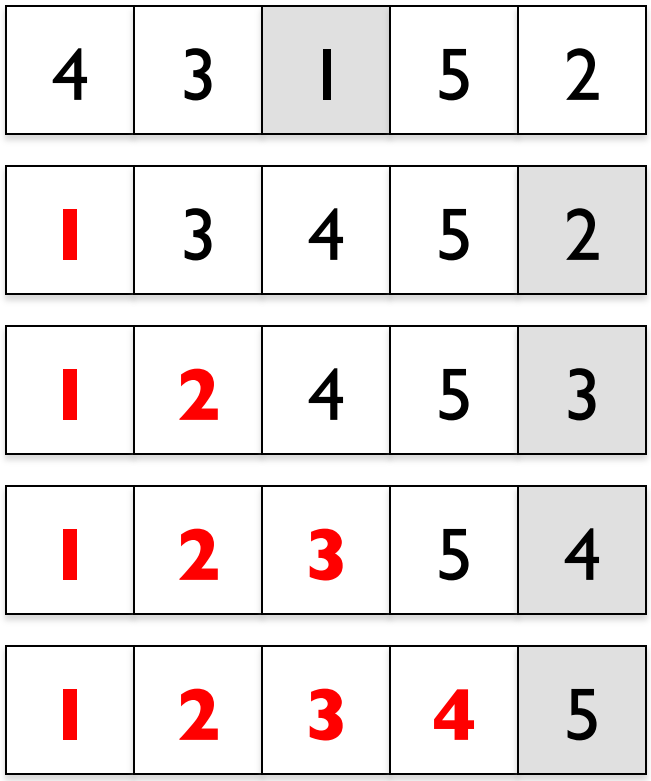
CS32 Discussion
Section 1B
Week 8

TA: Zhou Ren

Sorting Algorithms

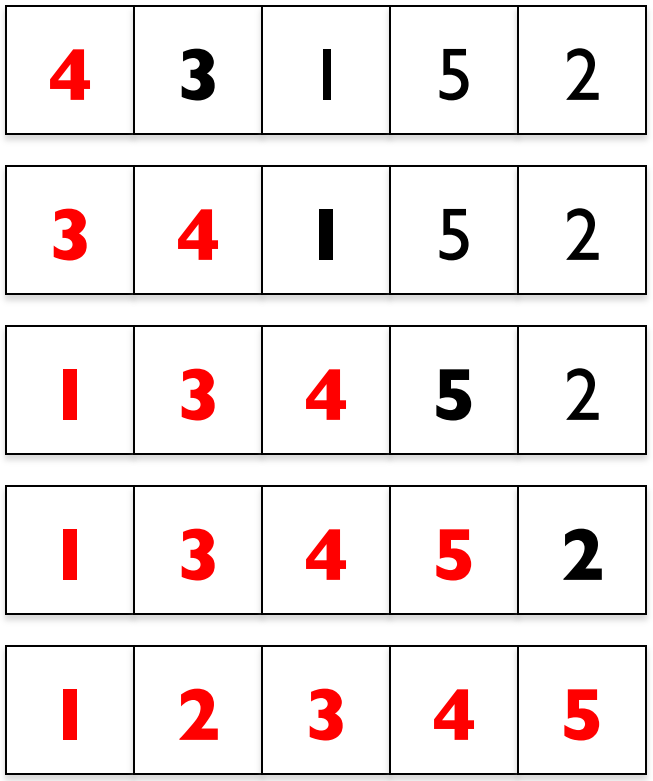
- We now switch gears and discuss some well known sorting algorithms.

Selection Sort



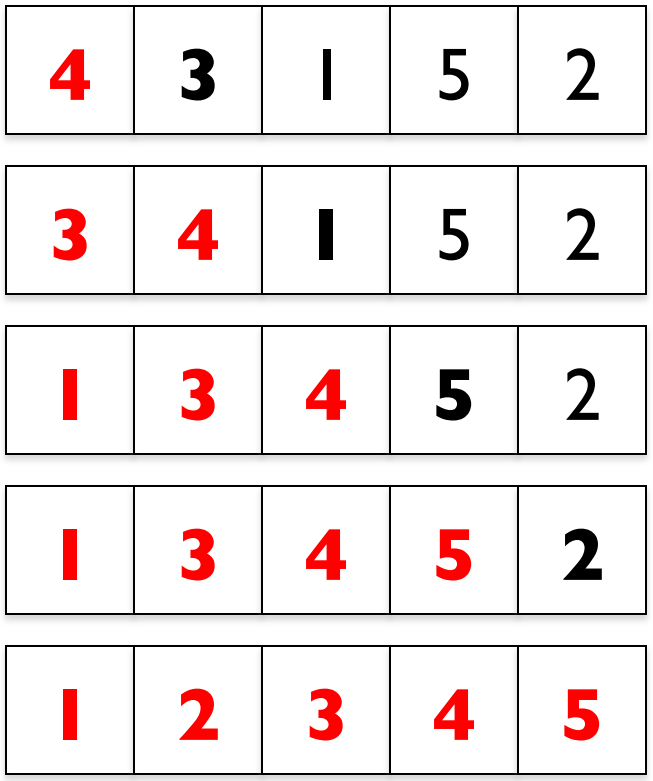
- Find the smallest item in the unsorted portion, and place it in front.
- What is the running time (complexity) of this algorithm?

Insertion Sort



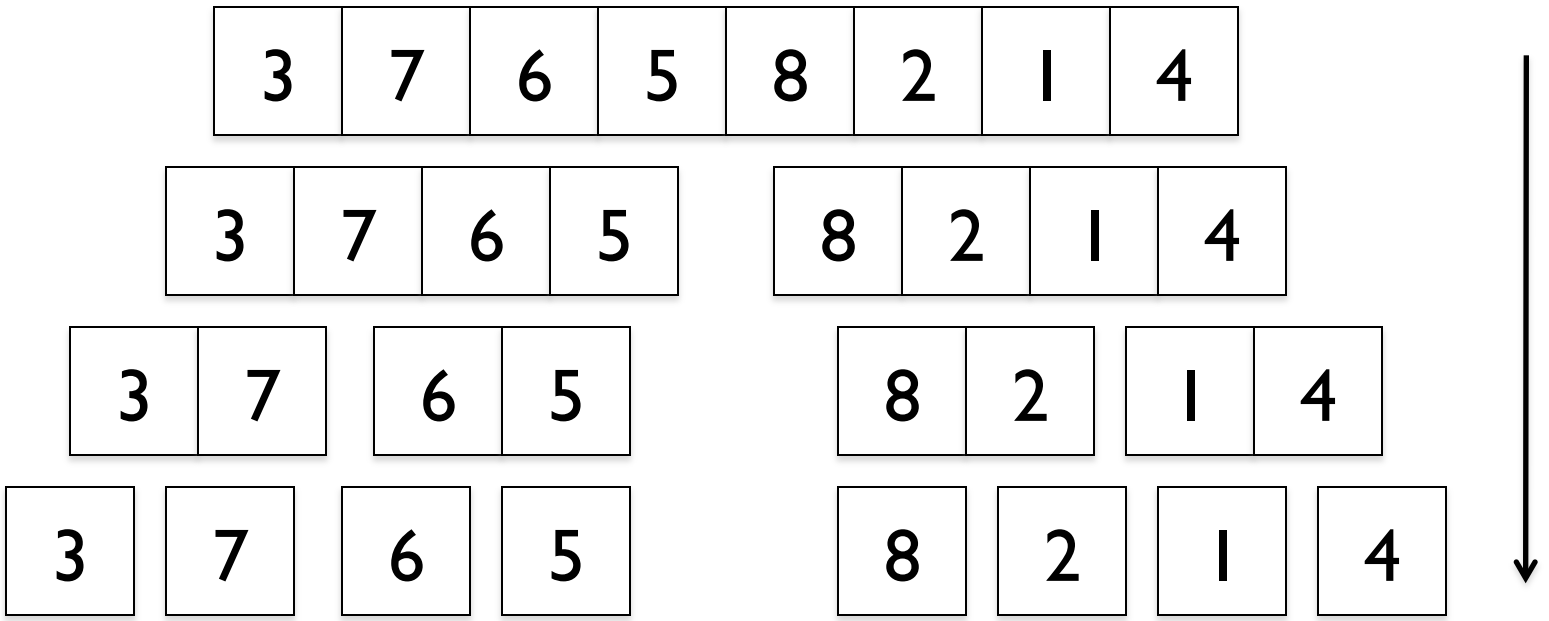
- Pick one from the unsorted part, and place it in the "right" position in the sorted part.
- Best case?
- Avg. case?
- Worst case?

Insertion Sort



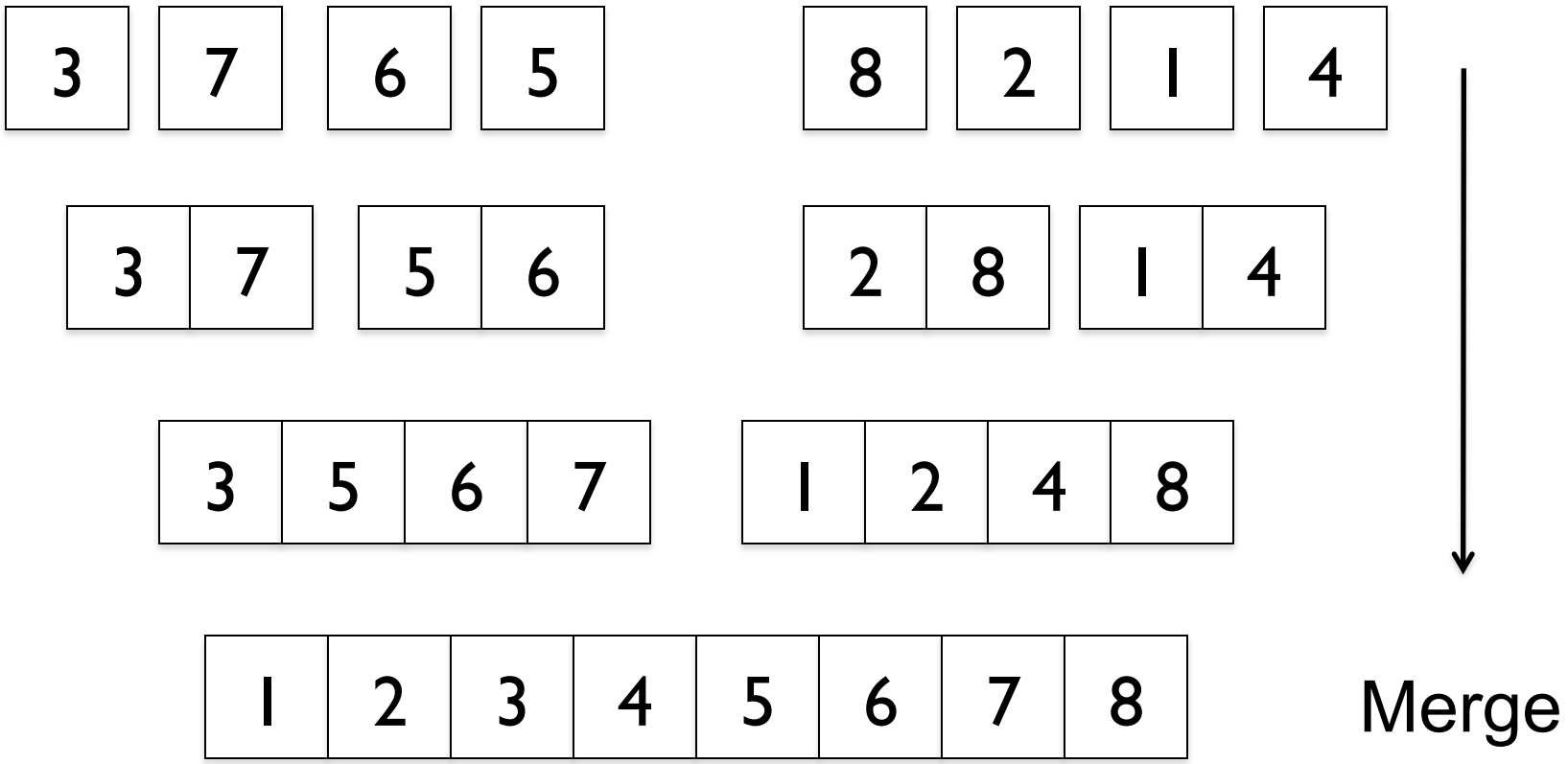
- Pick one from the unsorted part, and place it in the "right" position in the sorted part.
- Best case? **O(n)**
- Avg. case? **$O(n^2)$**
- Worst case? **$O(n^2)$**

Merge Sort

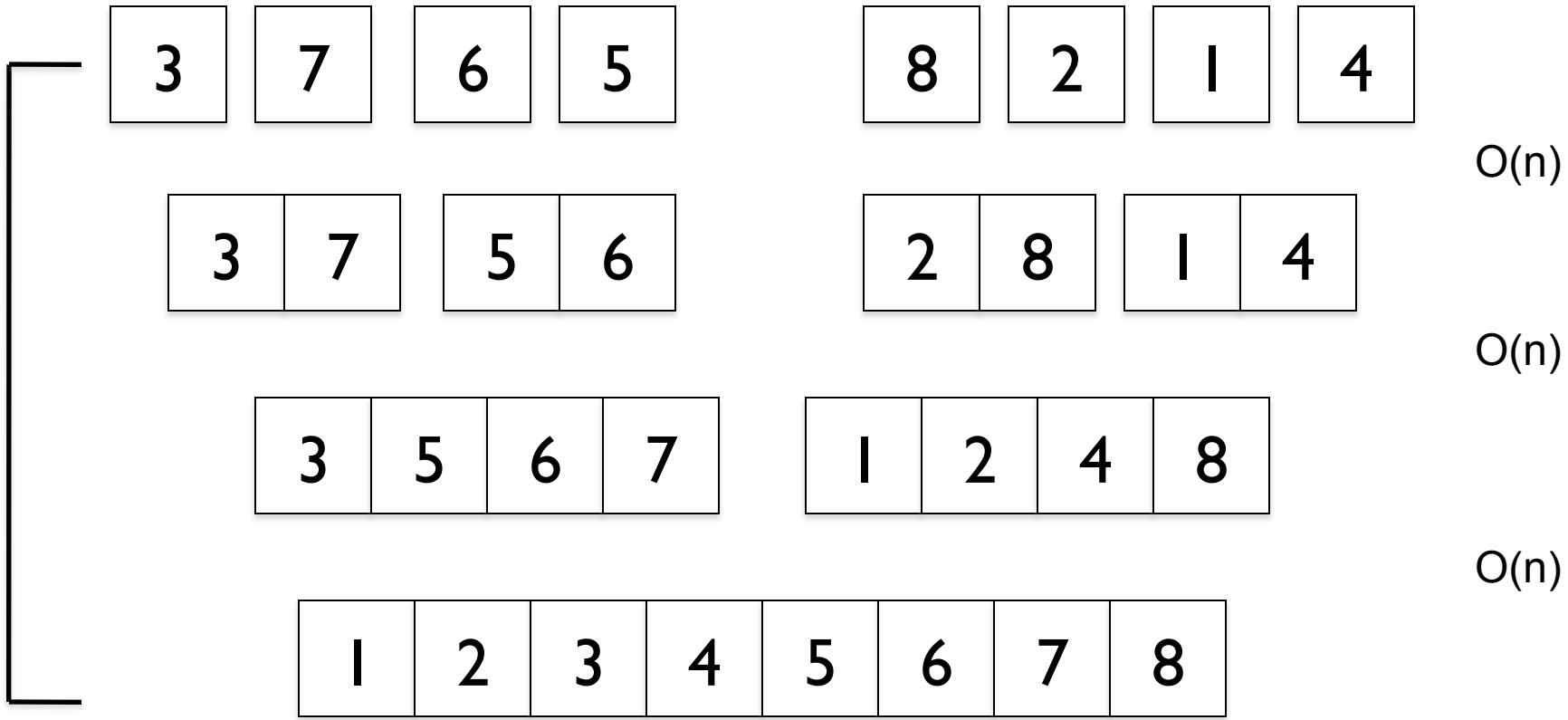


Keep splitting

Merge Sort



Merge Sort: Running Time?

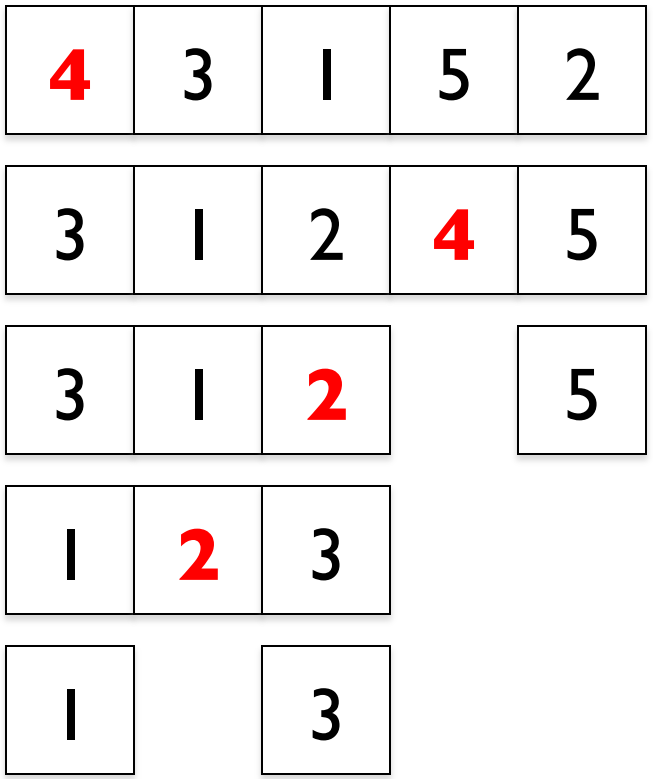


O(n)O(log n) = **O(n log n)**

General Sorting: Running Time

- O(n log n) is faster than $O(n^2)$ – merge sort is more efficient than selection sort or insertion sort.
- O(n log n) is the best average complexity that a general (comparison) sorting algorithm can get (assuming you know nothing about the data set).
- With more information about the data set provided, you can sometimes sort things almost linearly.

Quick Sort



- Pick a **pivot**, and move numbers that are less than the pivot to front, and ones that are greater than the pivot to end. (Does this sound familiar?)
- On average, O(n log n)
- Depending on how you pick your pivots, it can be as bad as $O(n^2)$

Quick Questions

- Given an unsorted array of n items, what is the best you can do to search for an item, if you are to run this search only once?
- Given an unsorted array of n items, what is the best you can do to search for an item, if you are to run this search 100 times? (assume: n >> 100)
- Given an unsorted array of n items, what is the best you can do to search for an item, if you are to run this search n times?