LECTURE 16

MERGESORT

MCS 275 Spring 2021

Emily Dumas

LECTURE 16: MERGESORT

Course bulletins:

- Starting with Quiz 6, you will have 48 hours for quizzes (Noon Sunday to Noon Tuesday).
- Project 2 description updated with sample data and modules policy.
- Project 2 due 6pm CST Friday, February 26.
- Check out the recursion sample code.
- Worksheet 7 will explore recursive maze solver / generator in more depth.

PLAN

- Discuss the theory of
  - Divide and conquer
  - Sorting
  - Mergesort
- Implement mergesort

DIVIDE AND CONQUER

A strategy that often involves recursion.

- **Split** a problem into parts.
- **Solve** for each part.
- **Merge** the partial solutions into a solution of the original problem.

Not always possible or a good idea. It only works if merging partial solutions is easier than solving the entire problem.

COMPARISON SORT

Suppose you have a list of objects that can be compared with $==$, $>$, $<$.

You'd like to reorder them in increasing order.

This problem is called **comparison sort**. There are many solutions.

MERGESORT

A divide-and-conquer solution to comparison sort.

It is a fast solution, often used in practice.

Key: It is pretty easy to take two sorted lists and merge them into a single sorted list.

So, let's divide our list into halves, sort each one (recursively), then merge them.

Now we'll formalize this.

Algorithm `mergesort`:

Input: list `L` whose elements support comparison.

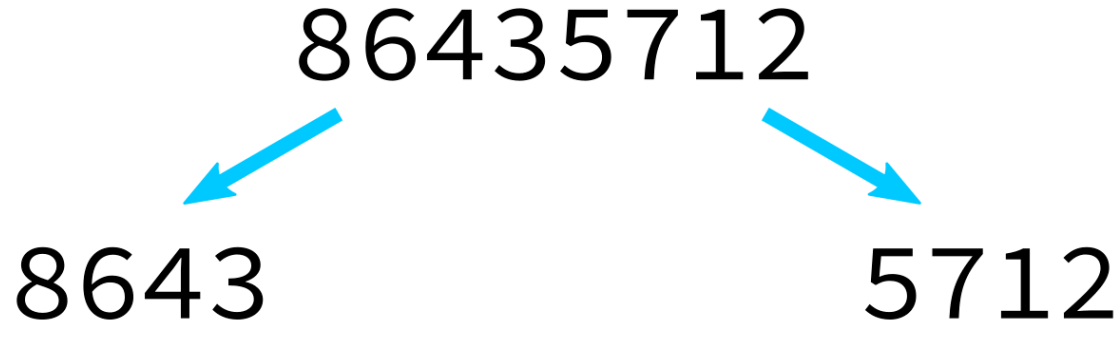
Goal: reorder the elements of `L` in place to achieve sorted order.

1. If `L` has 0 or 1 elements, it is already sorted. Do nothing.
2. Otherwise, copy the first half of `L` into a new list `L1`, and the rest into `L2`.
3. Use recursive calls to sort `L1` and `L2` (in place).
4. Use `merge_sorted_lists` to merge `L1` and `L2` into `L`.

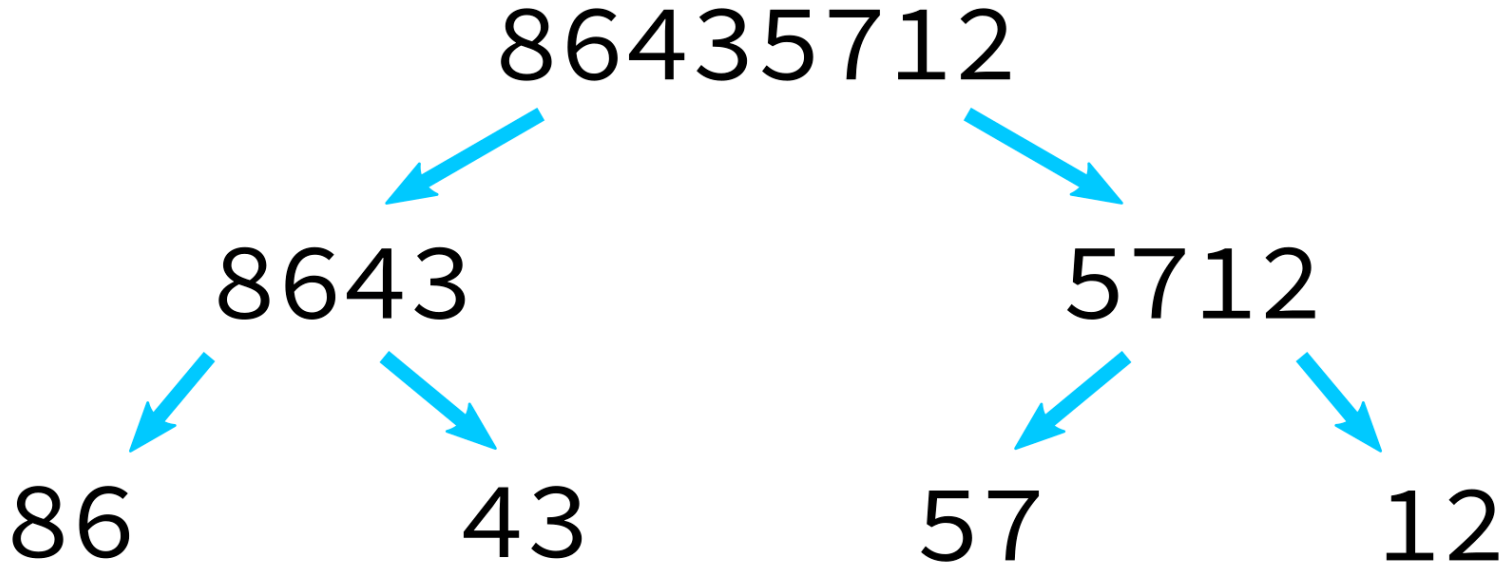
MERGESORT EXAMPLE

86435712

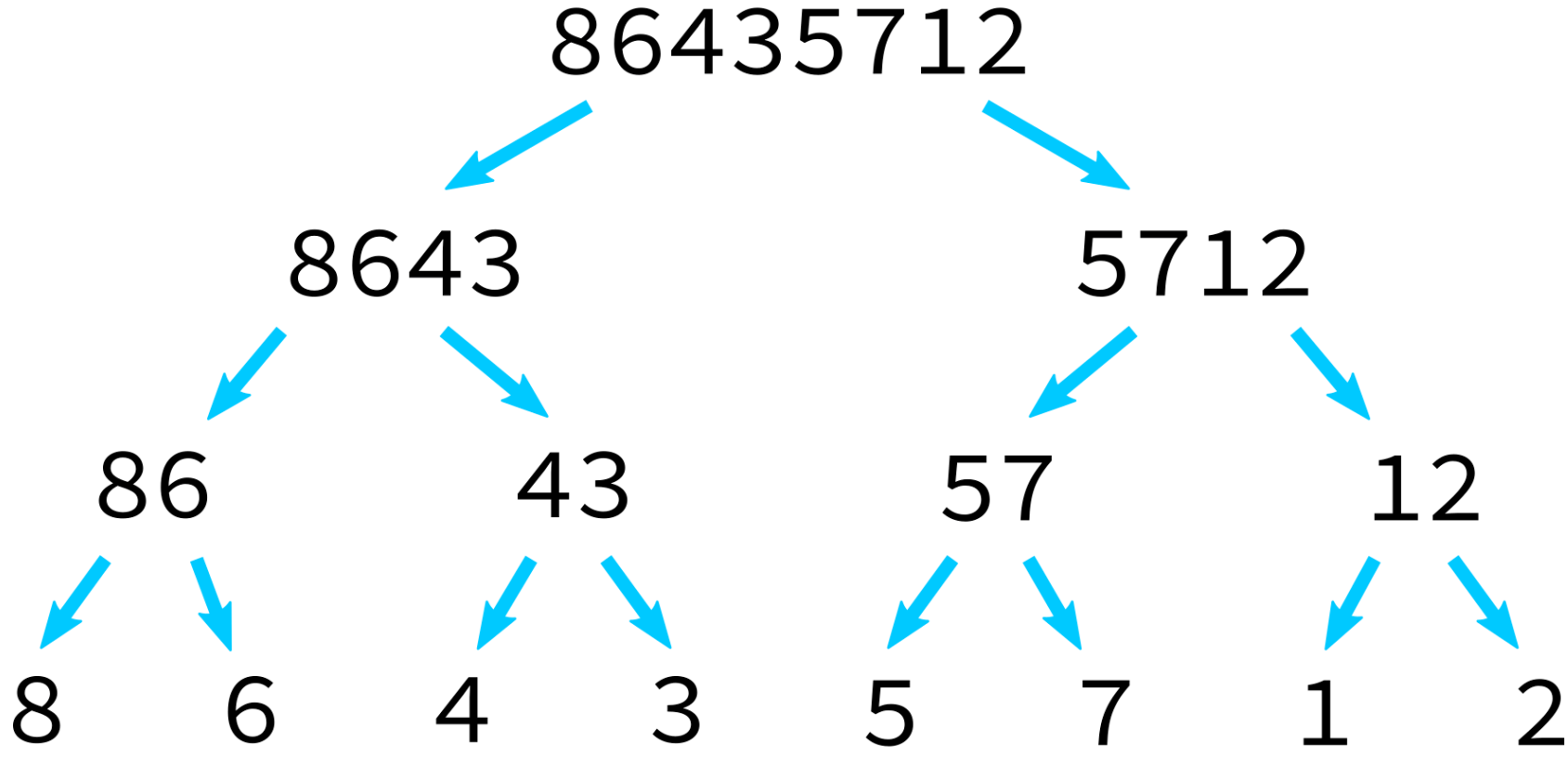
MERGESORT EXAMPLE



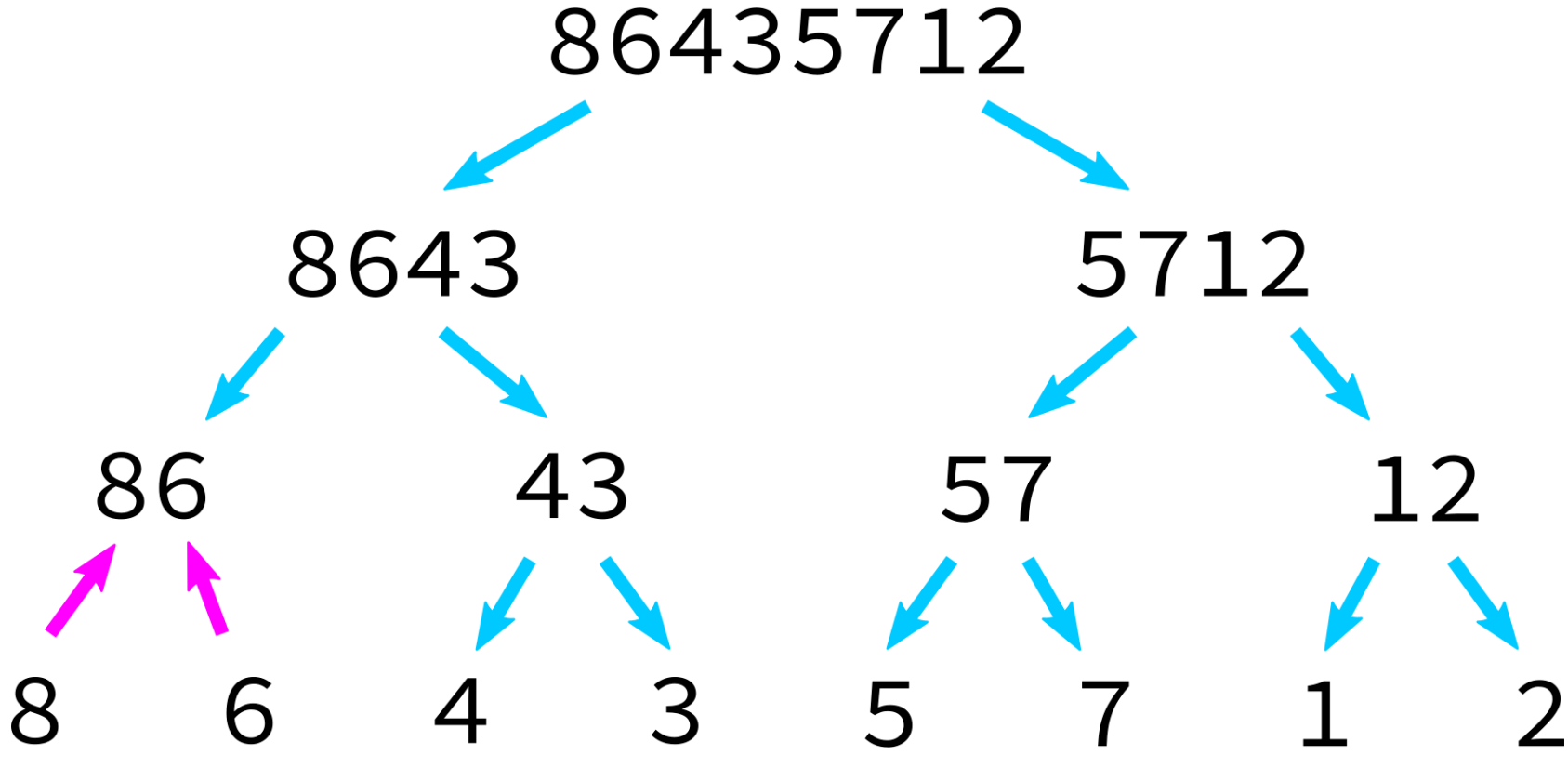
MERGESORT EXAMPLE



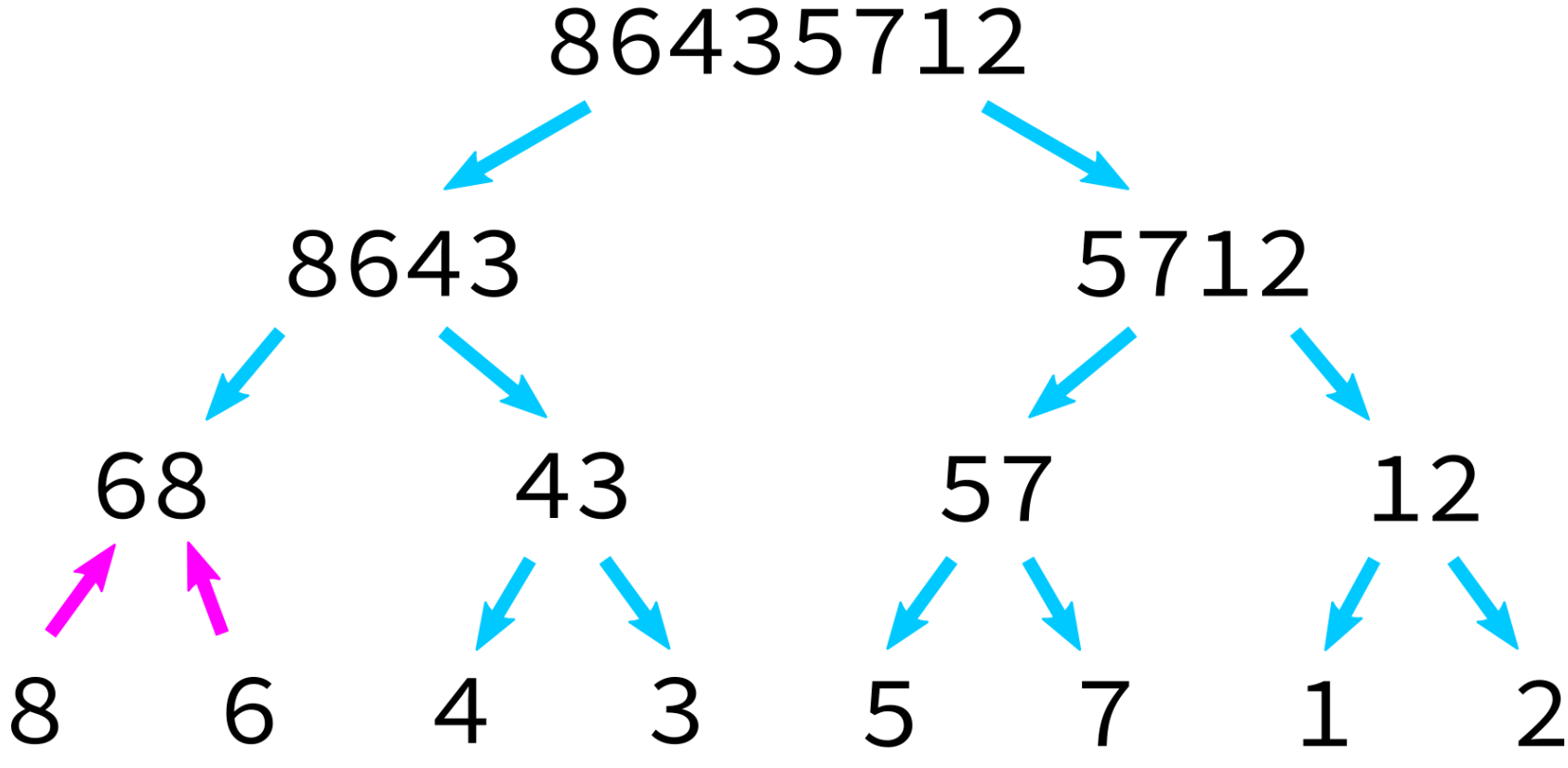
MERGESORT EXAMPLE



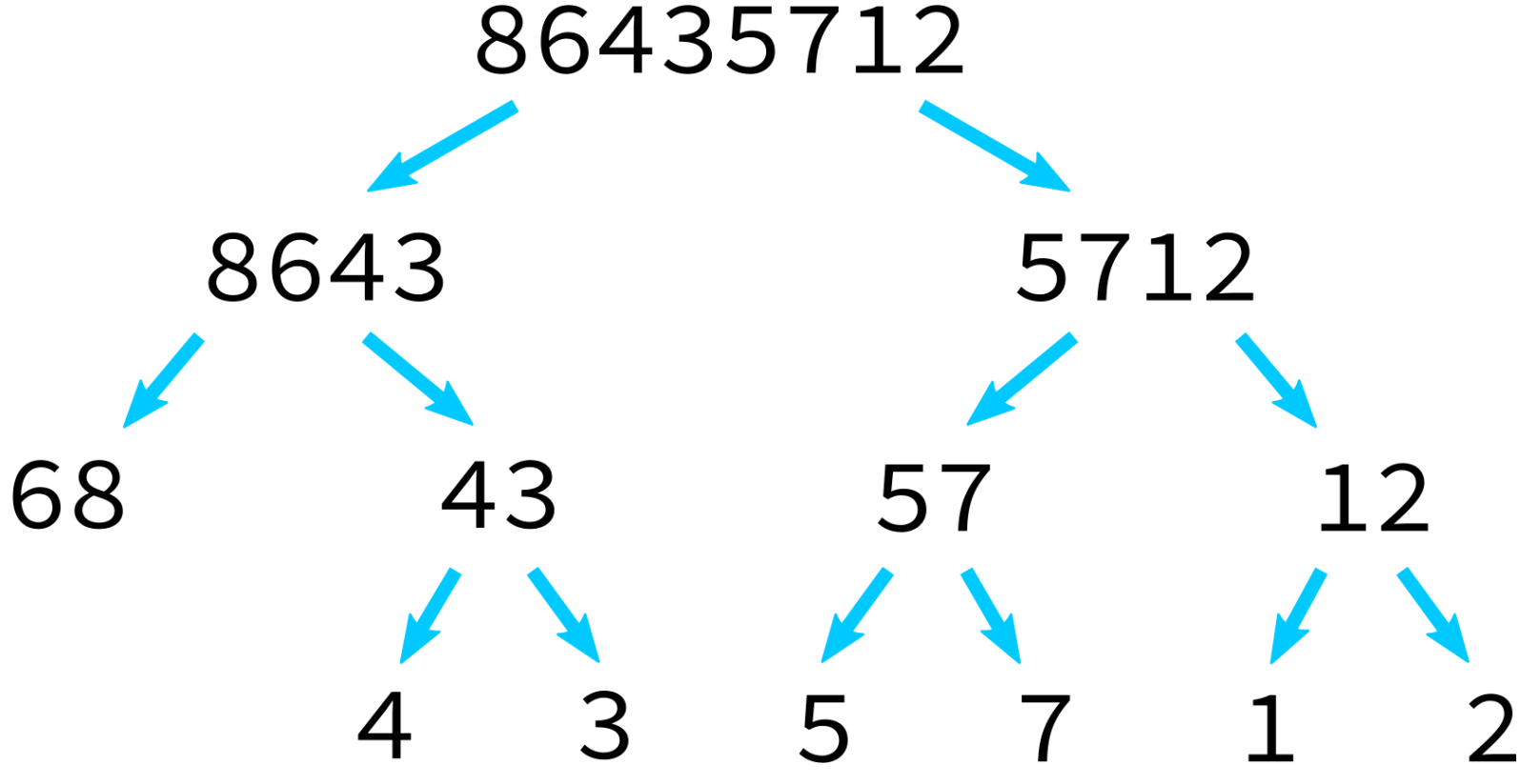
MERGESORT EXAMPLE



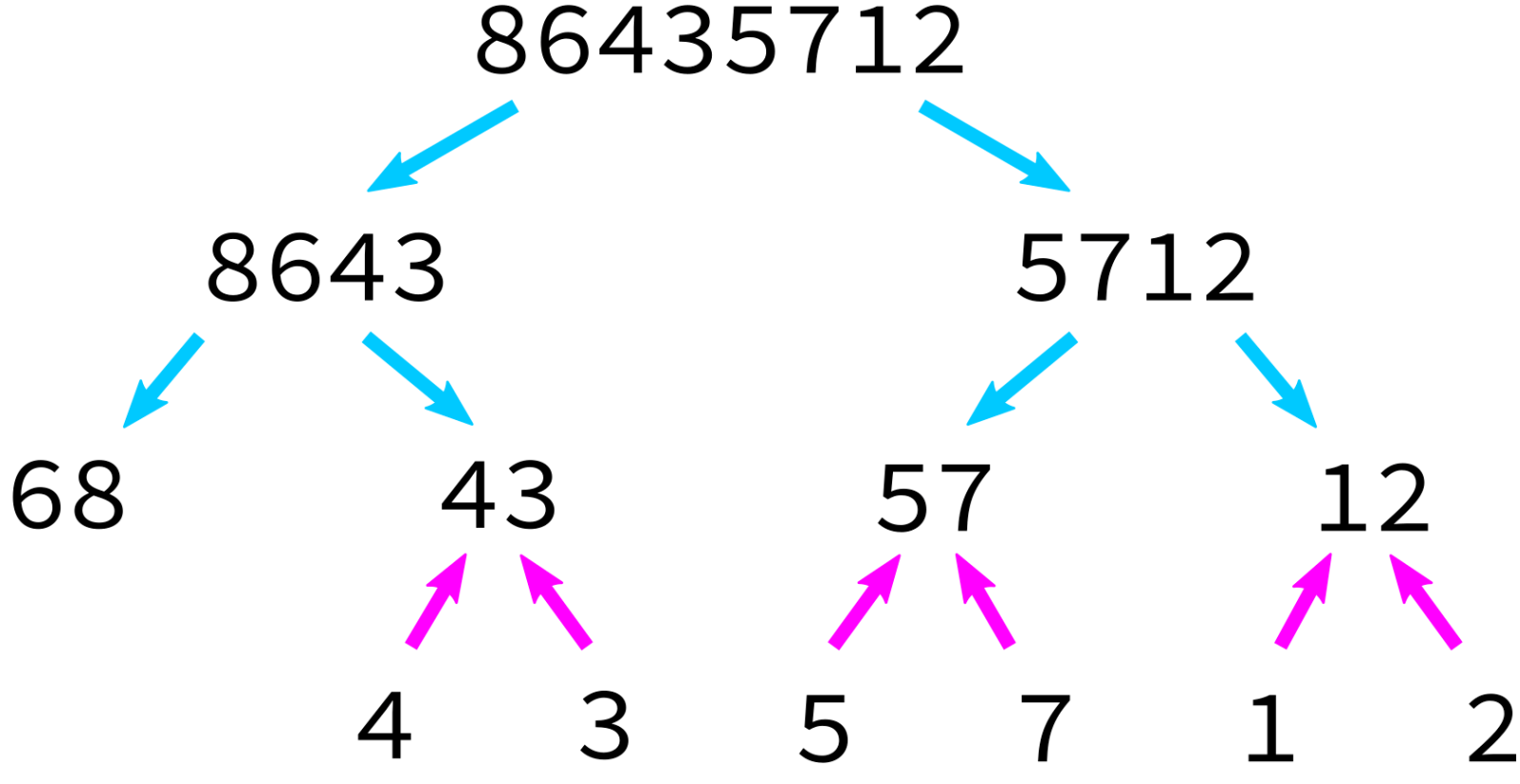
MERGESORT EXAMPLE



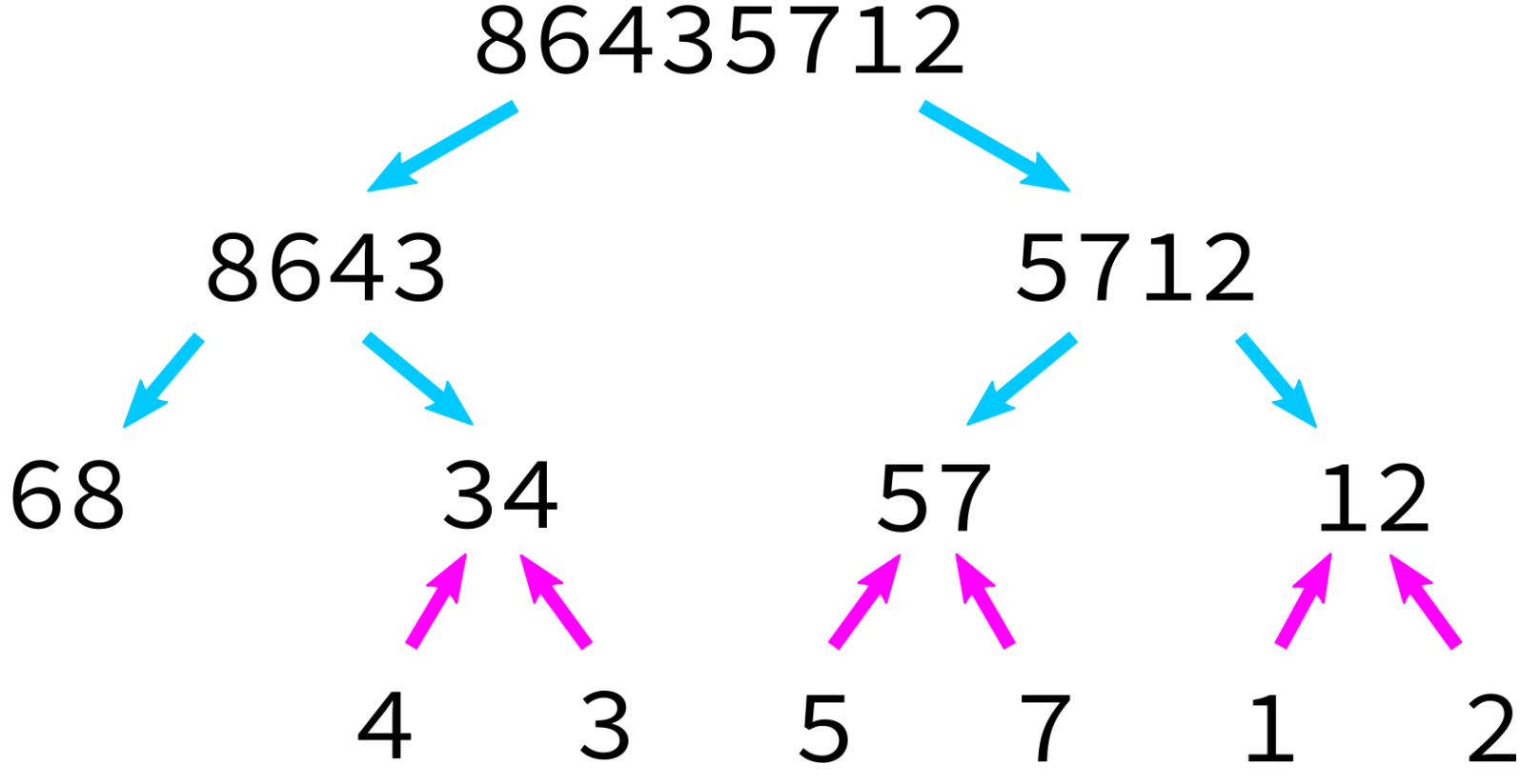
MERGESORT EXAMPLE



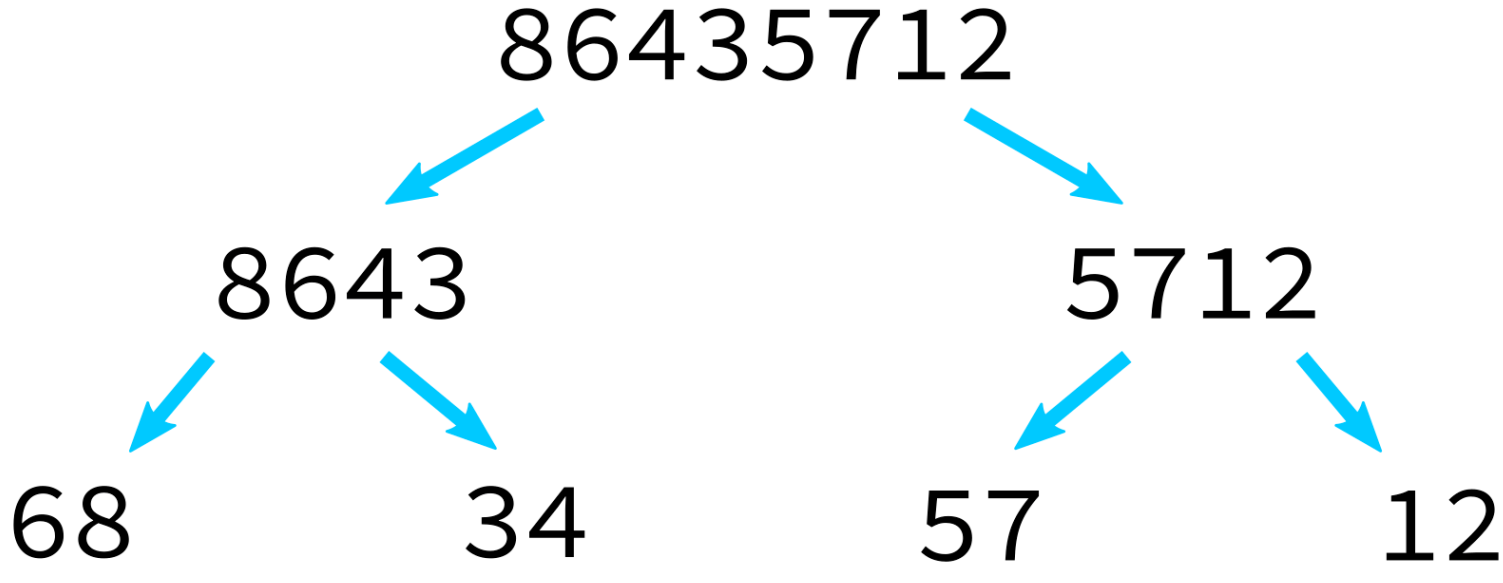
MERGESORT EXAMPLE



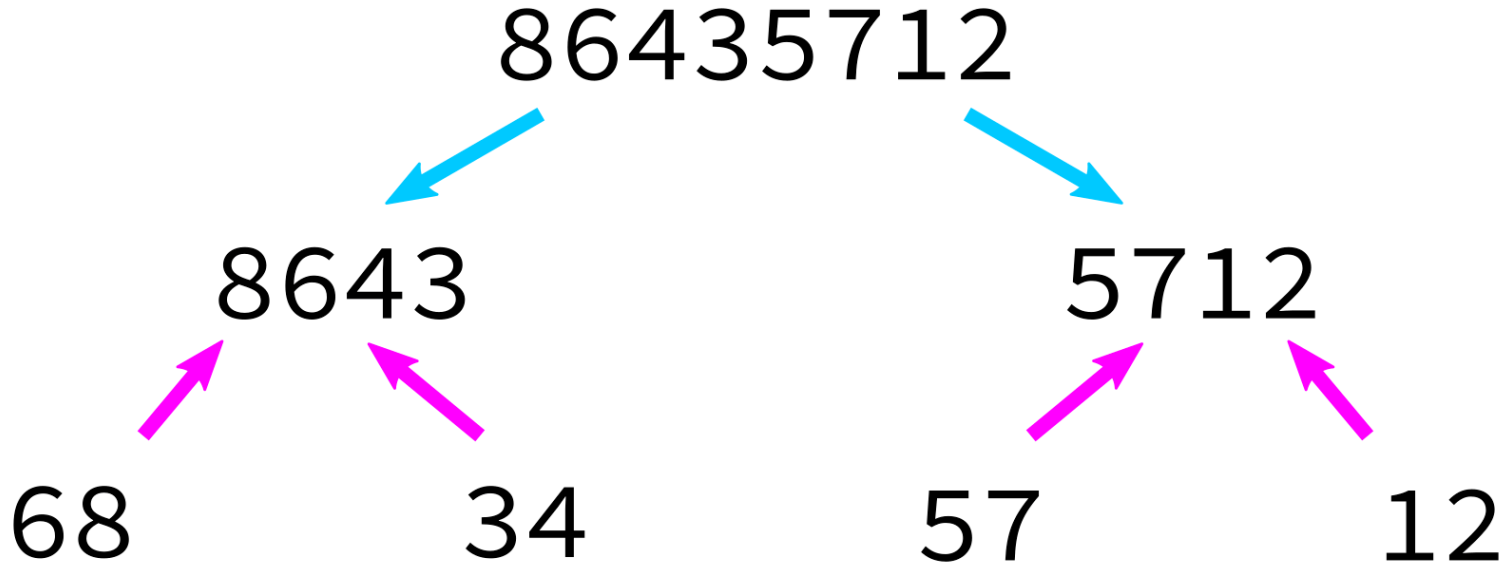
MERGESORT EXAMPLE



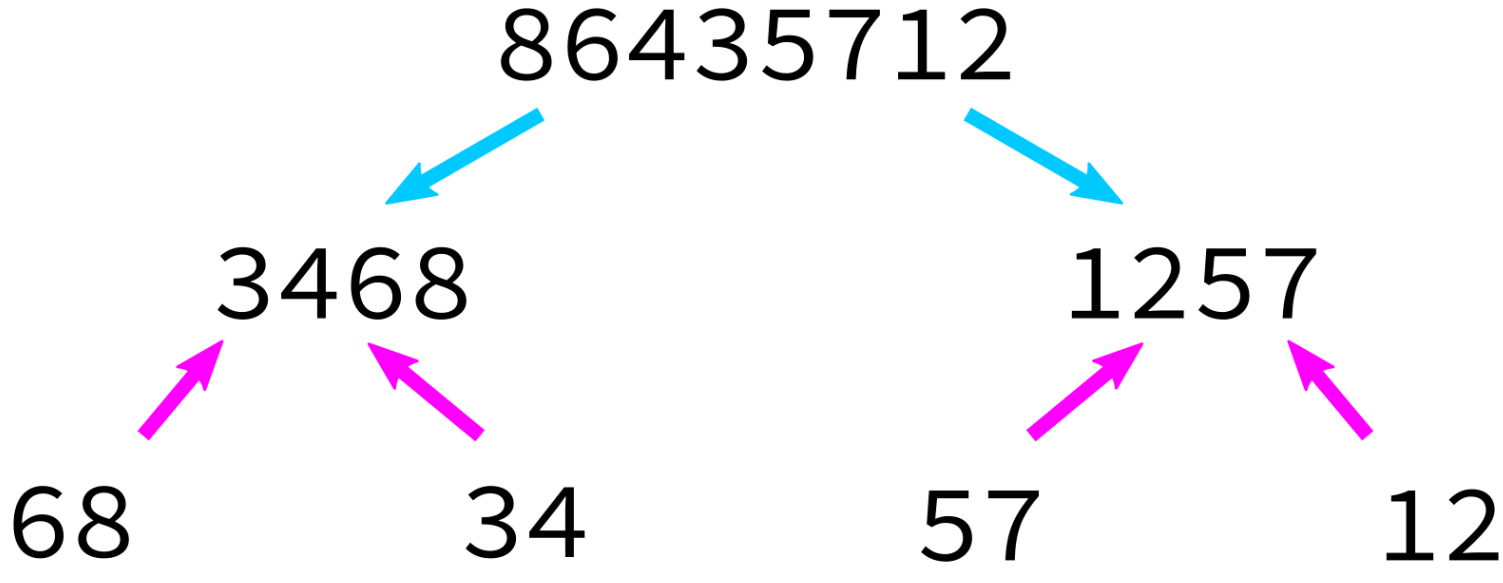
MERGESORT EXAMPLE



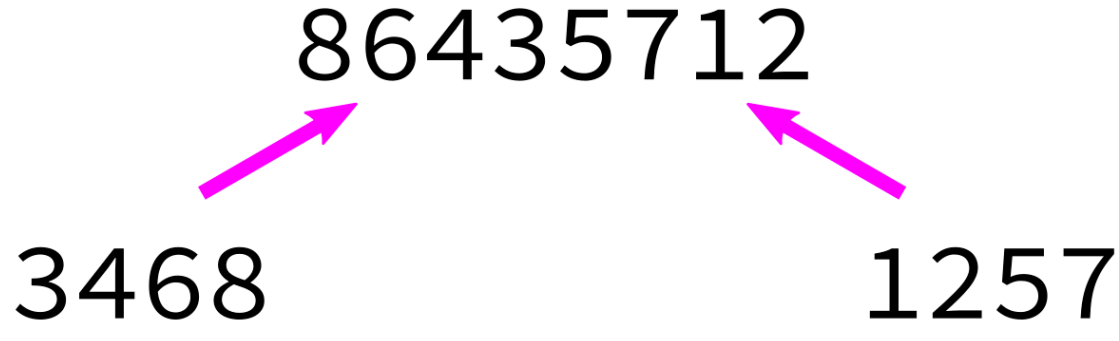
MERGESORT EXAMPLE



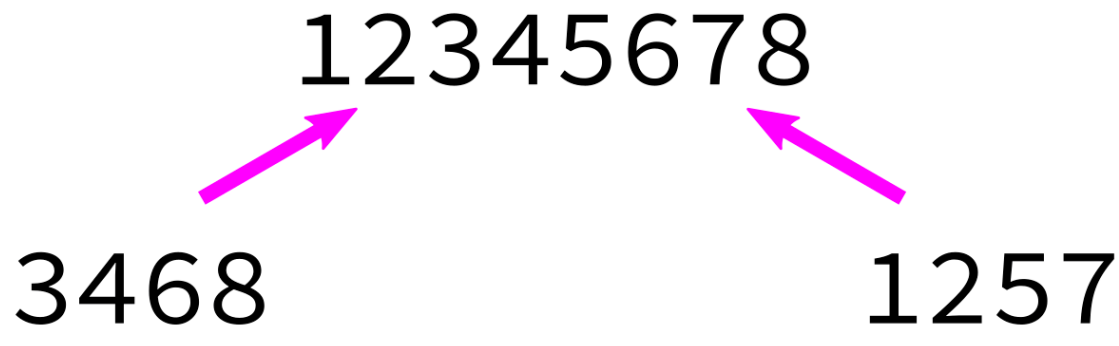
MERGESORT EXAMPLE



MERGESORT EXAMPLE



MERGESORT EXAMPLE



MERGESORT EXAMPLE

12345678

BUT HOW TO MERGE?

This algorithm depends on having a function `merge_sorted_lists` that can merge to sorted lists into a single sorted list.

Algorithm `merge_sorted_lists`:

Input: sorted lists L1 and L2, and a list `L` of the proper length in which to store the results.

Goal: copy all elements of `L1` and `L2` into `L` in increasing order.

1. Make three integer variables `i1`, `i2`, `i` to keep track of current position in `L1`, `L2`, `L` respectively. Set all to zero.
2. While `i1 < len(L1)` and `i2 < len(L2)`, do the following:
   - Check which of `L1[i1]` and `L2[i2]` is smaller.
   - Store the smaller one in `L[i]`.
   - Increment whichever one of `i1`, `i2` was used.
   - Increment `i`
3. Copy any remaining portion of `L1` into `L`.
4. Copy any remaining portion of `L2` into `L`.

MERGING SORTED LISTS

86435712

MERGING SORTED LISTS

86435712

8643 5712

MERGING SORTED LISTS

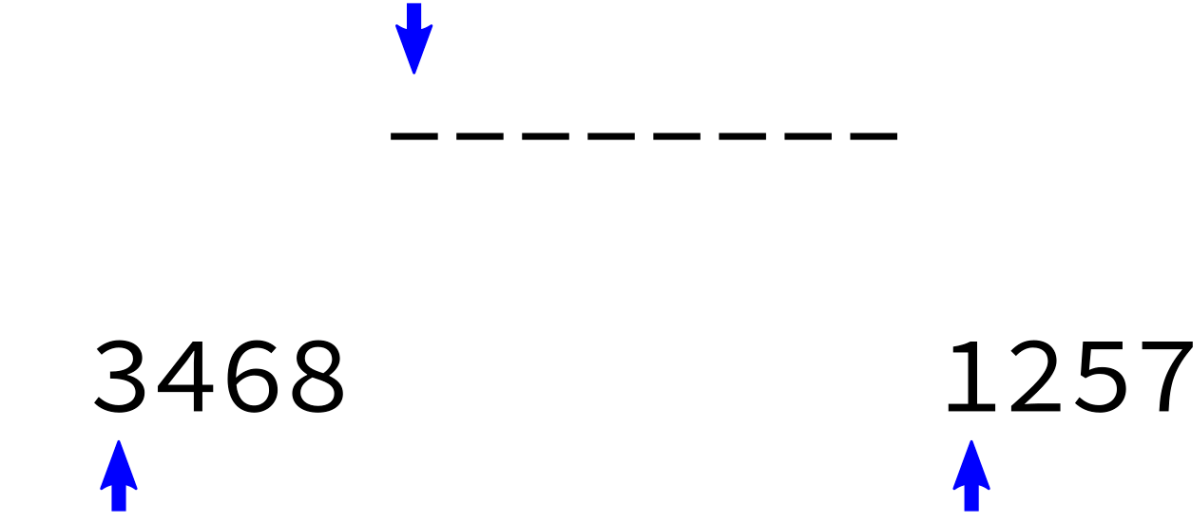
86435712

3468 1257

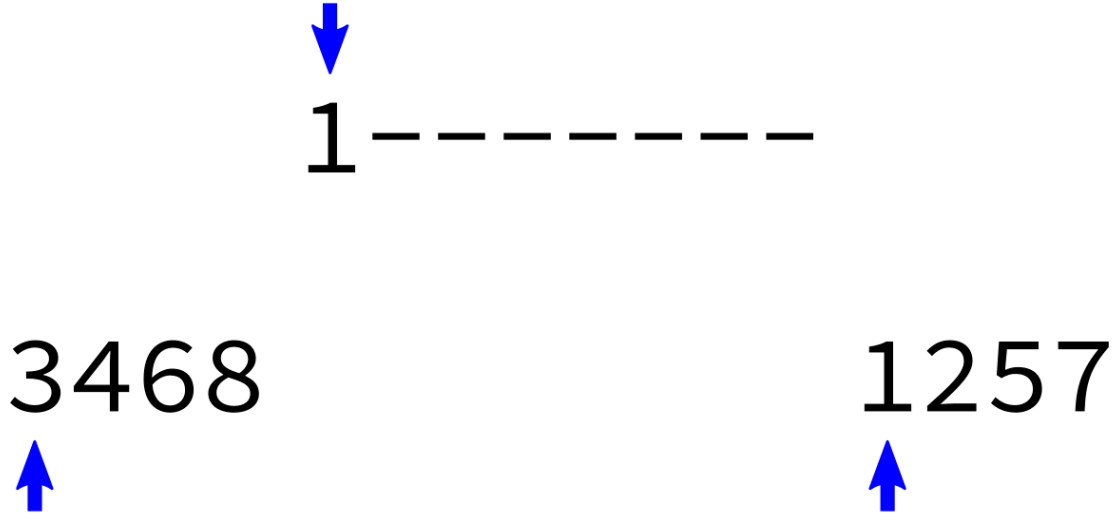
MERGING SORTED LISTS

3468 1257

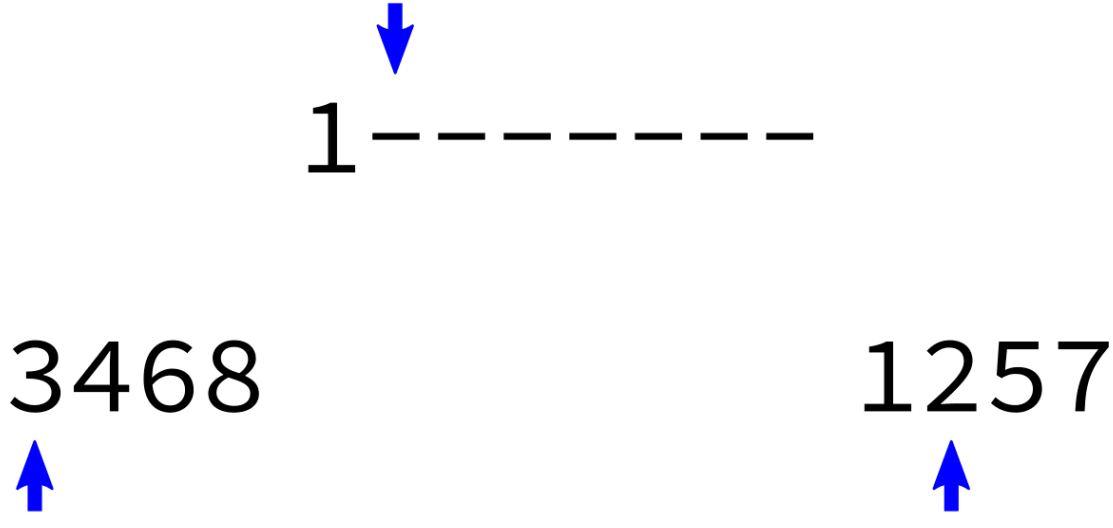
MERGING SORTED LISTS



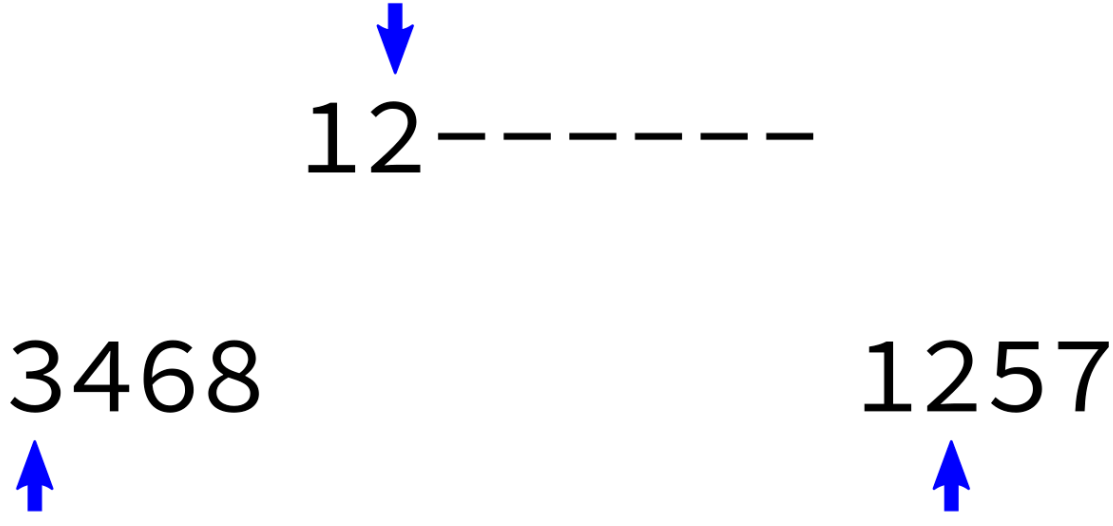
MERGING SORTED LISTS



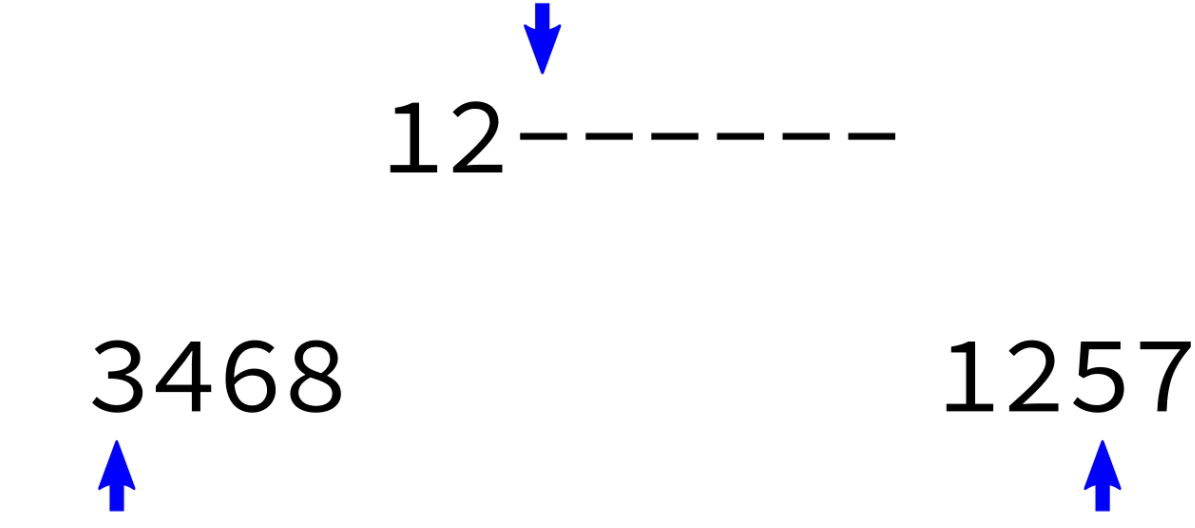
MERGING SORTED LISTS



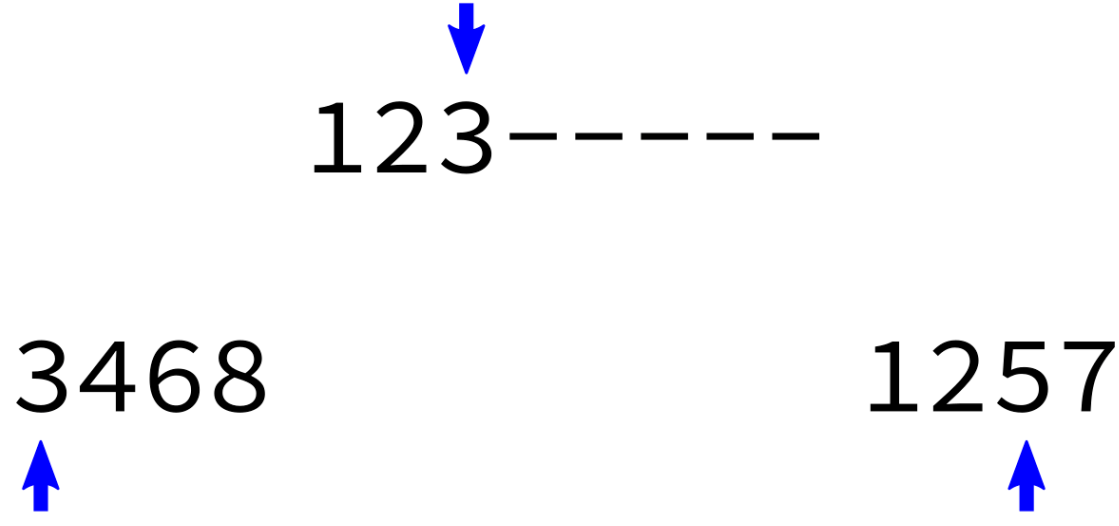
MERGING SORTED LISTS



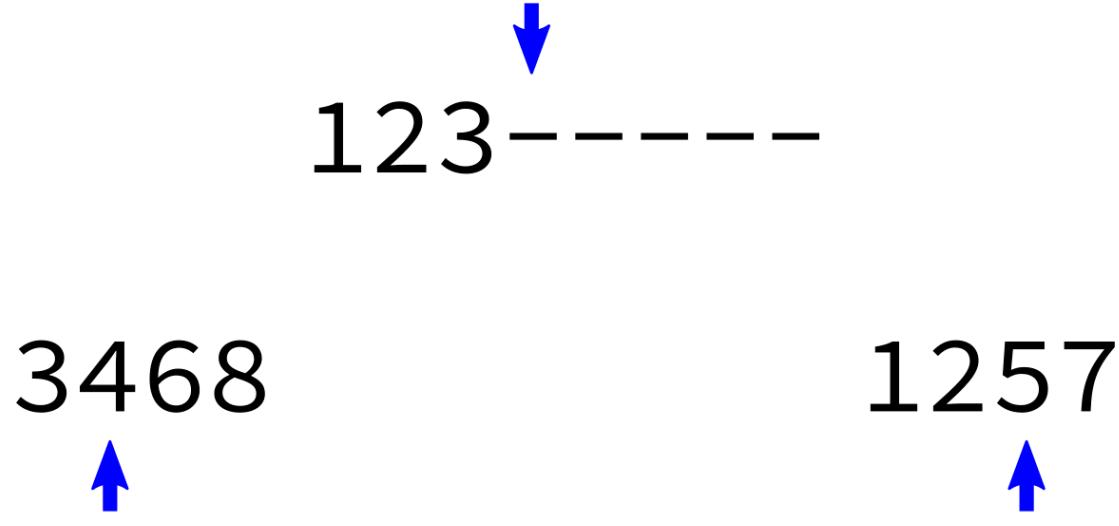
MERGING SORTED LISTS



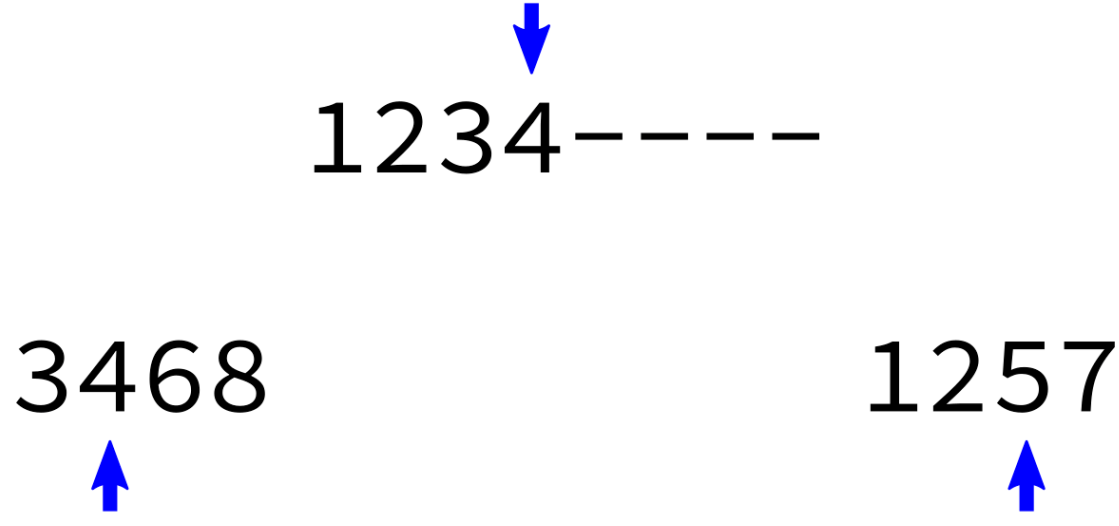
MERGING SORTED LISTS



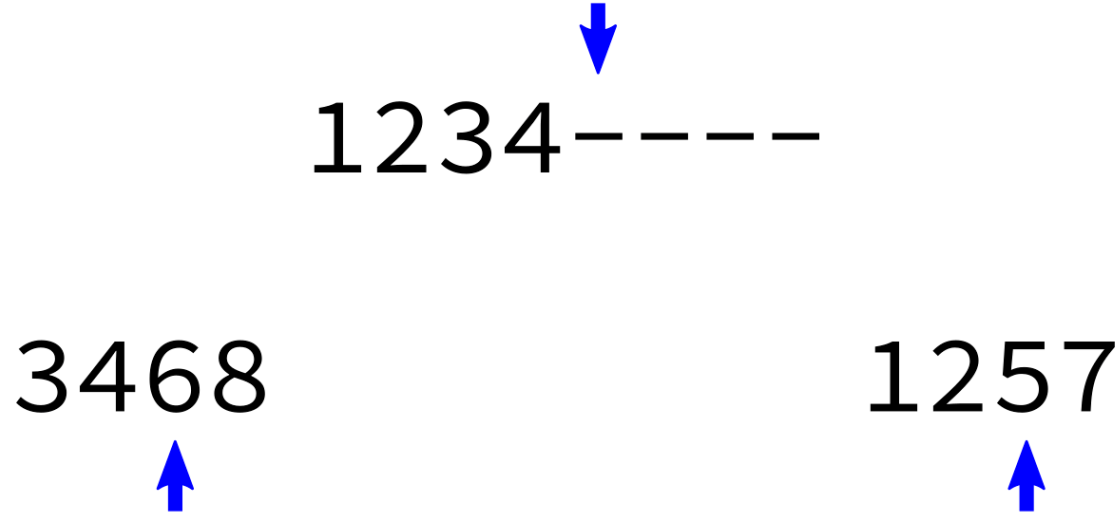
MERGING SORTED LISTS



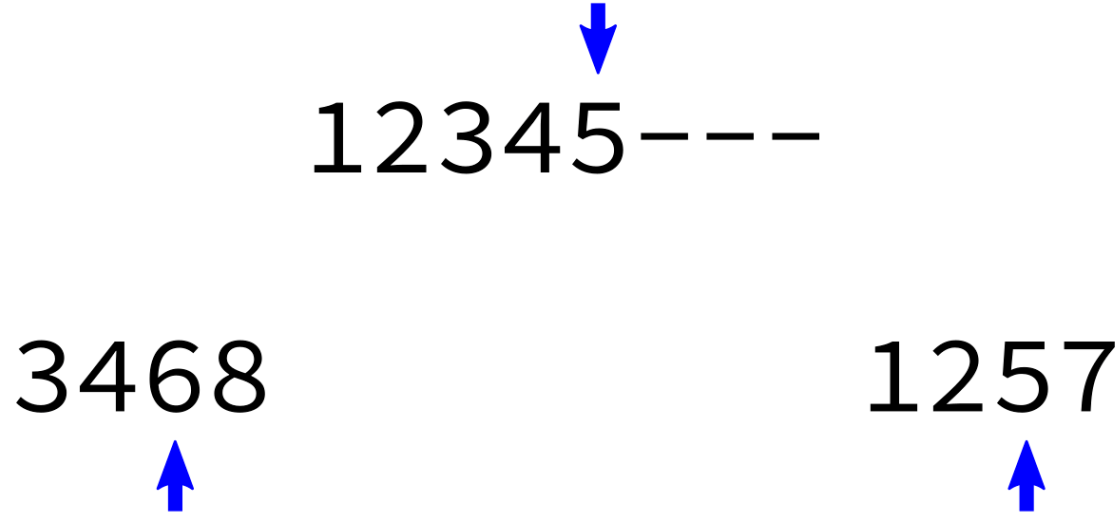
MERGING SORTED LISTS



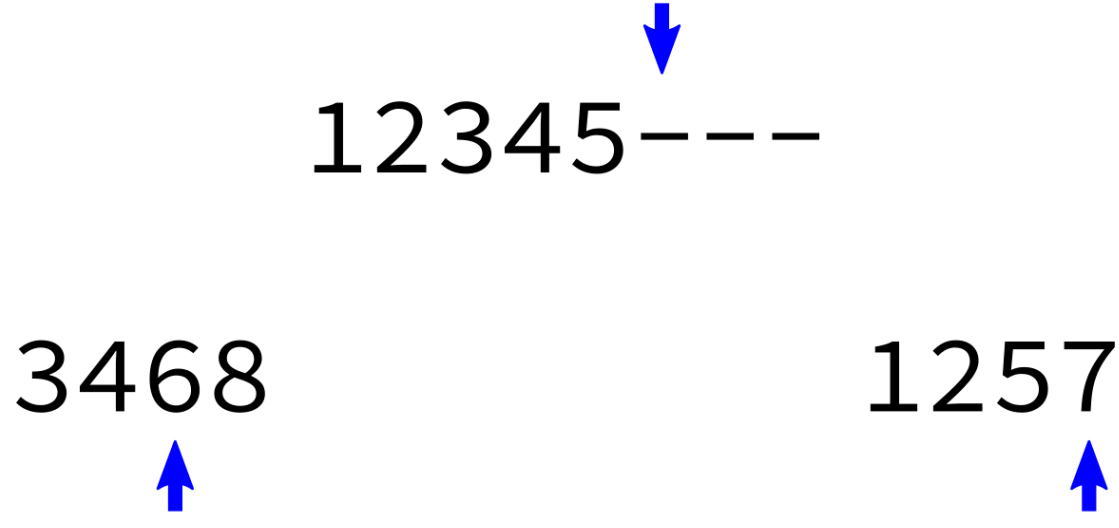
MERGING SORTED LISTS



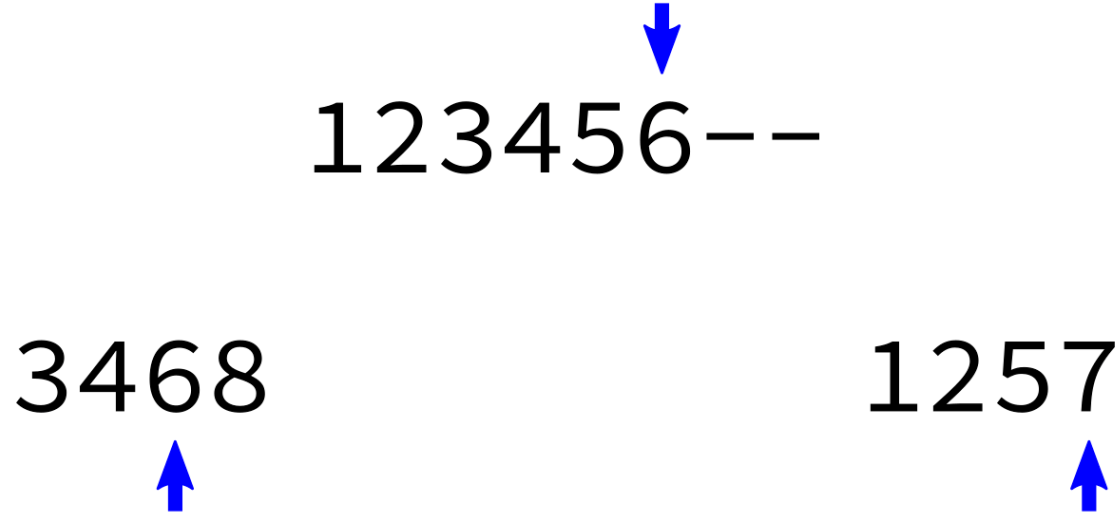
MERGING SORTED LISTS



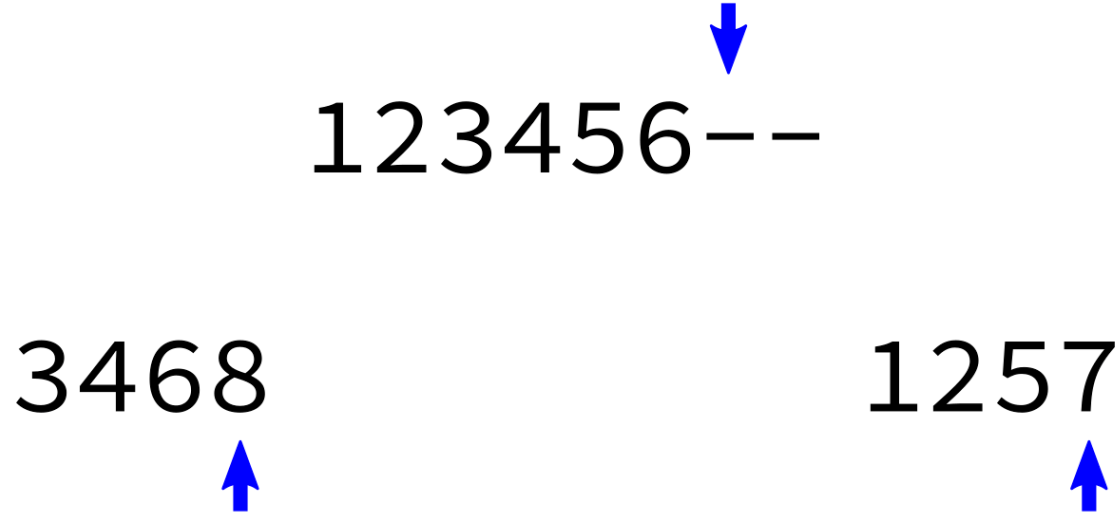
MERGING SORTED LISTS



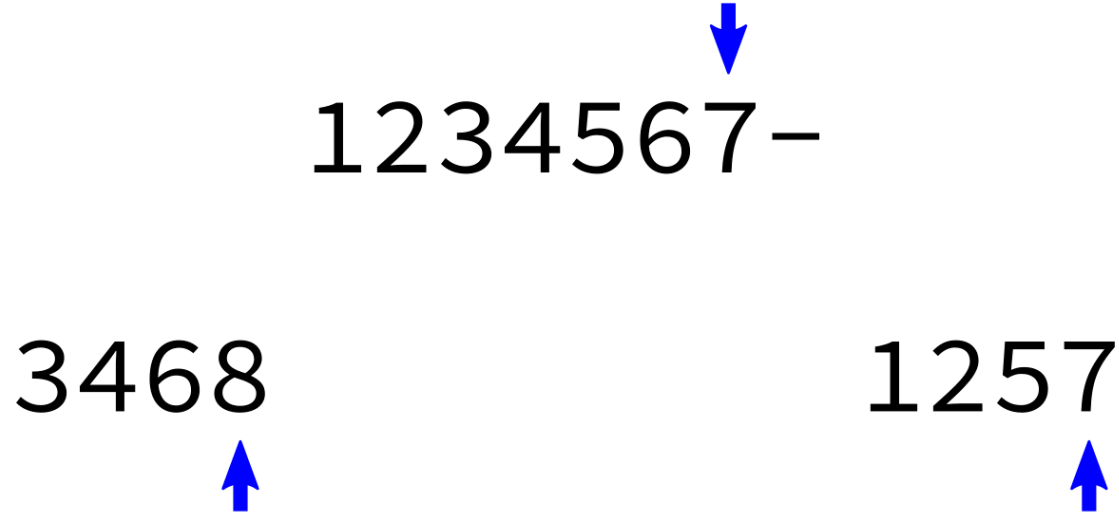
MERGING SORTED LISTS



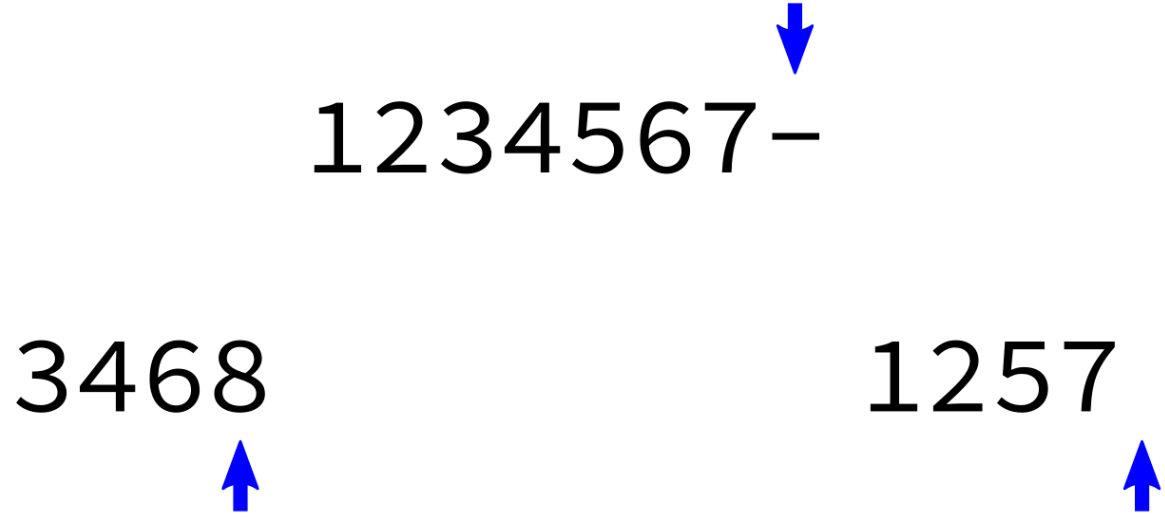
MERGING SORTED LISTS



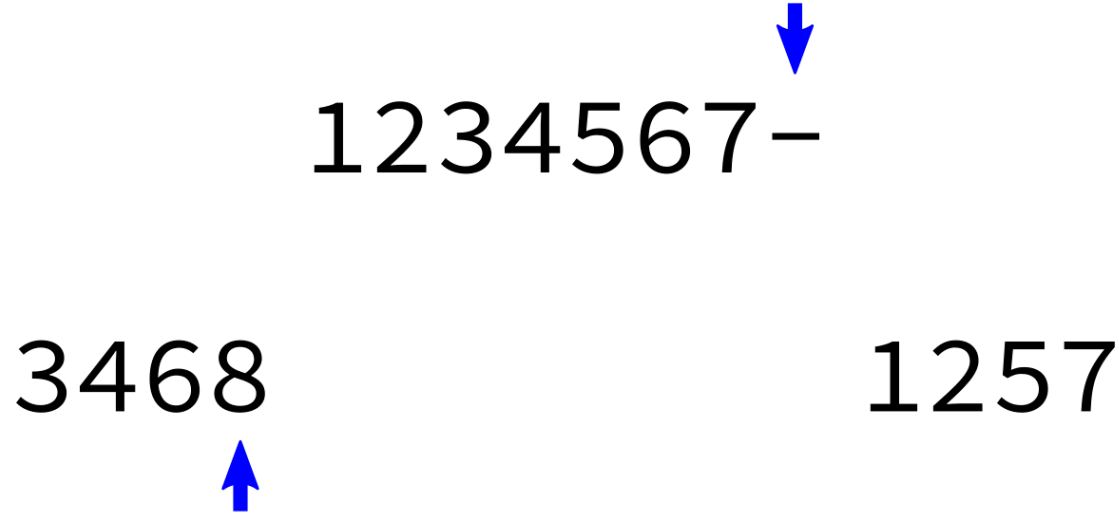
MERGING SORTED LISTS



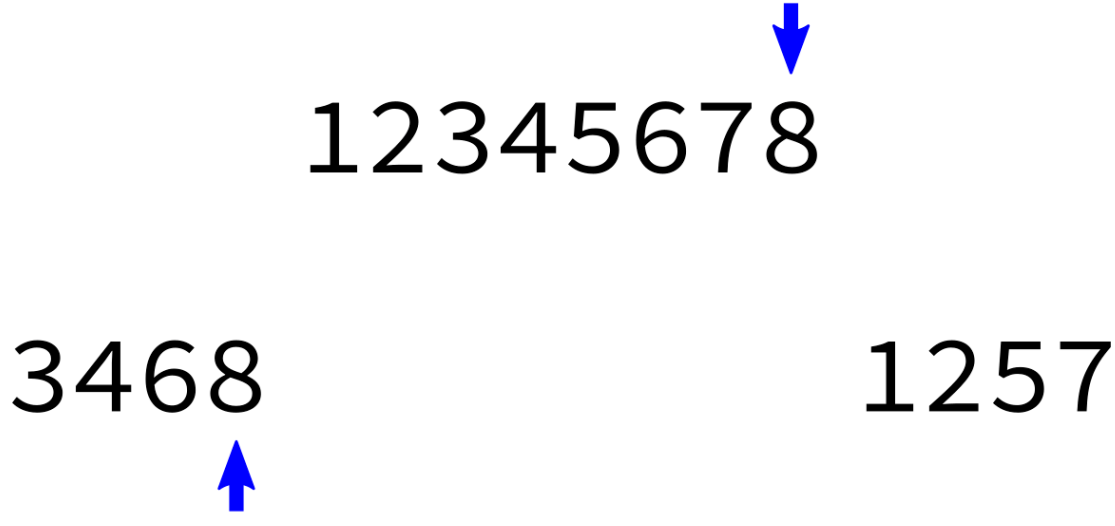
MERGING SORTED LISTS



MERGING SORTED LISTS



MERGING SORTED LISTS



MERGING SORTED LISTS

12345678

3468 1257

MERGING SORTED LISTS

12345678

CODING TIME

Let's implement `mergesort` in Python.

REFERENCES

No changes to the references from Lecture 13

- Algorithms by Jeff Erickson, Chapter 1. Mergesort is example 1.4.
- Lutz discusses recursive functions in Chapter 19 (pages 555-559 in the print edition).
- Intro to Python for Computer Science and Data Science by Deitel and Deitel, Chapter 11.
- Think Python, 2ed, by Allen B. Downey, Sections 5.8 to 5.10.
- Computer Science: An Overview by Brookshear and Brylow, Section 5.5.

REVISION HISTORY

- 2021-02-18 Move unused slides to Lecture 17
- 2021-02-17 Initial publication