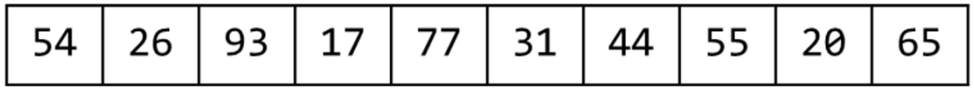
Searching and Sorting

by Kittipitch Kuptavanich

adapted to English by Prakarn Unachak

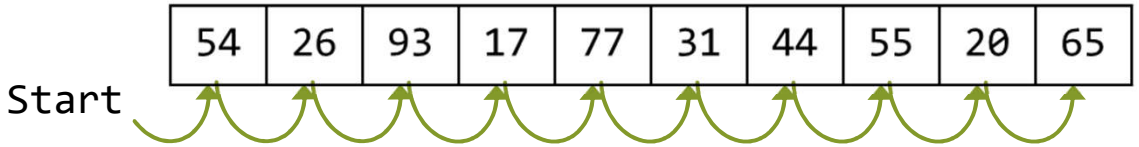
Searching

Searching



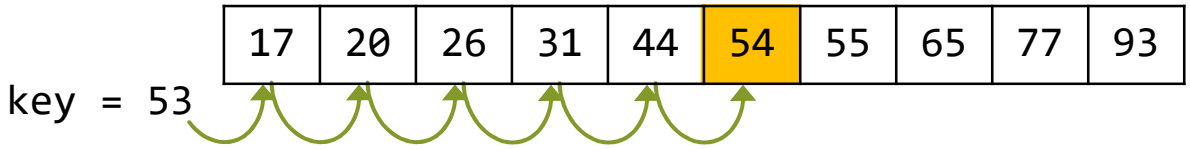
- One of the most encountered problem in computing
- In basic searching, we will try to find out whether an item is in a collection of items or not.
  - True/false – Does 18 exist in the list? (No)
  - With modification, it can also return the location of the item. – Where is 77? (5th item)

Linear Search



- If the collection is unsorted, i.e. not ordered, we can use linear search (also known as sequential search) to look for an item (called key).
  1. Start from the beginning, then:
  2. Look at the current item, if matches the key, return yes.
  3. If the current item does not match the key, check if this is the last item in the list, if so, return no. (since we have reach the last item and hasn't found the item yet.)
  4. If we are not at the last item, move to the next item and go back to 2.

Linear Search (2)



- However, if the list is sorted, we don't have to search until we have reached the end.
- For example, if the list is in ascending order (from smallest number to largest number), we can stop once the current item is larger than the key.
- But if we know that the list is sorted, there is a better way to do searching.

Source: Problem Solving with Algorithms and Data Structures Using Python - Miller and Ranum

Linear Search (3) - Exercise

- **We want to find key = 18 using linear search in an <u>unsorted list:</u>**

15	17	2	19	18	0	8	13	19	14

- **Which numbers to do we need to compare with the key before we can find 18?**
- **Try again with key = 12**

Linear Search (4) - Exercise

- **We want to find key = 13 using linear search in an <u>sorted list:</u>**

3	5	6	8	11	12	14	15	17	18

- **Which numbers to do we need to compare with the key before we can stop?**

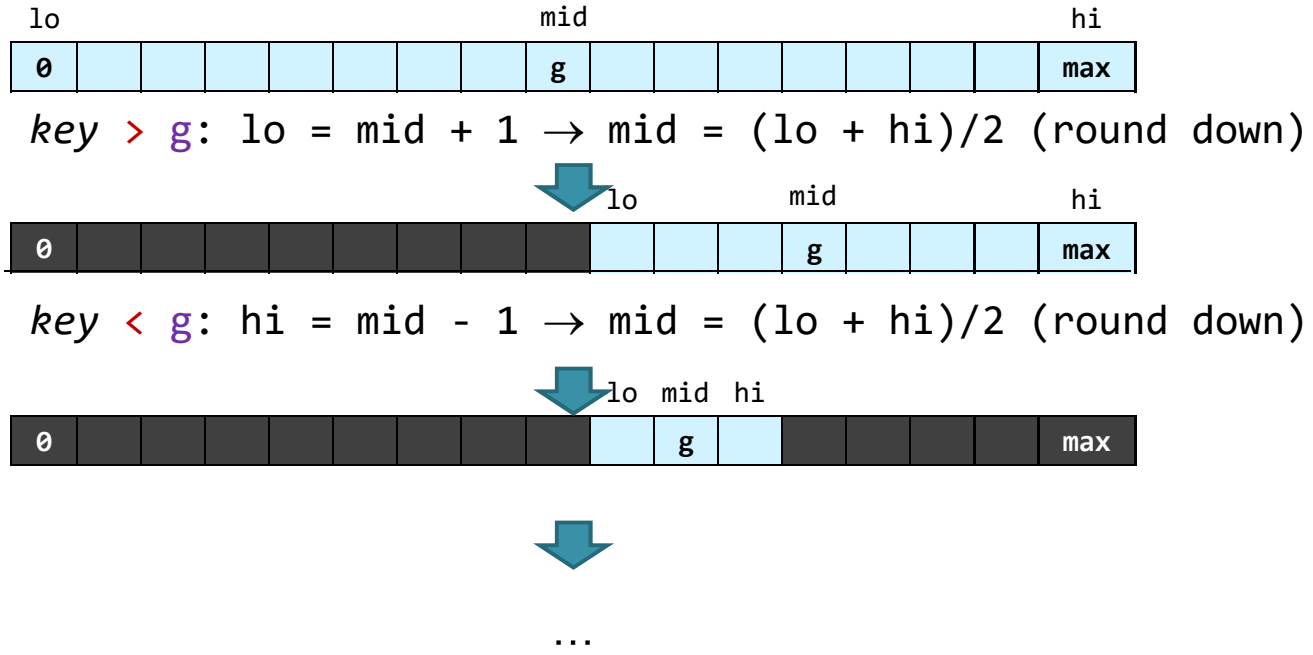
Binary Search

- **Considering looking for a word (key) in the dictionary.**
  - **The words are ordered. The dictionary contains a sorted list of words.**
  - **If we open a page (g) randomly:**
    - **If the word came before the key, we can just look for the word in pages after that page.**
    - **If the word came after the key, we can just look for the word in pages before that page.**
    - **So, we can basically divide the size of the list in half in each step.**

Source: Problem Solving with Algorithms and Data Structures Using Python - Miller and Ranum

Binary Search (2)

- **or Bisection Search, which can be use to find an item in ordered list.**



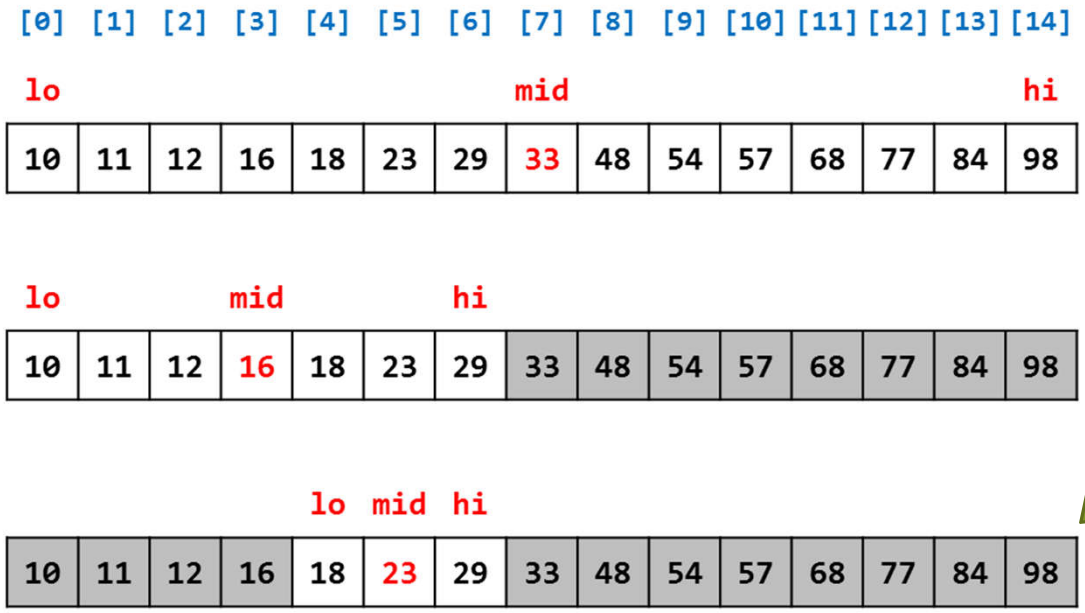
Source: Problem Solving with Algorithms and Data Structures Using Python - Miller and Ranum

Binary Search (3)

- For example, we have a sorted list of numbers

10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

- And want to search for key = 23



Binary search will stop when:

(1) An item matching the key is found, or

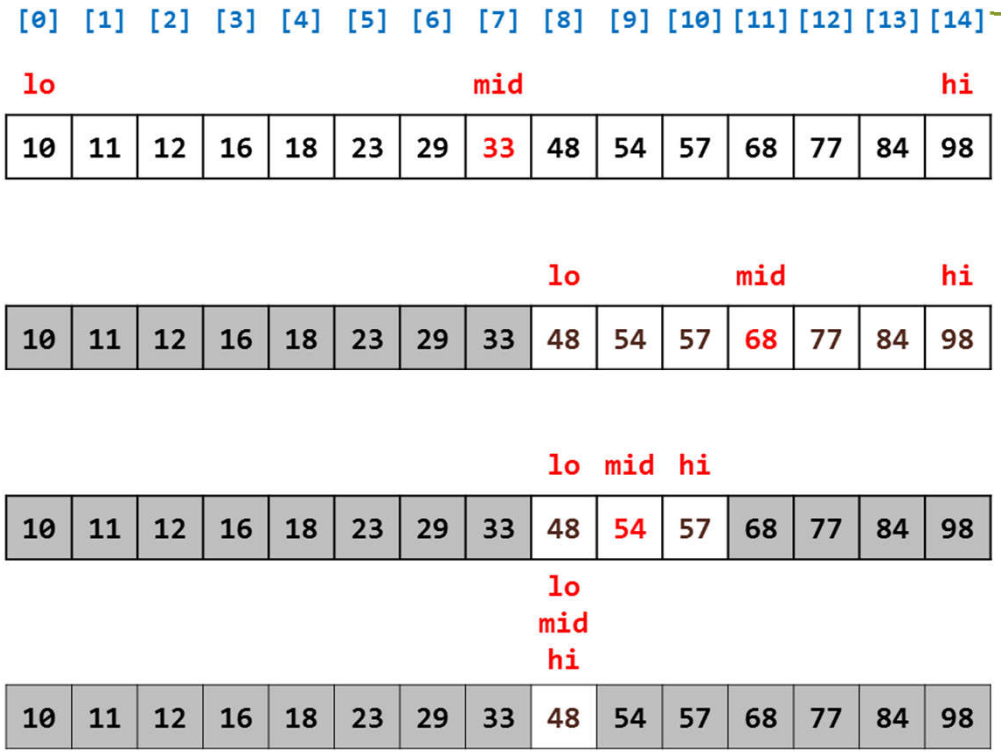
(2) lo = hi, and the item does not match the key.

Source: Problem Solving with Algorithms and Data Structures Using Python - Miller and Ranum

Binary Search (3)

10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

- **And we want to find key = 50**



Binary search will stop when:

(1) An item matching the key is found, or

(2) lo = hi, and the item does not match the key.

Source: Problem Solving with Algorithms and Data Structures Using Python - Miller and Ranum

Binary Search (4) - Exercise

- **If we want to find key = 8 using binary search in an ordered list**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
3	4	7	8	11	14	16	17	20	23	26	29	30	31	34

- **Which numbers will be compared with the key?**

Source: Problem Solving with Algorithms and Data Structures Using Python - Miller and Ranum

Binary Search (5) - Exercise

- **If we want to find key = 15 using binary search in an ordered list**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
3	4	7	8	11	14	16	17	20	23	26	29	30	31	34

- **Which numbers will be compared with the key?**

Source: Problem Solving with Algorithms and Data Structures Using Python - Miller and Ranum

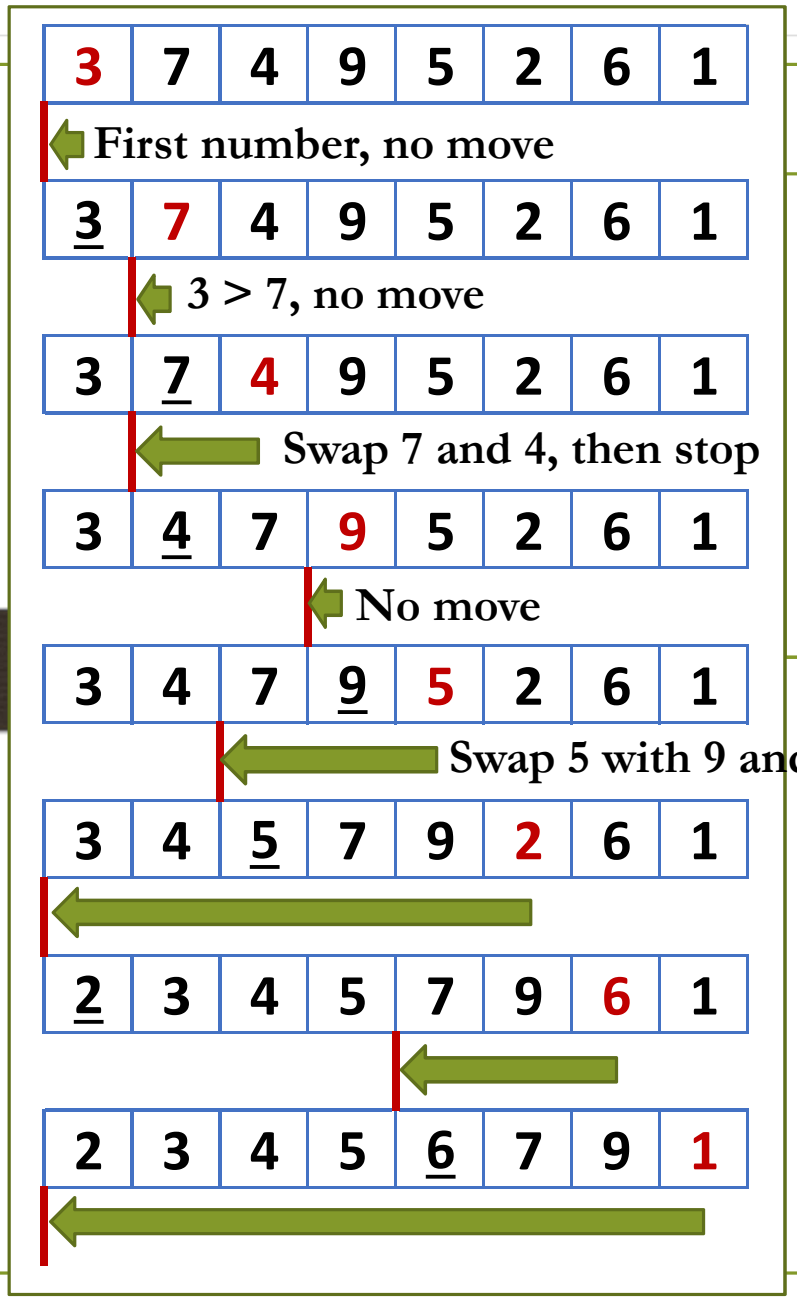
Sorting

Sorting

- As you can see, if we can have an ordered list of data, we can use binary search, which can greatly reduce search time.
- So, we need to sort an unordered list, to get an ordered one.
- We will talk about two examples of sortings:
  - Insertion sort
  - Merge sort.

Insertion Sort

- Starting from the leftmost item.
- Move it to the front until we find an item smaller than it, then put it after that item, or the beginning, if no such item is found.

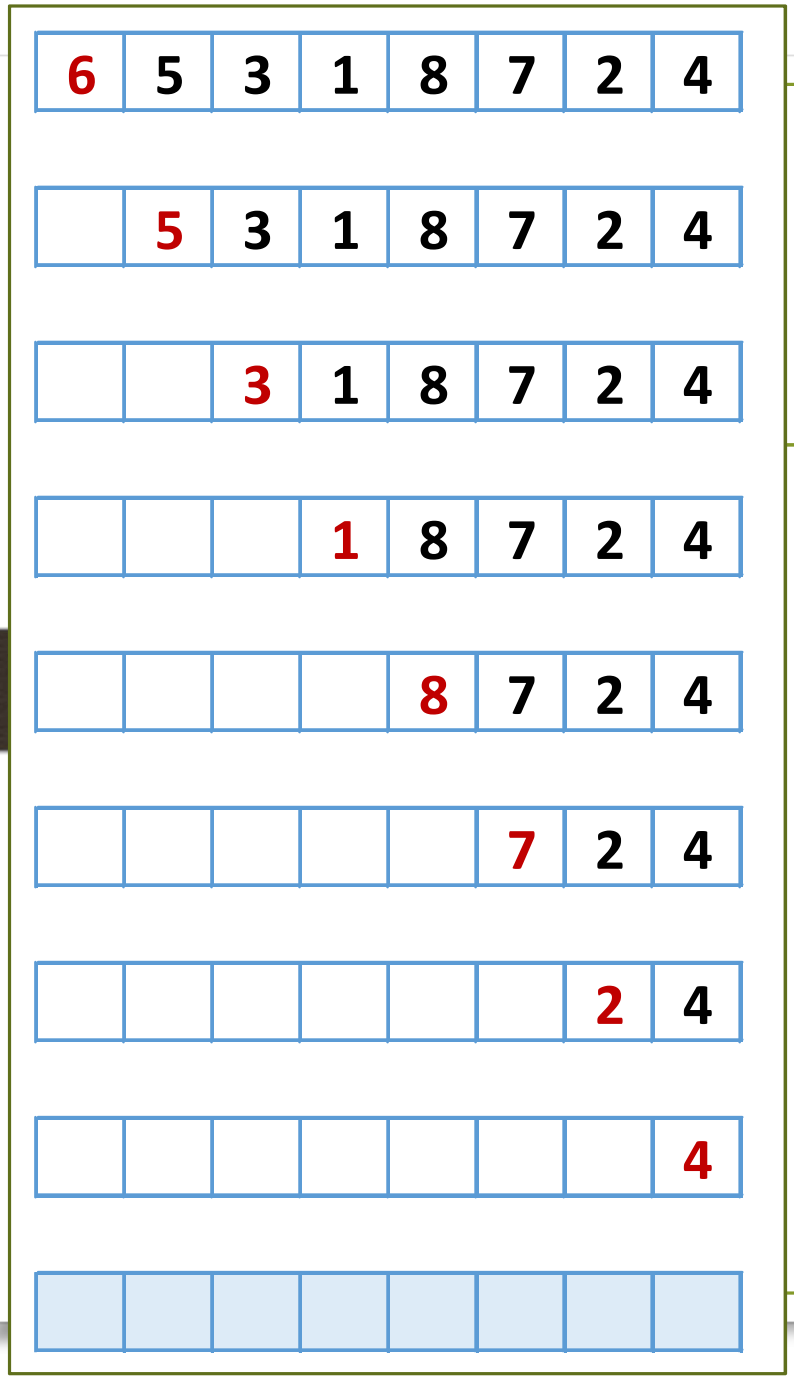


Sorted Section

To insert 5:
(1) 9 > 5, so we swap 9 and 5
(2) 7 > 5, so we swap 7 and 5
(3) 4 < 5 → stop

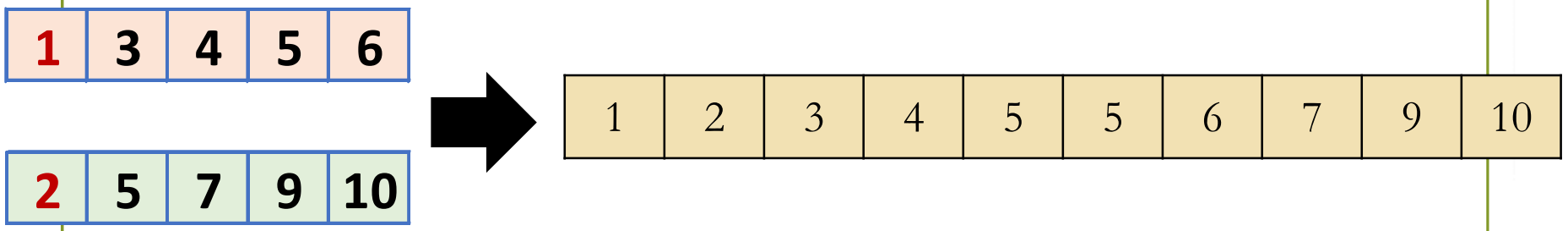
Insertion Sort

Practice 1

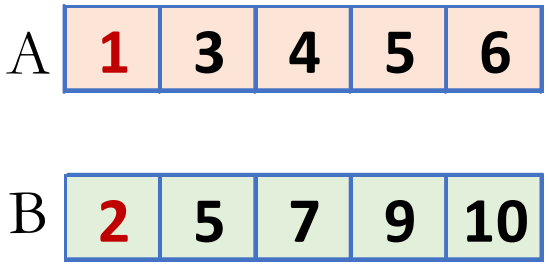


Merge Algorithm

- If we have two ordered lists, we have a way to combine them to get a bigger, but still ordered list.
  - The bigger list will contains all members of the two lists.
  - This process is called merging.

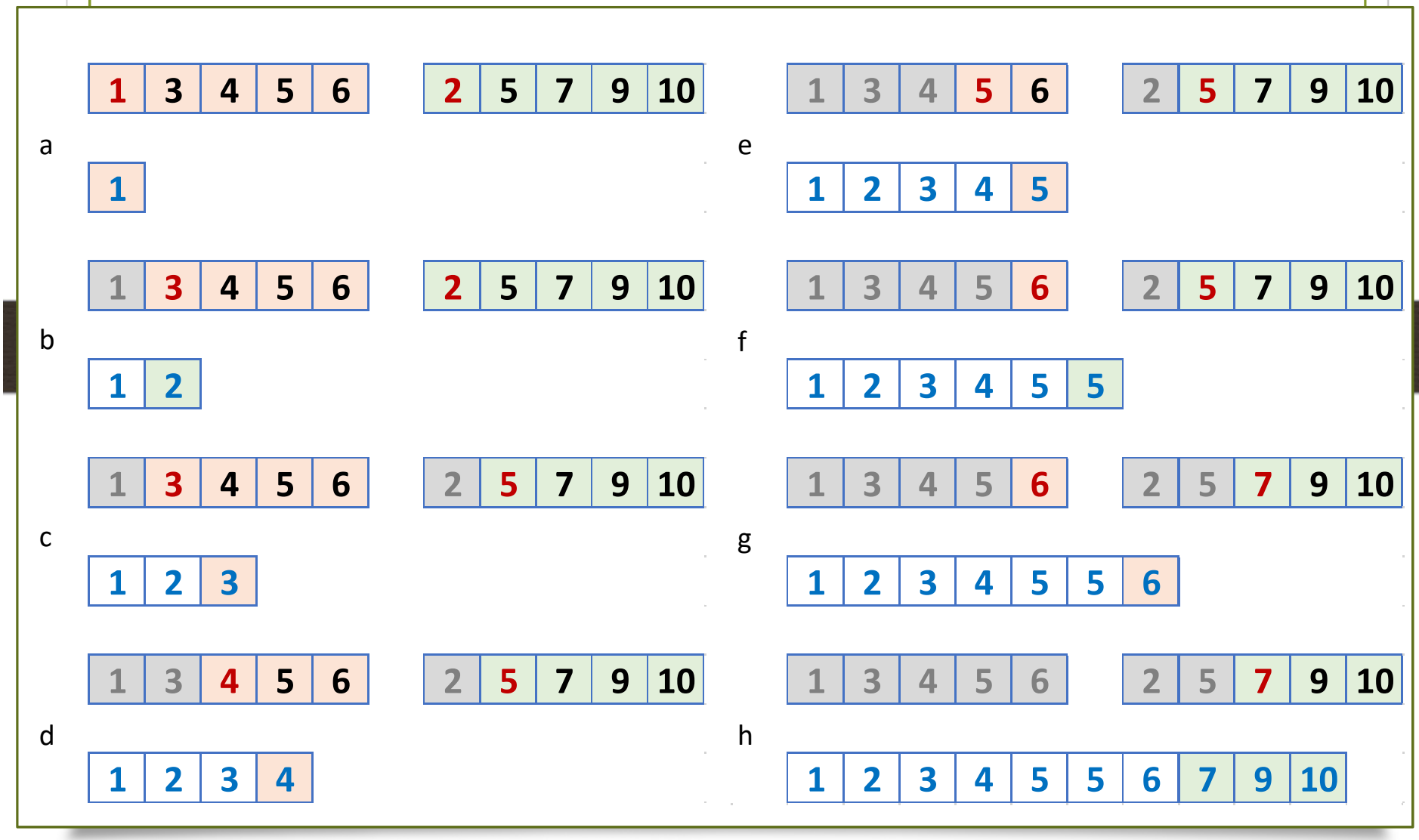


Merge Algorithm [2]

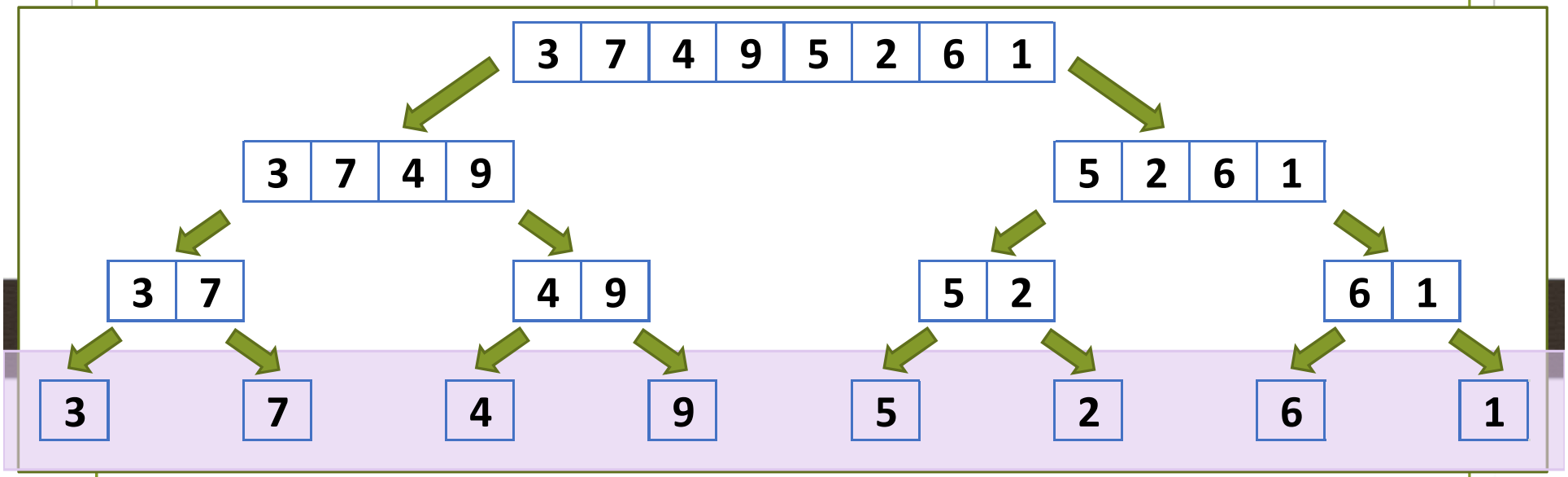


- We have order list A and B
- Let C be an empty list
- We will start looking at the beginning of each list
  - let *a* be the current item at list A and *b* be the current item at list B
  - If *a* < *b* add *a* into list C and look at the next element in A
  - Else, add *b* into list C and look at the next element in B
- Repeat until a list is empty, then we add the rest of the other list to C.

Merge Algorithm [3]



Merge Sort



- For Merge Sort, we divide the list up into smaller list, then keep dividing until we reach list of individual elements, use merging to combine them back up into sorted list.

Merge Sort