Middlebury

CSCI 201: Data Structures

Spring 2025

Lecture 5W: More Sorting

Goals for today:

- Implement the steps in `MergeSort` and analyze the runtime complexity.
- List the steps in `QuickSort` and analyze the runtime complexity.

divide
and
conquer

IDEA

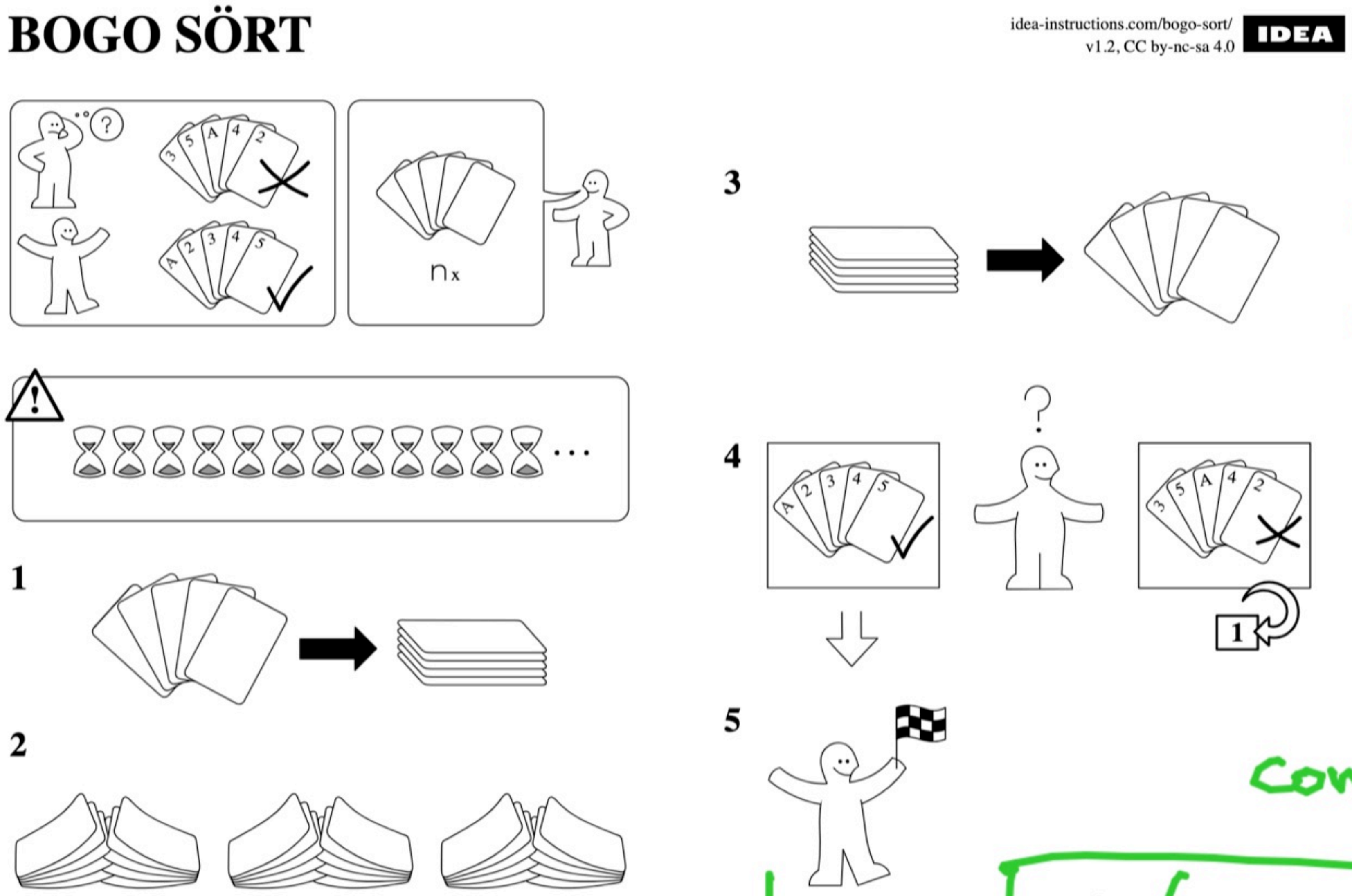
A series of nonverbal algorithm assembly instructions.

https://idea-instructions.com/

which sorting algorithms have we seen so far?

- selection
- insertion
- bucket
- radix

BogoSort: one of the worst sorting algorithms.



n cards shuffle randomly but let's assume we go through all permutations

$n!$

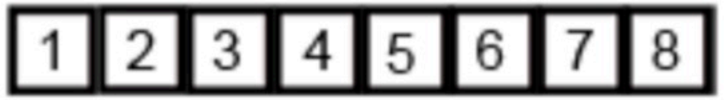
how much work to determine if cards are sorted?

$\sim n$ comparisons

$$\boxed{O(n \cdot n!)}$$

Sorting Algorithm #5: MergeSort.

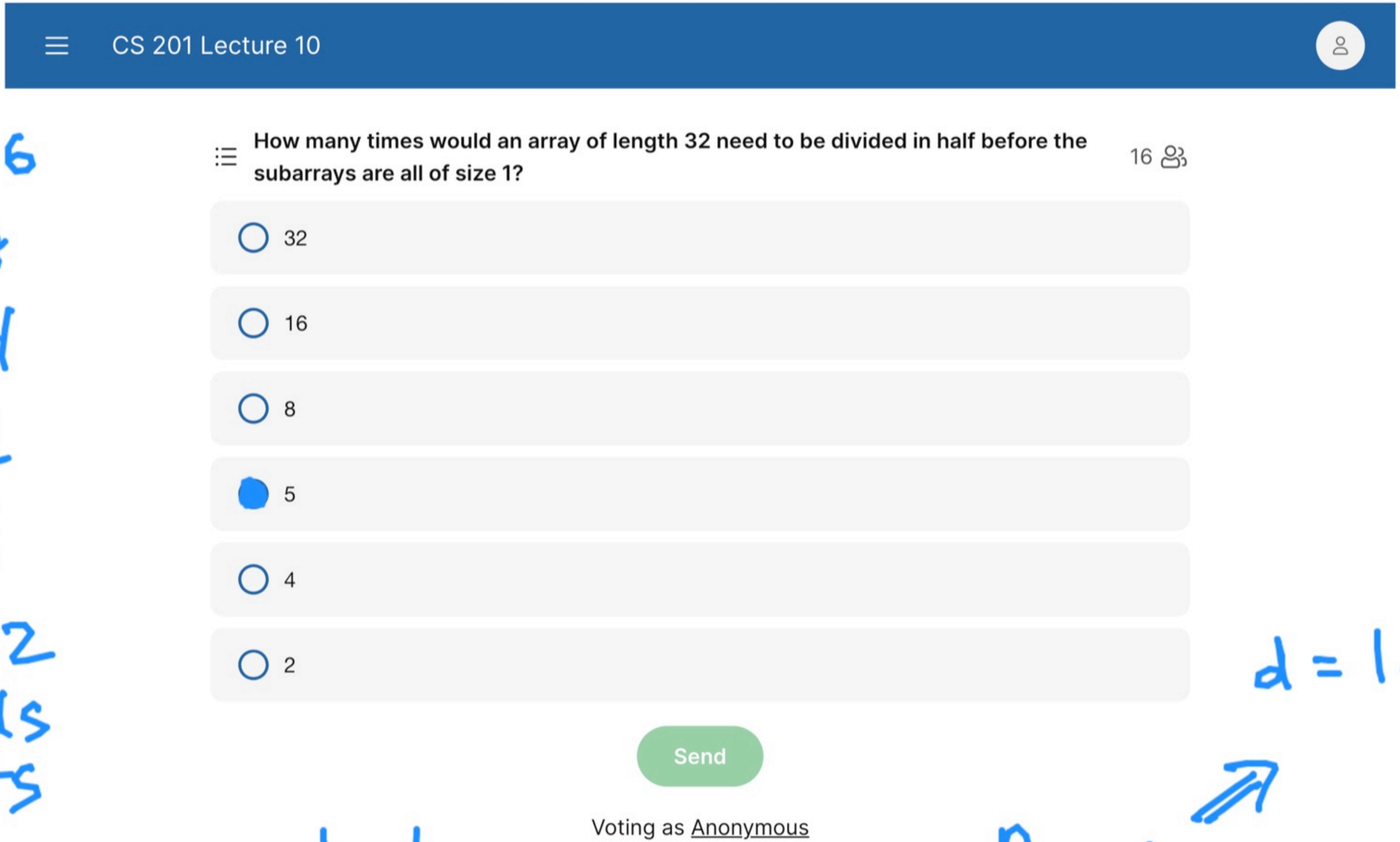
1. Divide input array into two subarrays.
2. Call MergeSort on each subarray.
3. Merge the result. ← today



1	2	3	4	5	6	7	8

(from Wikipedia)

Analyzing MergeSort (Part 1). See slido.com # 1175134



How many times would an array of length 32 need to be divided in half before the subarrays are all of size 1?

- ○ 32
- ○ 16
- ○ 8
- ● 5
- ○ 4
- ○ 2

Voting as Anonymous

$32/2 = 16$

$16/2 = 8$

$8/2 = 4$

$4/2 = 2$

$2/2 = 1$

for $n = 32$
→ 5 levels of splits

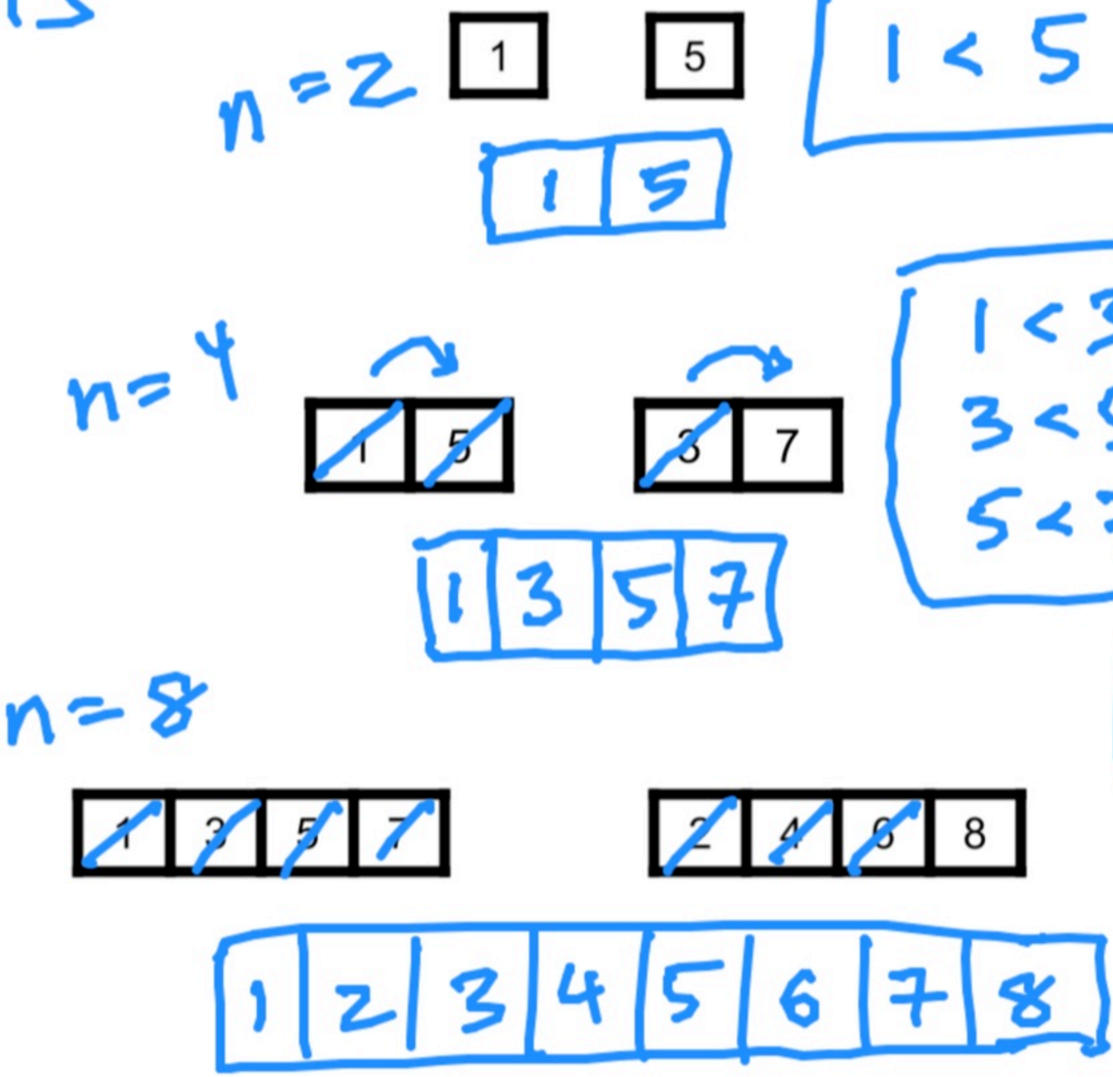
For n : $n \cdot \frac{1}{2} \cdot \frac{1}{2} \cdots \frac{1}{2} = 1 \rightarrow \frac{n}{2^d} = 1$ ⇗ $d = \log_2 n$

Analyzing MergeSort (Part 2).

How many times do you need to ask "which leading element is smallest?" when merging these two subarrays?

comparisons is $n-1$

$O(n)$ work during merge



$1 < 5$

$1 < 3$
$3 < 5$
$5 < 7$

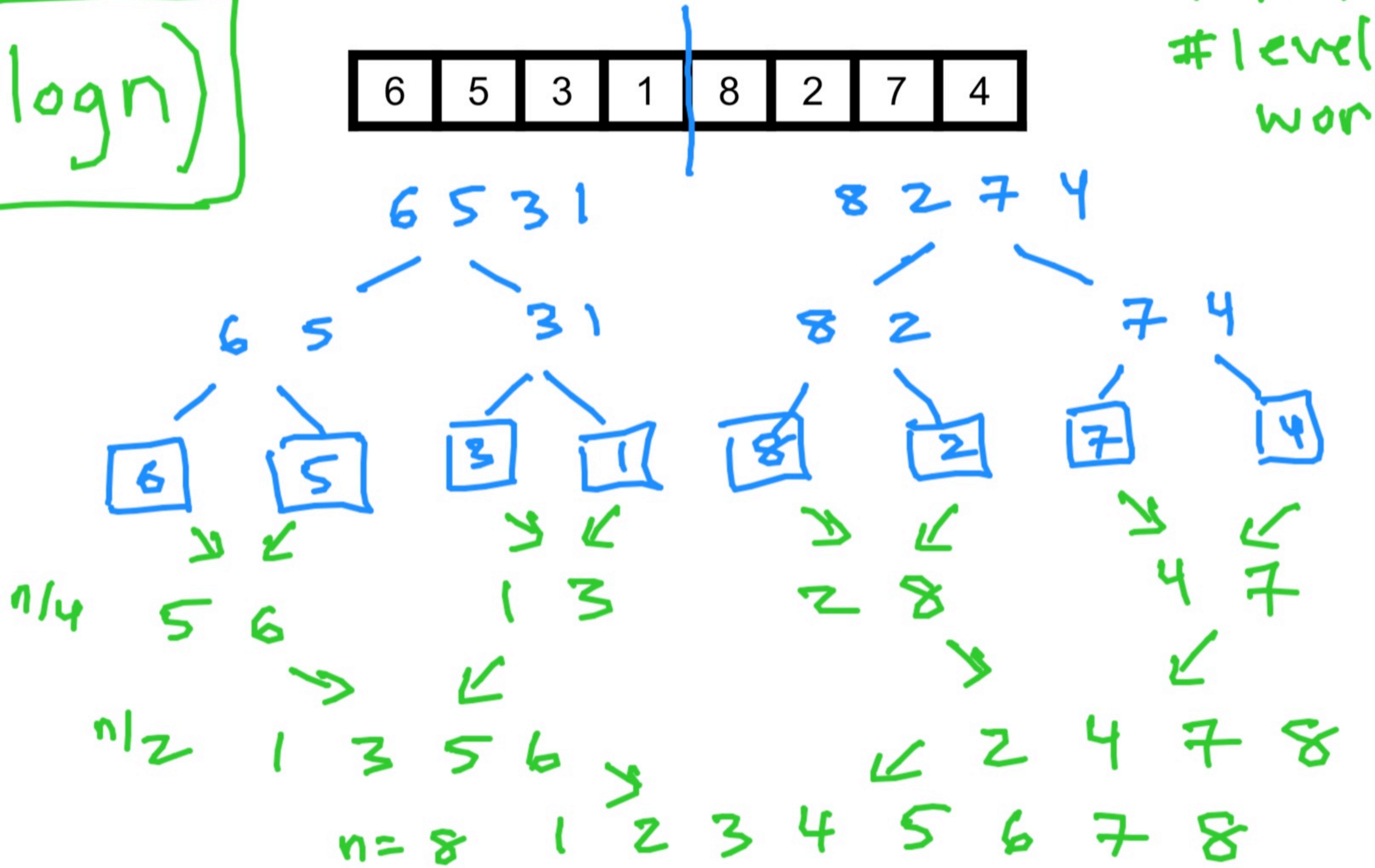
$1 < 2$
$2 < 3$
$3 < 4$
$4 < 5$
$5 < 6$
$6 < 7$
$7 < 8$

comparisons
1
3
7

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

$\boxed{O(n\log n)}$

total work
#levels *
work/level

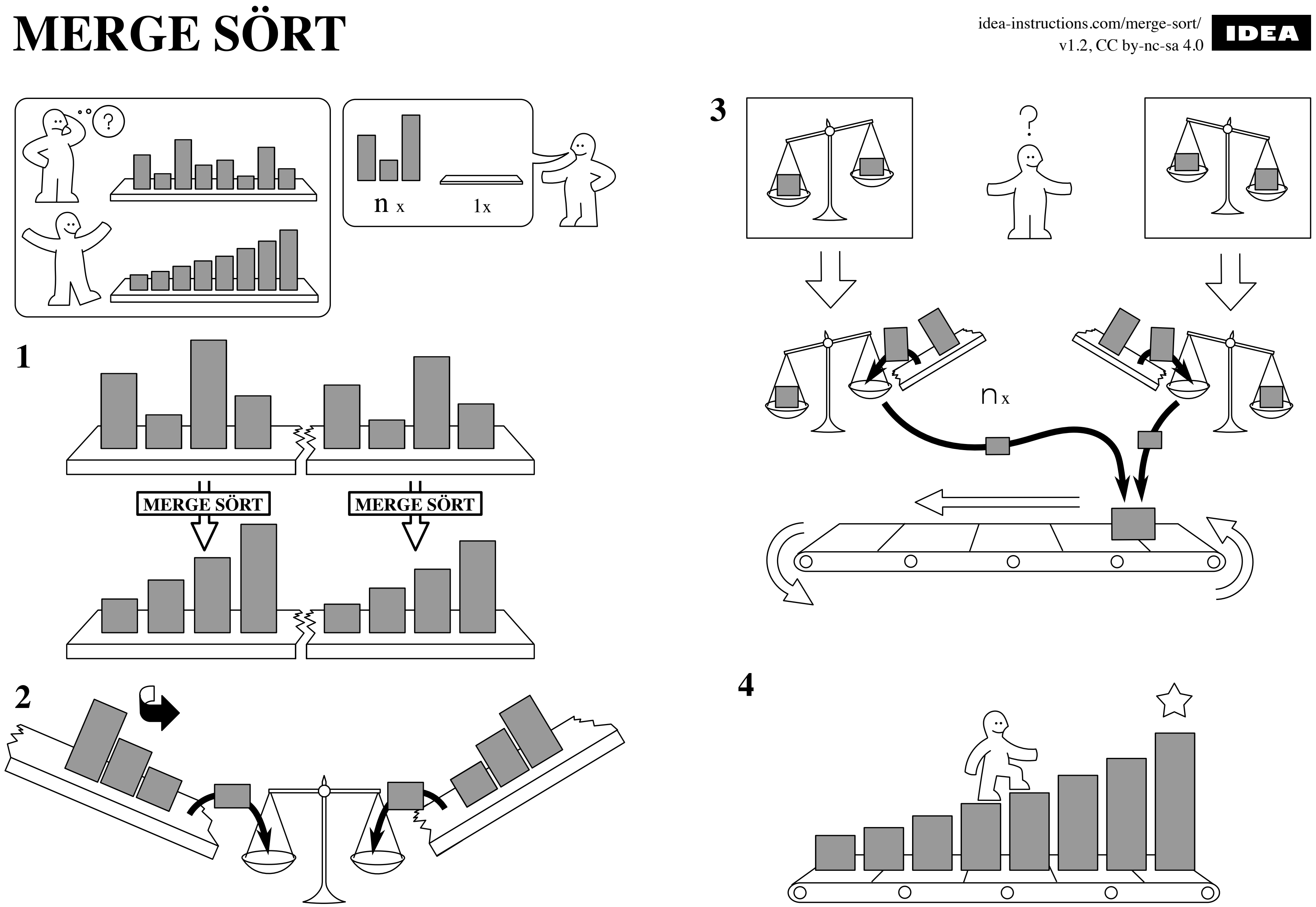


Possible implementation of MergeSort.

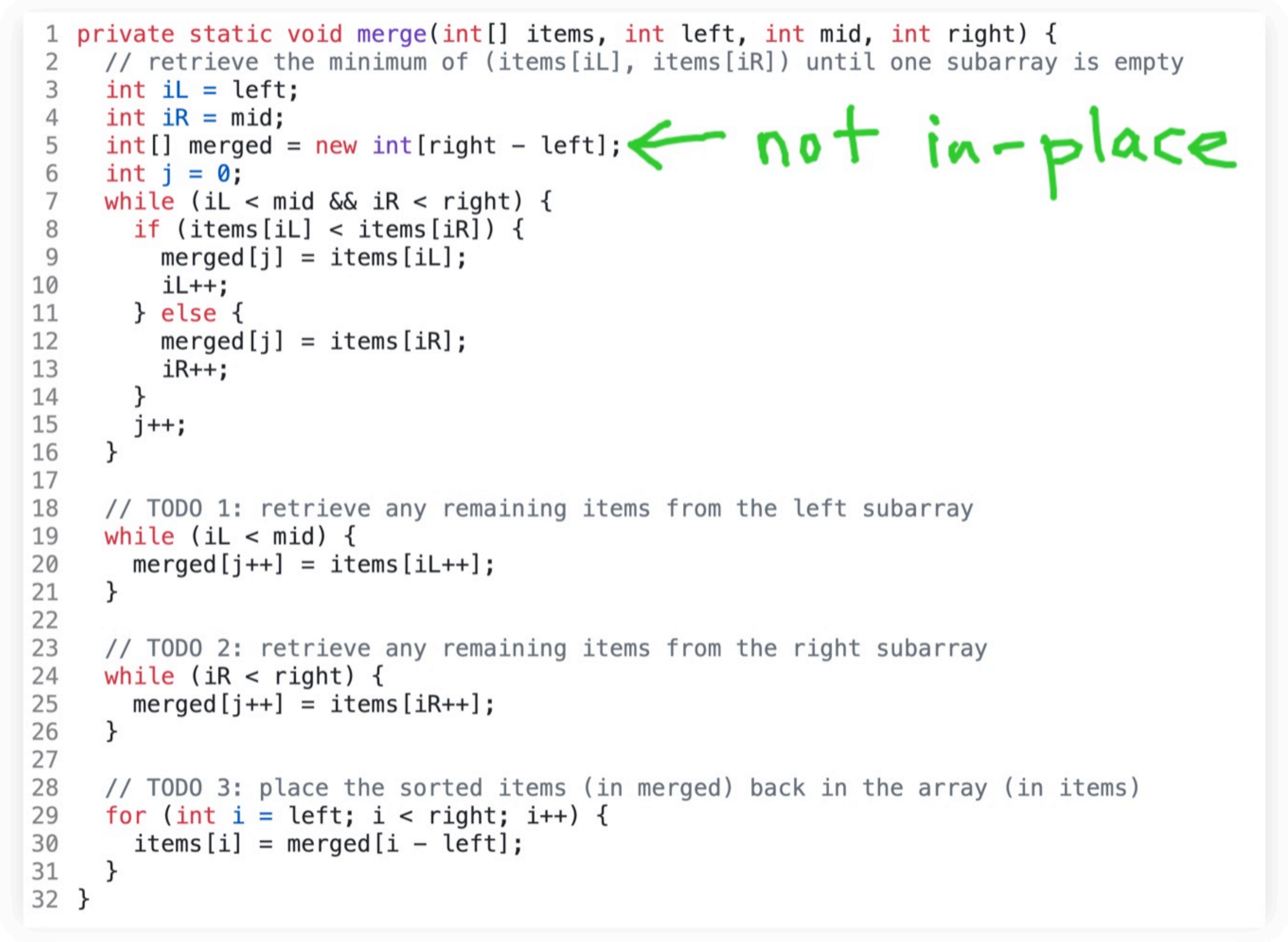
```java
/**
 * Sorts an array of items using merge-sort.
 */
public static void sort(int[] items) {
  mergesortHelper(items, 0, items.length);
}

/**
 * Runs merge-sort on a portion of the items, starting
 * at the "left" item, up to (but not including) the "right" item.
 *
 * @param items - entire array to be sorted.
 * @param left - left index of the subarray to be sorted.
 * @param right - right index of the subarray to be sorted.
 */
private static void mergesortHelper(int[] items, int left, int right) {
  int n = right - left;
  if (n <= 1) return;

  // recursively call merge sort on left and right subarrays
  int mid = left + n / 2;
  mergesortHelper(items, left, mid);
  mergesortHelper(items, mid, right);

  // merge the two subarrays
  merge(items, left, mid, right);
}
```

Exercise: complete the `merge` step in `MergeSorter.java`.

Study the existing code and see the `TODO` comments for what steps are missing.



Possible implementation of merge.

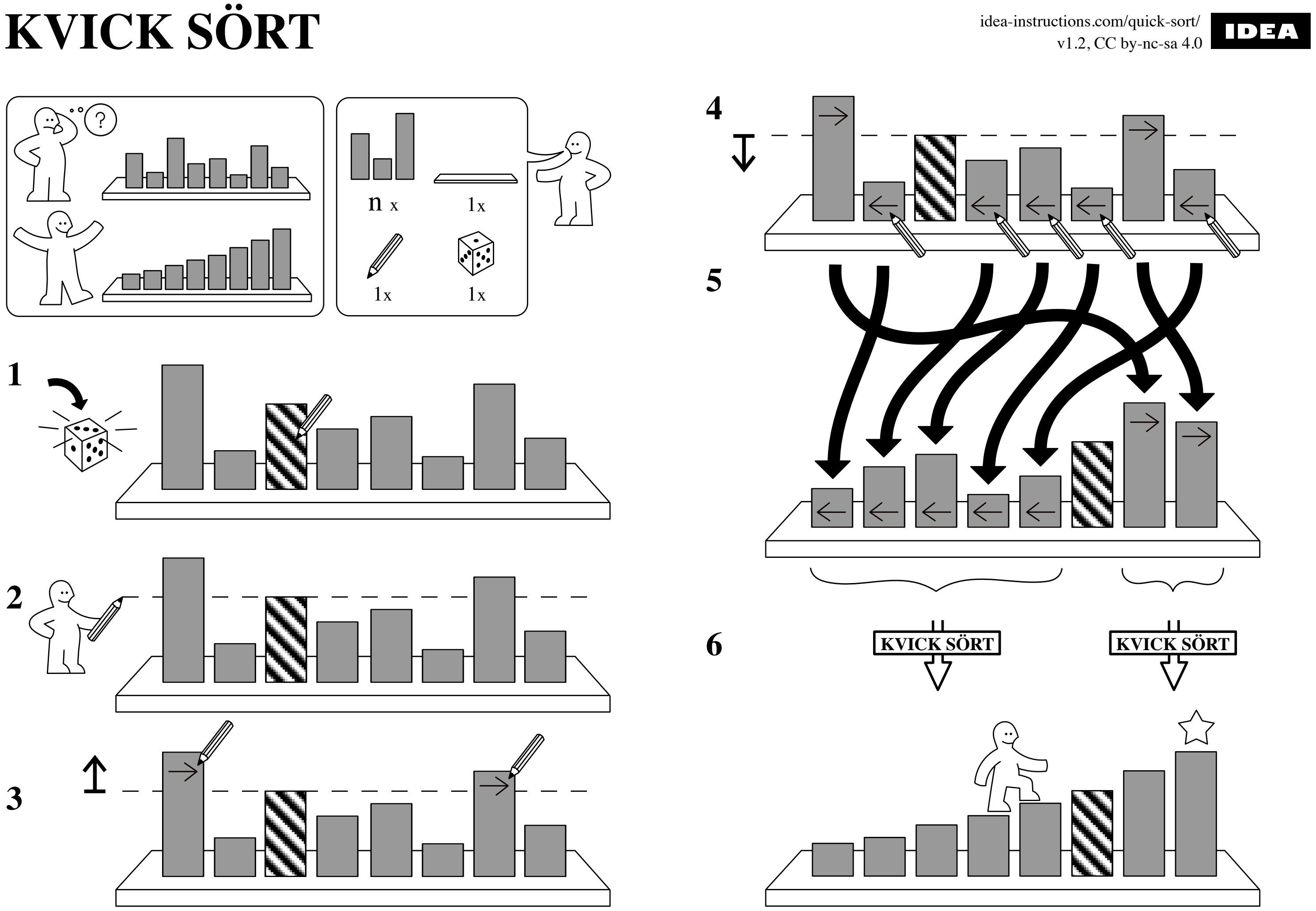


```java
private static void merge(int[] items, int left, int mid, int right) {
  // retrieve the minimum of (items[iL], items[iR]) until one subarray is empty
  int iL = left;
  int iR = mid;
  int[] merged = new int[right - left];
  int j = 0;
  while (iL < mid && iR < right) {
    if (items[iL] < items[iR]) {
      merged[j] = items[iL];
      iL++;
    } else {
      merged[j] = items[iR];
      iR++;
    }
    j++;
  }

  // TODO 1: retrieve any remaining items from the left subarray
  while (iL < mid) {
    merged[j++] = items[iL++];
  }

  // TODO 2: retrieve any remaining items from the right subarray
  while (iR < right) {
    merged[j++] = items[iR++];
  }

  // TODO 3: place the sorted items (in merged) back in the array (in items)
  for (int i = left; i < right; i++) {
    items[i] = merged[i - left];
  }
}
```

← not in-place

Sorting algorithm #6: `QuickSort`.

1. Pick a pivot.
2. Partition items so than any item < pivot is in left subarray and any item > pivot is in the right subarray.
3. Call `QuickSort` on each subarray.



Sorting algorithm #6: QuickSort.

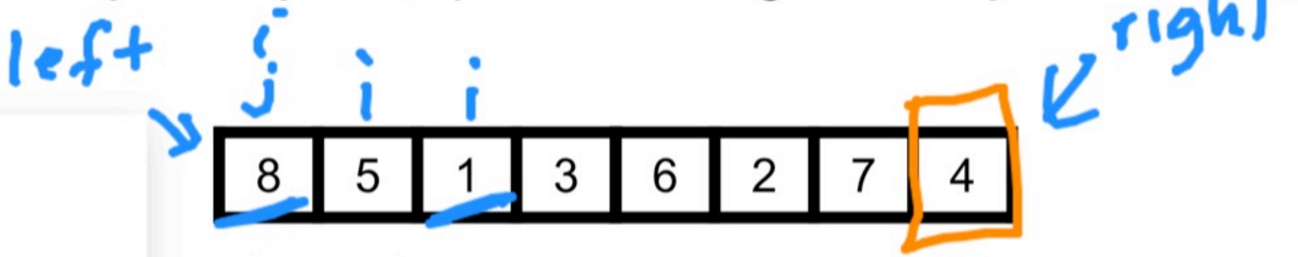
best case: O(n log n)

worst case: $O(n^2)$

data already sorted

1. Pick a pivot. let's pick last element as pivot
2. Partition items so than any item < pivot is in left subarray and any item > pivot is in the right subarray.
3. Call QuickSort on each subarray.

```java
public static void sort(int[] items) {
  quicksortHelper(items, 0, items.length - 1);
}

private static void quicksortHelper(int[] items,
                                    int left, int right) {
  if (left >= right) return;

  // pick a pivot and partition items to the left/right
  int p = partition(items, left, right);

  // call quicksort on left and right subarrays
  quicksortHelper(items, left, p - 1);
  quicksortHelper(items, p + 1, right);
}
```



8 5 1 3 6 2 7 4

is items[i] < pivot ?

if no → no swap i++

if yes → swap items[i] with items[j]

i++, j++

```
  j       i
1 5   8   3   6   2   7 4

      j       i   i
1 3   8   5   6   2   7 4

          j           i i
1 3   2   5   6   8   7 4
```

last step: swap items[j] with pivot

1 3 2 4 6 8 7 5

See you Friday -- or, after Spring Break!

- Friday class meeting is optional; we will work on Homework 5
- Note that Lab 5 will be after Spring Break and due after Homework 5!
- Reminder that Noah (go/noah) and Smith (go/smith) have office hours throughout the week and the 201 Course Assistants have drop-in hours in the late afternoons/evenings (go/cshelp). No drop-in sessions on Sunday March 16.