CS134 Lecture 33:
Sorting Wrap Up and Java

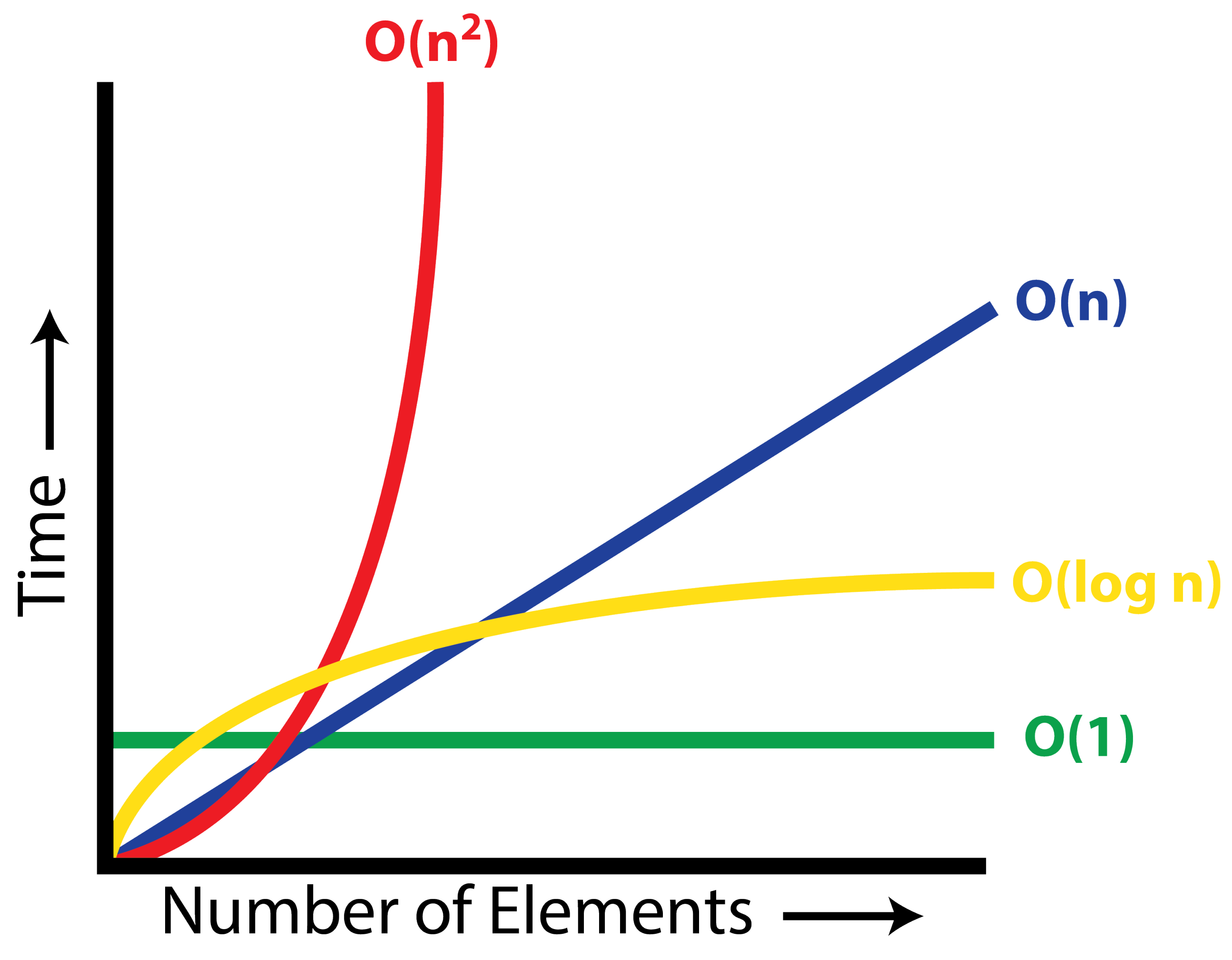
Announcements & Logistics

- **HW 10 released**, due tonight @ 10 pm
- Lab 8 graded feedback returned
- **Lab 10** released
  - Handout will be posted soon: server issues
  - Very short lab on searching and sorting (today's lecture)
  - No prelab
  - Individual lab but can discuss strategies with lab mate
- CS134 Scheduled Final: **Friday, May 17, 9:30 AM**
  - Room: **TCL 123**

Do You Have Any Questions?

Last Time: Efficiency & Searching

- Discussed recursive code for binary search
- Discussed selection sort algorithm
    - get_min_index helper function: debug in Lab 10
- Analyzed selection sort
    - $O(n^2)$



This Week

- Today we will discuss an improved (optimal) sorting algorithm
    - ***Merge sort***
- Example of recursion: a divide-and-conquer sorting algorithm
- Two more lectures:
    - Comparison of Python vs Java
    - OOP Wrap up and review

More Efficient Sorting:

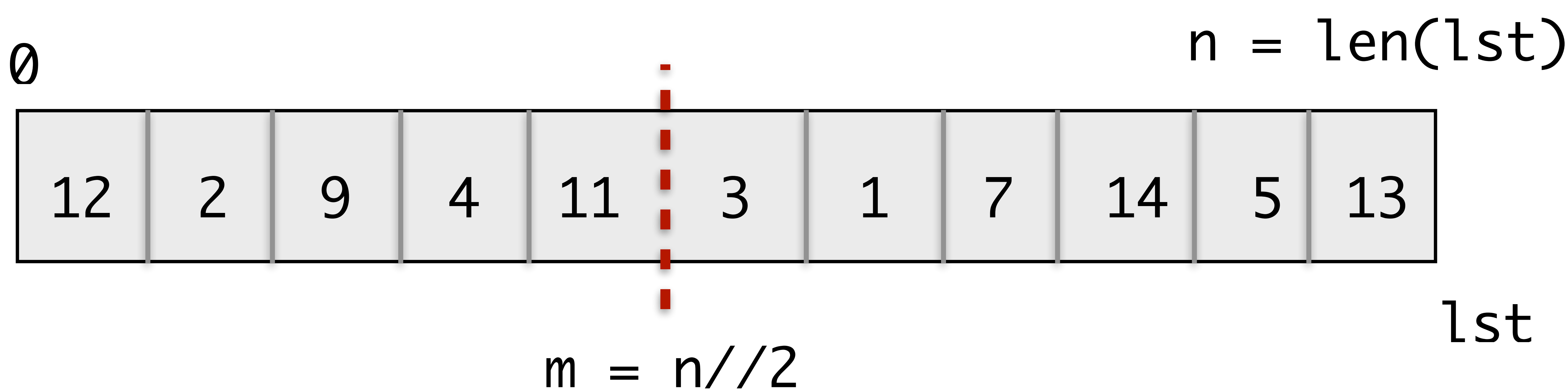
Merge Sort

Towards an $O(n \log n)$ Algorithm

- There are other sorting algorithms that compare and rearrange elements in different ways, but are still $O(n^2)$ steps
    - Any algorithm that takes n steps to move each item n positions (in the worst case) will take at least $O(n^2)$ steps
    - To do better than n^2, we need to move an item in fewer than n steps
- We can sort in $O(n \log n)$ time if we are clever: ***Merge sort algorithm*** (Invented by John von Neumann in 1945)

Merge Sort: Basic Idea

- If we split the list in half, sorting the left and right half are smaller versions of the same problem
- **Algorithm:**
  - **(Divide)** Recursively sort left and right half
  - **(Conquer)** Merge the sorted halves into a single sorted list

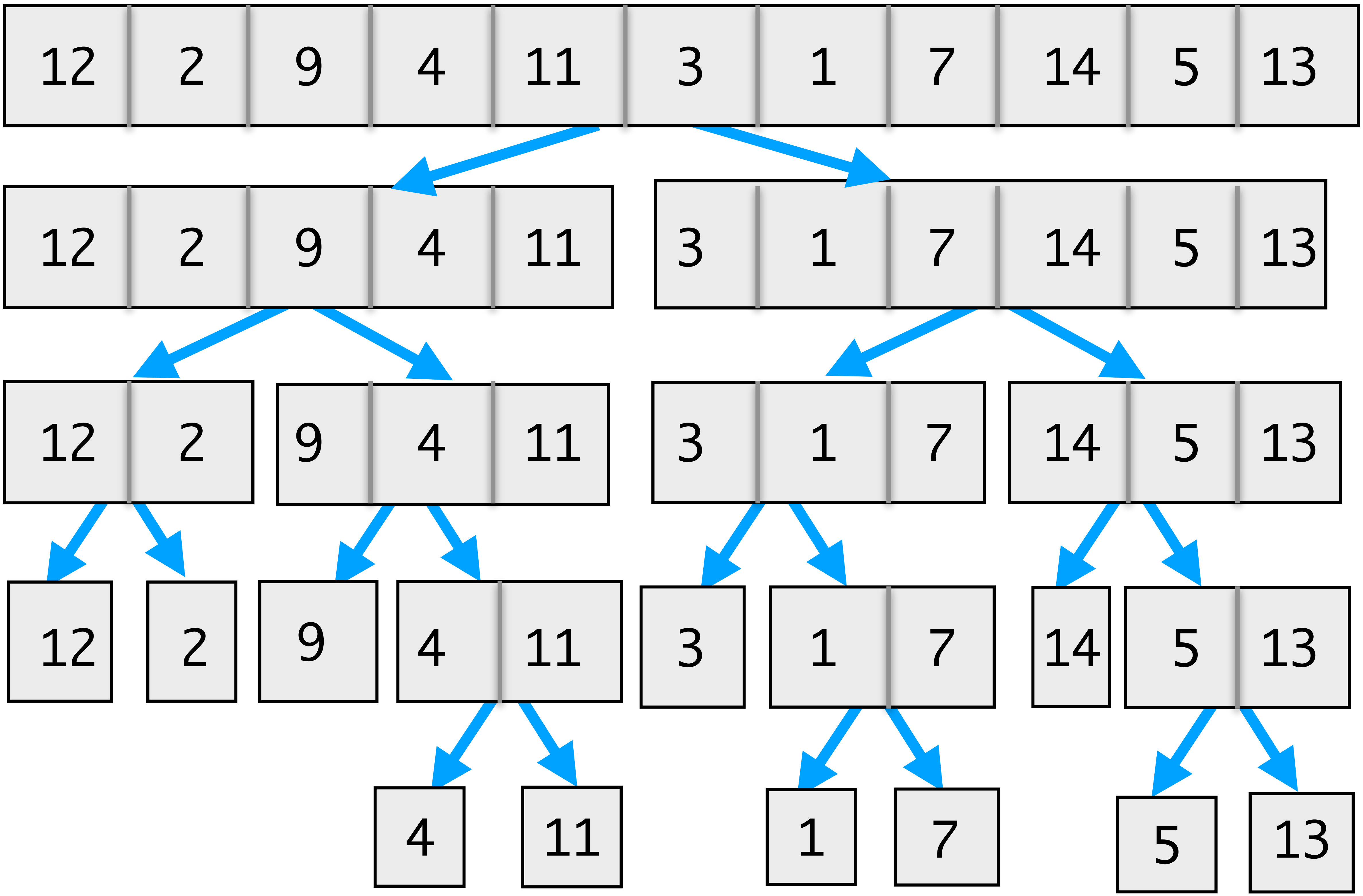


Merge Sort Algorithm

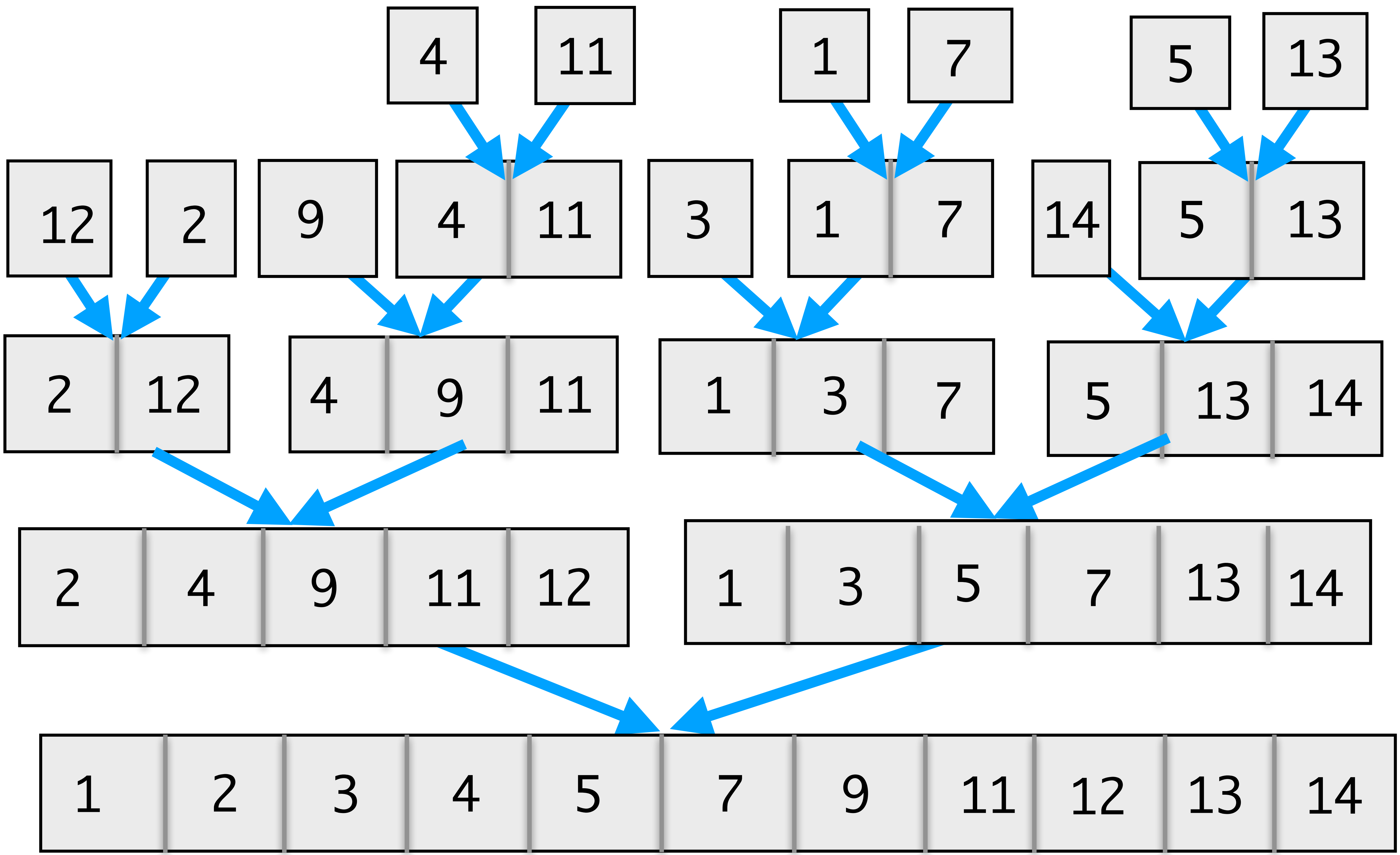
- **Base case:** If list is empty or contains a single element: it is already sorted
- **Recursive case:**
    - Recursively sort left and right halves
    - Merge the sorted lists into a single list and return it
- **Question:**
    - Where is the ***sorting*** actually taking place?

```python
def merge_sort(lst):
    """Given a list lst, returns
    a new list that is lst sorted
    in ascending order."""
    n = len(lst)

    # base case
    if n == 0 or n == 1:
        return lst

    else:
        m = n//2 # middle

        # recurse on left & right half
        sort_lt = merge_sort(lst[:m])
        sort_rt = merge_sort(lst[m:])

        # return merged list
        return merge(sort_lt, sort_rt)
```

Merge Sort Example

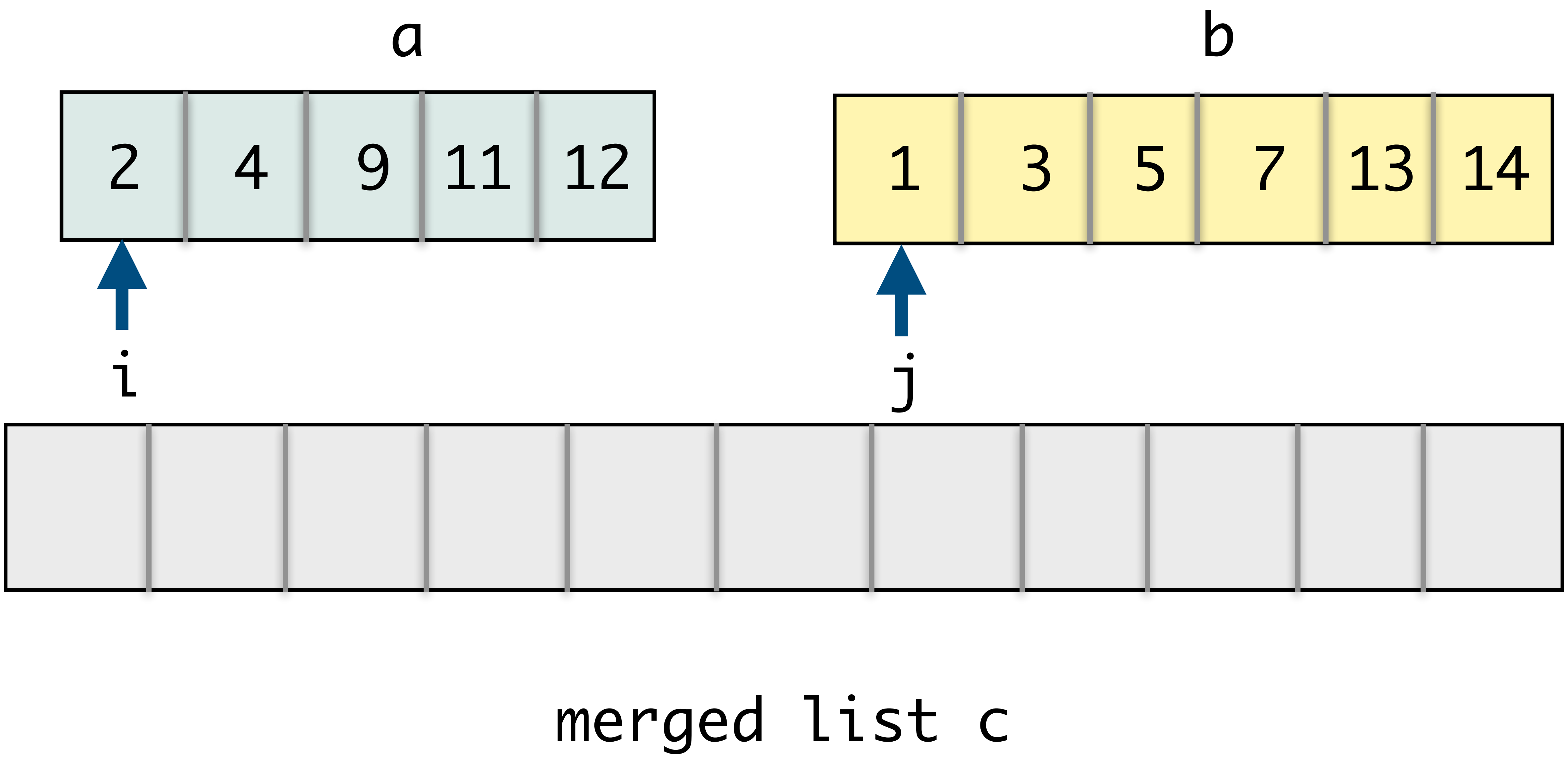


Merge Sort Example



Merging Sorted Lists

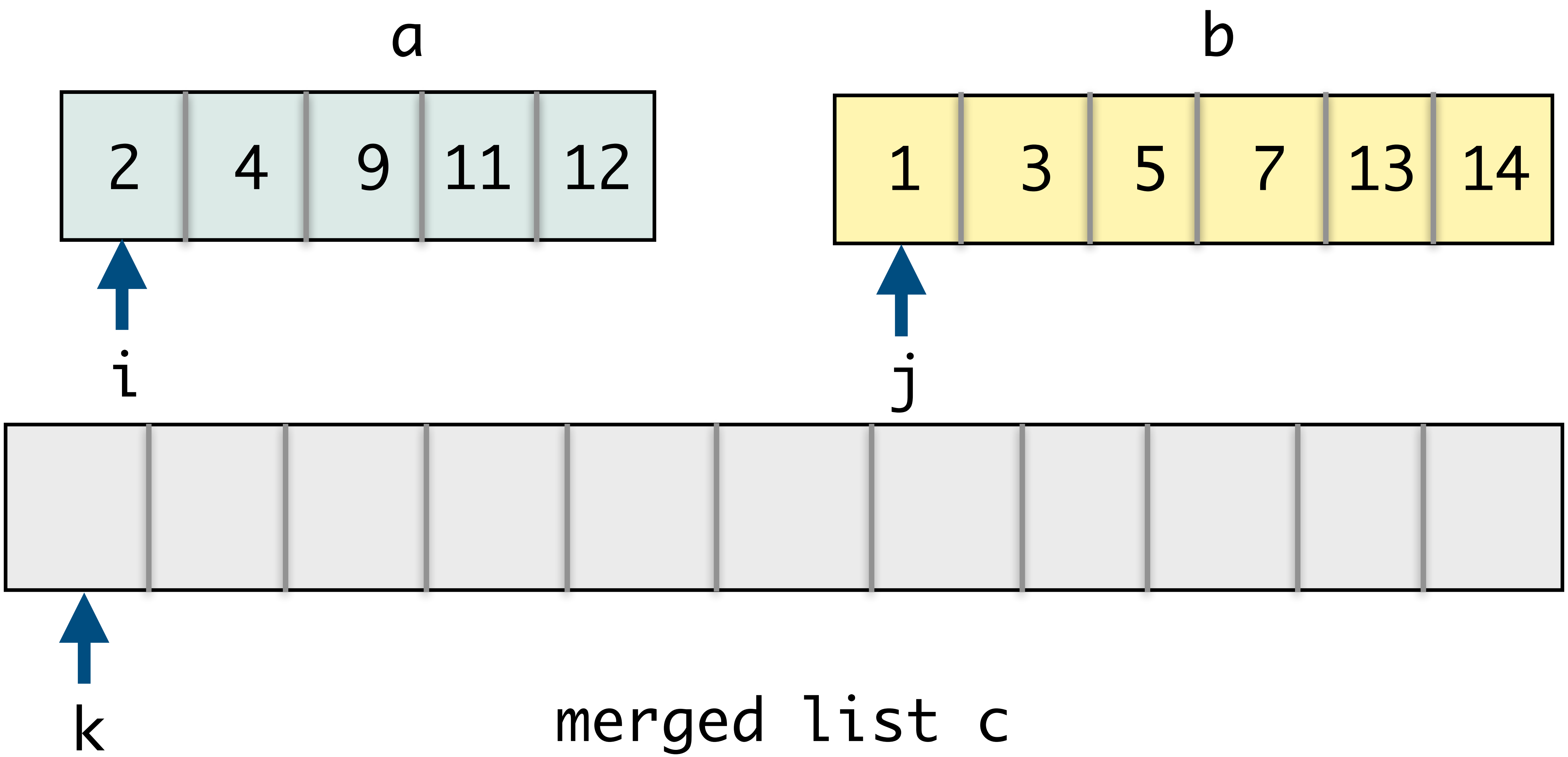
- **Problem.** Given two sorted lists `a` and `b`, how quickly can we merge them into a single sorted list?



Merging Sorted Lists

Is `a[i] <= b[j]` ?

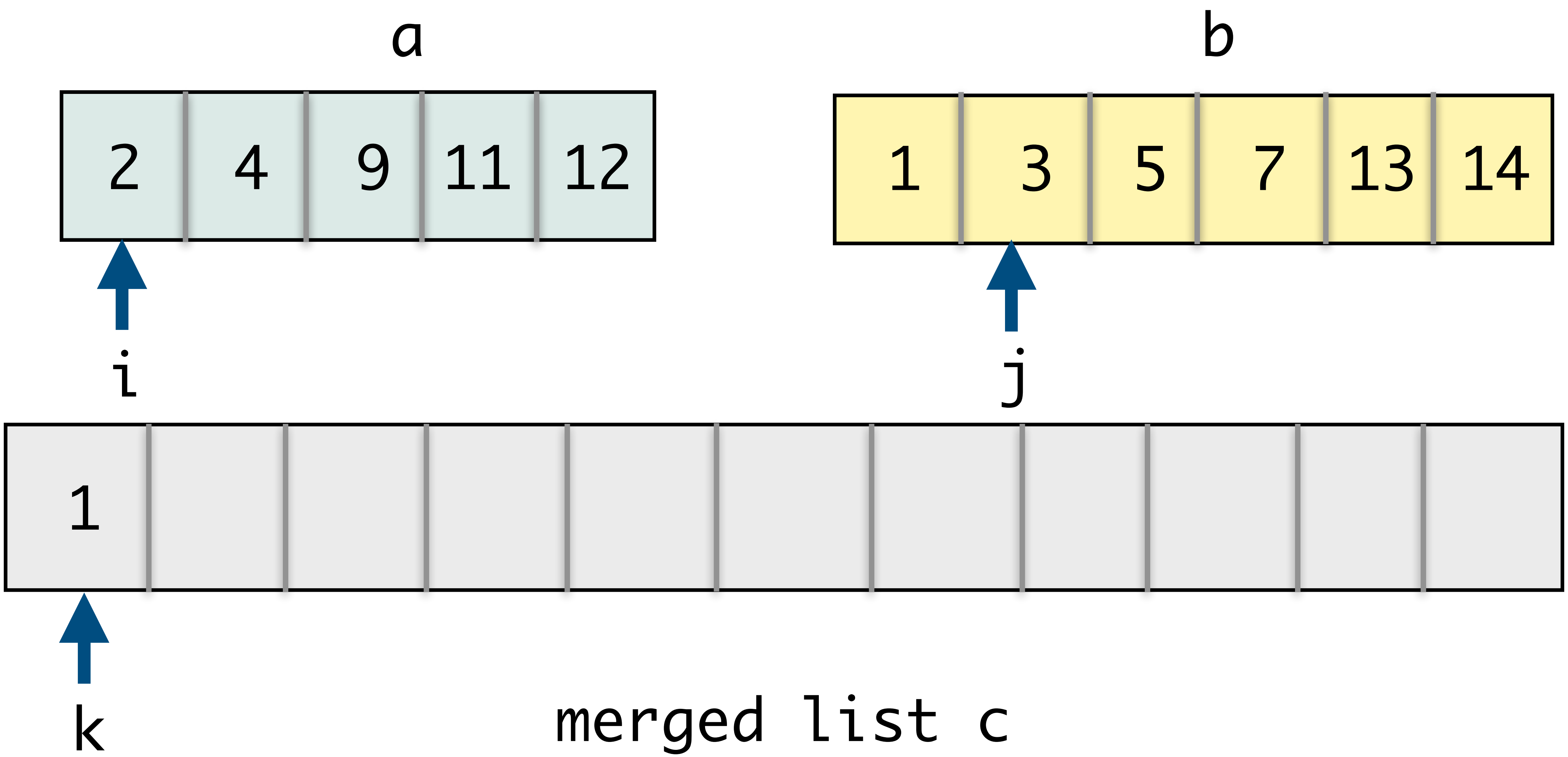
- Yes, `a[i]` appended to `c`
- No, `b[j]` appended to `c`



Merging Sorted Lists

Is `a[i] <= b[j]` ?

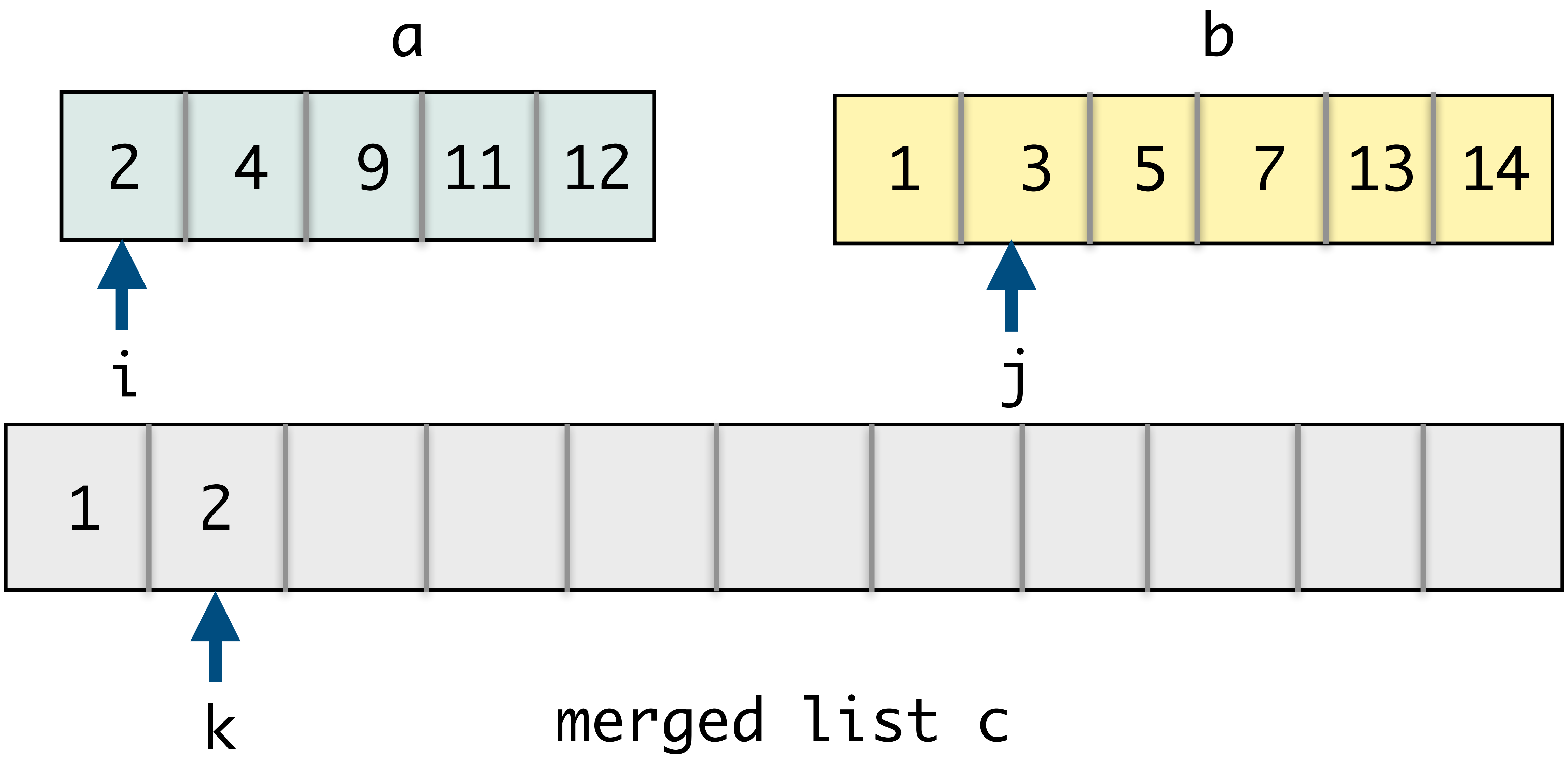
- Yes, `a[i]` appended to `c`
- No, `b[j]` appended to `c`



Merging Sorted Lists

Is `a[i] <= b[j]` ?

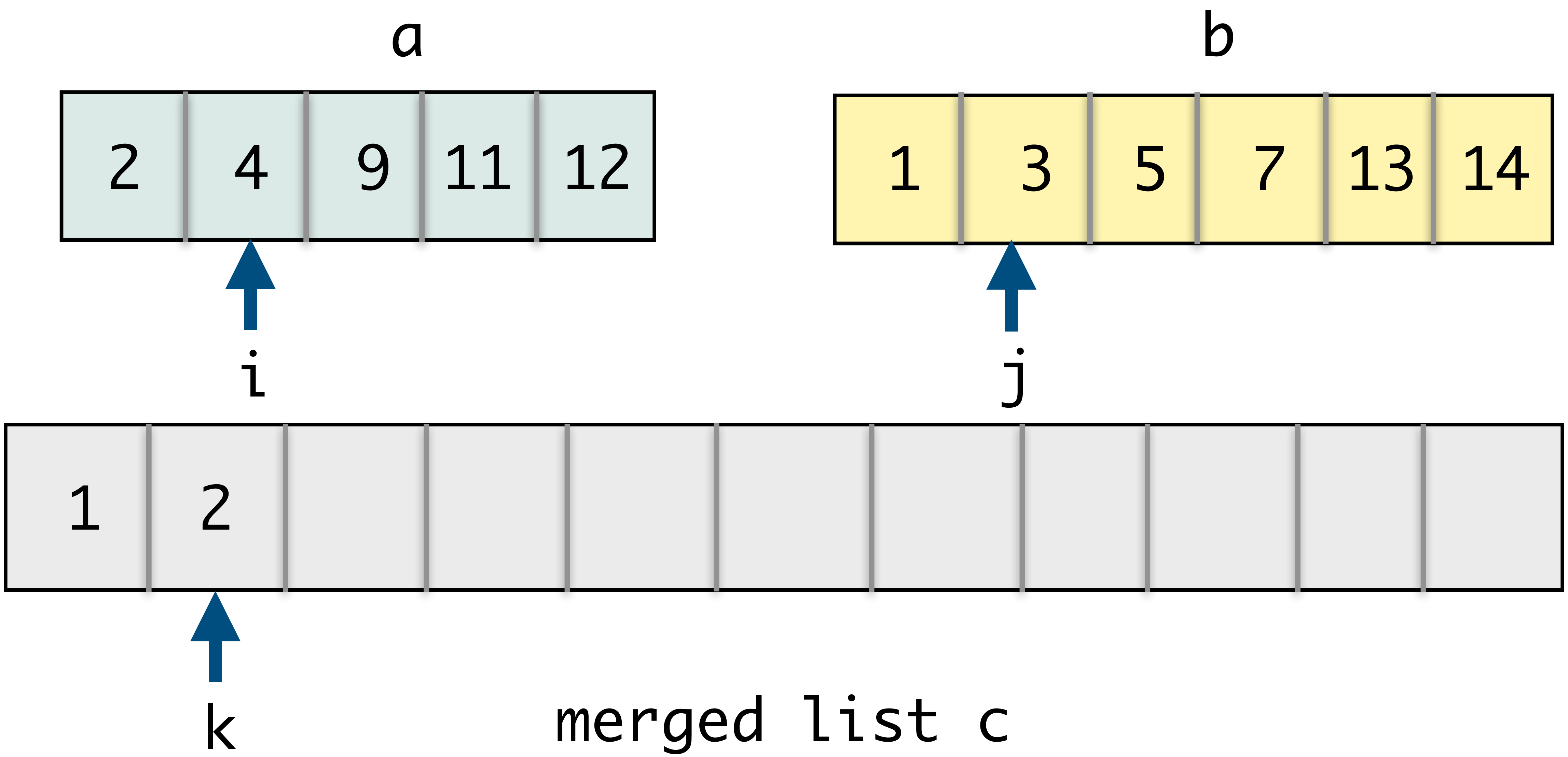
- Yes, `a[i]` appended to `c`
- No, `b[j]` appended to `c`



Merging Sorted Lists

Is `a[i] <= b[j]` ?

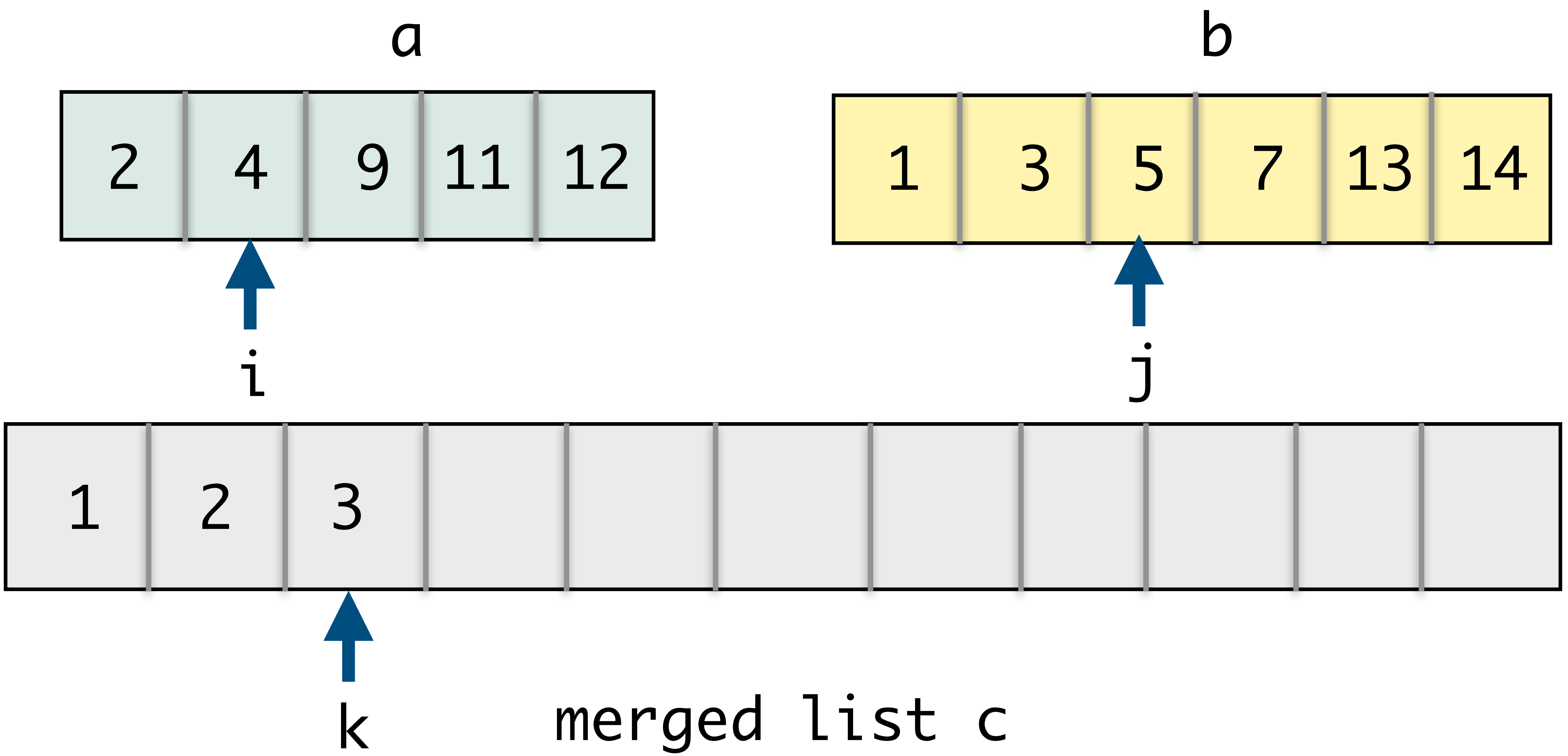
- Yes, `a[i]` appended to `c`
- No, `b[j]` appended to `c`



Merging Sorted Lists

Is `a[i] <= b[j]` ?

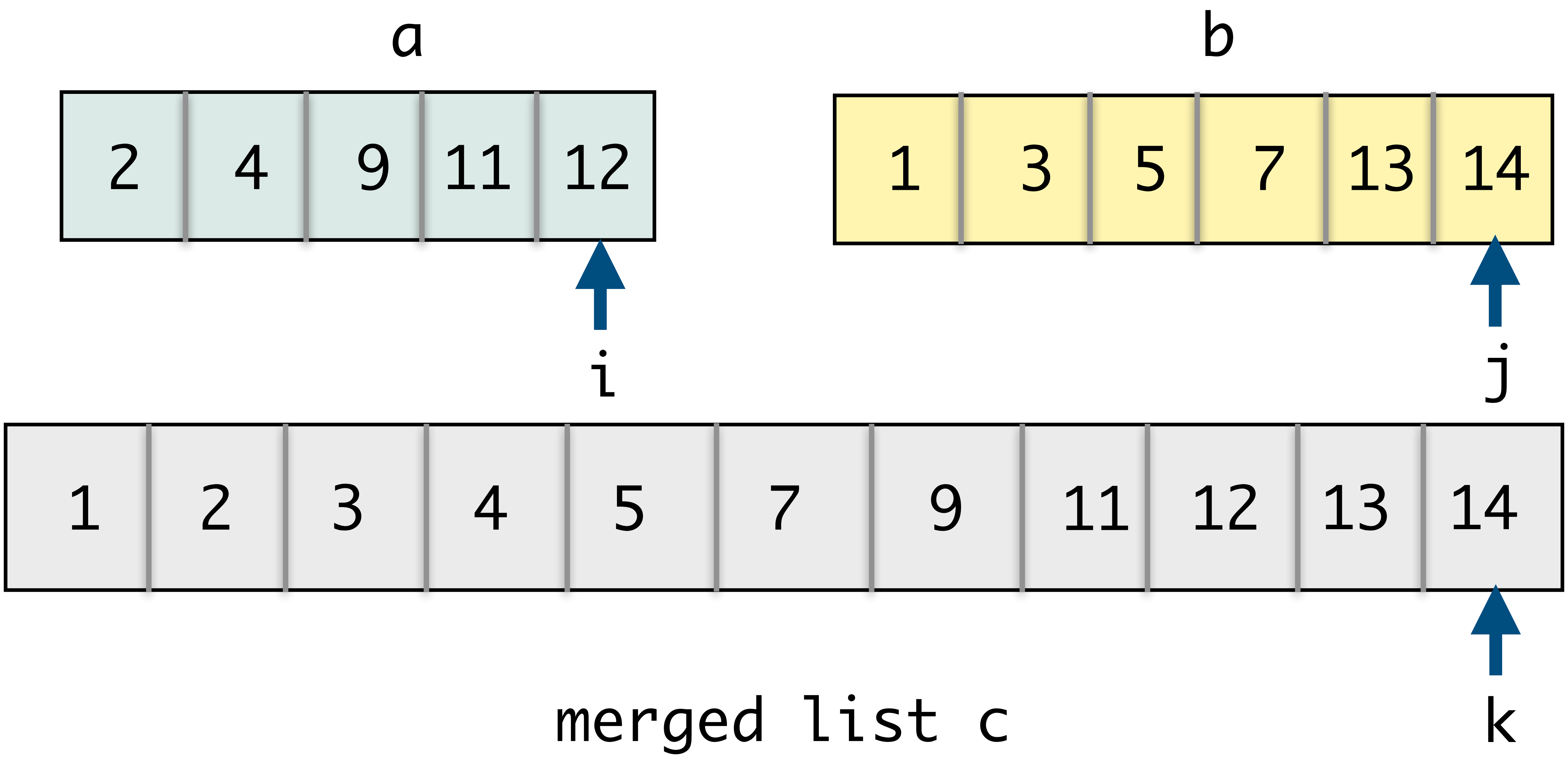
- Yes, `a[i]` appended to `c`
- No, `b[j]` appended to `c`



Merging Sorted Lists

Is `a[i] <= b[j]` ?

- Yes, `a[i]` appended to `c`
- No, `b[j]` appended to `c`



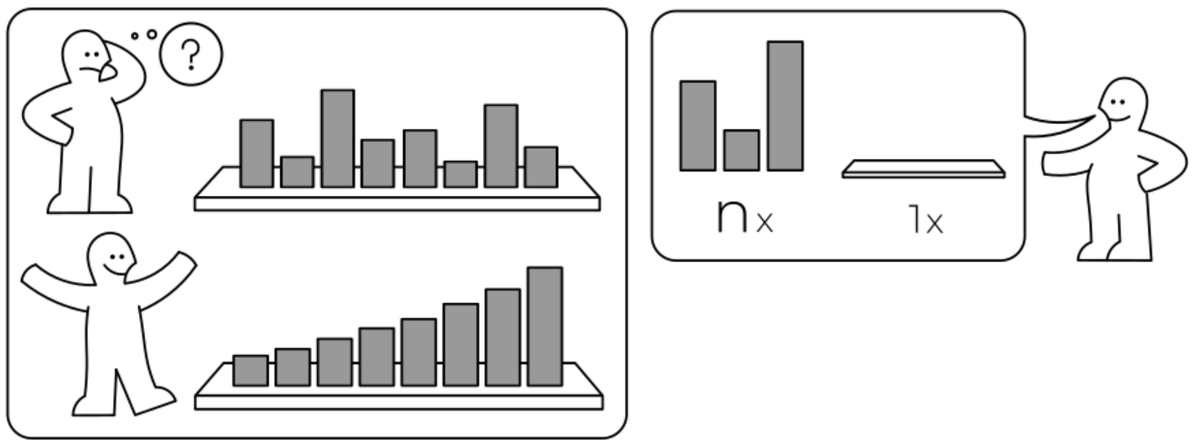
Merging Sorted Lists

- Walk through lists a, b, c maintaining current position of indices i, j, k
- Compare $a[i]$ and $b[j]$, whichever is smaller gets put in the spot of $c[k]$
- Merging two sorted lists into one is an $O(n)$ step algorithm!
- Can use this merge procedure to design our recursive merge sort algorithm!

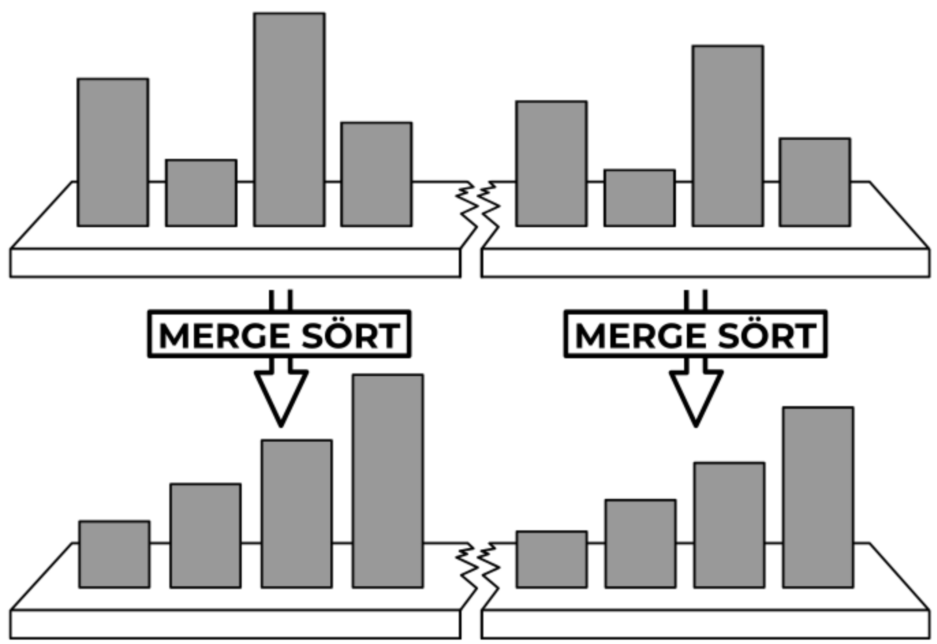
```python
def merge(a, b):
    """Merges two sorted lists a and b,
    and returns new merged list c"""
    # initialize variables
    i, j = 0, 0
    len_a, len_b = len(a), len(b)
    c = []
    # traverse and populate new list
    while i < len_a and j < len_b:

        if a[i] <= b[j]:
            c.append(a[i])
            i += 1
        else:
            c.append(b[j])
            j += 1

    # handle remaining values
    if i < len_a:
        c.extend(a[i:])

    elif j < len_b:
        c.extend(b[j:])

    return c
```

MERGE SÖRT

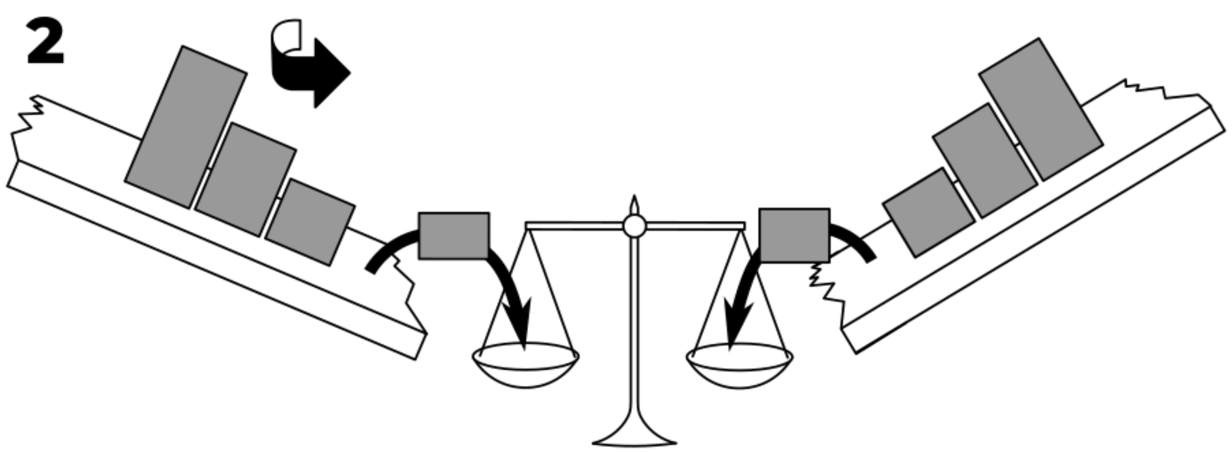
idea-instructions.com/merge-sort/



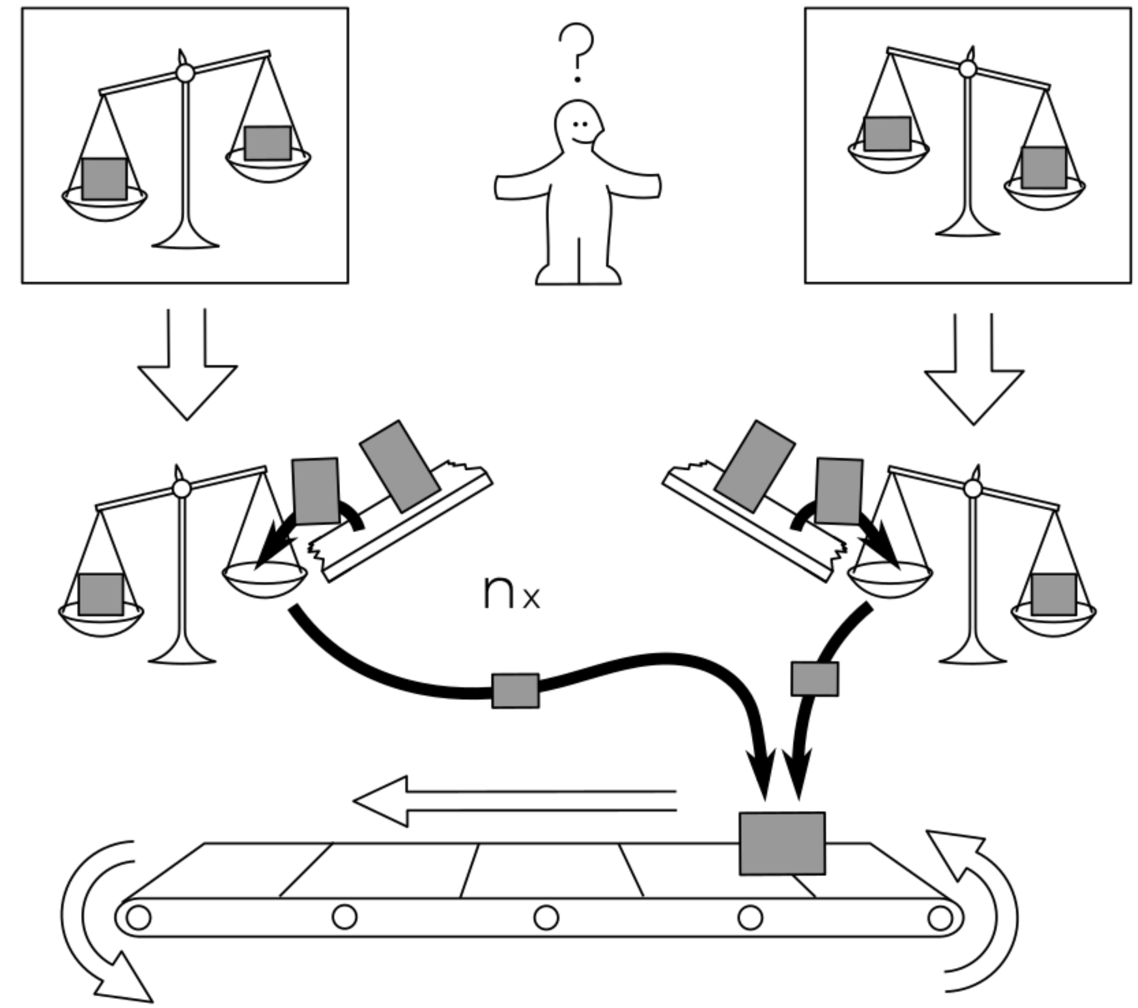
1



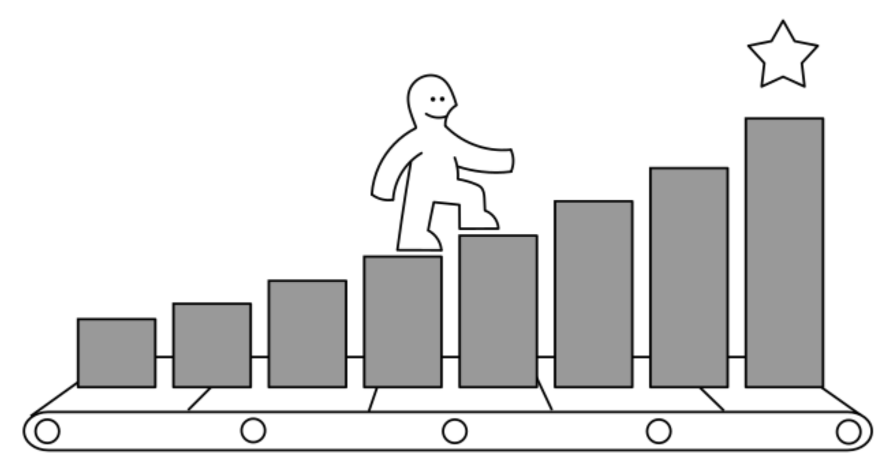
2



3



4



Merge Sort Analysis: Basic Idea

- If you take CS256 (Algorithms), you will learn how to analyze the Big Oh complexity of such recursive algorithms
- We'll give an intuitive explanation for now:
  - # times can we divide the list in half until we hit the base case?
    - $\approx \log_2 n$
  - # steps to merge two lists each of size $O(n)$?
    - $O(n)$
  - Merge occurs at every recursive step, so overall $O(n \log n)$ steps

Runtime Comparisons: Big Oh

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Summary: Searching and Sorting

- We have seen algorithms that are
  - $O(\log n)$: binary search in a sorted list
  - $O(n)$: linear searching in an unsorted list
  - $O(n \log n)$: merge sort
  - $O(n^2)$: selection sort
- Important to think about efficiency when writing code!