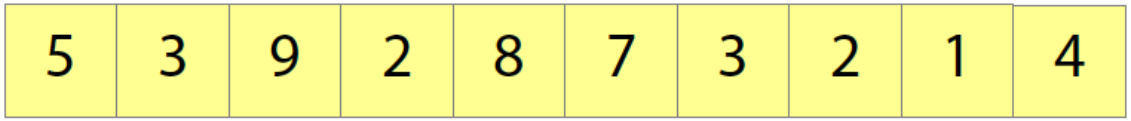
DIT181: Data Structures and Algorithms

Binary Search and Sorting Algorithms

Gül Calikli
Email: calikli@chalmers.se

Binary Search

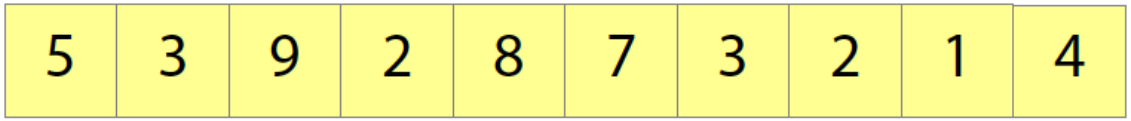
- Suppose you are supposed to find “4”, in the following array:



- One possible way would be *linear search* → look at each element in the array.

Binary Search

- Suppose you are supposed to find “4”, in the following array:

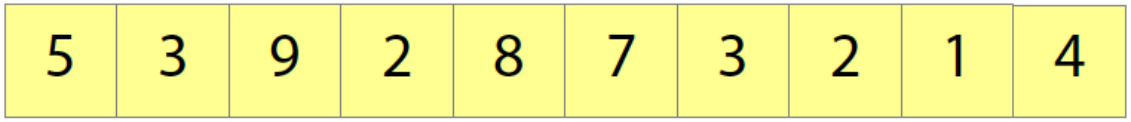


- One possible way would be *linear search* → look at each element in the array.

- In-Class Exercise 3.1: What is the complexity of linear search? (Use Big-O notation)

Binary Search

- Suppose you are supposed to find “4”, in the following array:



- One possible way would be *linear search* → look at each element in the array.

- In- Class Exercise 3.2: What is the complexity of linear search? (Use Big-O notation)
- Answer: In the worst case, you may have to look at every element in array. Hence, it is O(n)

Binary Search

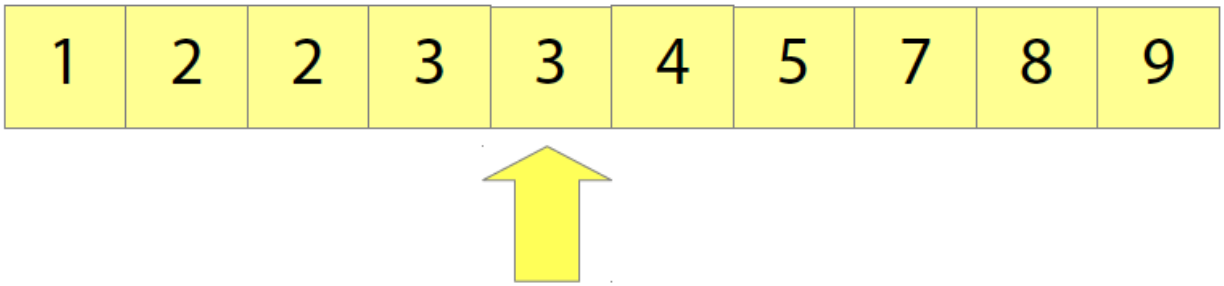
- But what if the array is sorted?

1	2	2	3	3	4	5	7	8	9

- Then, we can use binary search

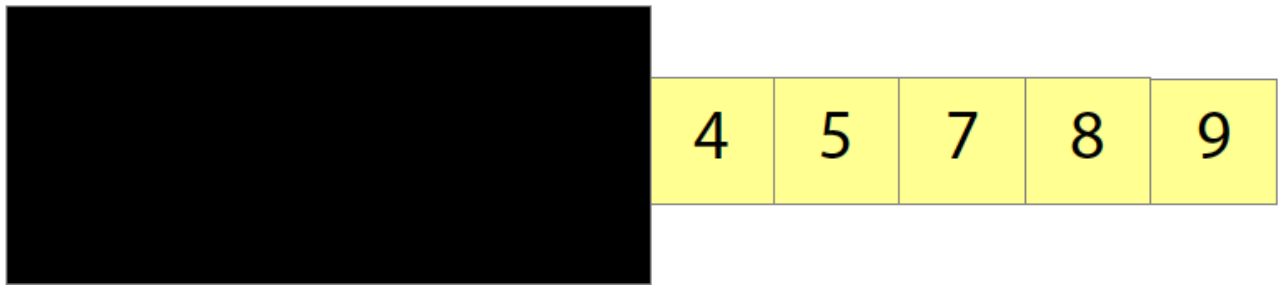
Binary Search

- Suppose we want to look for **4**.
- We start by looking at the element half way along the array, which happens to be 3.



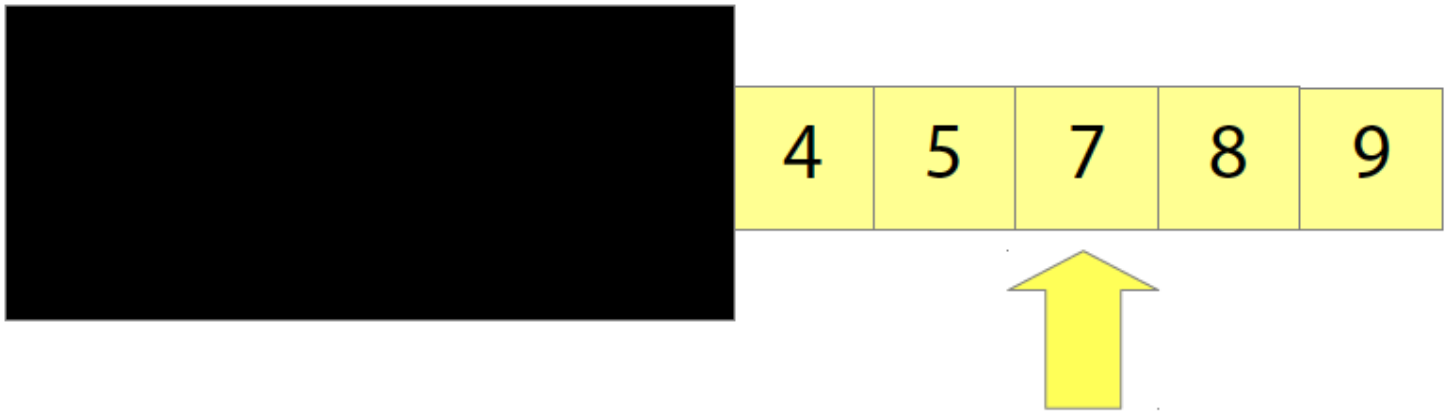
Binary Search

- 3 is less than 4.
- Since the array is sorted, we know that 4 must come after 3.
- We can ignore everything before 3.



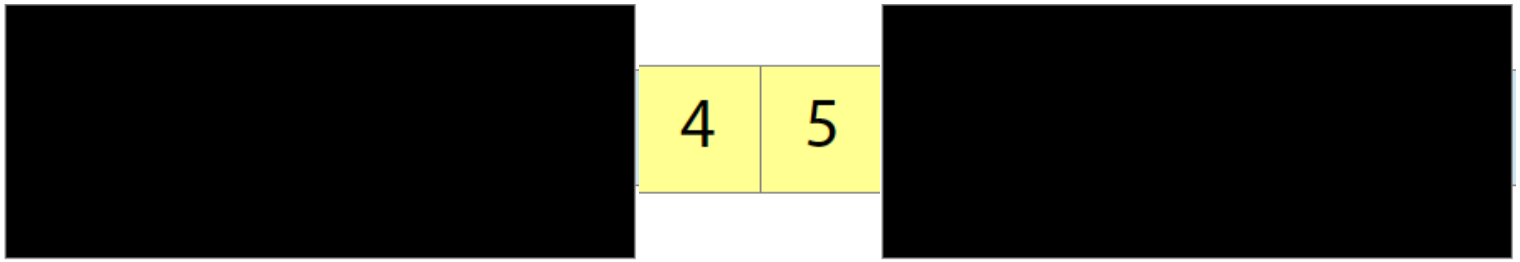
Binary Search

- Now we repeat the process.
- We look at the element half way along what's left of the array. This happens to be 7.



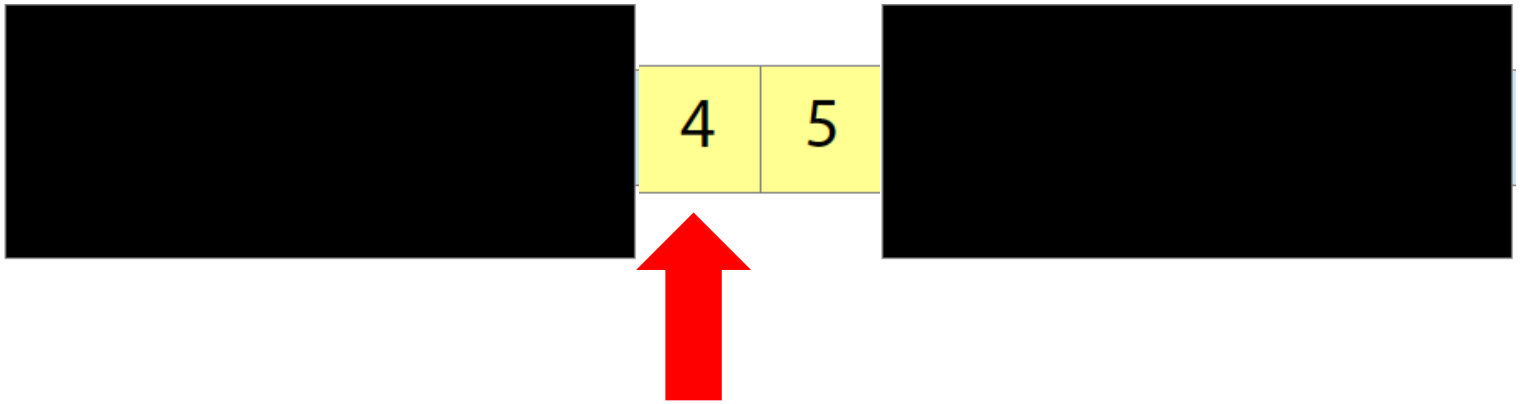
Binary Search

- 7 is greater than 4.
- Since the array is sorted, we know that 4 must come before 7, we can ignore everything after 7.



Binary Search

- We repeat the process.
- We look half way along the array again.
- We find **4!**



Binary Search

- In- Class Exercise 3.2: What is the complexity of binary search? (Use Big-O notation)

Binary Search

- In- Class Exercise 3.2: What is the complexity of binary search? (Use Big-O notation)
- **Hint:** With an array of size 2^n, we are down to 1 element after n steps.

Binary Search

- In- Class Exercise 3.2: What is the complexity of binary search? (Use Big-O notation)
- **Hint:** With an array of size 2^n, we are down to 1 element after n steps.

search 1: n elements in search space
search 2: $n/2$ elements in search space
search 3: $n/4$ elements in search space
...
search i: 1 element in search space.

Binary Search

- In- Class Exercise 3.2: What is the complexity of binary search? (Use Big-O notation)
- **Hint:** With an array of size 2^n, we are down to 1 element after n steps.

search 1: n elements in search space

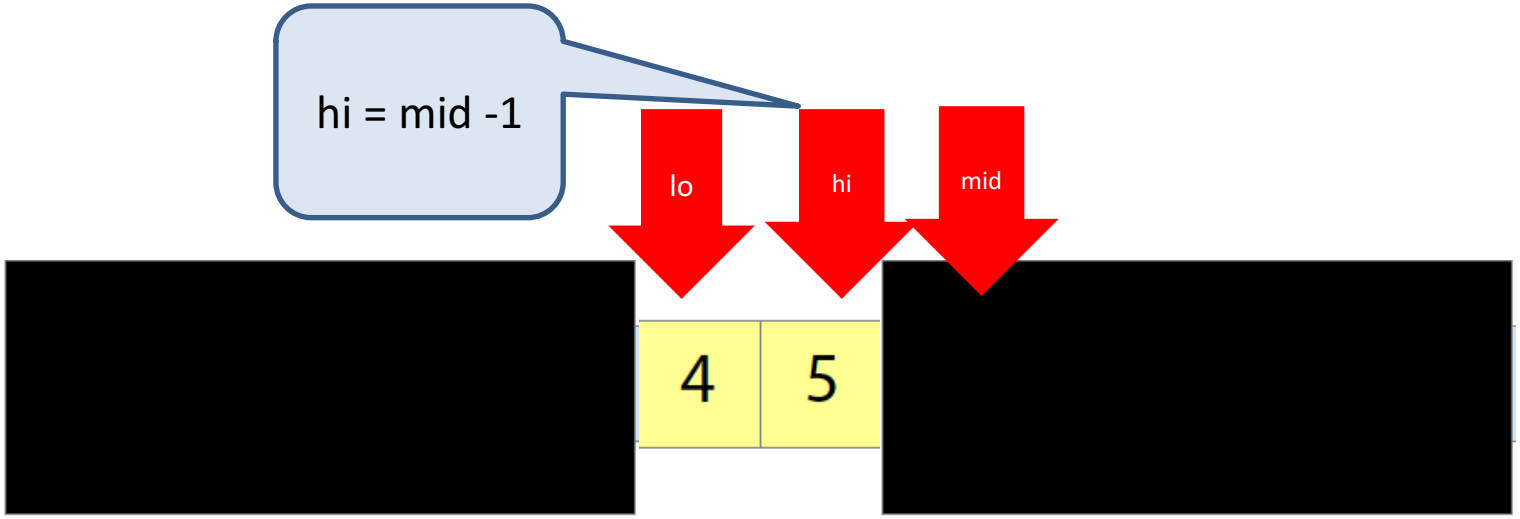
search 2: $n/2$ elements in search space
search 3: $n/4$ elements in search space
...
search i: 1 element in search space.

search i has $n/2^{(i-1)}$elements, and $\frac{n}{2^{(i-1)}} = 1$ so that $n = 2^{(i-1)}$

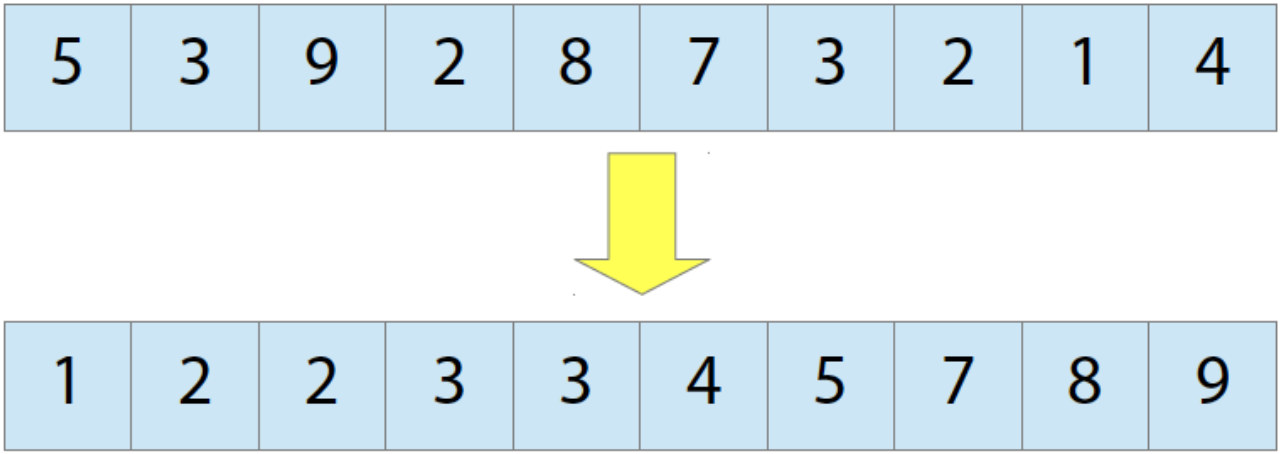
you solve for i then you get $i = log(n) + 1$. Hence, complexity is $\boldsymbol{O(\log n)}$

Implementing Binary Search

- Keep two indices lo and hi. They represent the part of the array to search.
- Let `mid = (lo + hi) / 2` and look at `a[mid]` – then either set `lo = mid+1`, or `hi = mid - 1` depending on the value of `a[mid]`



Sorting



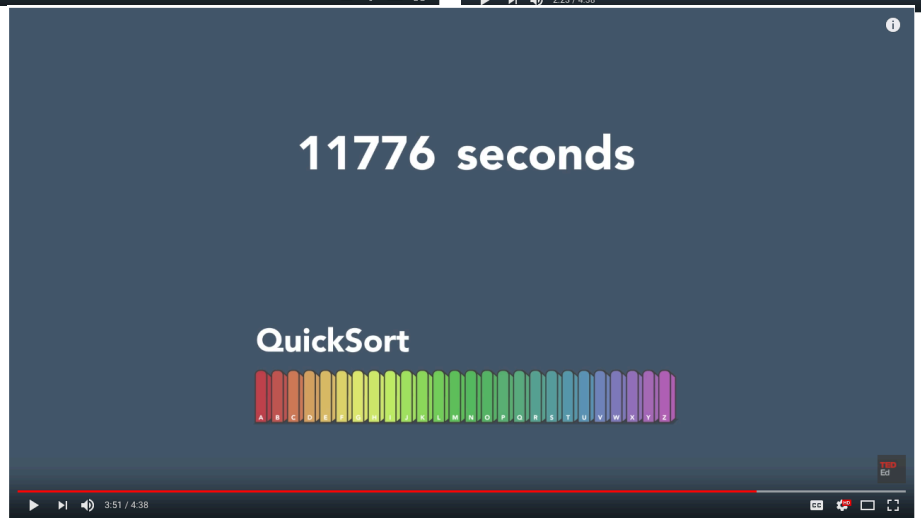
Why is sorting important?

- Because sorted data is much easier to deal with!

Searching

Finding Duplicates

Tutorial video



Sorting Algorithms

- There are many sorting algorithms. In this lecture, we will cover the following:
  - Bubble sort
  - Insertion sort
  - Mergesort
  - Quicksort (just a quick look!)

Sorting Algorithms

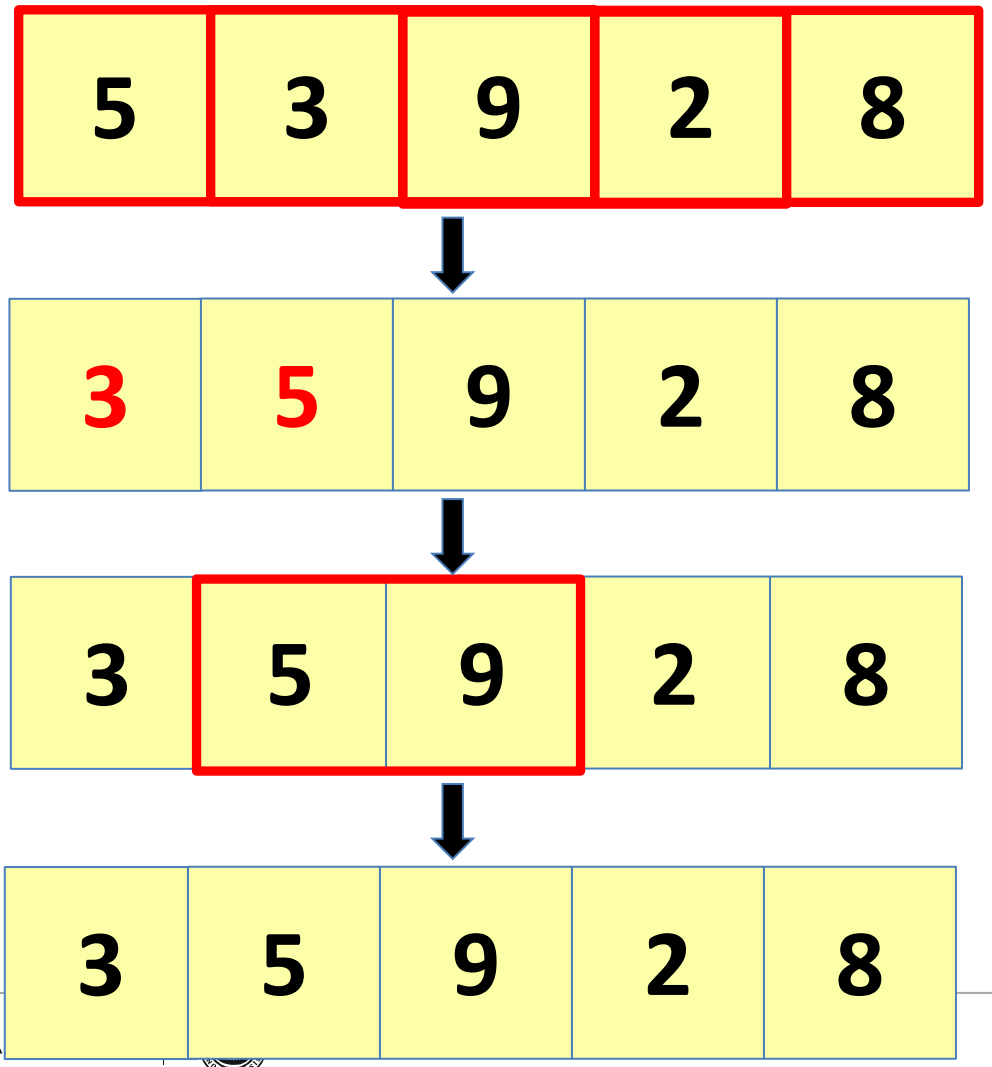
- There are many sorting algorithms. In this lecture, we will cover the following:
  - Bubble sort
  - Insertion sort
  - Mergesort
  - Quicksort (just a quick look!)

Bubble Sort

- Simplest sorting algorithm
- Works by repeatedly:
  - comparing adjacent items, and
  - swapping them if they are in wrong order.

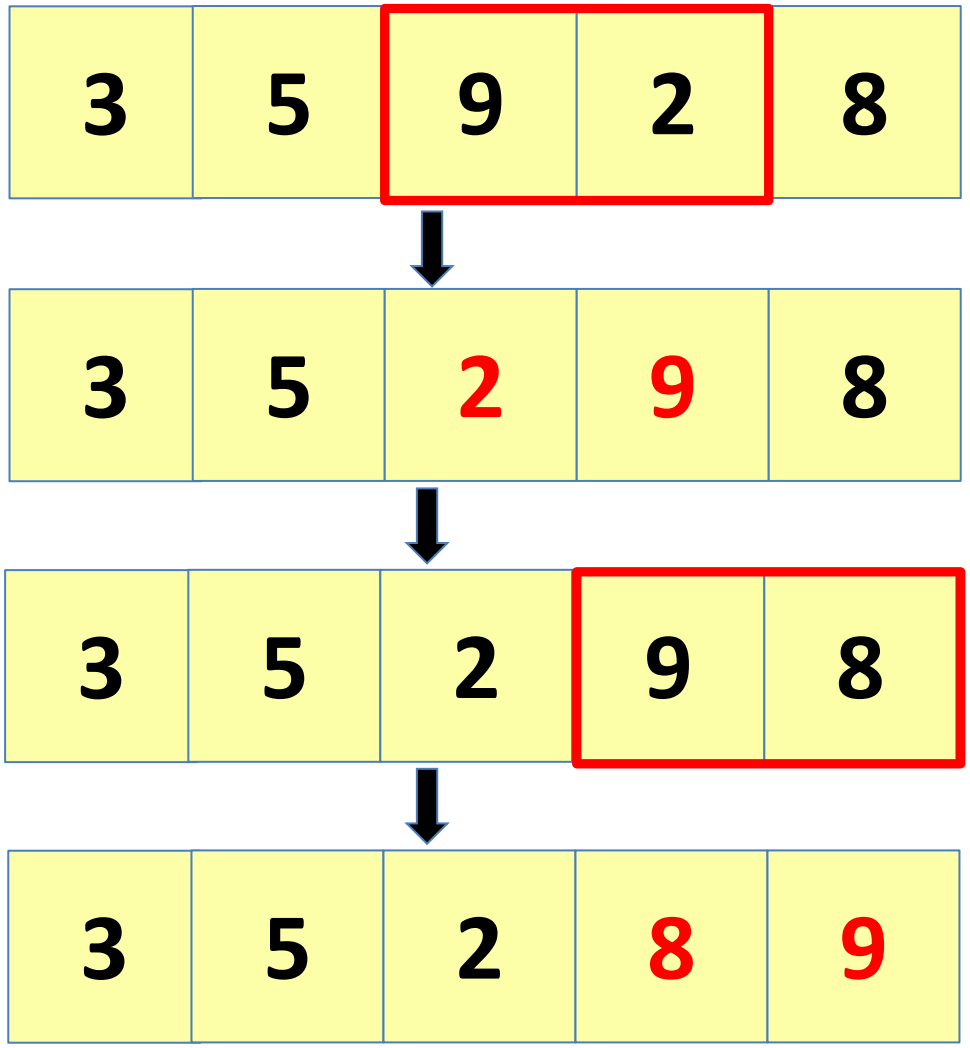
Bubble Sort (Example)

- We'll sort the following in ascending order

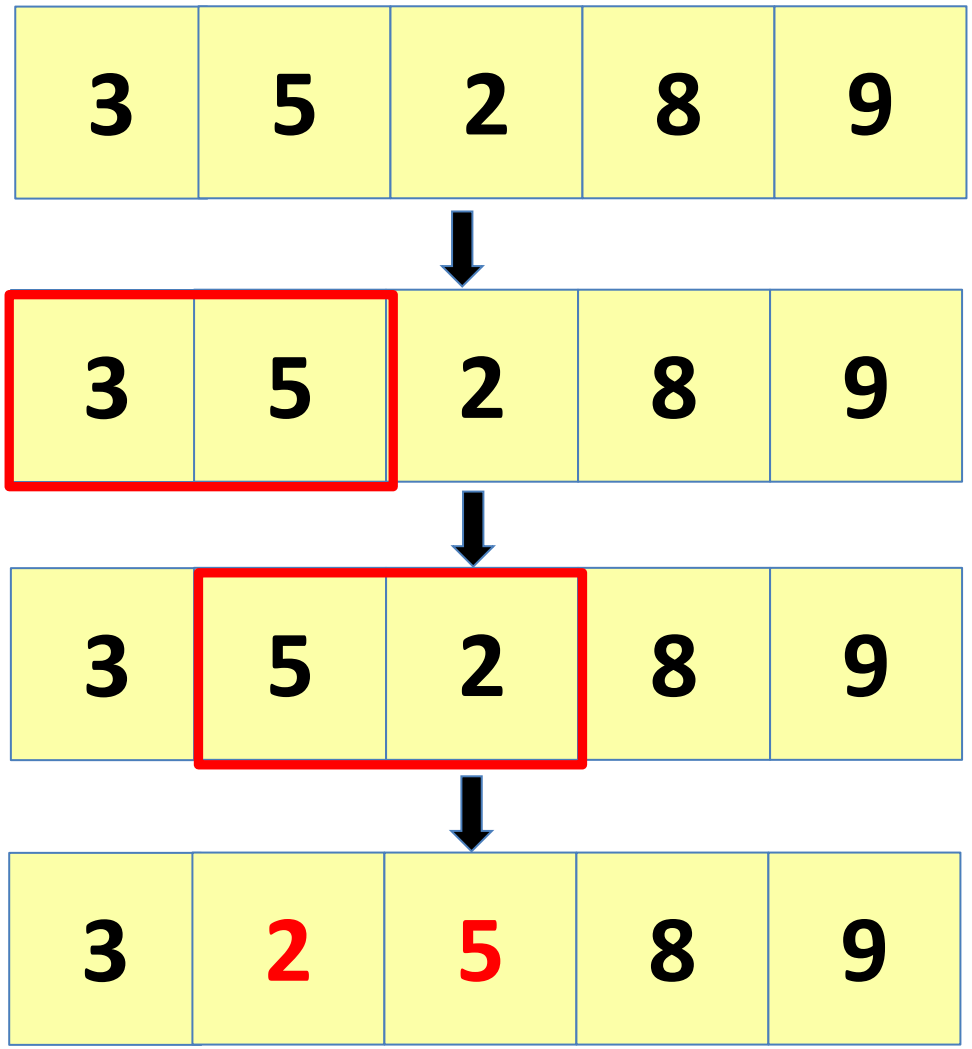


We'll **compare each adjacent pair** from left to right

Bubble Sort (Example)



Bubble Sort (Example)

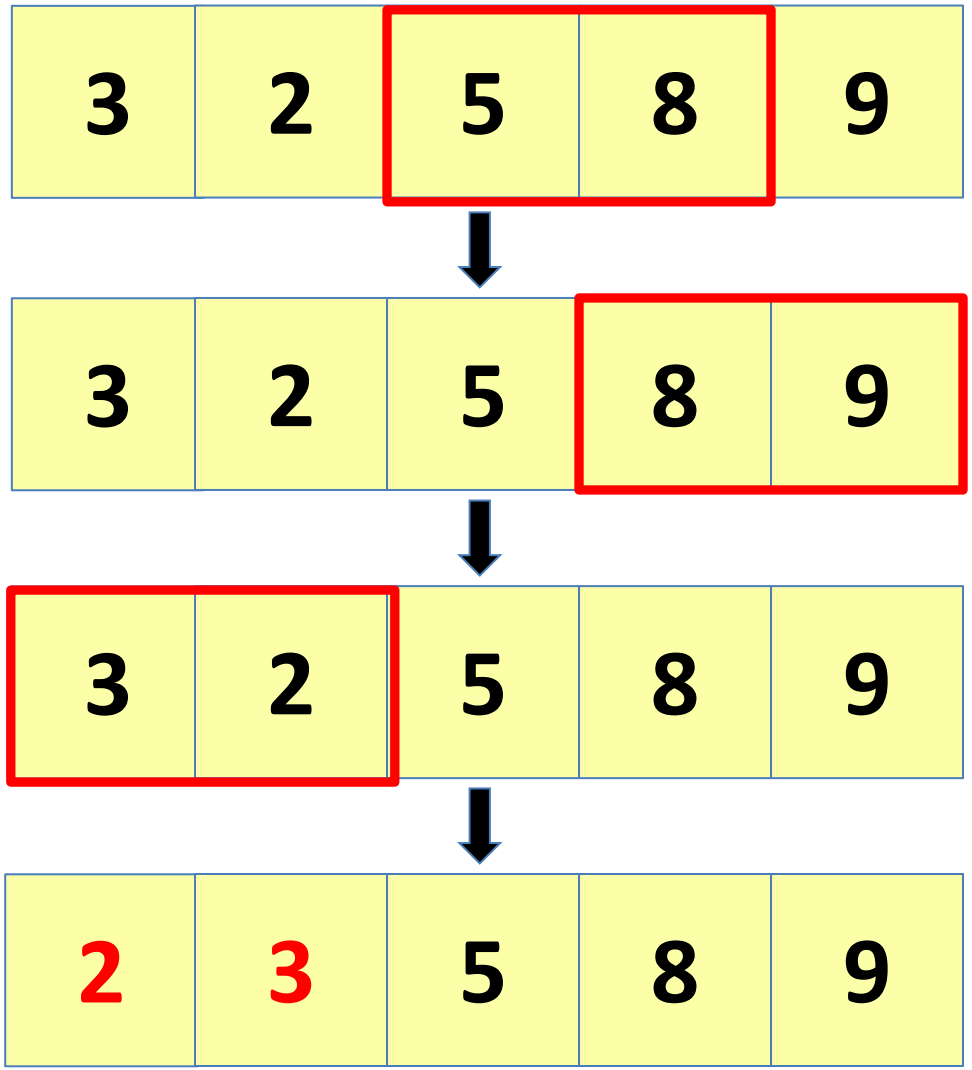


First pass over the sequence is over

But data is still not ordered

Hence, we **need a second pass**

Bubble Sort (Example)

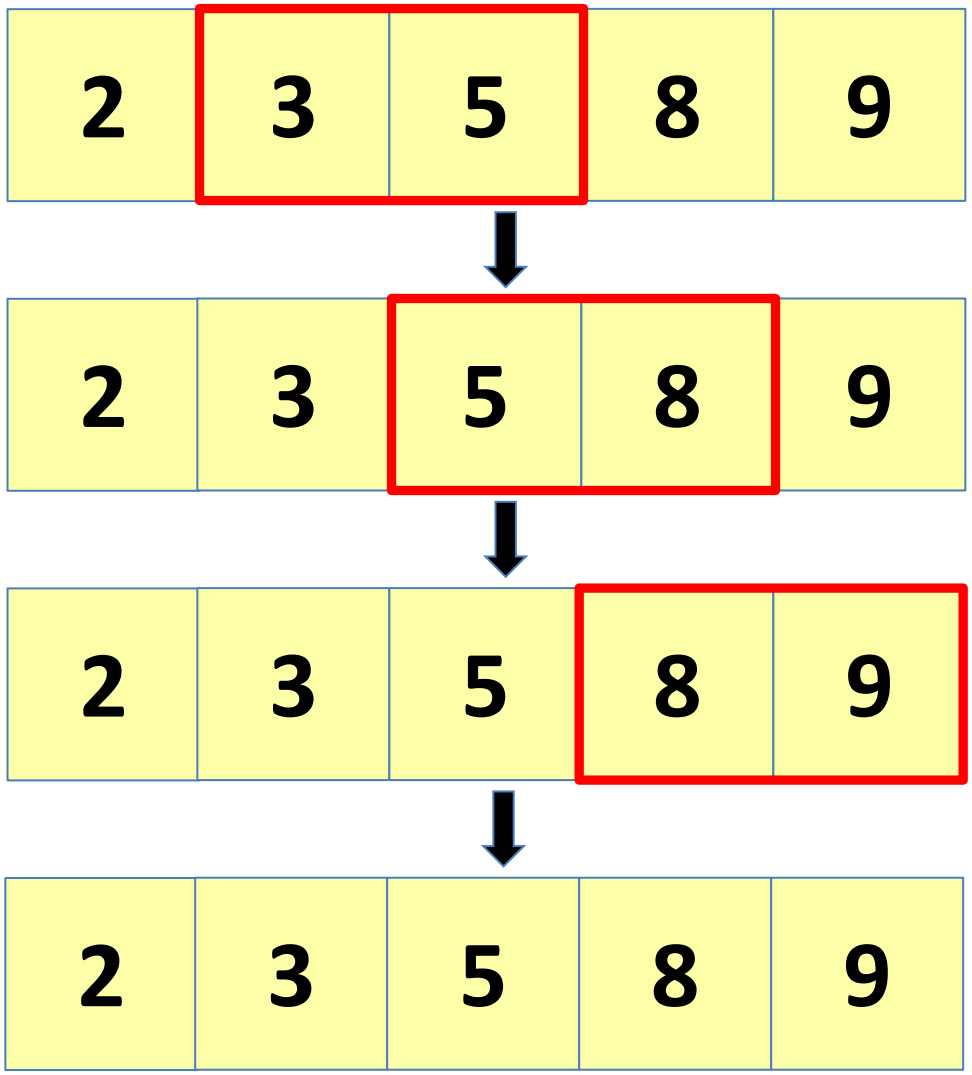


Second pass over the sequence is over

But data is still not ordered

Hence, we **need a third pass**

Bubble Sort (Example)



Third pass over the sequence is over

Finally, data is now sorted!!

Bubble Sort (Complexity)

- In- Class Exercise 3.3: What is best case and worst case the complexity of bubble sort? (Use Big-O notation)

Bubble Sort (Complexity)

- In- Class Exercise 3.3: What is best case and worst case the complexity of bubble sort? (Use Big-O notation)

Case 1) O(n) (Best case) This time complexity can occur if the array is **already sorted**, and that means that no swap occurred and only 1 iteration of n elements $\boldsymbol{O(n)}$

Bubble Sort (Complexity)

- In- Class Exercise 3.3: What is best case and worst case the complexity of bubble sort? (Use Big-O notation)

Case 1) O(n) (Best case) This time complexity can occur if the array is **already sorted**, and that means that no swap occurred and only 1 iteration of n elements $\boldsymbol{O(n)}$

Case 2) O(n^2) (Worst case) The worst case is if the array is **already sorted** but **in descending order**. This means that:

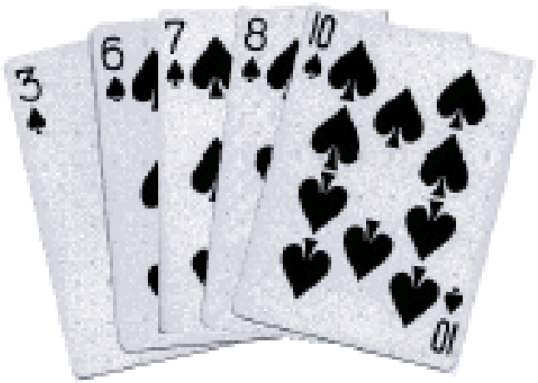
- in the 1^{st} iteration, it would have to look at n elements,
- then in the 2^{nd} iteration it would look $n - 1$ elements (since the biggest integer is at the end)
- …
- and so on and so forth till 1 comparison occurs.
- Hence in total: $n + n - 1 + n - 2 + n - 3 + \ldots + 1 = \sum_{i=1}^{n} i = \frac{n*(n+1)}{2} = \boldsymbol{O(n^2)}$

Sorting Algorithms

- There are many sorting algorithms. In this lecture, we will cover the following:
  - Bubble sort
  - Insertion sort
  - Mergesort
  - Quicksort (just a quick look!)

Insertion Sort

- Imagine in a card game, someone is dealing you cards
- Every time you get a new card, you put it in the right place in your hand



Insertion Sort (Example)

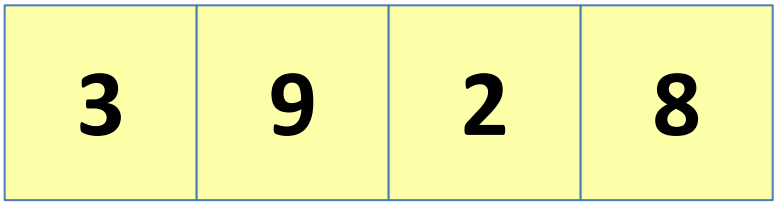
- We'll sort the following in ascending order

Insertion Sort (Example)

- We'll sort the following in ascending order

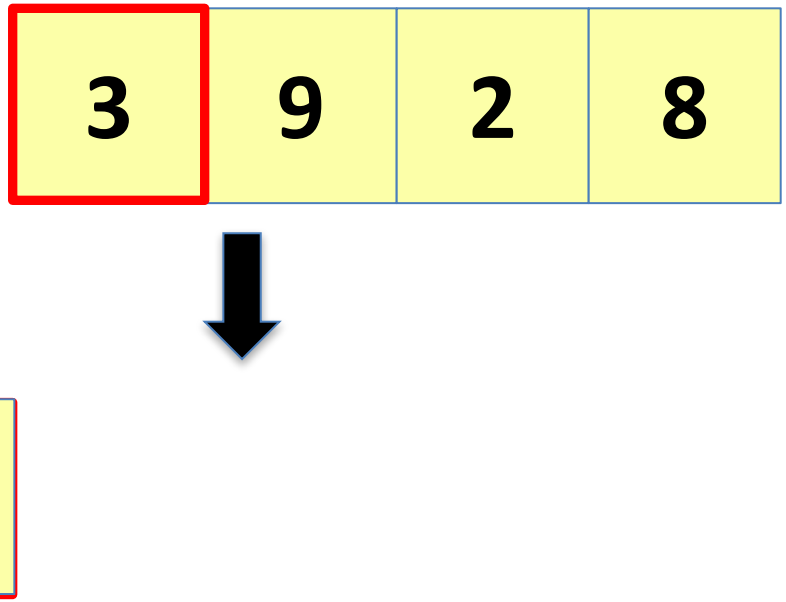
Insertion Sort (Example)

- We'll sort the following in ascending order



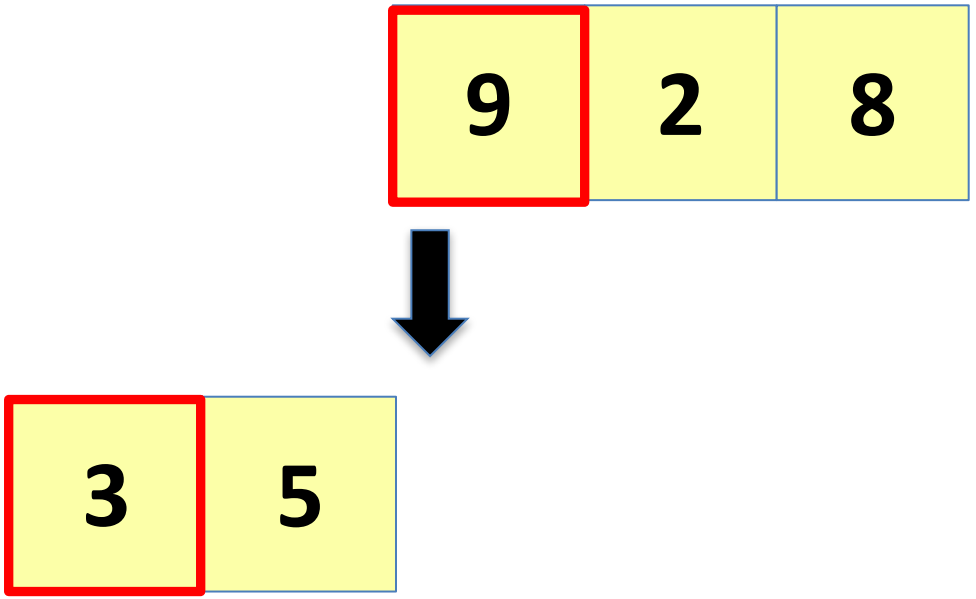
Insertion Sort (Example)

- We'll sort the following in ascending order



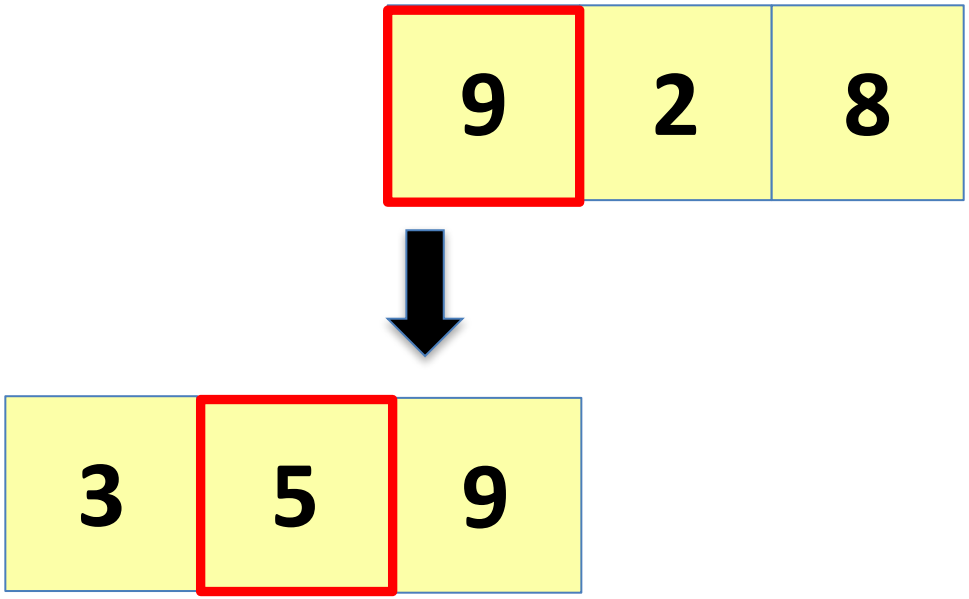
Insertion Sort (Example)

- We'll sort the following in ascending order



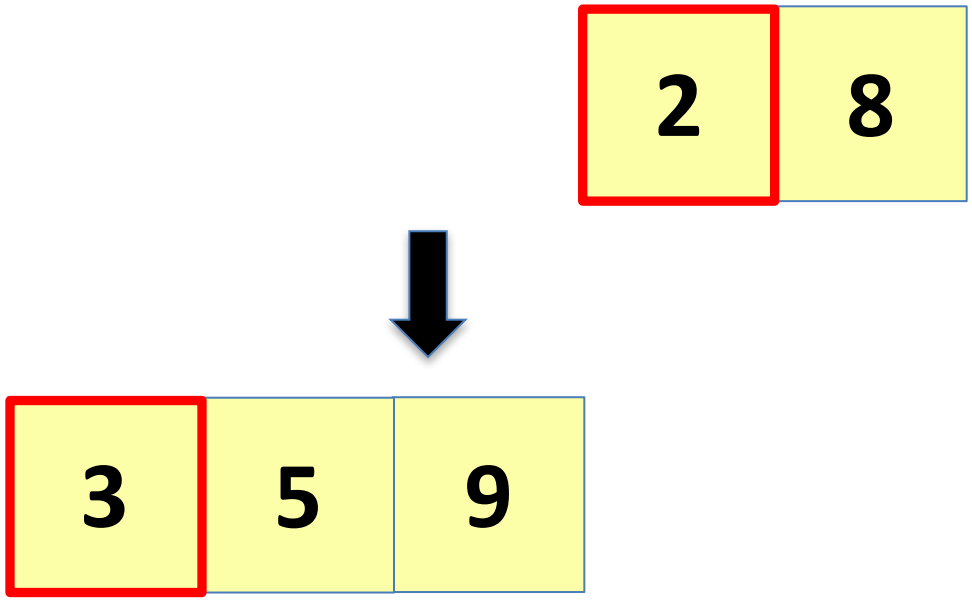
Insertion Sort (Example)

- We'll sort the following in ascending order



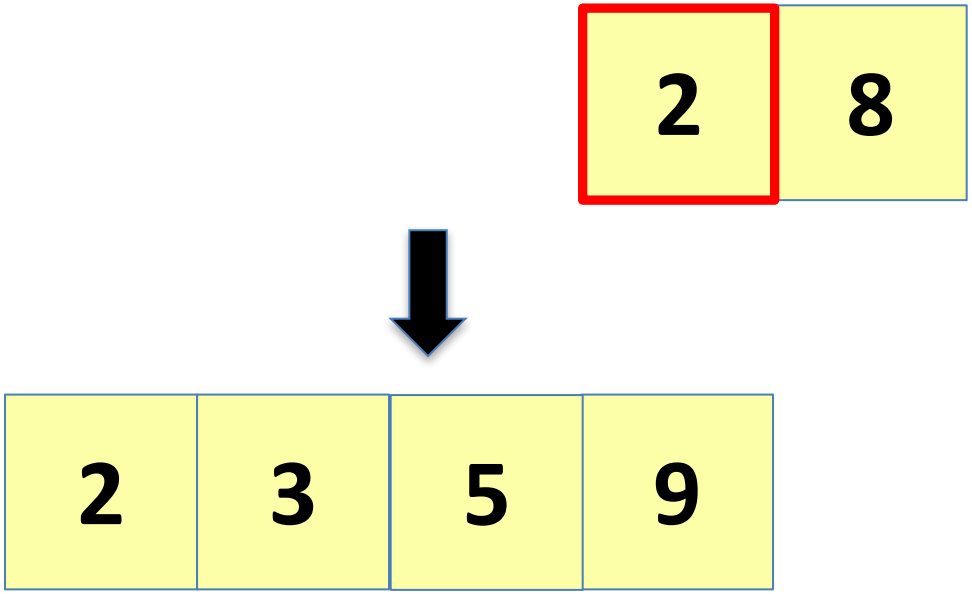
Insertion Sort (Example)

- We'll sort the following in ascending order



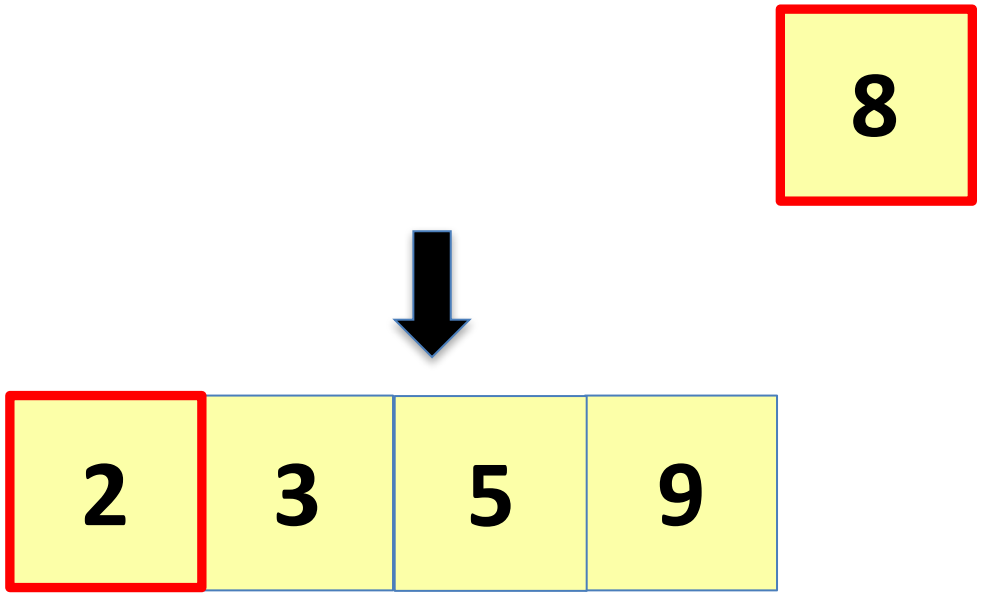
Insertion Sort (Example)

- We'll sort the following in ascending order



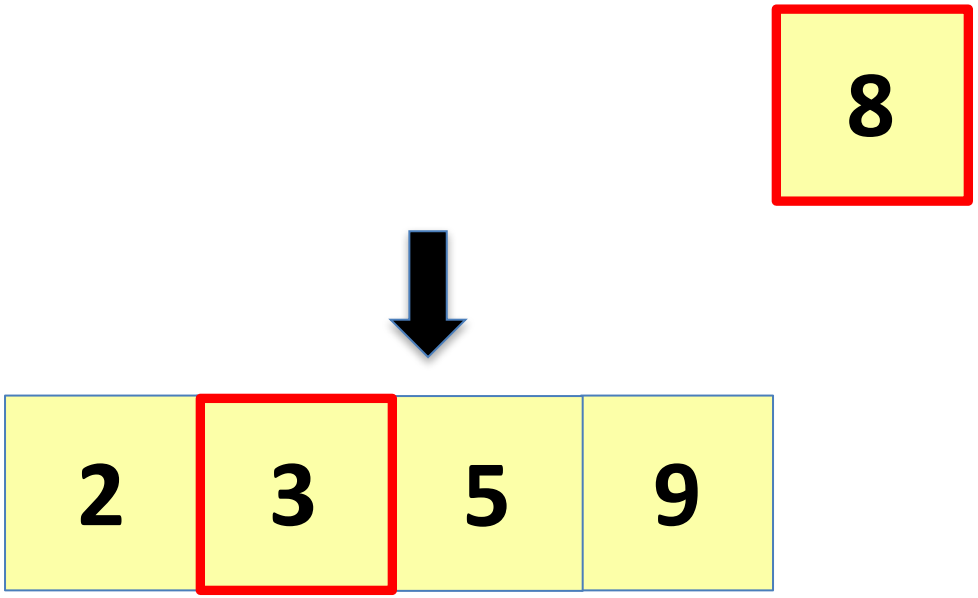
Insertion Sort (Example)

- We'll sort the following in ascending order



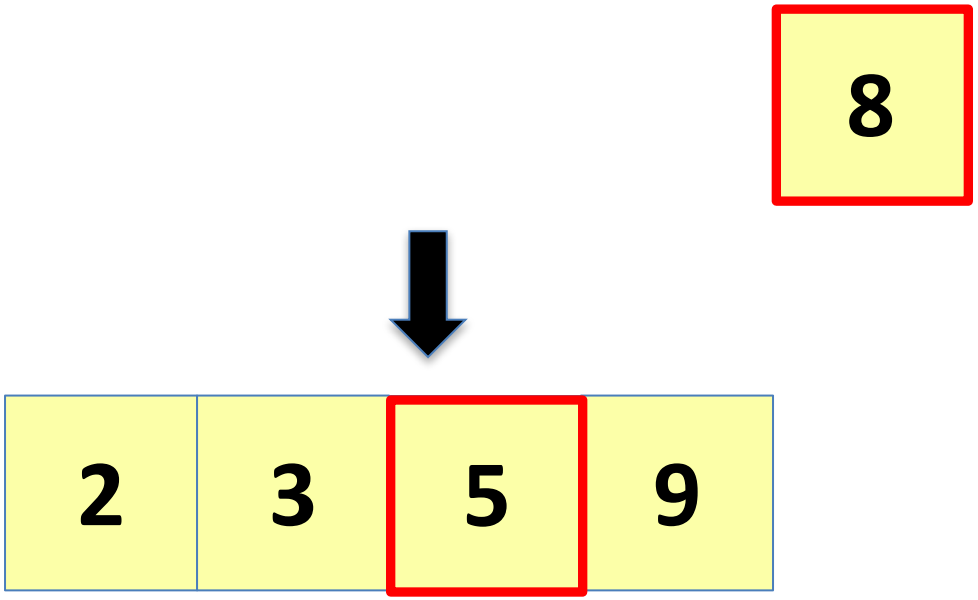
Insertion Sort (Example)

- We'll sort the following in ascending order



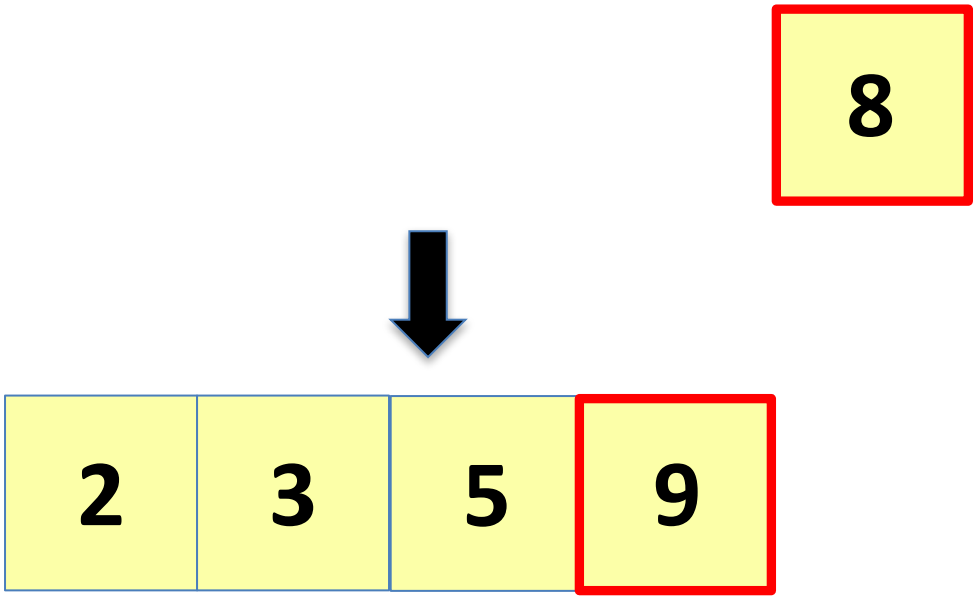
Insertion Sort (Example)

- We'll sort the following in ascending order



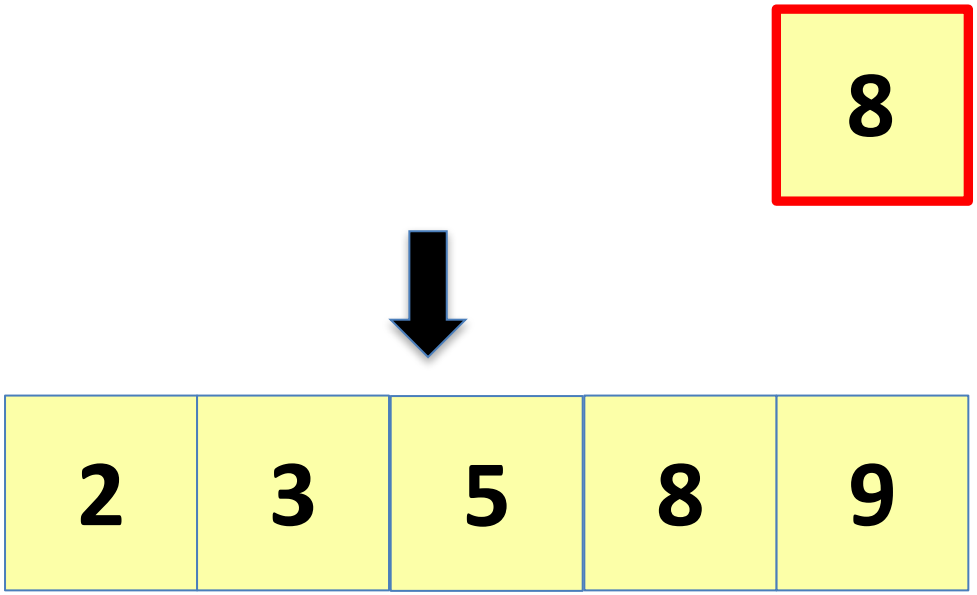
Insertion Sort (Example)

- We'll sort the following in ascending order



Insertion Sort (Example)

- We'll sort the following in ascending order



Insertion Sort (Complexity)

- In- Class Exercise 3.4: What is the complexity of insertion sort? (Use Big-O notation)

Insertion Sort (Complexity)

- In- Class Exercise 3.4: What is the complexity of insertion sort? (Use Big-O notation)

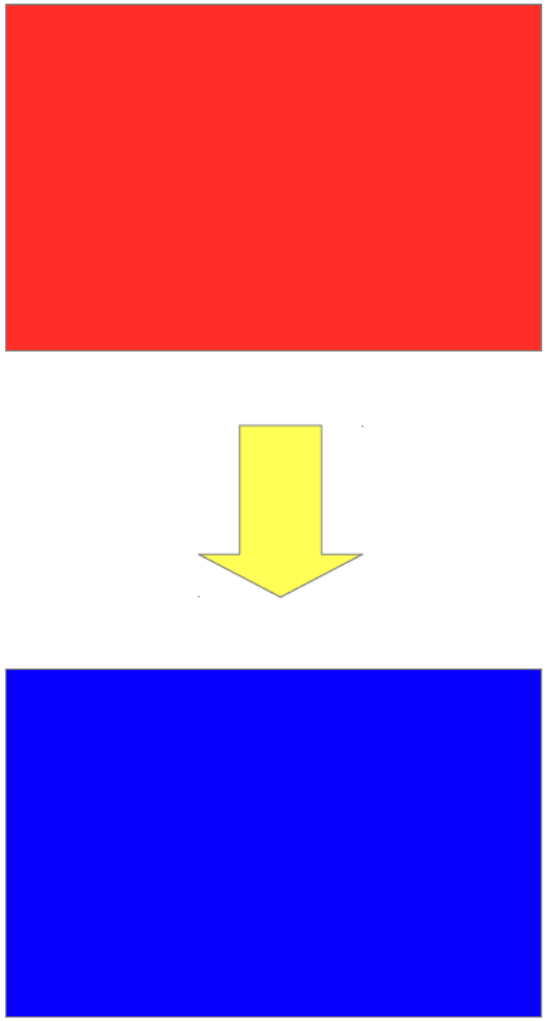
  - Insertion sort does n insertions for an array of size n
  - To insert into a sorted array, you must move all the elements up one, which is $O(n)$.
  - Thus total is $\boldsymbol{O(n^2)}$

Sorting Algorithms

- There are many sorting algorithms. In this lecture, we will cover the following:
  - Bubble sort
  - Insertion sort
  - Mergesort
  - Quicksort (just a quick look!)

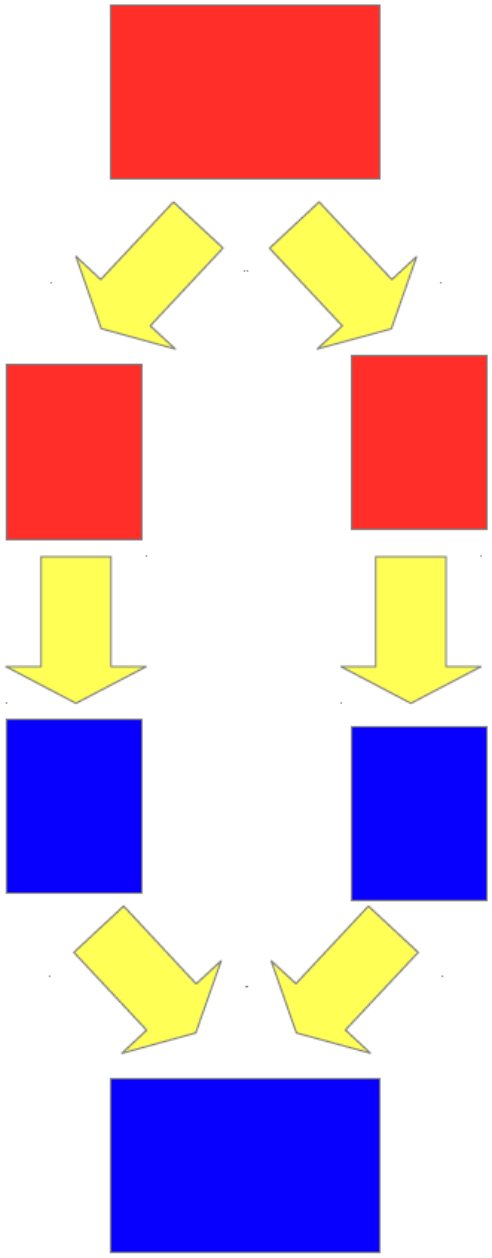
Mergesort

IDEA: To solve this…



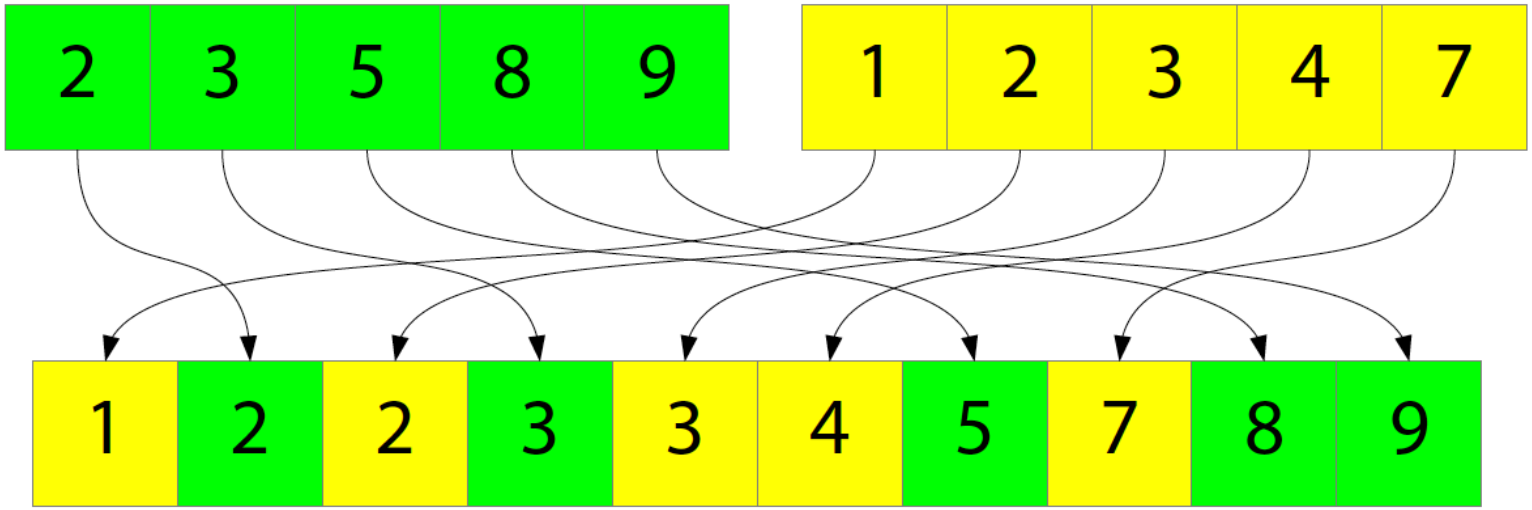
Mergesort

- Split the problem into 2 sub-problems
- Solve the problem for each solution
- Combine the solutions



Mergesort

- We can merge two sorted lists into one in linear time:

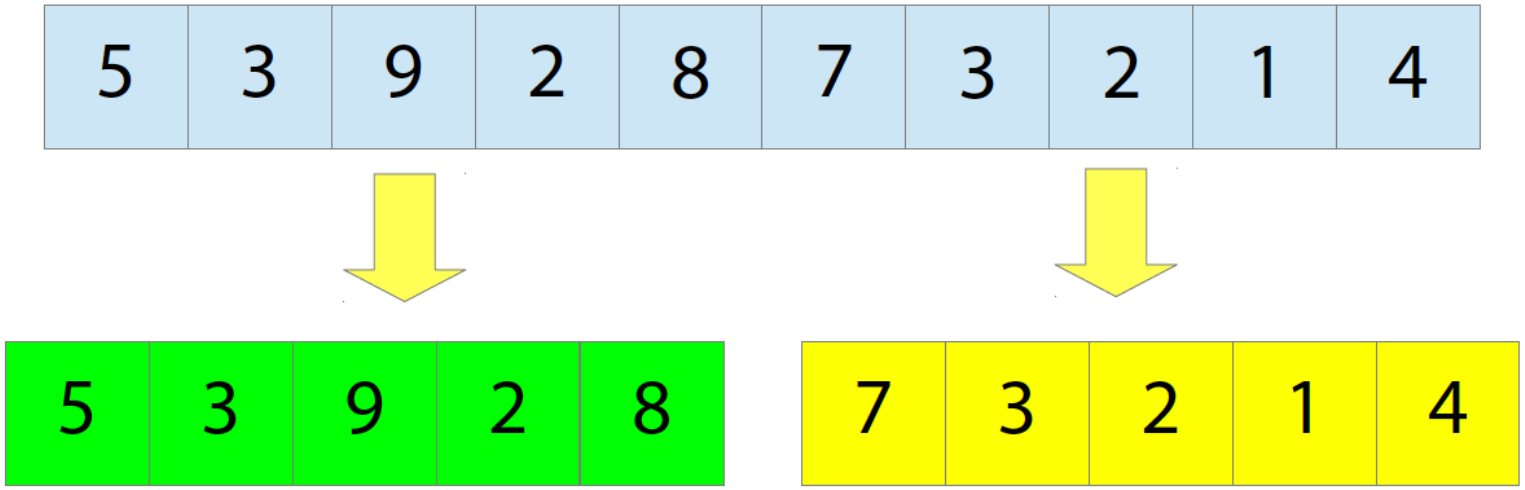


Mergesort

- A divide-and-conquer algorithm
- To mergesort a list:
  - Split the list into two equal parts
  - Recursively mergesort the two parts
  - Merge the two sorted lists together

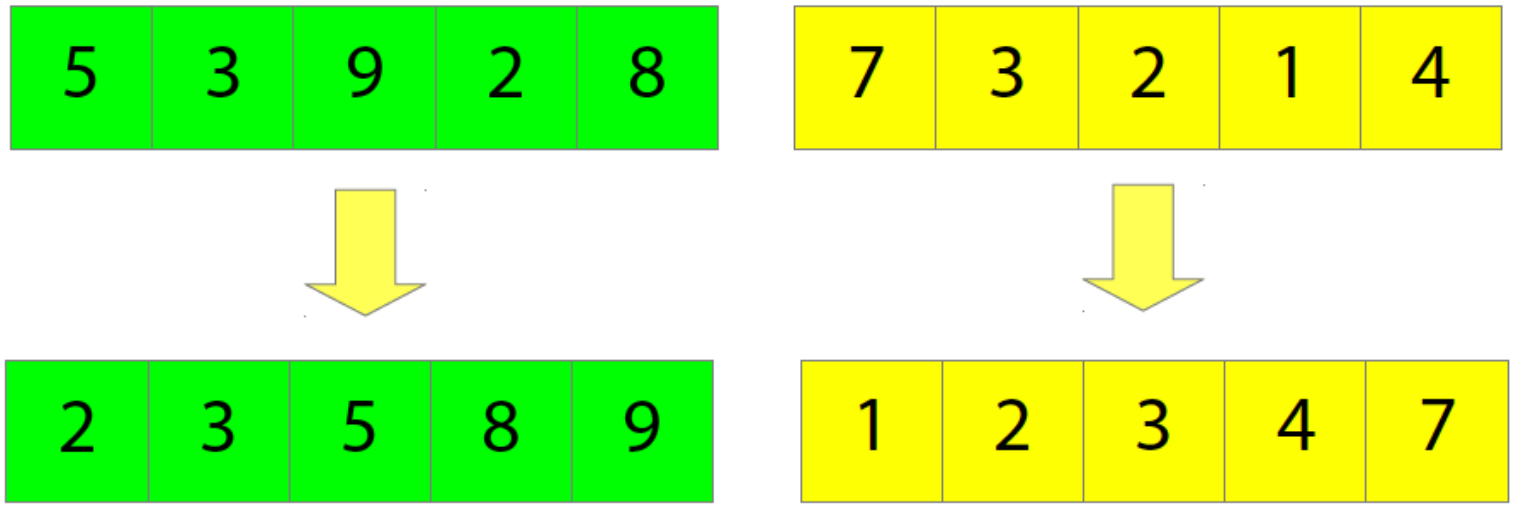
Mergesort

1. Split the list into two equal parts



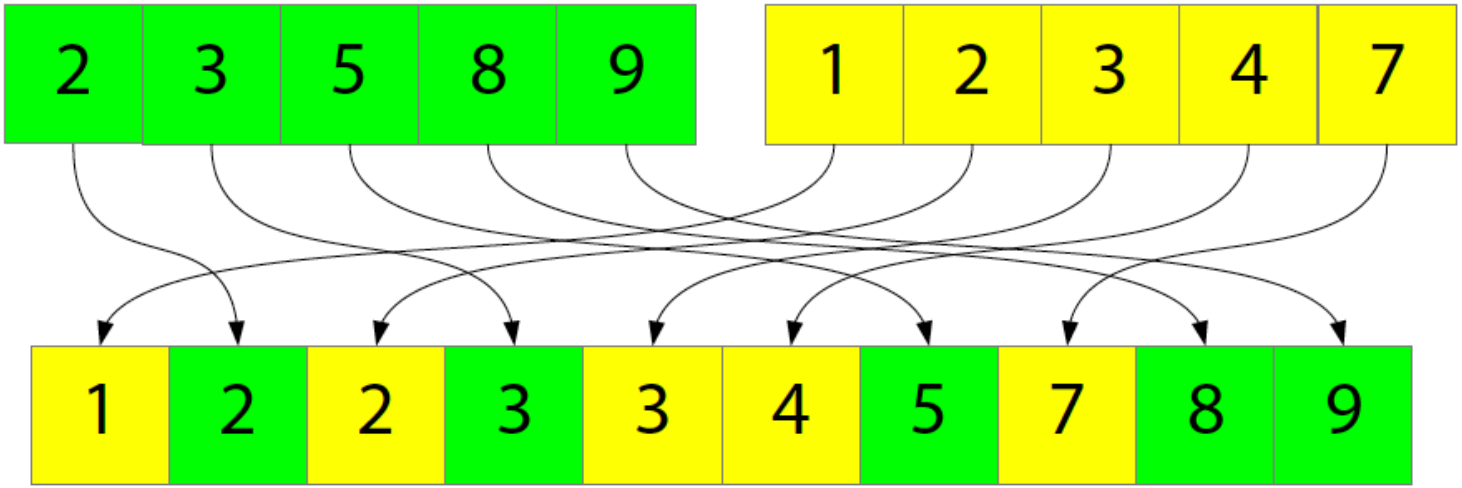
Mergesort

2. Recursively mergesort the two parts



Mergesort

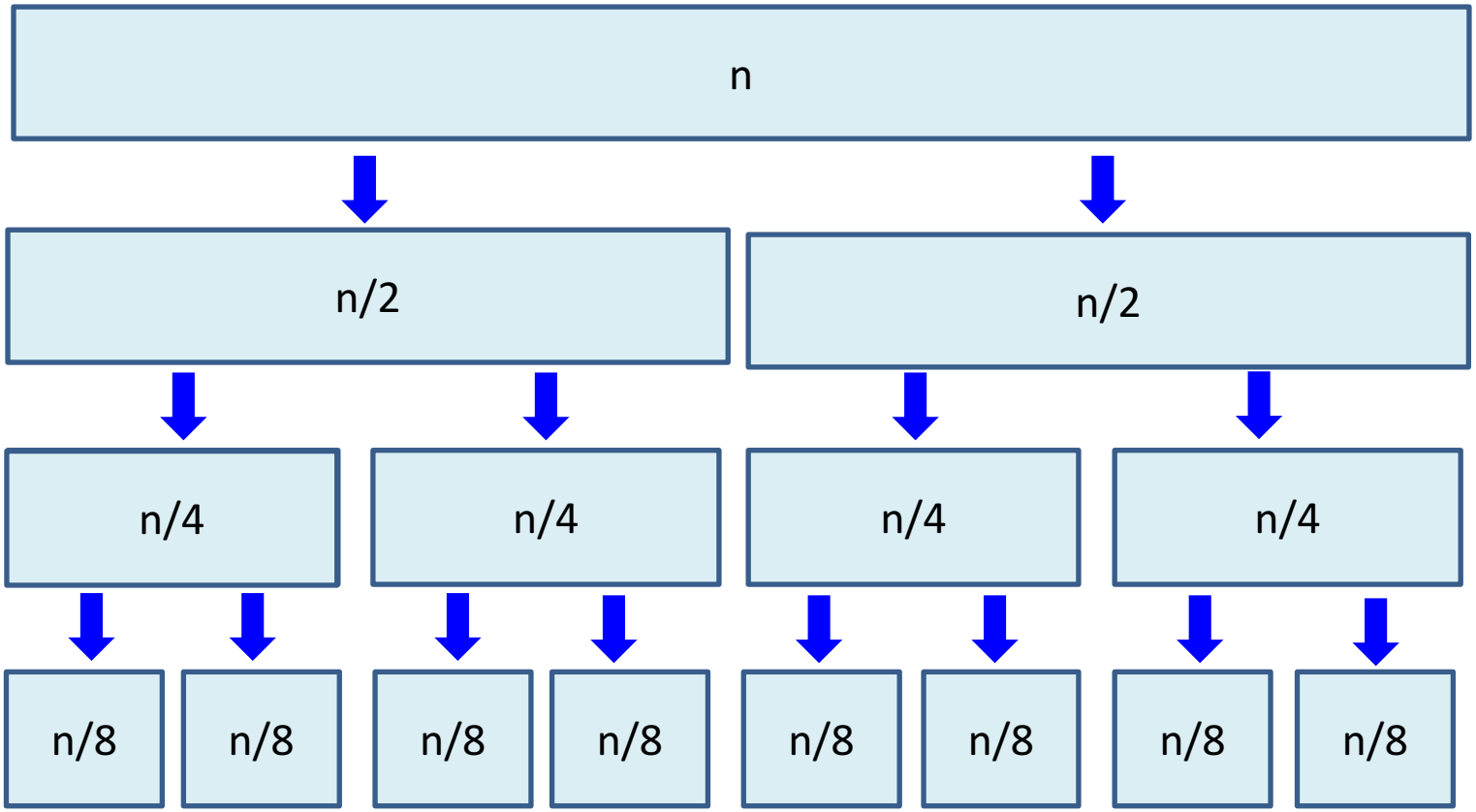
3. Merge the two sorted lists together



Merge Sort (Complexity)

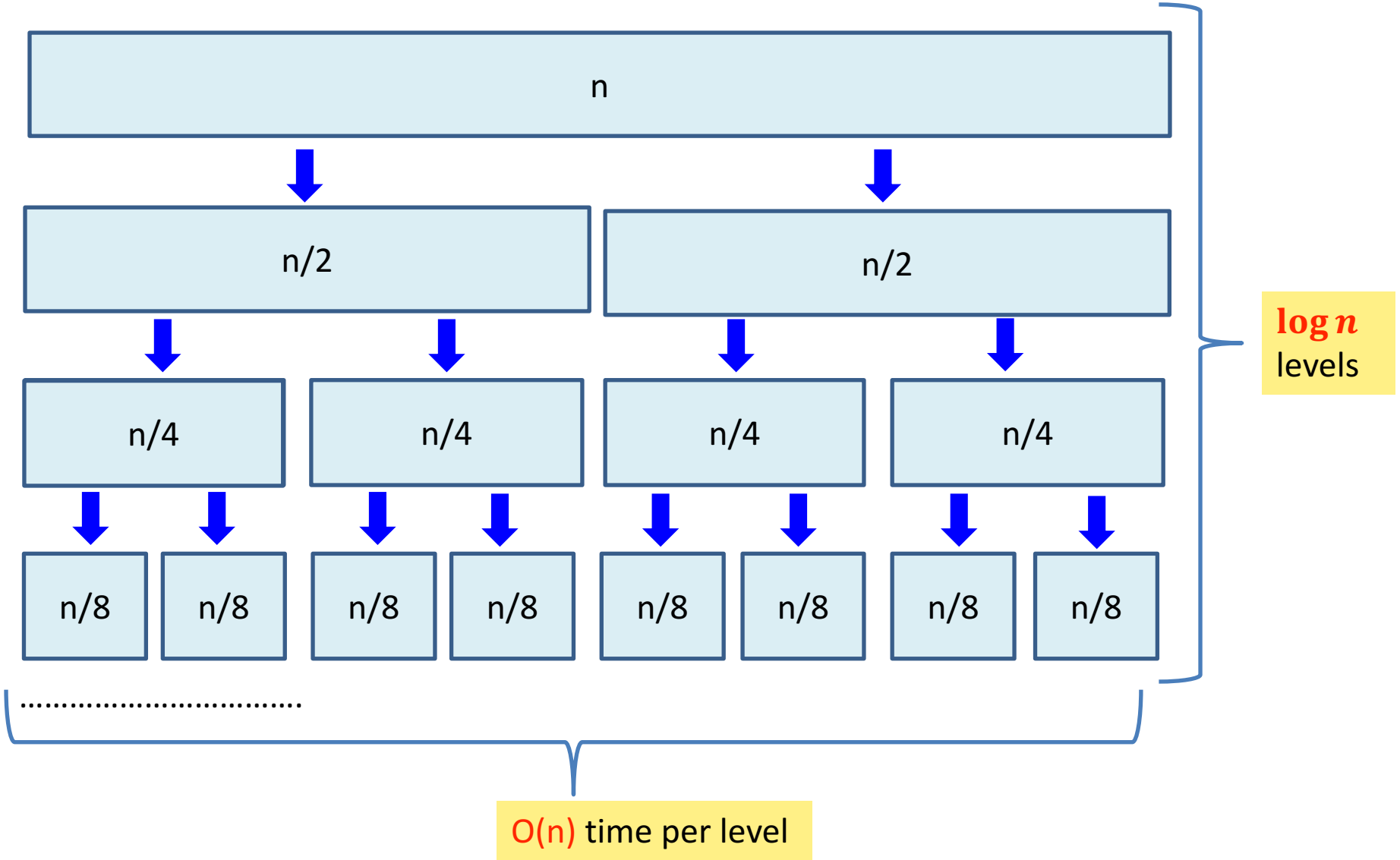
- In- Class Exercise 3.5: What is the complexity of merge sort? (Use Big-O notation)

Mergesort (Complexity)

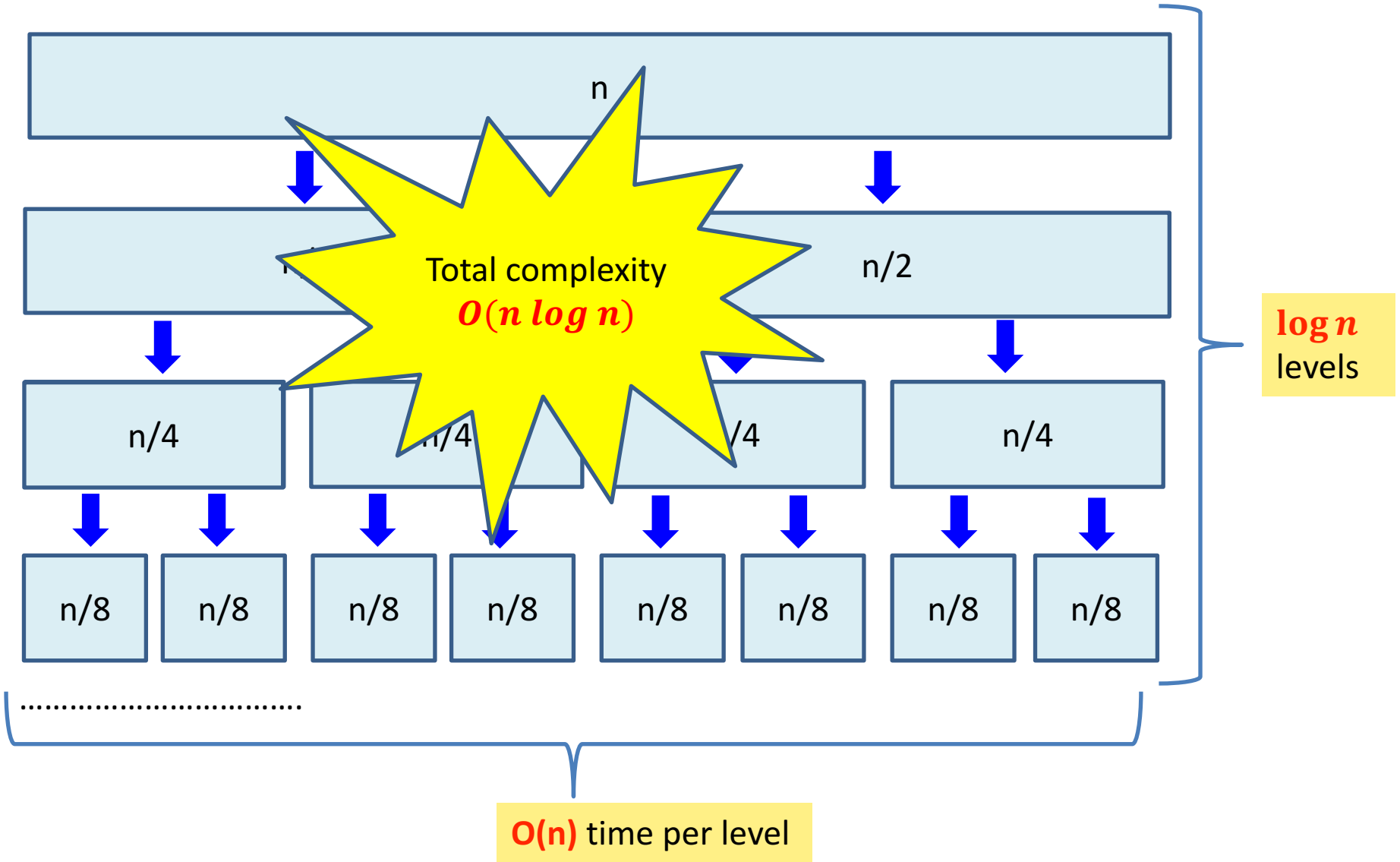


.....

Mergesort (Complexity)



Mergesort (Complexity)

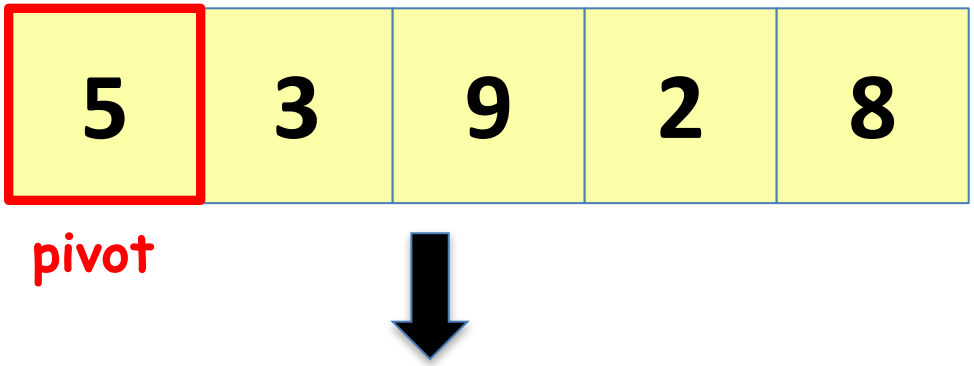


Sorting Algorithms

- There are many sorting algorithms. In this lecture, we will cover the following:
  - Bubble sort
  - Insertion sort
  - Mergesort
  - Quicksort (just a quick look!)

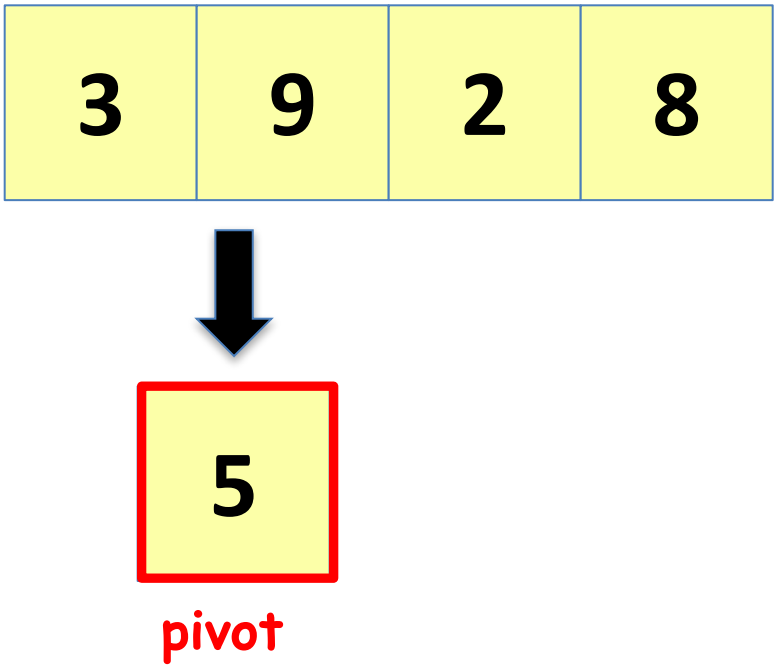
QuickSort (Example)

- We'll sort the following in ascending order



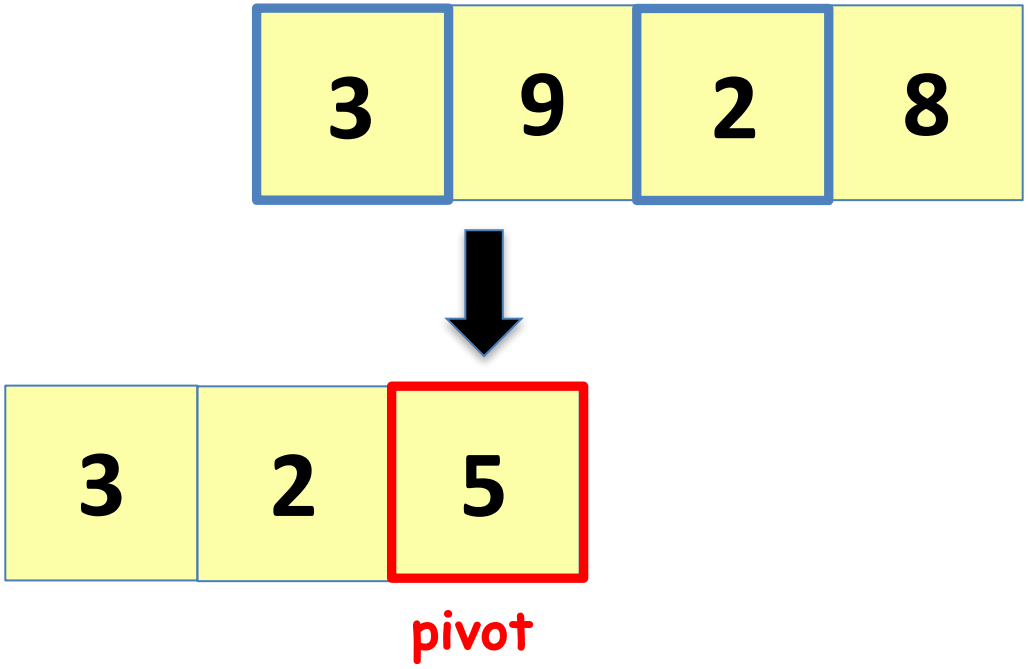
Quicksort (Example)

- We'll sort the following in ascending order



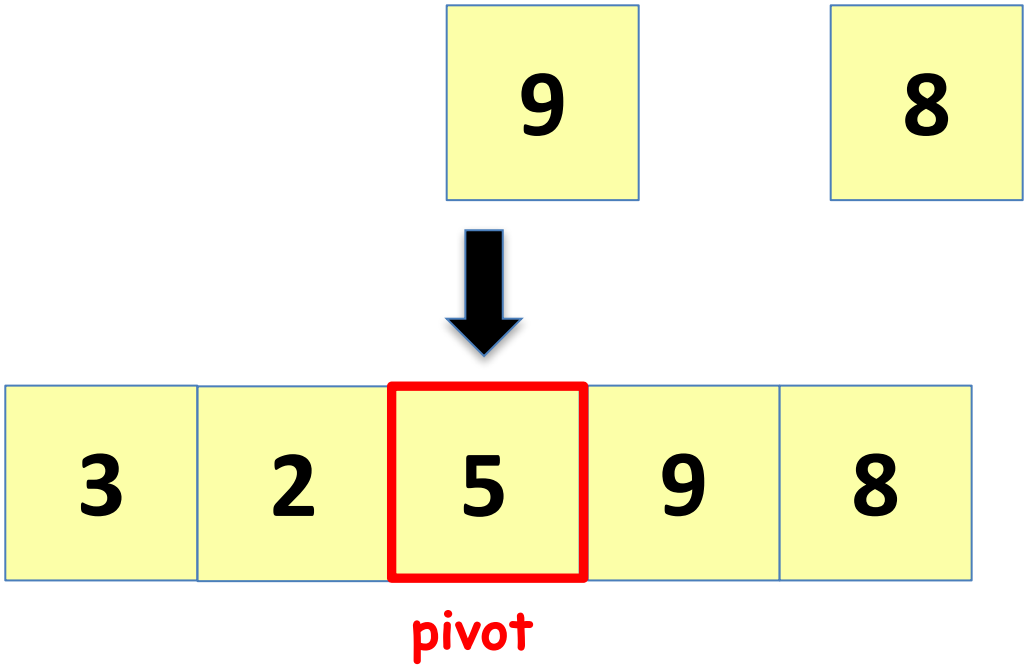
Quicksort (Example)

- We'll sort the following in ascending order



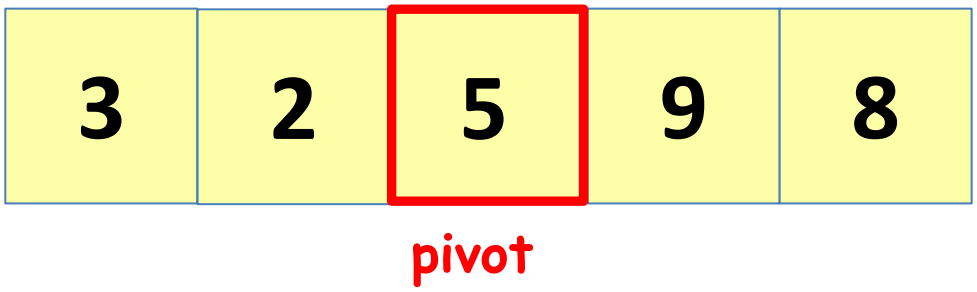
Quicksort (Example)

- We'll sort the following in ascending order



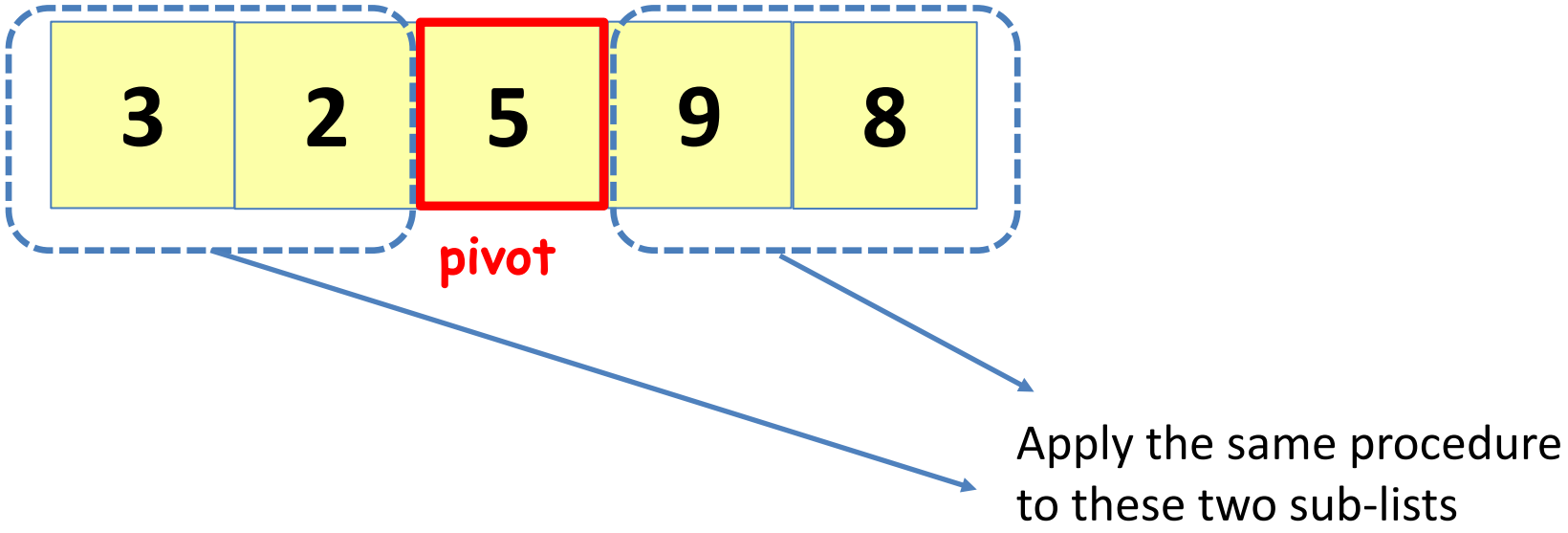
Quicksort (Example)

- We'll sort the following in ascending order



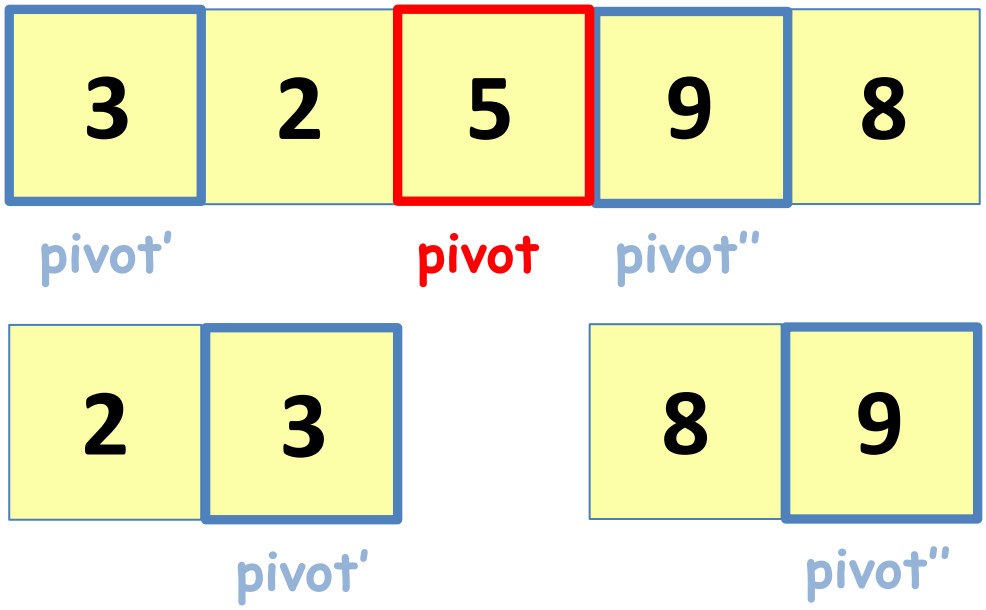
Quicksort (Example)

- We'll sort the following in ascending order



Quicksort (Example)

- We'll sort the following in ascending order

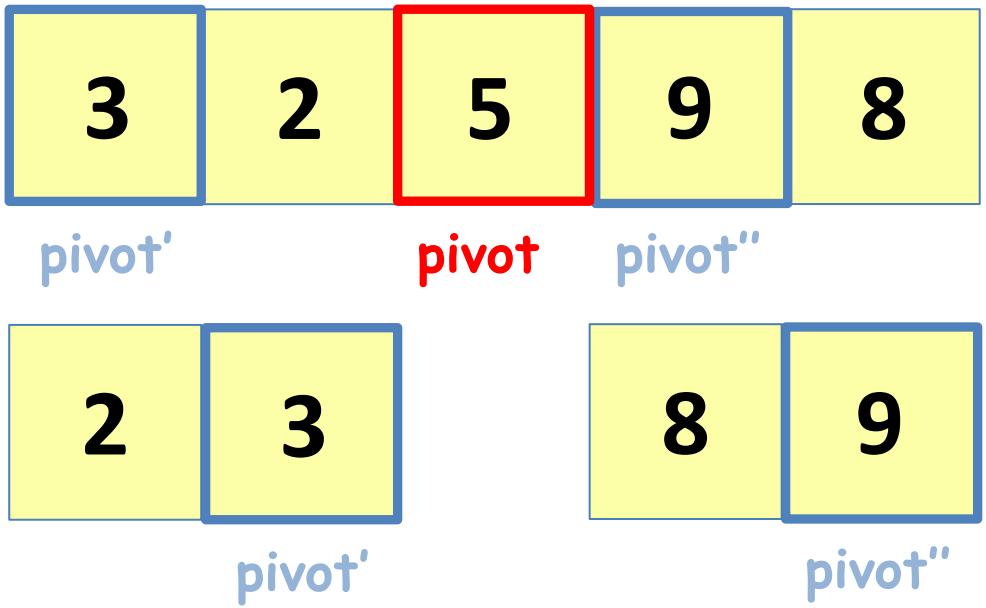


Quicksort (Example)

- We'll sort the following in ascending order

Quicksort (Example)

- We'll sort the following in ascending order



Quicksort (Example)

- We'll sort the following in ascending order

Tutorial video