", "language processing", "processing involves", "involves the", "the interaction", "interaction between", "between computers", "computers and", "and human", "human languages"]
- **Trigrams (n = 3)**: Three-word sequences.
    - ["Natural language processing", "language processing involves", "processing involves the", "involves the interaction", "the interaction between", "interaction between computers", "between computers and", "computers and human", "and human languages"]

N-grams



https://books.google.com/ngrams/

Why do we need N-grams?

Used in Language Modeling (specifically statistical Language Models)

Applications:

How can we predict a word?

- ***Autocompletion/ Text prediction***
  - To be or not to ____
  - The cat is sitting on the ____
- ***Speech Recognition***
  - "They're car is in the garage" → "Their car is in the garage"
  - "She gave me a peace of cake" → "She gave me a piece of cake"
- ***Machine Translation***
  - **Large** winds tonight << **High** winds tonight
- ***Spelling Correction***
  - The office is about fifteen **minuets** from my house → The office is about fifteen **minutes** from my house

How can we predict a word?

Human Prediction:

- Domain Knowledge
  - The doctor prescribed _____
  - Predicted word: medicine/ painkillers/ antibiotics (domain information)
- Syntactic Knowledge
  - The cat quickly ____
  - Predicted word: ran/ jumped/moved (verb after an adverb)
- Lexical Knowledge
  - He made a remarkable ___
  - Predicted word: achievement/ discovery/ invention (understanding frequent word combinations and associations in a language)

- In an N-gram model, the last word w_n depends only on the previous (n-1) words (w_1 ,... w_{n-1}) (Markov assumption)

Literature

- https://www.cse.iitd.ac.in/~mausam/courses/csl772/autumn2014/lectures/14-ir.pdf
- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008.
- https://www.khoury.northeastern.edu/home/vip/teach/IRcourse/2_indexing_ngrams/slides/indexing_1.pdf