



Middlebury

CSCI 146: Intensive Introduction to Computing

Fall 2025

Lecture 17: Searching and Sorting

Python uses **`sys.maxsize`** as the maximum size of a container (and hence maximum index).

On most modern computers, this max size is represented by an integer with 64 bits. So what's **`sys.maxsize`**?

```
>>> import sys
>>> sys.maxsize
9223372036854775807
```

$$2^{63} - 1$$

- Account for negative indices.
- 1 bit used for sign.

But computers use "two's complement" to encode negative and positive numbers.

Use the most-significant (leftmost) bit (MSB) to represent the sign (0: positive, 1: negative).

- But we don't just use this sign bit with the unchanged number (e.g. -6 would not be 1110).
- Instead, start with absolute value of number (e.g. 6 is 0110).
- Then flip all the bits: 1001 .
- Add one: 1010 . This is the two's-complement representation of -6 .

Check? Convert as usual but use $-$ on MSB term.

int64_t
uint64_t

6 : 0110
-6 : 1110 X not
start: 0110 flip bits
1001 add one
1010 $\rightarrow -6$ using two's complement
 $-2^3 \quad 2^2 \quad 2^1 \quad 2^0 = -8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$
 $= -6$

Representing floating-point numbers.

```
>>> 0.1 + 0.2 <= 0.3  
False
```

64-bit floating-point number:

- 1 bit for sign s
- 11 bits for exponent e (offset from some bias e_0)
- 52 bits for mantissa m

$$s * 1.m * 2^{e-e_0}$$

Question 1: Will $0.125 + 0.25 \leq 0.375$ evaluate to **True** (as expected mathematically)?

A. Yes

B. Possibly, I'd have to try it out.

C. No

Handwritten notes illustrating floating-point representation:

$$S * 1.m * 2^{e-e_0}$$

Labels with arrows pointing to the components:

- sign (points to S)
- mantissa (points to m)
- exponent (points to $e - e_0$)

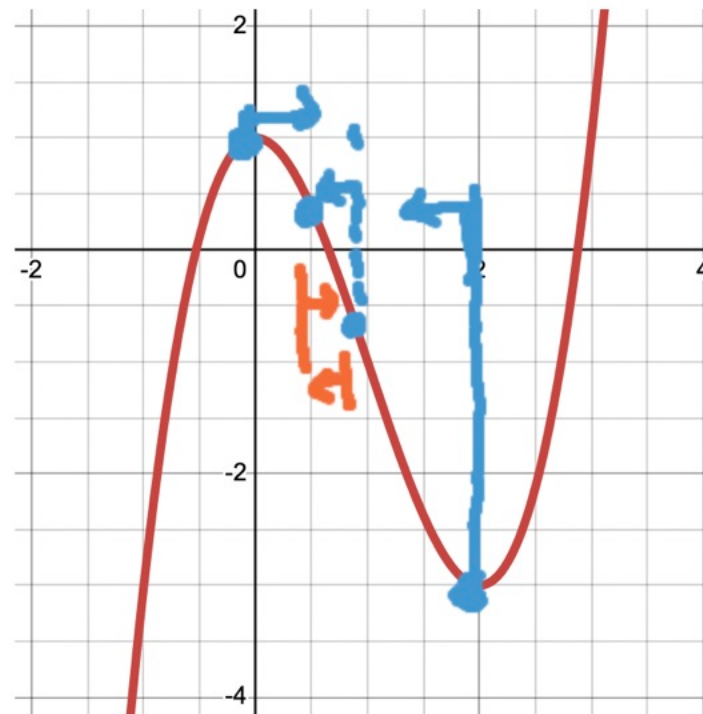
Handwritten notes showing binary representations:

$$0.125 = 2^{-3}$$
$$0.25 = 2^{-2}$$

Goals for today

- Describe algorithms for linear and binary search.
- Describe algorithms for selection sort, insertion sort and merge sort.
- Recall the asymptotic runtime complexity of different search and sort algorithms.

Why is searching important?



bisection method

What if we want the root of $f(x) = x^3 - 3x^2 + 1$ in the interval $[0, 2]$?

Determining the index of an **item** in a list with *linear search*.

```
def linear_search(a_list, item):  
    """Return index of item in a_list or None if not found  
  
    Args:  
        a_list: A sequence of comparable values  
        item: A value to search for in a_list  
  
    Returns:  
        Index of item or None if not found  
    """  
    for i in range(len(a_list)):  
        if item == a_list[i]:  
            return i  
    return None
```

- [31, 98, 80, 87, 94, 22, 32, 35]
- [11, 14, 21, 23, 36, 49, 51, 79]

find 94
find 14

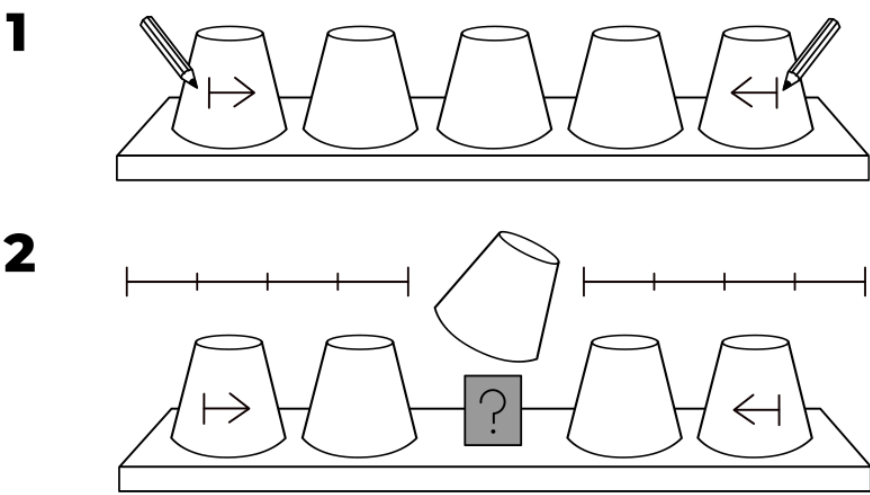
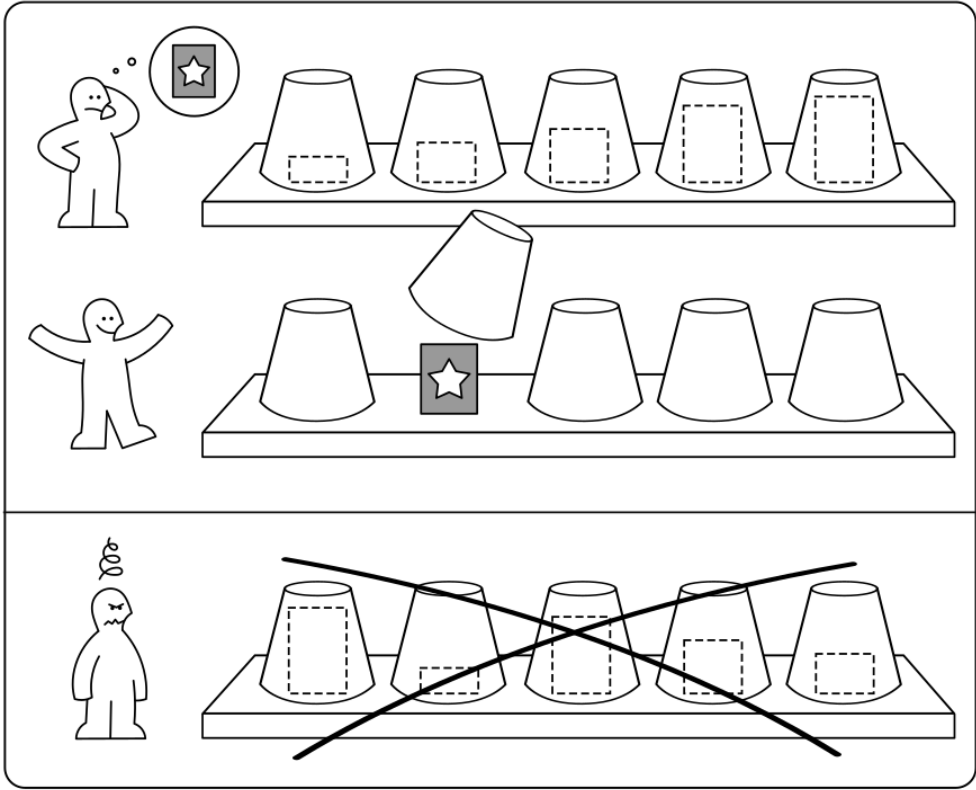
Complexity

$O(n)$ worst
 $O(n)$ average
 $O(1)$ best

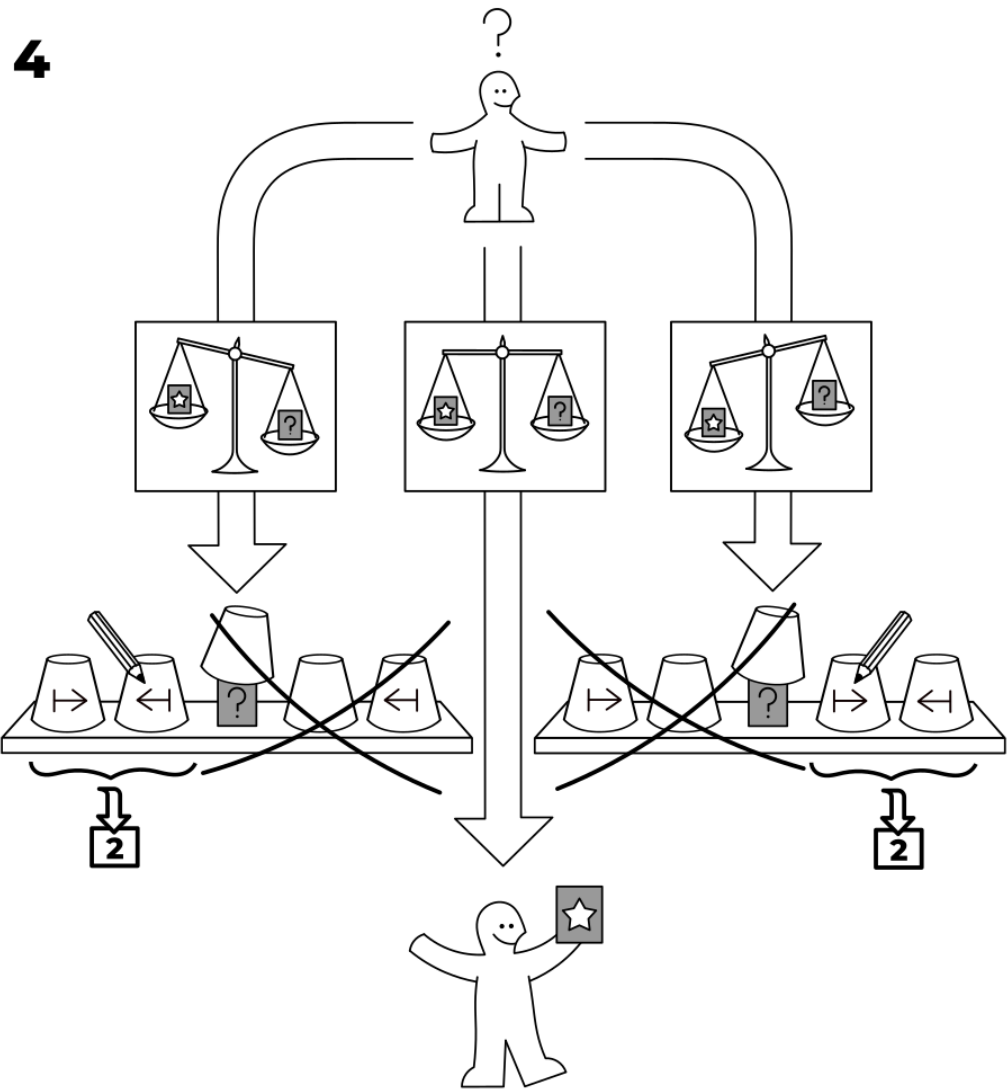
↑
item we are looking
for is at the
beginning

What if the list is sorted? Can we use this to our advantage?
Yes: *binary search*.

BINÄRY SEARCH



idea-instructions.com/binary-search/
v1.1, CC by-nc-sa 4.0 **IDEA**



What if the list is sorted? Can we use this to our advantage?

Yes: *binary search*.

$$\log_2(n) = \frac{\log(n)}{\log(2)}$$

```
def binary_search(a_list, item, lo, hi):  
    """  
    Return index of item in a sorted a_list or None if not found  
    Args:  
        a_list: A sequence of values sorted in ascending order  
        item: A value to search for in a_list  
        lo: Inclusive start index to search in a_list  
        hi: Inclusive end index to search in a_list  
    Returns:  
        Index of item or None if not found  
    """  
    if lo > hi:  
        return None  
    else:  
        middle = (lo + hi) // 2  
        middle_elem = a_list[middle]  
        if item == middle_elem:  
            return middle  
        elif item < middle_elem:  
            return binary_search(a_list, item, lo, middle - 1)  
        else:  
            return binary_search(a_list, item, middle + 1, hi)
```

0 1 2 3 4 5 6 7
[2, 3, 7, 9, 10, 12, 18, 19]

find 9 middle = (0+7)//2 = 3

best case: O(1)

find 2
0 1 2
[2, 3, 7]

lo = 0 hi = 2
middle = (0+2)//2
 = 1

0
[2]
return 0

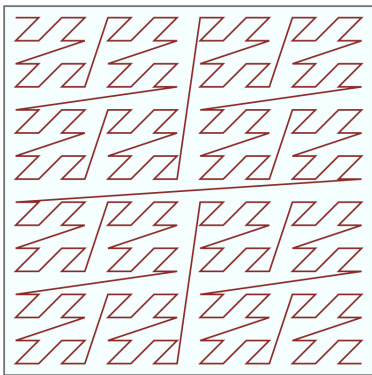
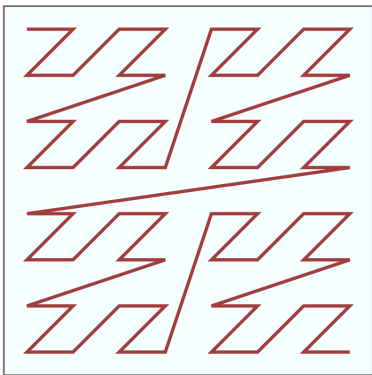
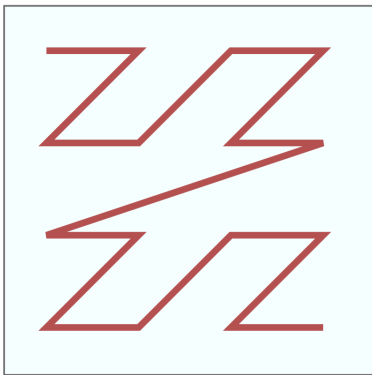
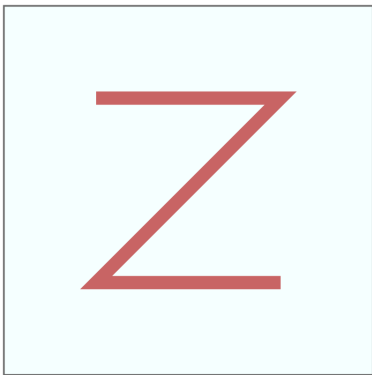
lo = 0 hi = 0
middle = (0+0)//2
 = 0

how many times does the list get halved until we get to a sublist of length 1?

when is $\frac{n}{2^k} = 1 \rightarrow k = \log_2 n$

$O(\log n)$ worst and average

How do we get sorted lists in the first place? And what are other applications of sorting?



TITLES NAMES COLLABORATIONS

Movie X Animation X Keyword: "pixar" X

Action · 5 Adventure · 30 Animation X Biography · 0 Comedy · 27 Crime · 0 Documentary · 1 Drama · 12 Family · 30 Fantasy · 20 Film-Noir · 0 Game-Show · 0 History · 0 Horror · 0 Music · 3 Musical · 0 Mystery · 2 News · 0 Reality-TV · 0 Romance · 2 Sci-Fi · 5 Short · 0 Sport · 2 Talk-Show · 0 Thriller · 0 War · 0 Western · 0

Exclude
☐ Documentary ☐ Short

Awards & recognition

Page topics

Companies

Instant watch options

US certificates

Color info

5. Brave

2012 1h 33m PG
★ 7.1 (450K) ☆ Rate 69 Metascore

Determined to make her own path in life, Princess Merida defies a custom that brings chaos to her kingdom. Granted one wish, Merida must rely on her bravery and her archery skills to undo a beastly curse.

6. The Good Dinosaur

2015 1h 33m PG
★ 6.7 (130K) ☆ Rate 66 Metascore

In a world where dinosaurs and humans live side-by-side, an Apatosaurus named Arlo makes an unlikely human friend.

7. Inside Out

2015 1h 35m PG
★ 8.1 (831K) ☆ Rate 94 Metascore

After young Riley is uprooted from her Midwest life and moved to San Francisco, her emotions - Joy, Fear, Anger, Disgust and Sadness - conflict on how best to navigate a new city, house, and school.

8. A Bug's Life

1998 1h 35m G
★ 7.2 (320K) ☆ Rate 78 Metascore

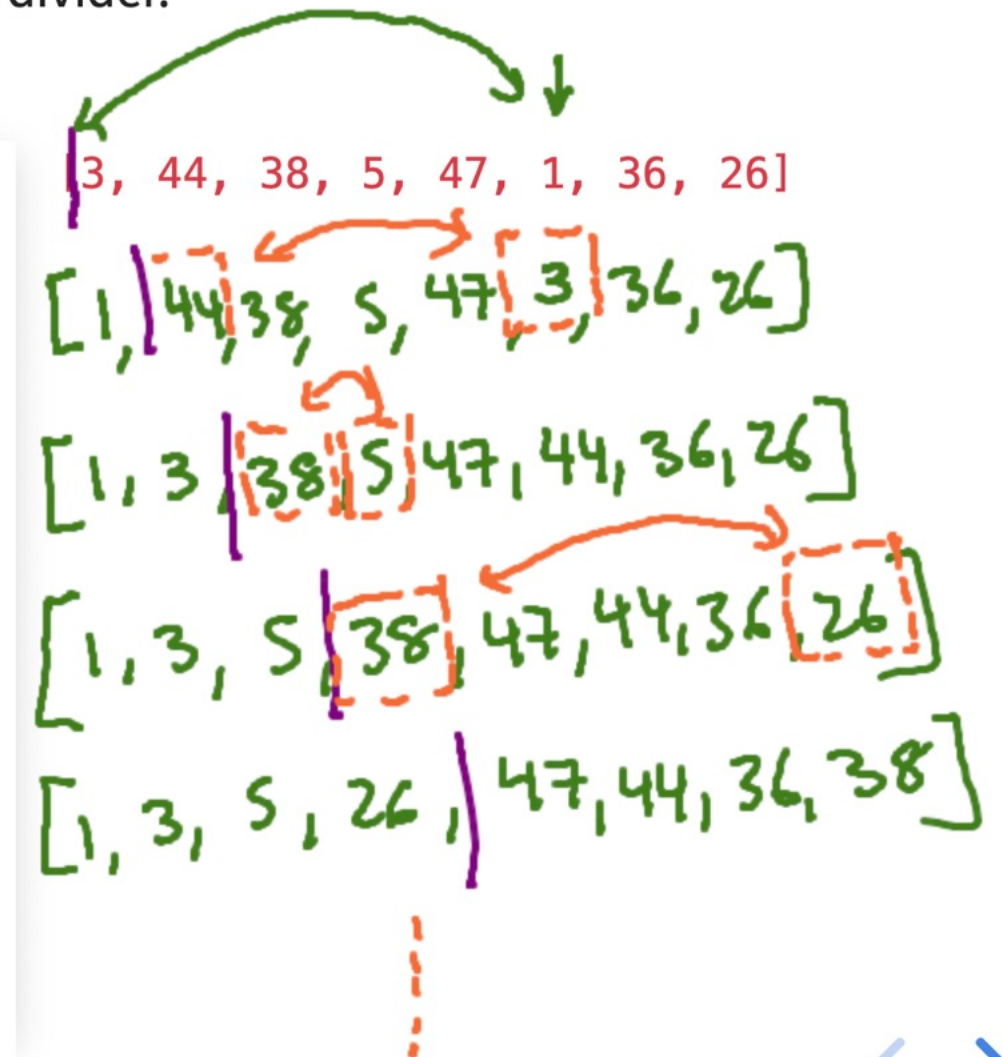
A misfit ant, looking for "warriors" to save his colony from greedy grasshoppers, recruits a group of bugs that turn out to be

Sorting Algorithm #1 (Selection Sort):

Main idea: maintain sorted elements on the left (of some imaginary divider) and unsorted elements on the right.

1. Find the smallest element in unsorted part.
2. Swap this smallest element with the element to the right of this divider.
3. Move the divider to the right (by one) and go back to Step 1.

```
1 def selection_sort(a_list):
2     """
3     Sort list in place using the selection sort algorithm
4
5     Args:
6         a_list : List to sort in place
7     """
8     # In each iteration, find the next smallest element in the list
9     # and swap it into the appropriate place
10    for i in range(len(a_list)):
11        # Find the index of the smallest value from i onwards
12        min_index = i
13        min_value = a_list[i]
14
15        for j in range(i+1, len(a_list)):
16            if a_list[j] < min_value:
17                min_index = j
18                min_value = a_list[j]
19
20        # Swap i and min_index
21        a_list[i], a_list[min_index] = a_list[min_index], a_list[i]
```

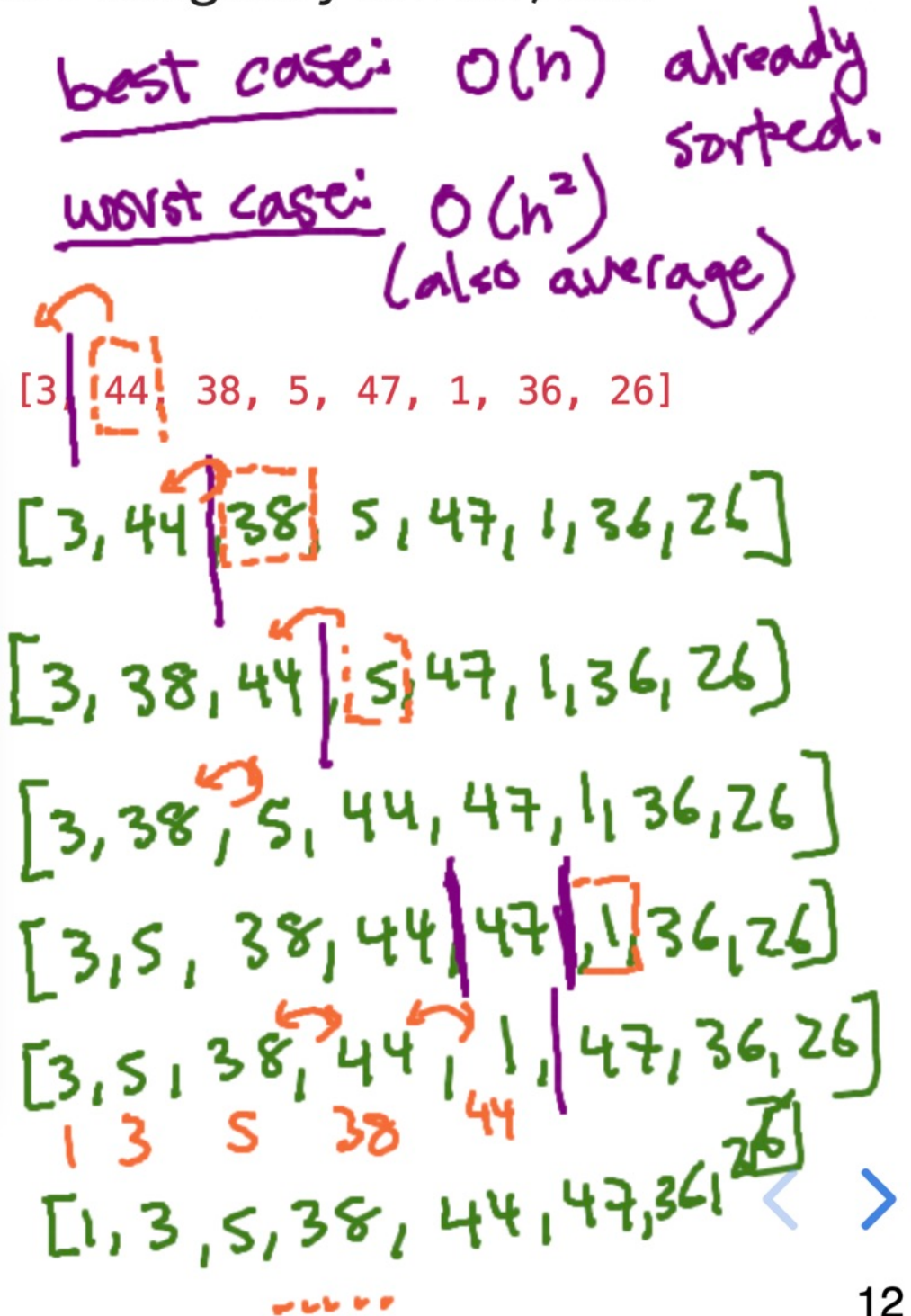


Sorting Algorithm #2 (Insertion Sort):

Main idea: maintain sorted elements on the left (of some imaginary divider) and unsorted elements on the right.

1. Look at first element in unsorted part (to the right of divider).
2. Iteratively swap this into the correct place in the sorted part.
3. Move the divider to the right (by one) and go back to Step 1.

```
1 def insertion_sort(a_list):
2     """
3     Sort list in place using the insertion sort algorithm
4
5     Args:
6         a_list : List to sort in place
7     """
8     for i in range(1, len(a_list)):
9         # Values at [0, i-1] are sorted already
10        # Shift up all values in [0, i-1] greater than a_list[i]
11        value = a_list[i]
12        index = i
13
14        while index > 0 and a_list[index-1] > value:
15            a_list[index] = a_list[index-1]
16            index -= 1
17        # Now insert value (old a_list[i]) in its proper place
18        a_list[index] = value
19        # Now everything from 0...i is sorted
```



Runtime analysis of selection and insertion sort.

```
def selection_sort(a_list):
    for i in range(len(a_list)):
        min_index = i
        min_value = a_list[i]

        for j in range(i+1, len(a_list)):
            if a_list[j] < min_value:
                min_index = j
                min_value = a_list[j]

        a_list[i], a_list[min_index] = a_list[min_index], a_list[i]
```

$$\begin{aligned}
 & \approx 1+2+3+\dots+(n-1)+n \\
 & i=0 \quad + \quad n-1 \quad < \quad \frac{n(n+1)}{2} \\
 & i=1 \quad + \quad n-2 \quad < \\
 & i=2 \quad + \quad n-3 \quad < \quad O(n^2) \\
 & \vdots \quad + \quad \vdots \\
 & i=n-2 \quad + \quad 1 \quad < \\
 & i=n-1 \quad + \quad 0 \quad < \\
 & \text{best, worst, average.}
 \end{aligned}$$

```
def insertion_sort(a_list):
    for i in range(1, len(a_list)):
        value = a_list[i]
        index = i

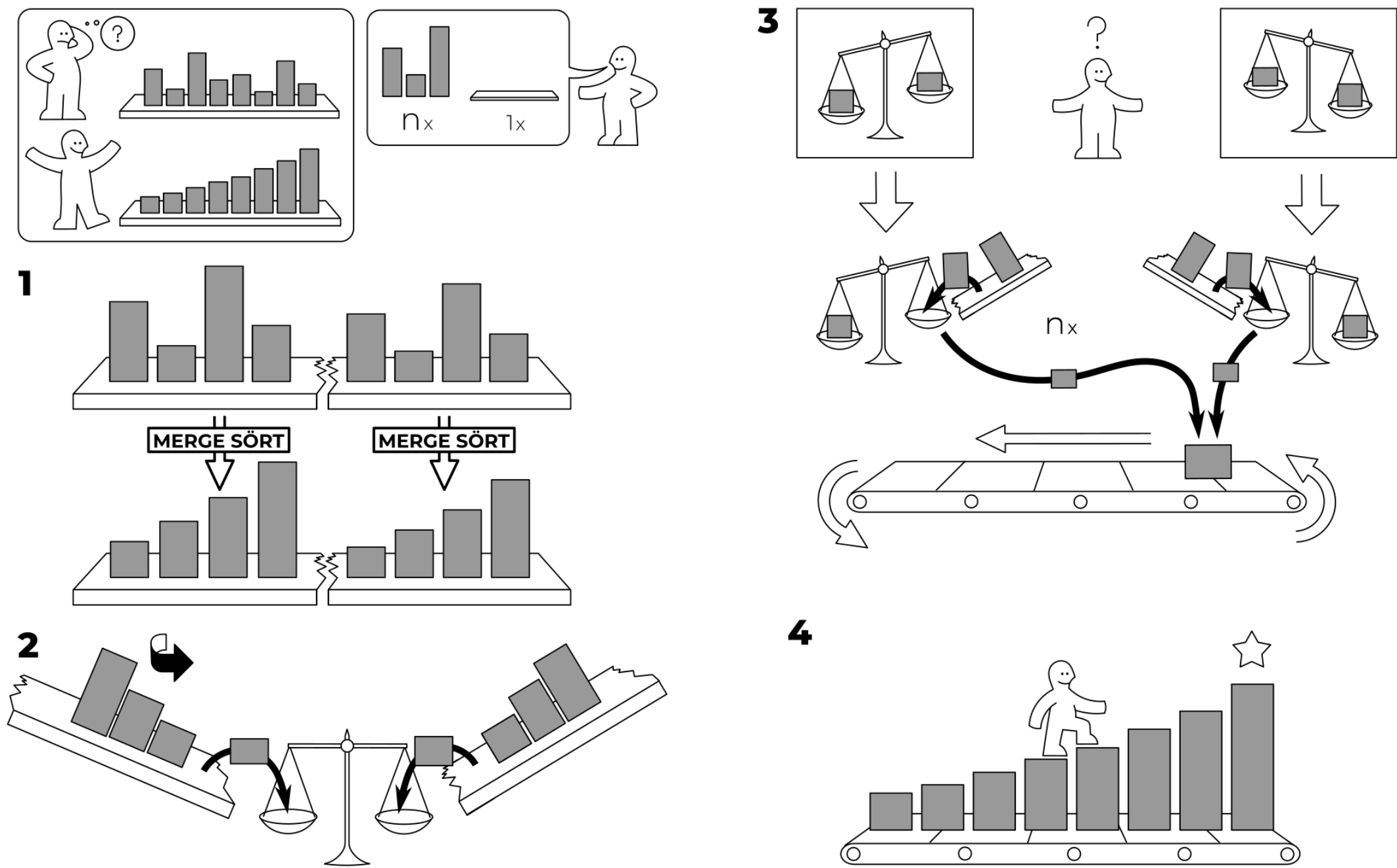
        while index > 0 and a_list[index-1] > value:
            a_list[index] = a_list[index-1]
            index -= 1
        a_list[index] = value
```

$$\begin{aligned}
 & i=0 \quad + \quad 0 \quad < \\
 & i=1 \quad + \quad 1 \quad < \\
 & \vdots \\
 & i=n-1 \quad + \quad n-1 \\
 & O(n^2) \text{ worst, average} \\
 & O(n) \text{ best (sorted)}
 \end{aligned}$$

Sorting Algorithm #3: merge_sort.

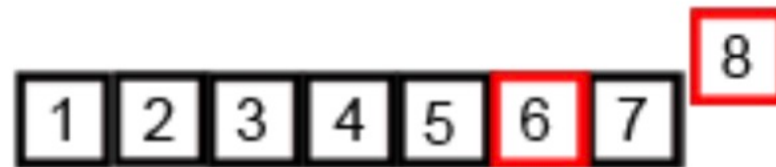
MERGE SÖRT

idea-instructions.com/merge-sort/
v1.2, CC by-nc-sa 4.0 **IDEA**



Sorting Algorithm #3: **merge_sort**.

1. Divide input list into two sublists.
2. Call **merge_sort** on each sublist.
3. Merge the result.



How many times do we need to halve a list of length n until we get a sublist of length 1?

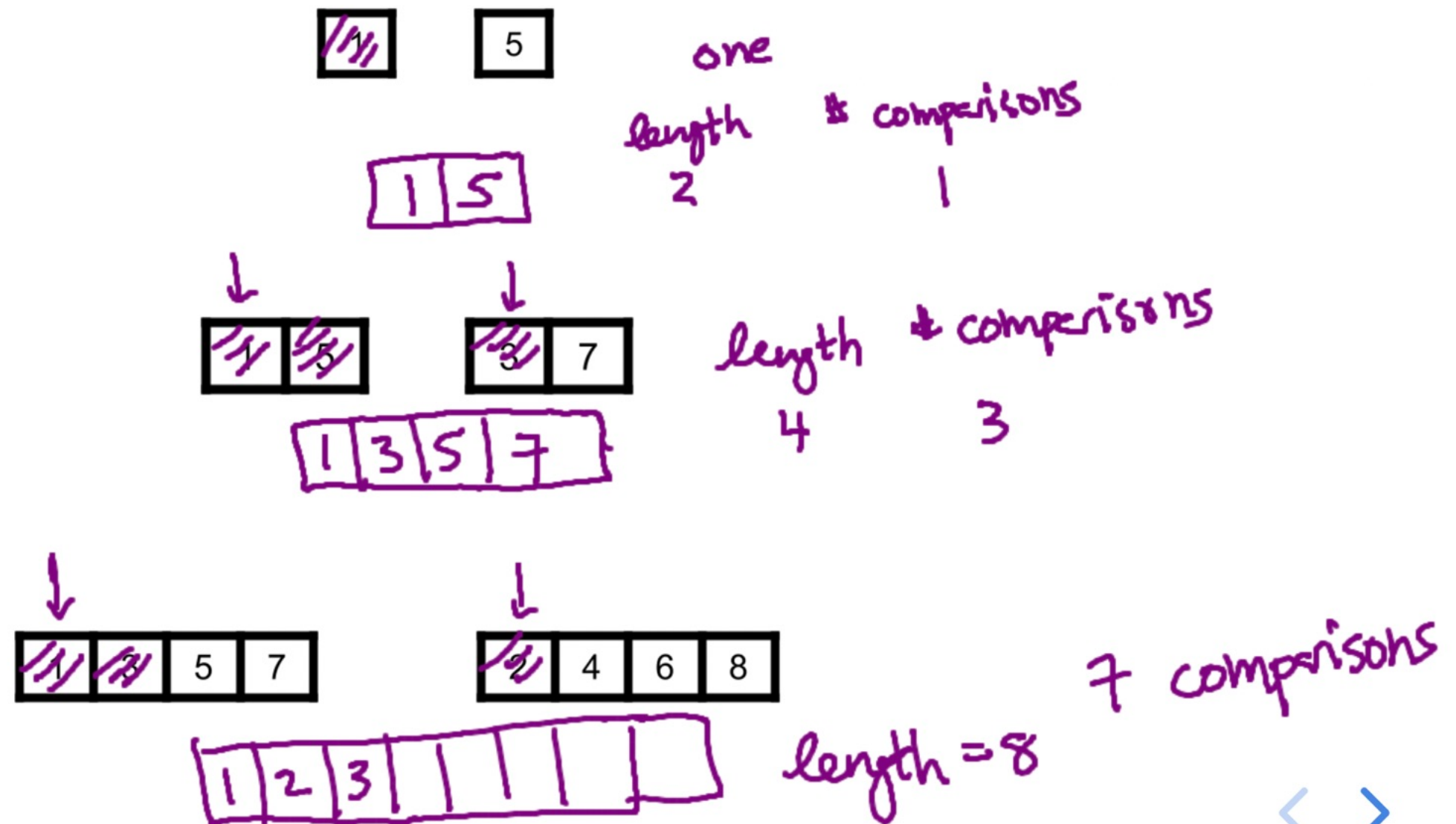
$$\frac{n}{2^k} = 1$$

$$k = \log_2 n$$

(from Wikipedia)

Analyzing the merge in **merge_sort**.

How many times do you need to ask: "*which leading element is smallest?*"
(when merging two sublists)

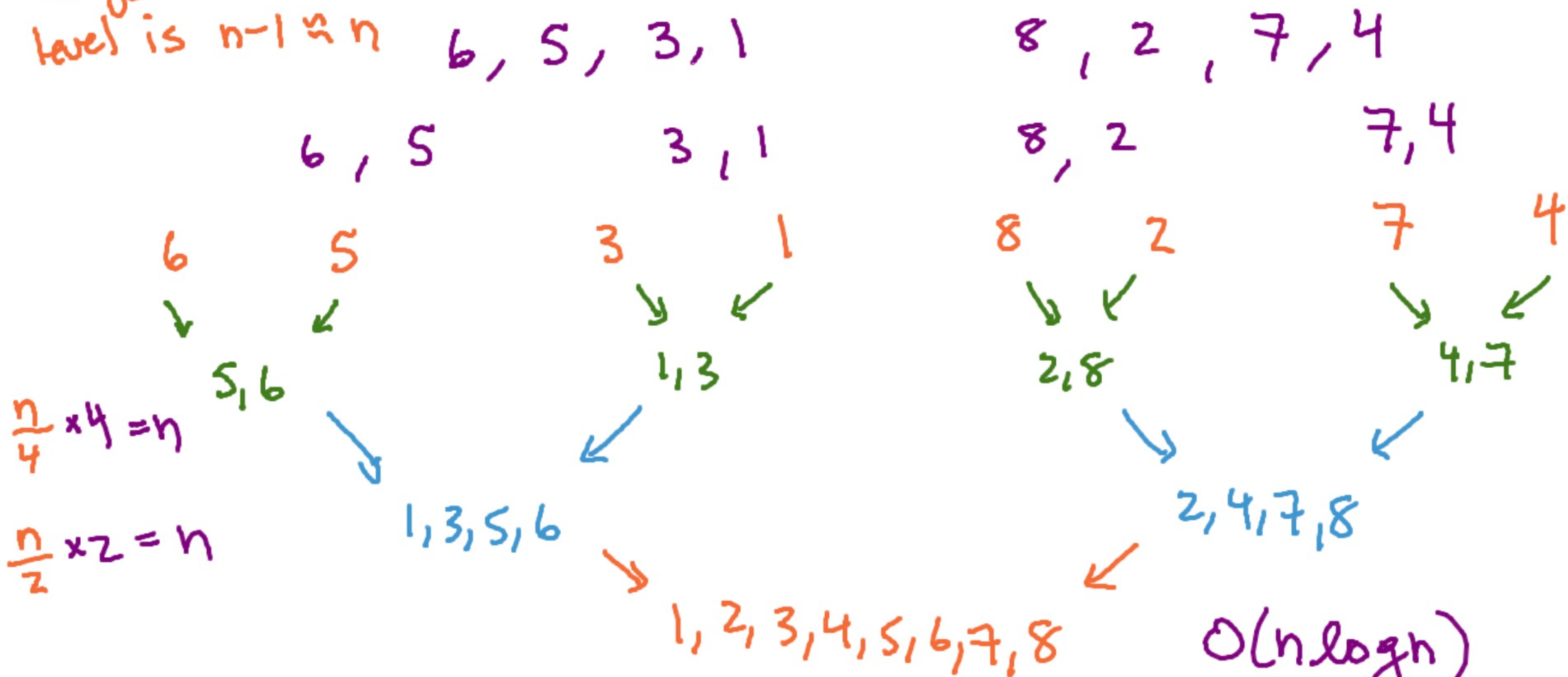


Analyzing **merge_sort**: adding up the number of **<** from each level.

$$\# \text{levels} = \log_2 n$$

6	5	3	1	8	2	7	4
---	---	---	---	---	---	---	---

comparisons to merge at each level is $n-1 \approx n$



$$\frac{n}{4} \times 4 = n$$

$$\frac{n}{2} \times 2 = n$$

$$n = 8$$

$O(n \log n)$

best, worst, average.

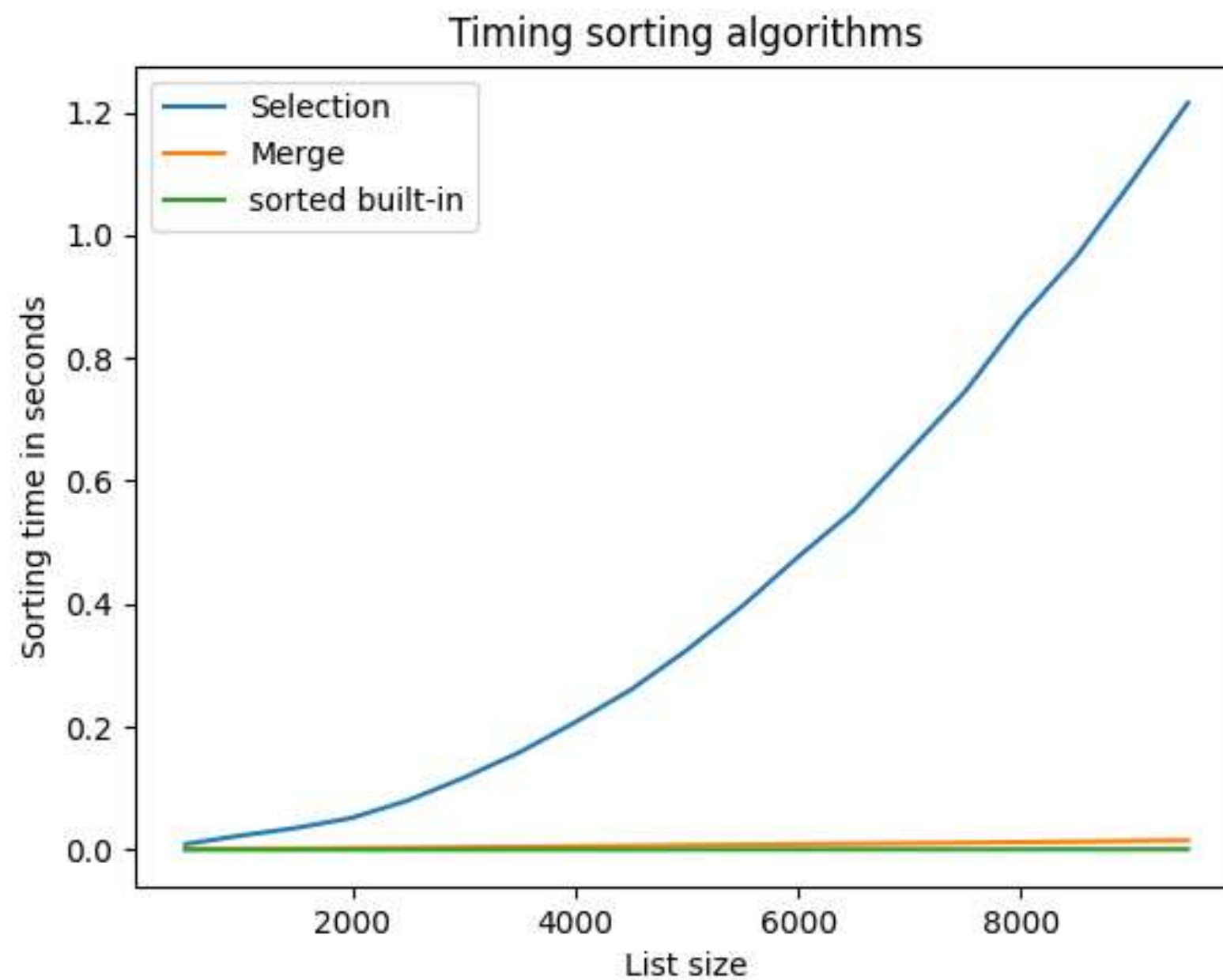


Possible implementation of `merge_sort`.

```
1 def merge_sort(a_list):
2     """
3     Sort list using the merge sort
4     algorithm returning a new sorted list.
5
6     Args:
7         a_list: List to sort
8
9     Returns:
10        New sorted list
11    """
12
13    if len(a_list) <= 1:
14        # Base case: List with single
15        # value is already sorted
16        return a_list
17    else:
18        # Recursive case: Split list in half,
19        # sort each half then merge
20        # the resulting lists
21        mid_index = len(a_list) // 2
22        left = merge_sort(a_list[:mid_index])
23        right = merge_sort(a_list[mid_index:])
24
25        merged = merge(left, right)
26        return merged
```

```
1 def merge(list1, list2):
2     """
3     Return a sorted list produced from merging
4     two sorted lists
5
6     Args:
7         list1, list2: Sorted lists to merge
8
9     Returns:
10        Sorted, merged, list
11    """
12    result = []
13    index1 = 0
14    index2 = 0
15
16    # Iterate each of the lists an item at a time
17    while index1 < len(list1) and index2 < len(list2):
18        if list1[index1] < list2[index2]:
19            # If the current item in list1 is smaller,
20            # copy and advance current item in list1
21            result.append(list1[index1])
22            index1 += 1
23        else:
24            # Otherwise, do the same in list2
25            result.append(list2[index2])
26            index2 += 1
27
28    # Append any remaining elements in list1 or list2.
29    # Only one of these lists should have any remaining
30    # elements, the other slicing operation
31    # will produce an empty list
32    result += list1[index1:]
33    result += list2[index2:]
34
35    return result
```

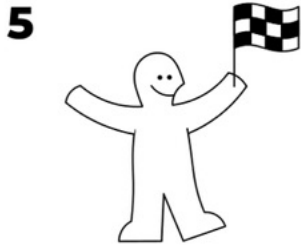
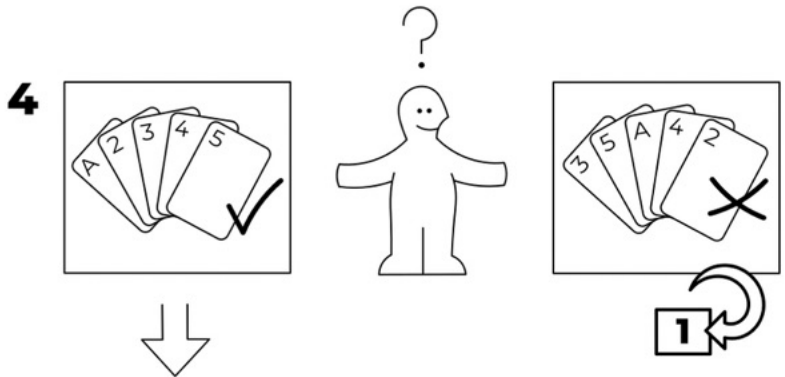
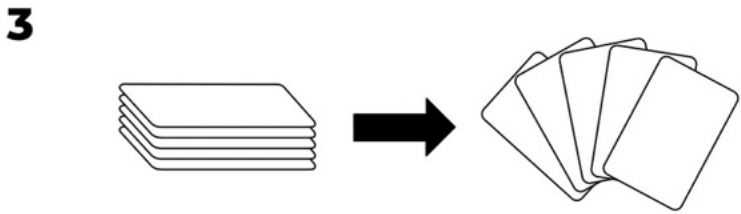
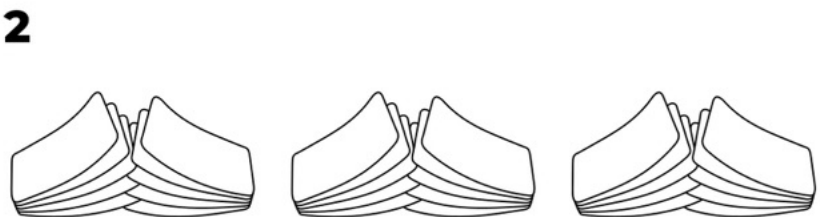
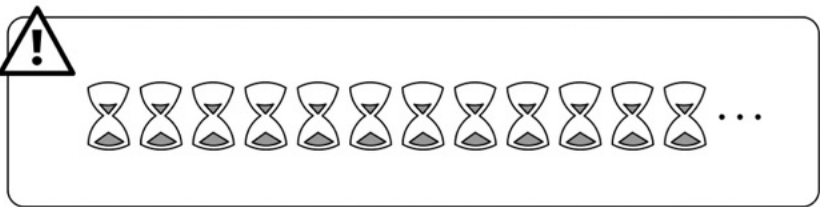
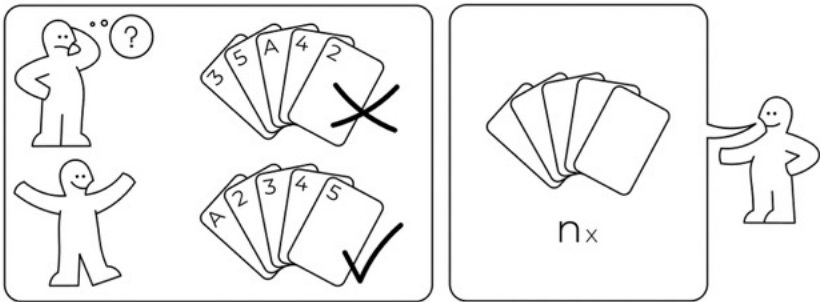
See **sorting.py** (linked in reading) for a performance comparison.



bogo_sort: one of the worst sorting algorithms.

BOGO SÖRT

idea-instructions.com/bogo-sort/
v1.2, CC by-nc-sa 4.0 **IDEA**



$n!$ permutations
of items
 n to check if a list is
sorted

$O(n \times n!)$

Summary and Reminders

- **Linear Search:** best is $O(1)$, worst is $O(n)$, average is $O(n)$.
- **Binary Search:** best is $O(1)$, worst is $O(\log n)$, average is $O(\log n)$.
- **Selection Sort:** best is $O(n^2)$, worst is $O(n^2)$, average is $O(n^2)$.
- **Insertion Sort:** best is $O(n)$, worst is $O(n^2)$, average is $O(n^2)$.
- **Merge Sort:** best is $O(n \log n)$, worst is $O(n \log n)$, average is $O(n \log n)$.
- Programming Assignment 7 initial due date on Thursday.
- Quiz 8 this Friday includes retakes from Quizzes 4 - 7 + new Quiz 8 topics + Midterm 1 retakes of **Questions 6 and Question 7:**
 - Midterm 1 Question 6: "Finding errors"
 - Midterm 1 Question 7: "Writing functions with sequences"
- Only Quiz 8 cheat sheet (linked on calendar) will be allowed for Quiz 8 + Midterm 1 retakes.