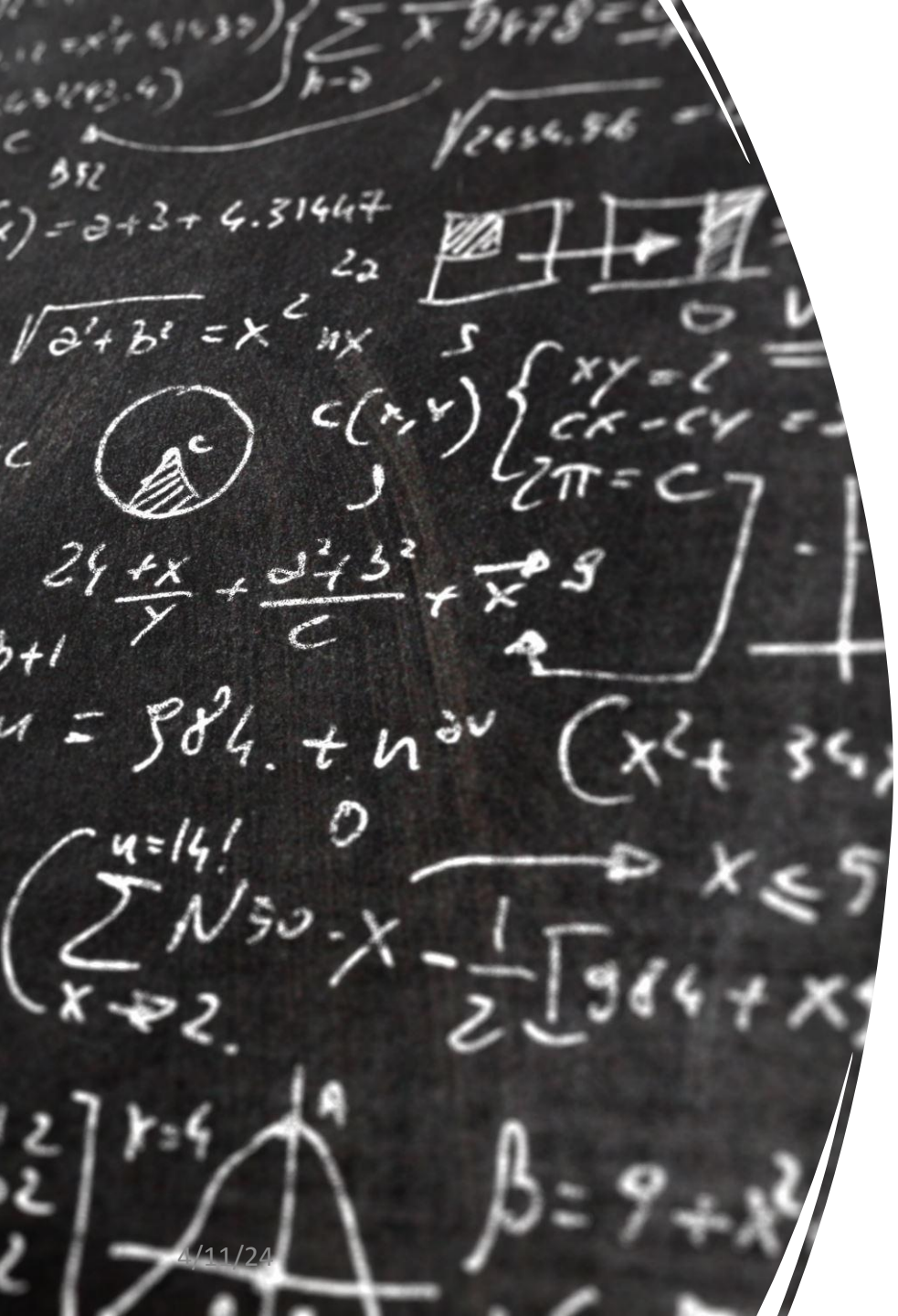


CS 113 – Computer Science I

Lecture 21 – Binary Search II

Tuesday 4/11/2024

Announcements

HW7 – Class Design and Lab7 due tonight
HW8 will be released tonight

Outline

- Binary Search Review
- Sorting
  - Bubble Sort
  - Selection Sort

Binary Search

- What is linear search?
  - Why is it inefficient?

- What is binary search?
  - Why is it more efficient?
  - What does it require about the list we are searching?

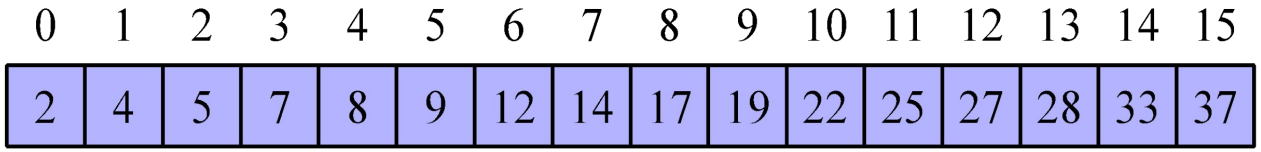
Binary Search

1. Calculate midpoint

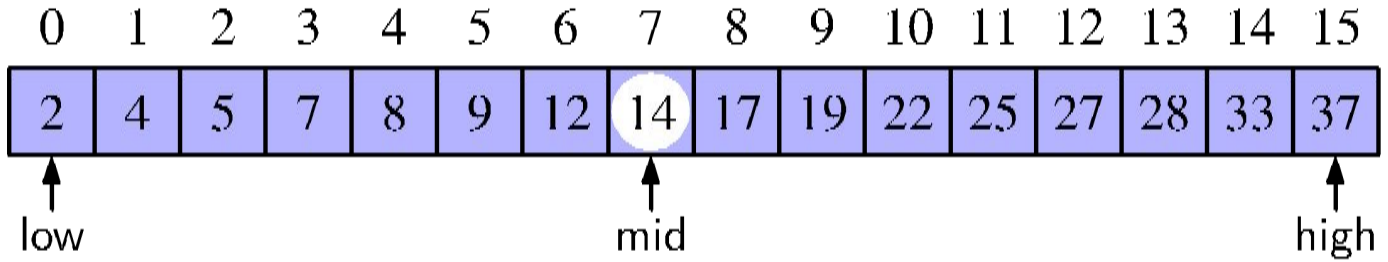
2. Compare the value at the midpoint with the target value
   a. if equal:
      i. return index
   b. if target value < midpoint value:
      i. **search** the left portion of the list
   c. if target value > midpoint value:
      i. **search** the right portion of the list

Binary Search

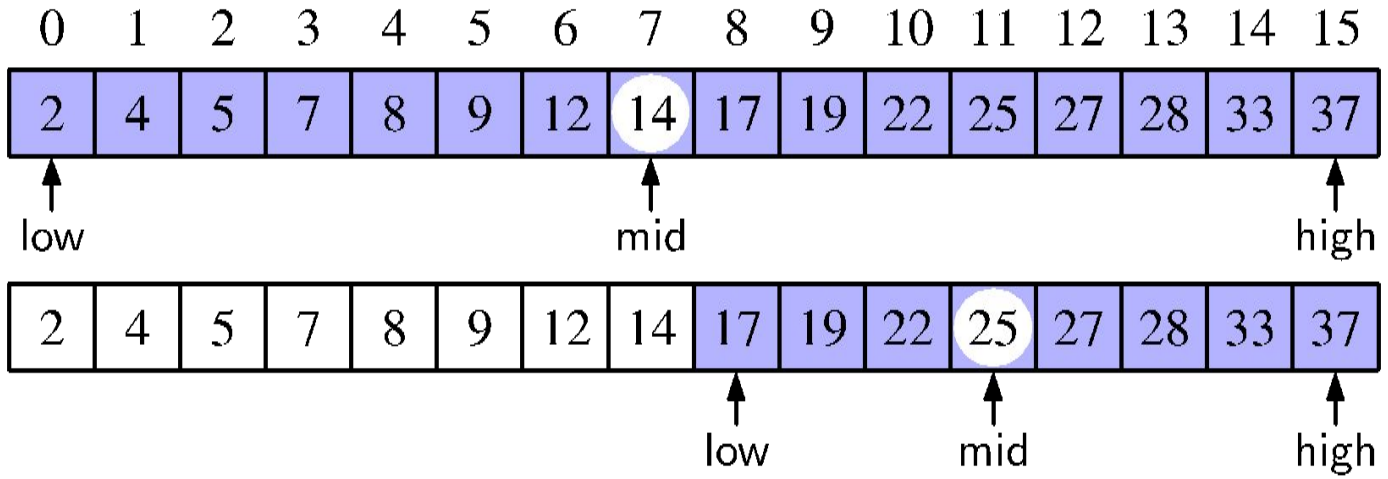
Search for an integer (22) in an ordered list



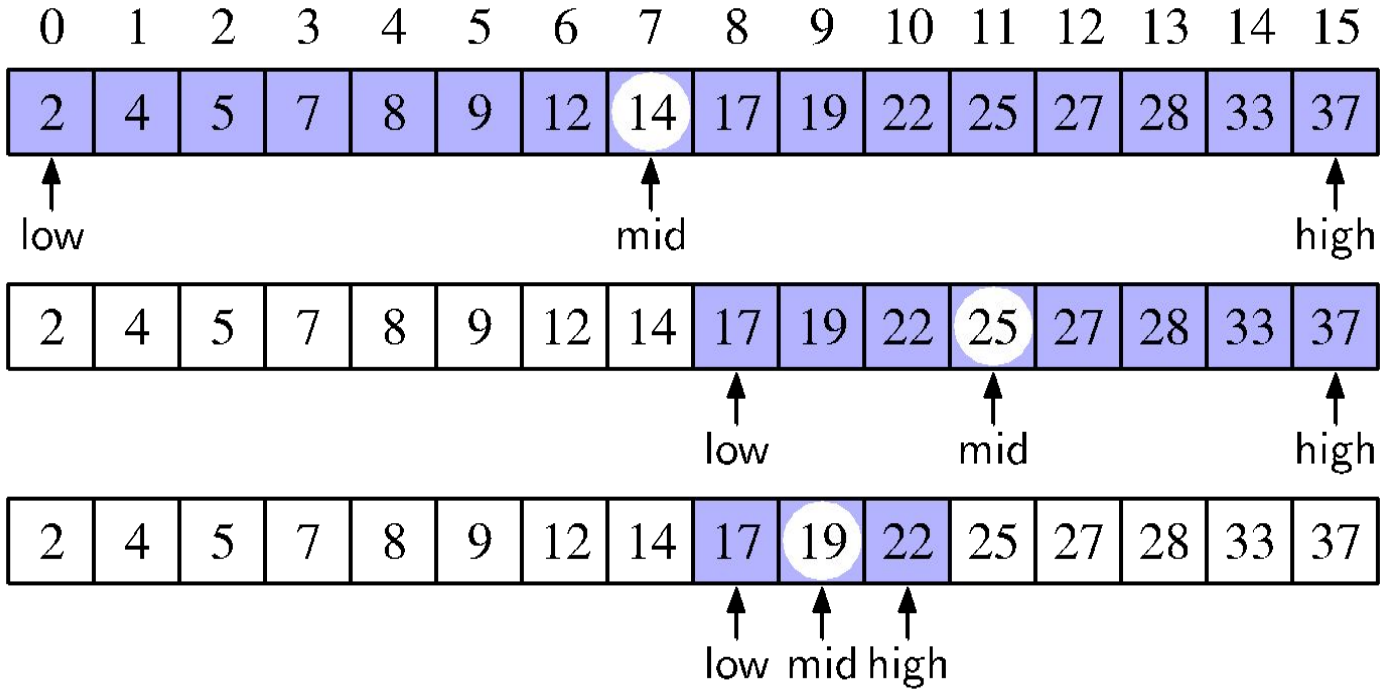
target = 22


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high

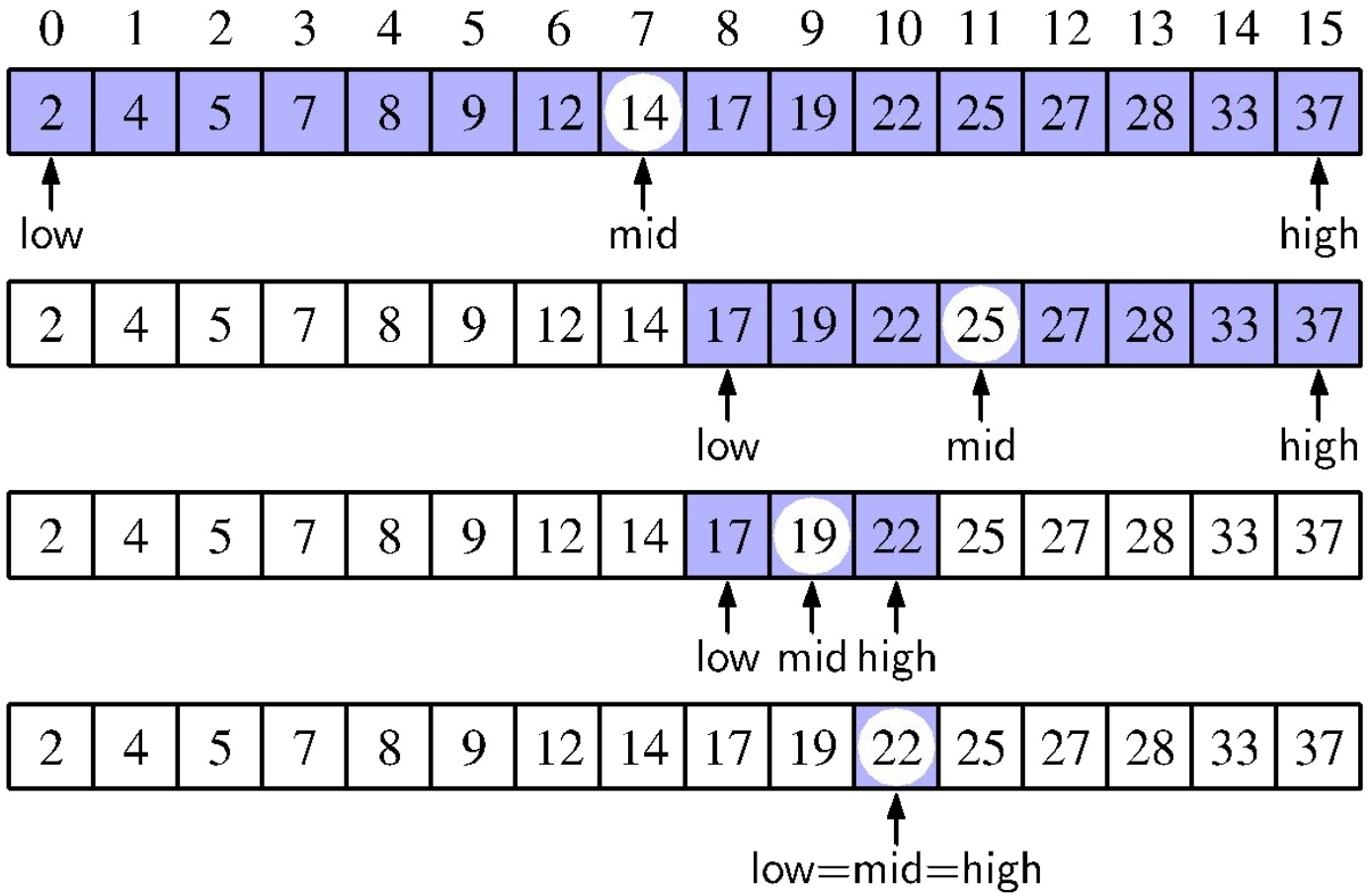
target = 22


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high

target = 22


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low mid high

target = 22


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low mid high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low=mid=high

Binary search

String[] ls = {-20, -4, 44, 58, 99, 145}

Search for 99

low	mid	high	ls[mid]

Binary search w/ Strings

String[] ls = {"bear", "bird", "bug", "cat", "cow", "dog", "fish, "lion“};

Search for “cow”

low	mid	high	ls[mid]

Binary Search Implementation

Binary Search Runtime

- In **Linear Search** how do the # of checks increase as a function of the # of elements in the list?
  - If we have 1 element in the list how many checks will we do?
  - What about 100 elements?
  - 1000?
  - 1mil?
  - **Checks increase linearly with the # of elements**

Binary Search Runtime

- In **Binary Search** how do the # of checks increase as a function of the # of elements in the list?
  - If we have 8 element in the list how many checks will we do?
  - What about 16 elements?
  - 32?
  - 64?
  - **Checks increase logarithmically with the # of elements**

- Time increases by $\log_2 n$. Why?

Runtime Complexity

Analyzing runtime complexity is an important aspect of computer science.

We analyze the runtime as a *function of the input size*

Consider the *worst case*

Outline

- Binary Search
- **Sorting**
  - Bubble Sort
  - Selection Sort

Who do we care about sorting?

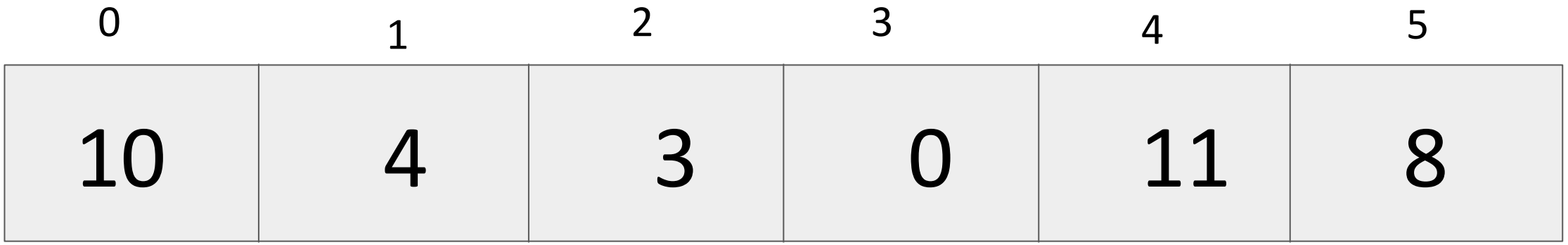
1. Makes searching faster
   a. Binary search!

2. Efficient data retrieval
   a. In many cases we want to print things in some order (alphabetical names, increasing scores etc)

3. Fundamental in the way computers work
   a. Operating systems
   b. Data structures (arrays, etc)
   c. ...

Sorting

How might we sort the list of numbers below in ascending order?



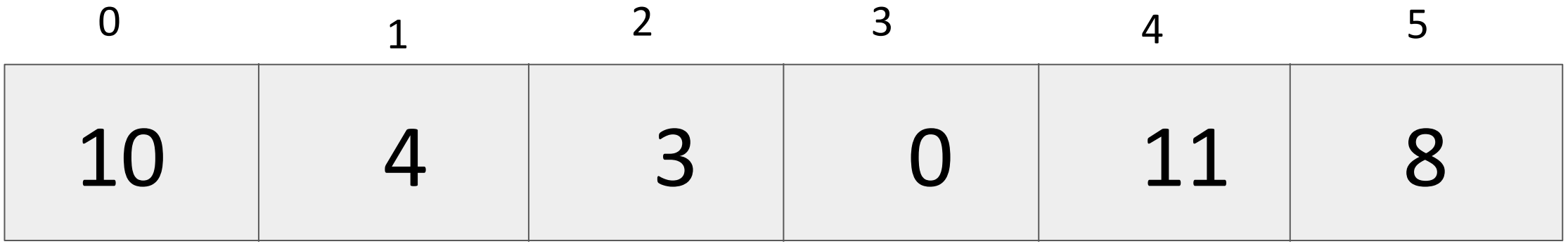
Bubble Sort

- Step through the input list element by element
- Compare the element with the one next to it
  - Swap values if needed
- Larger elements “bubble” to the back of the list

Bubble Sort

1. Start with the first element in the list
2. Compare the cur element with the next element
3. If cur > next, swap them
4. Move to the next pair of elements and repeat steps 2 and 3 until the end of the list is reached.
5. Repeat

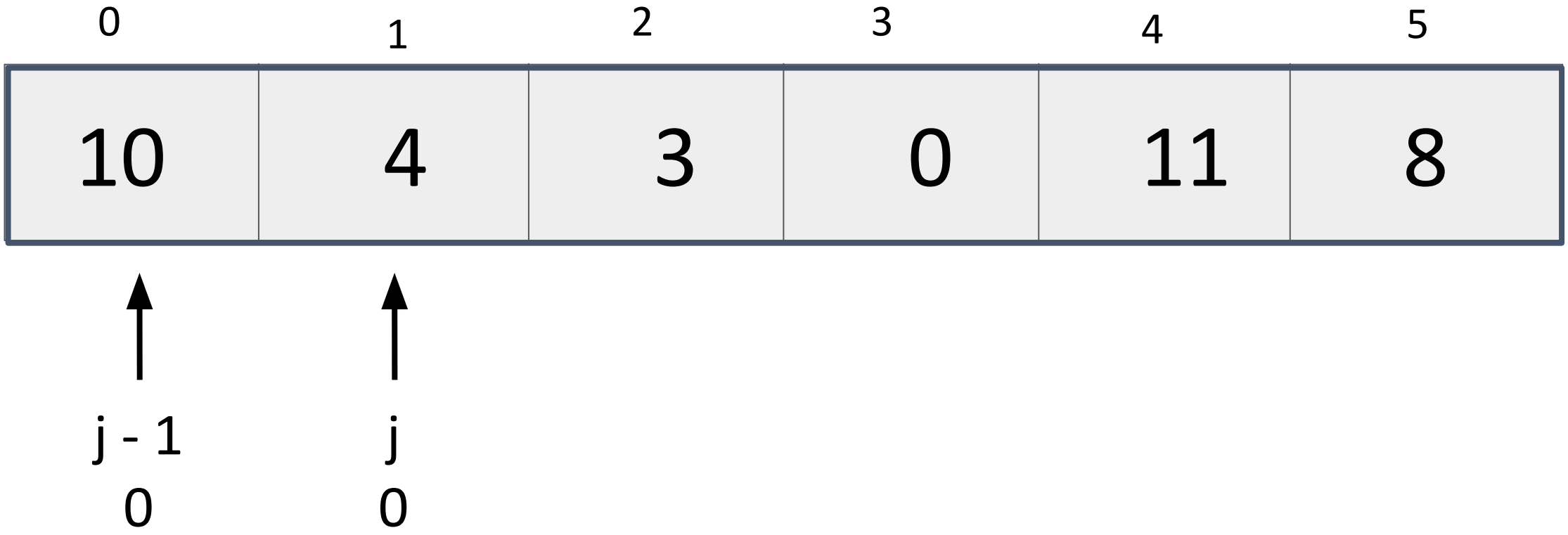
Bubble Sort



What do we do first?

Bubble Sort

len = 6

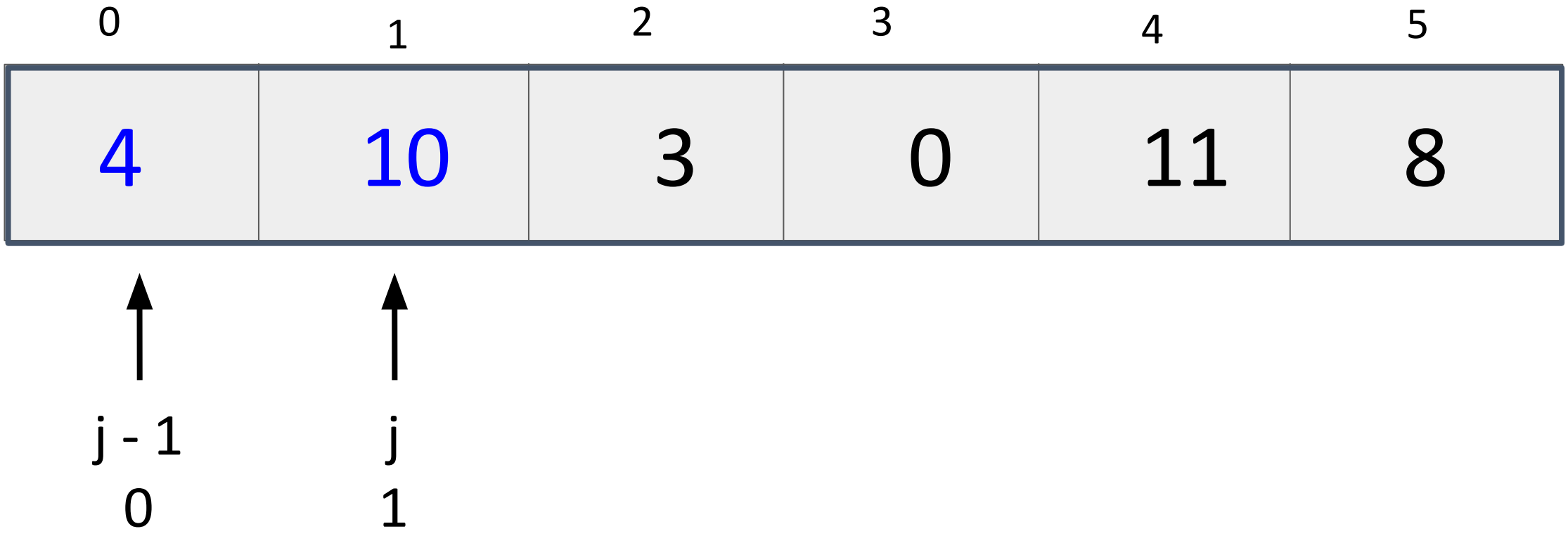


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

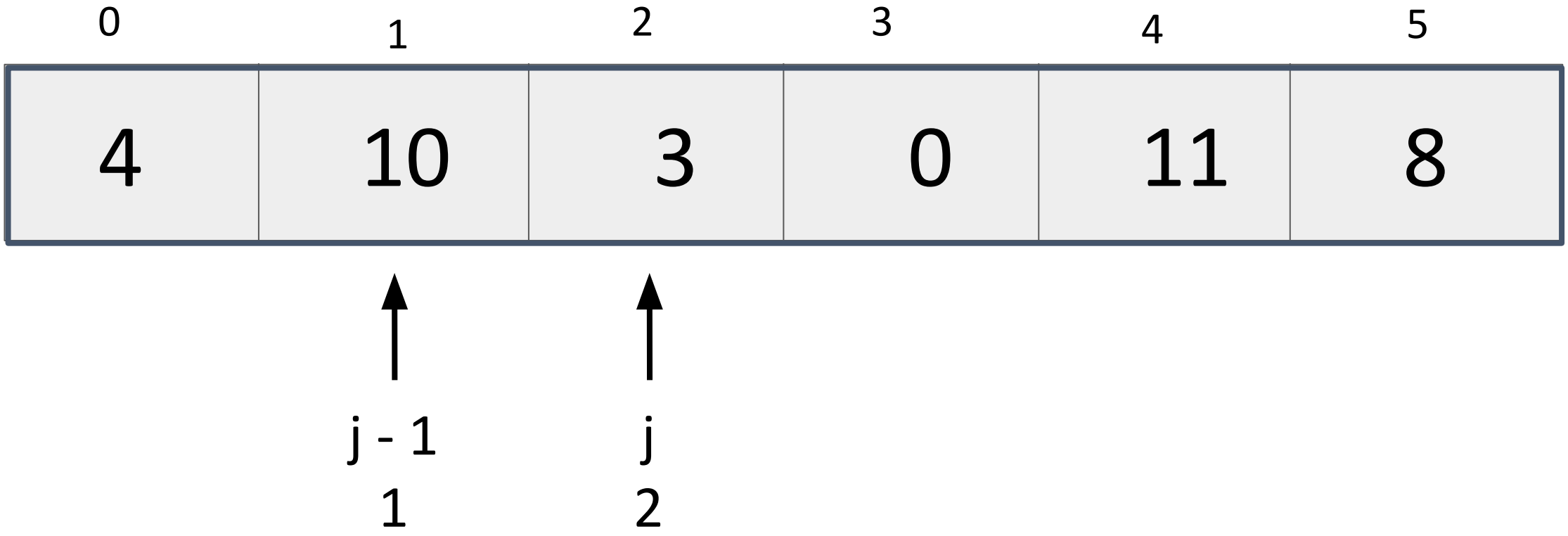


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

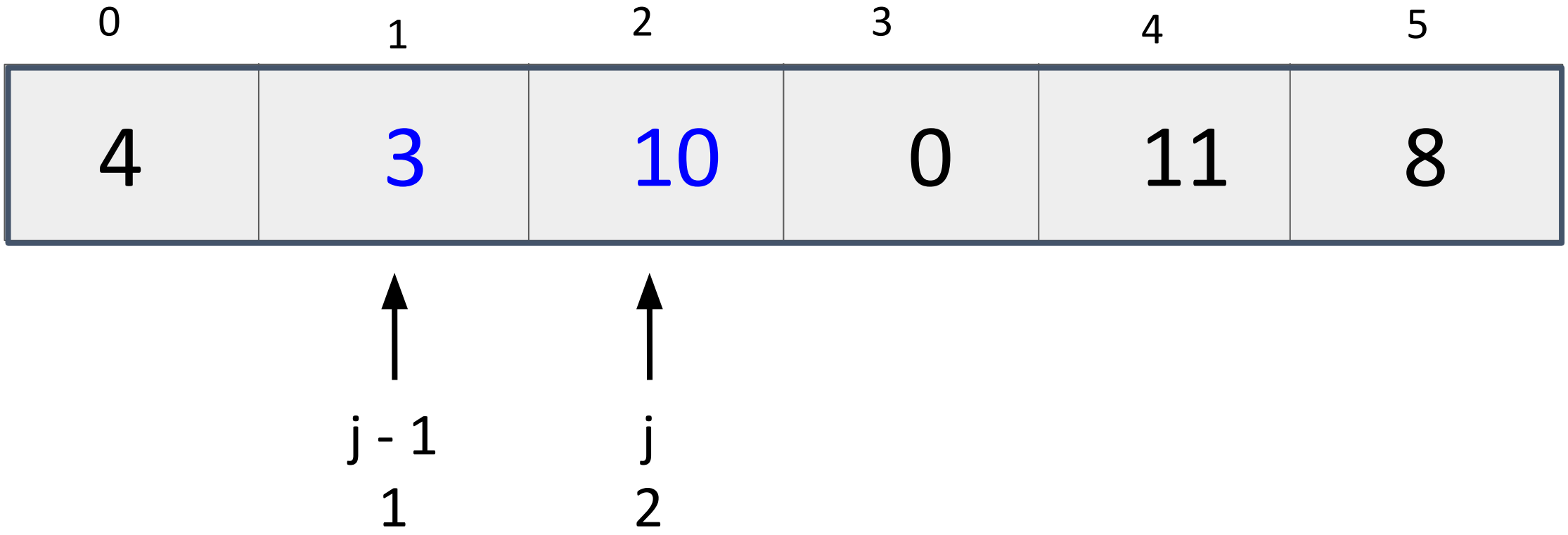


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

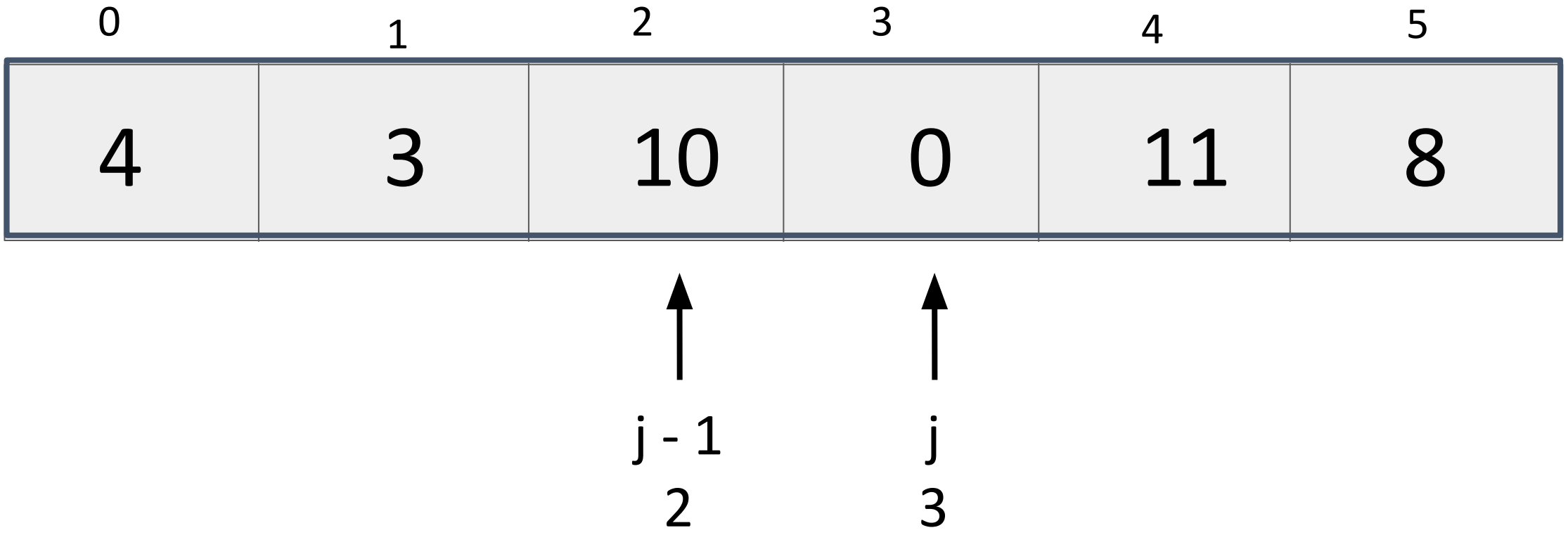


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

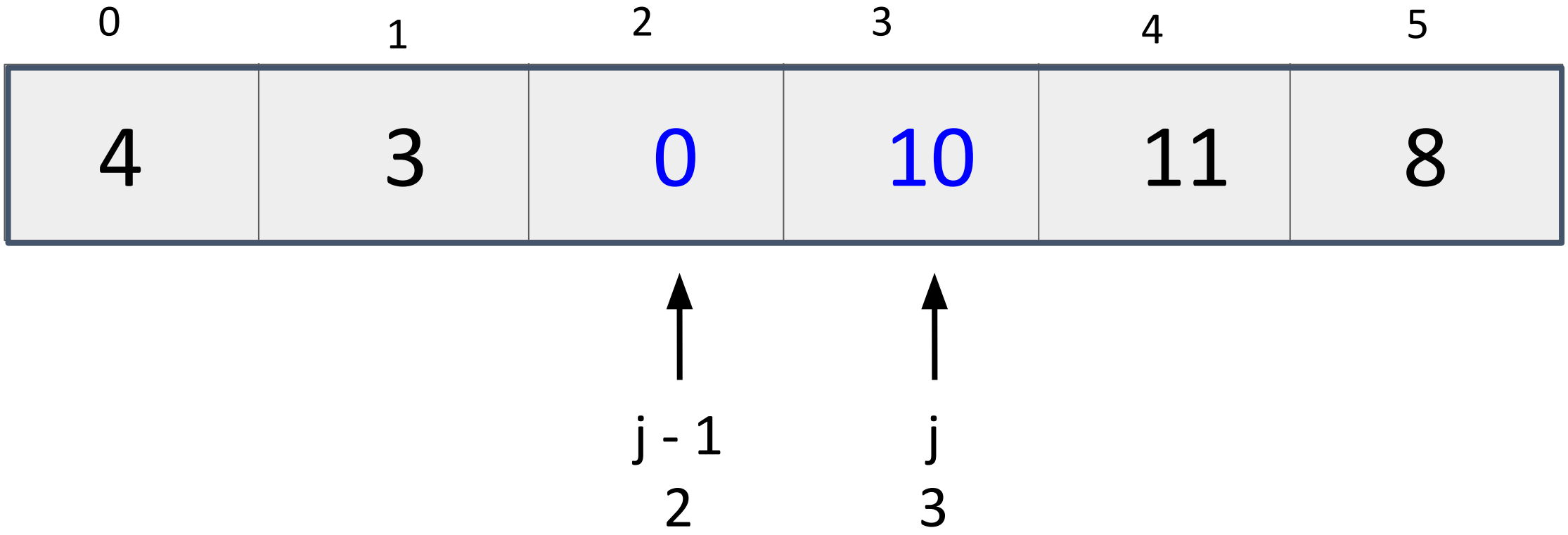


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

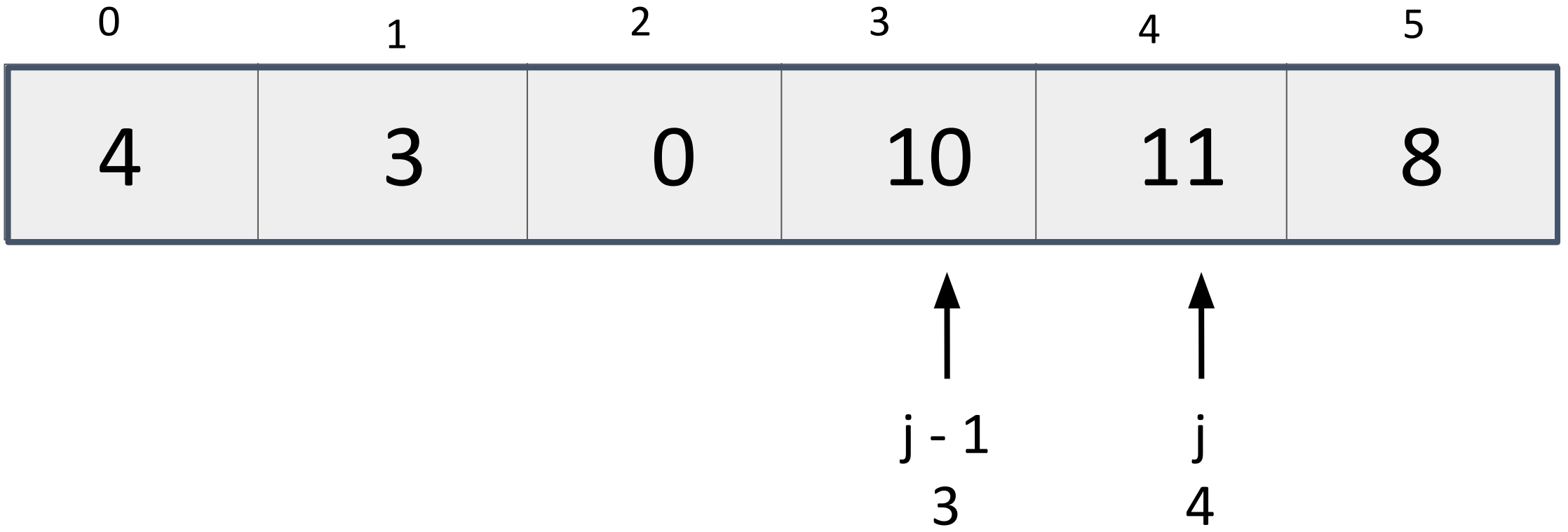


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

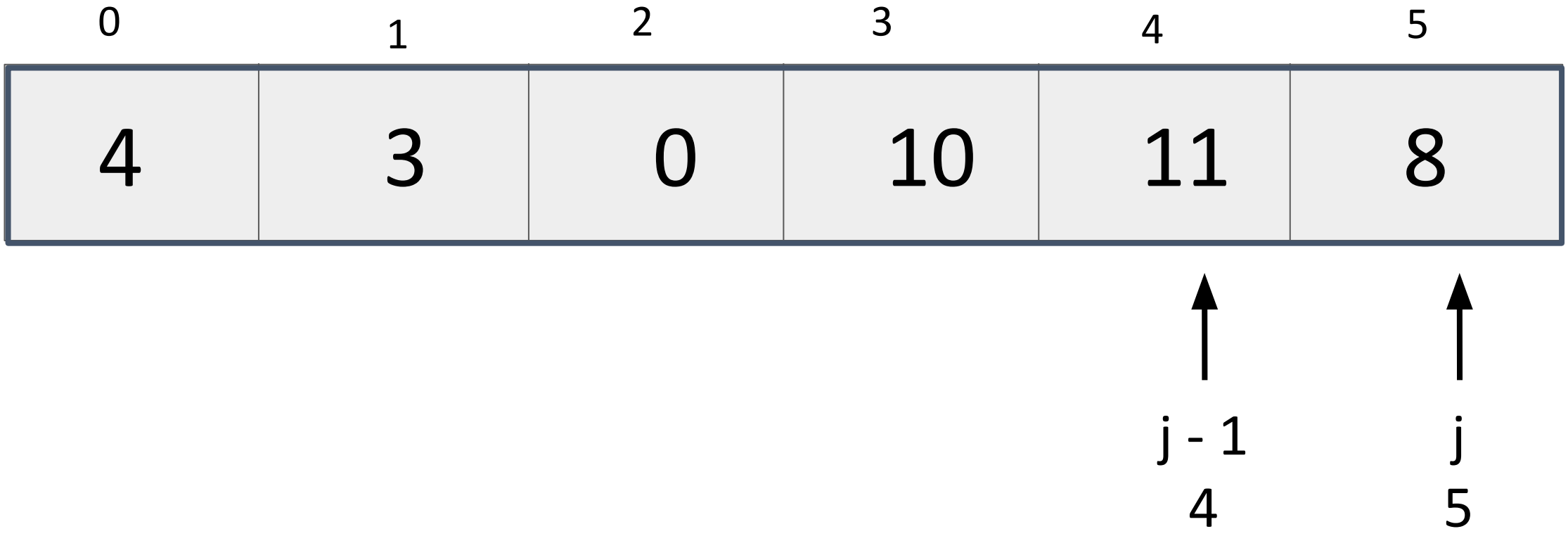


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

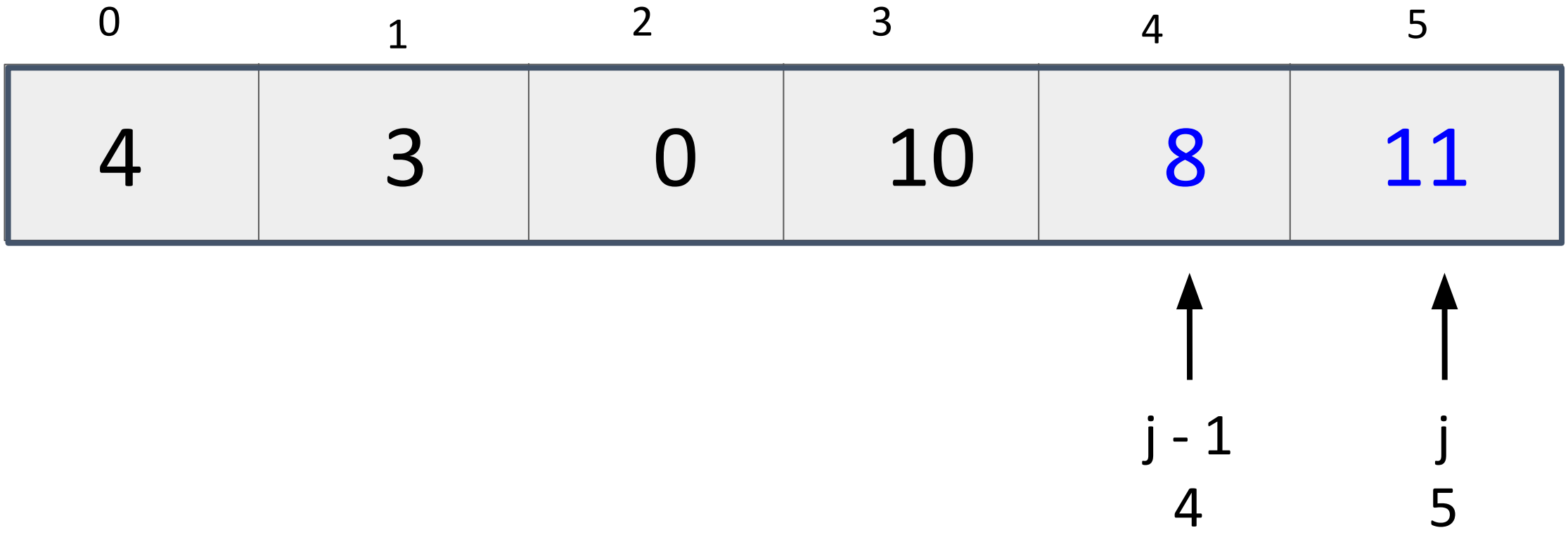


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

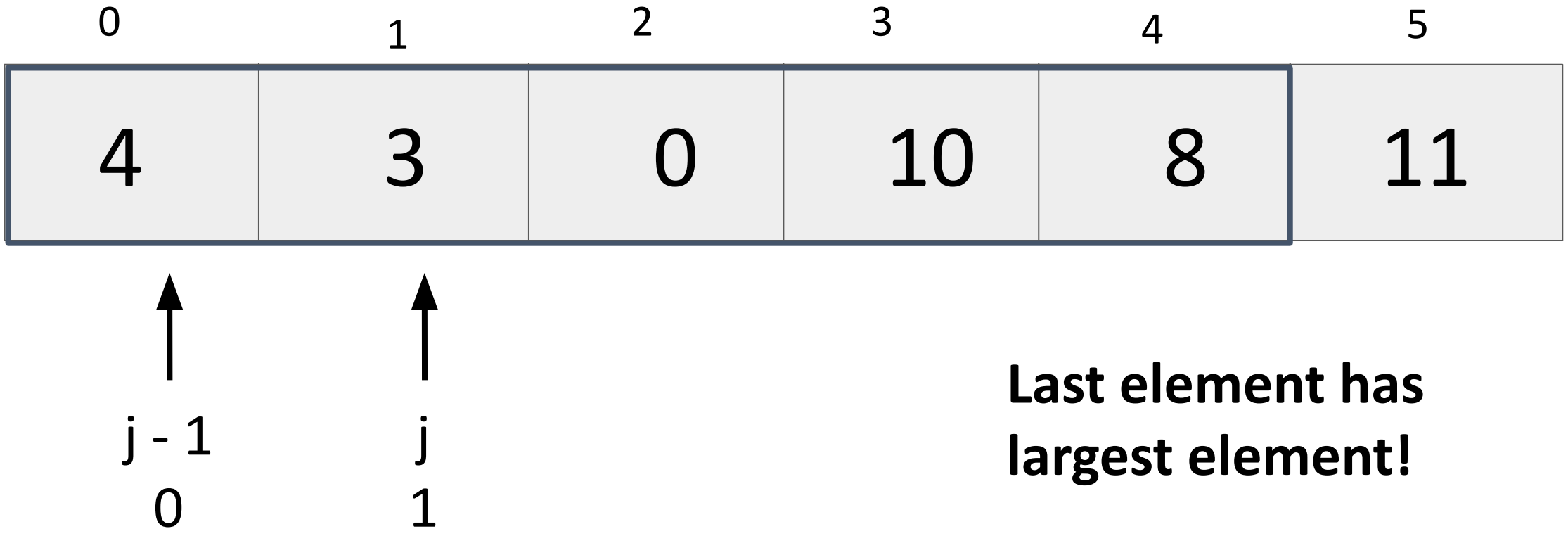


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5



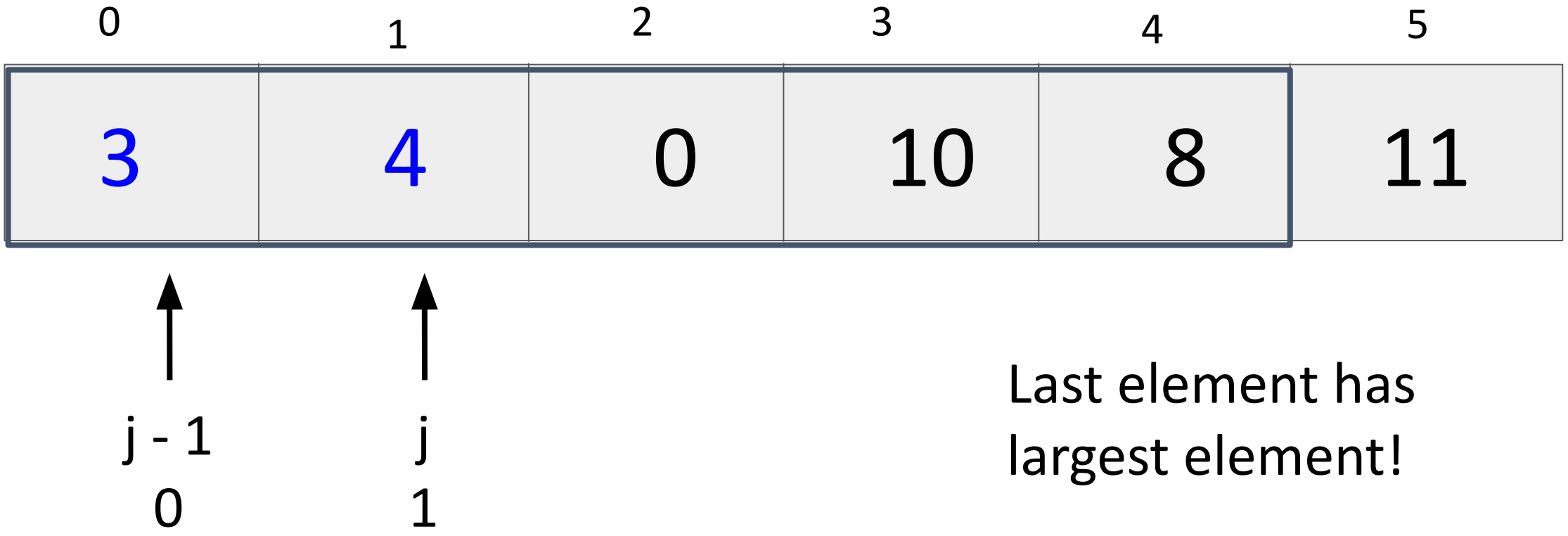
Last element has largest element!

Reset and compare pairs with shorter list!

What next?

Bubble Sort

len = 5



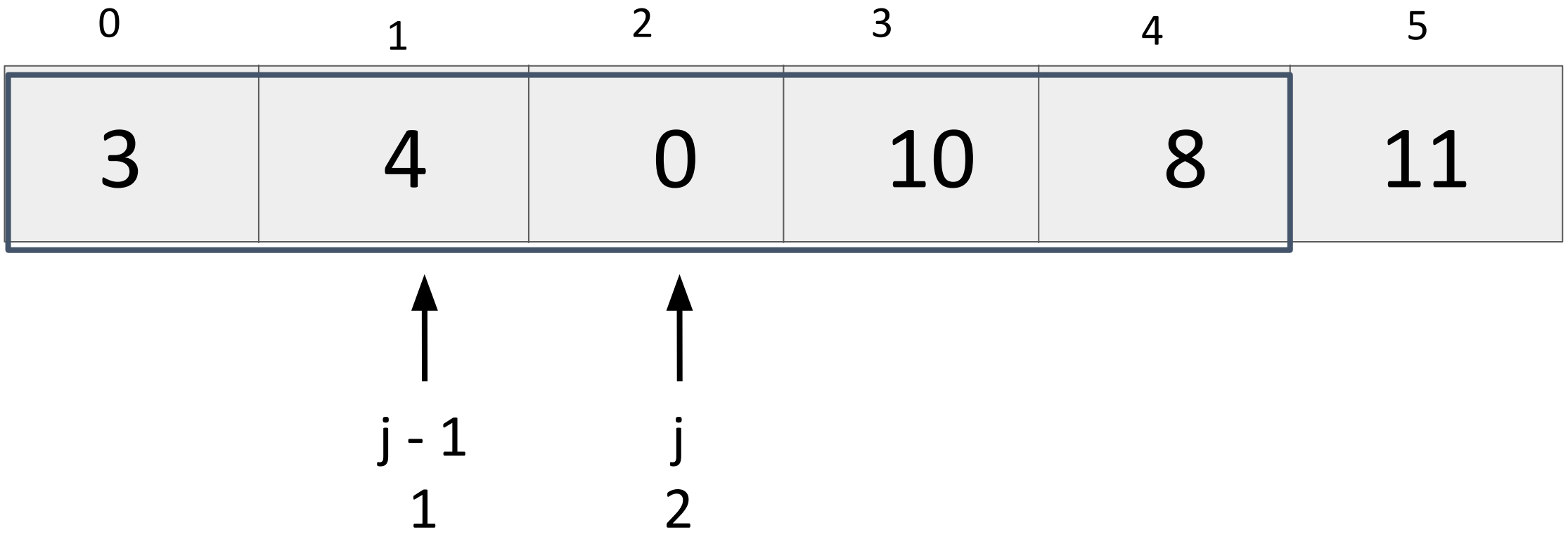
Last element has largest element!

Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5

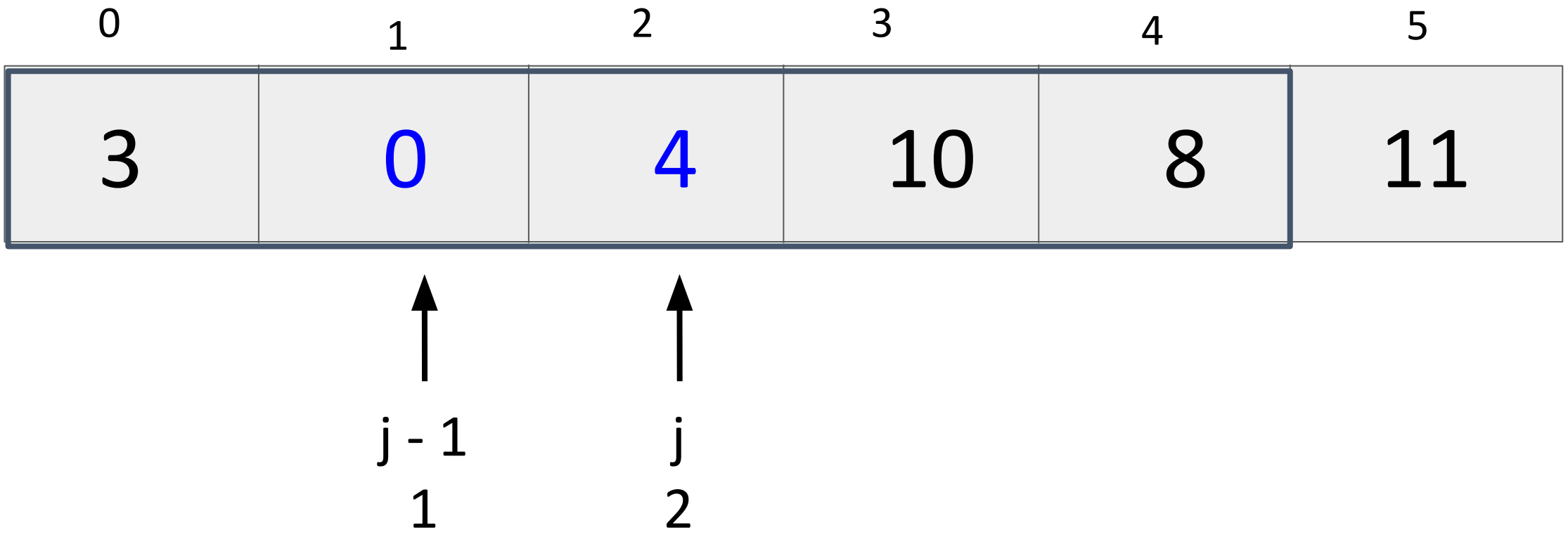


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5

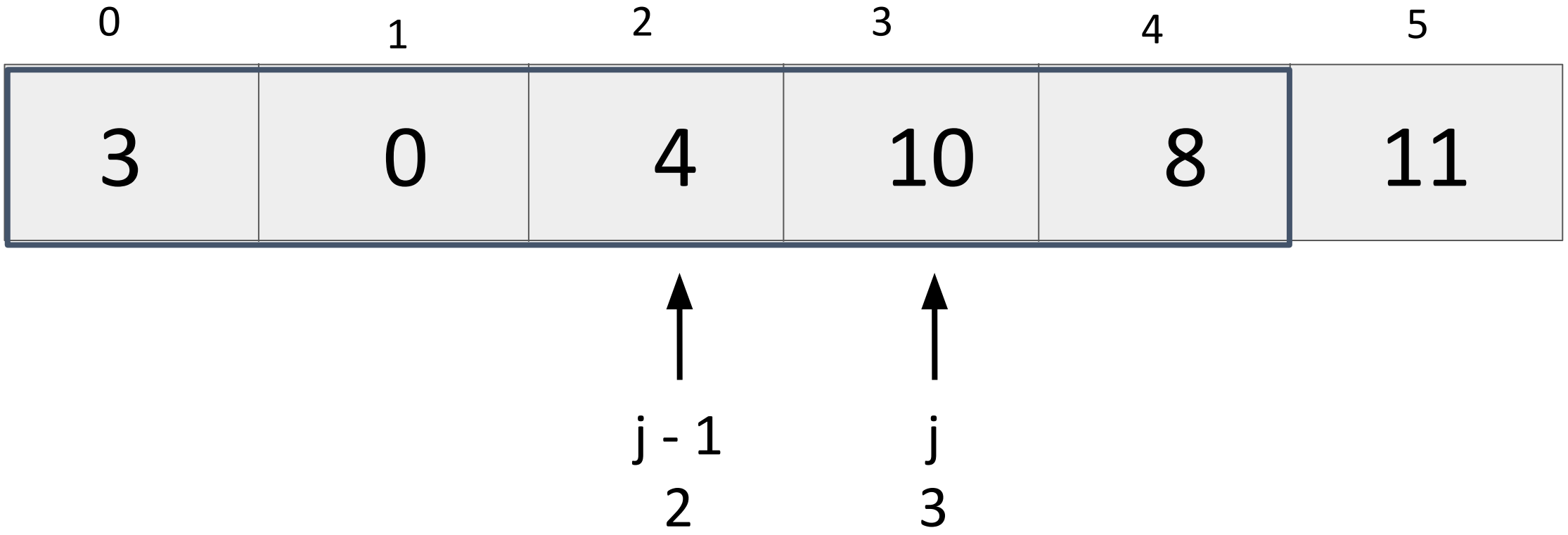


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5

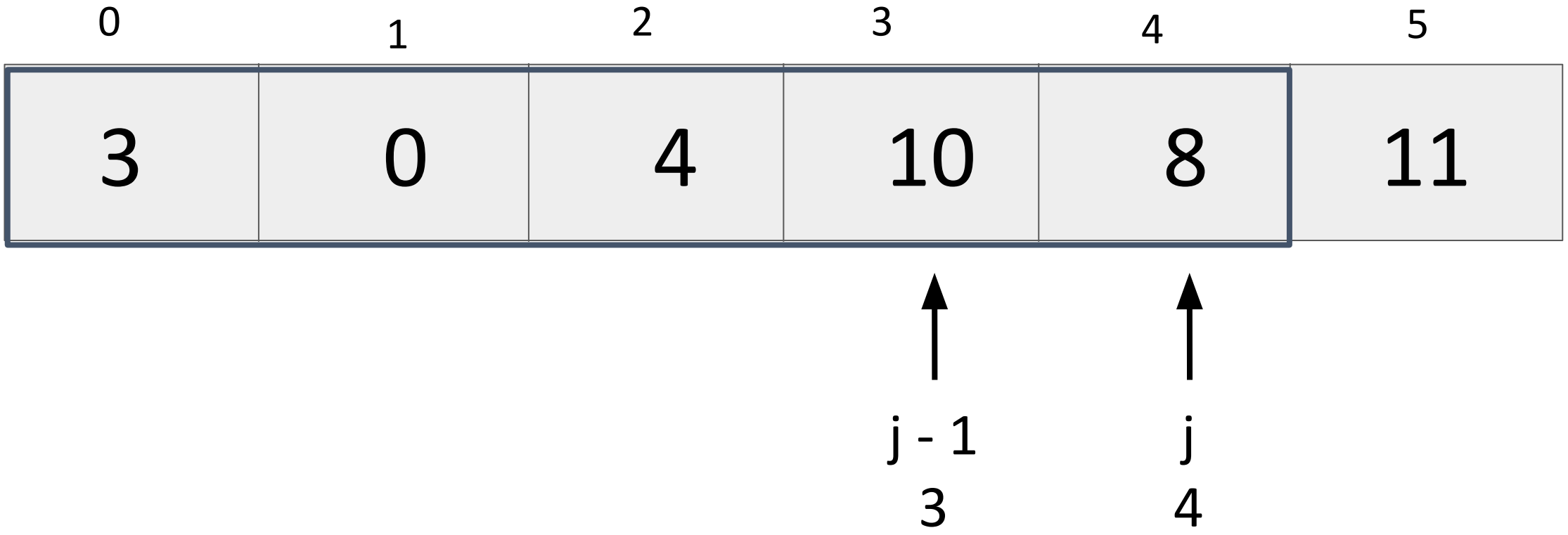


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5

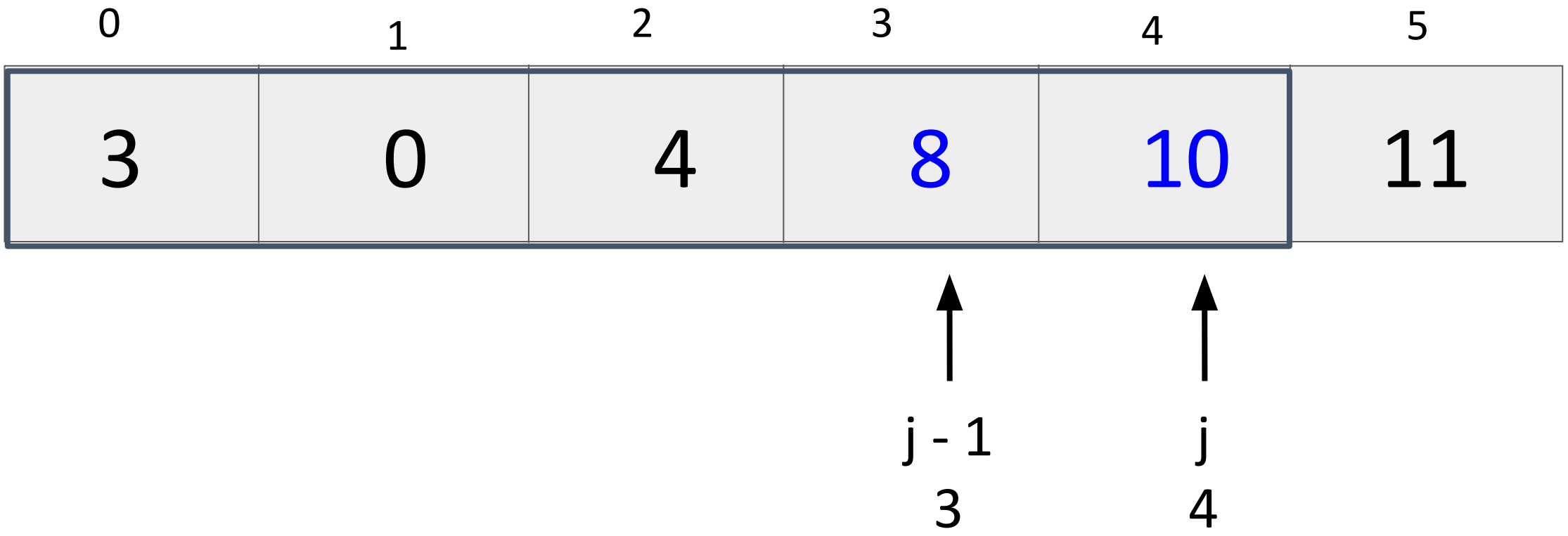


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5

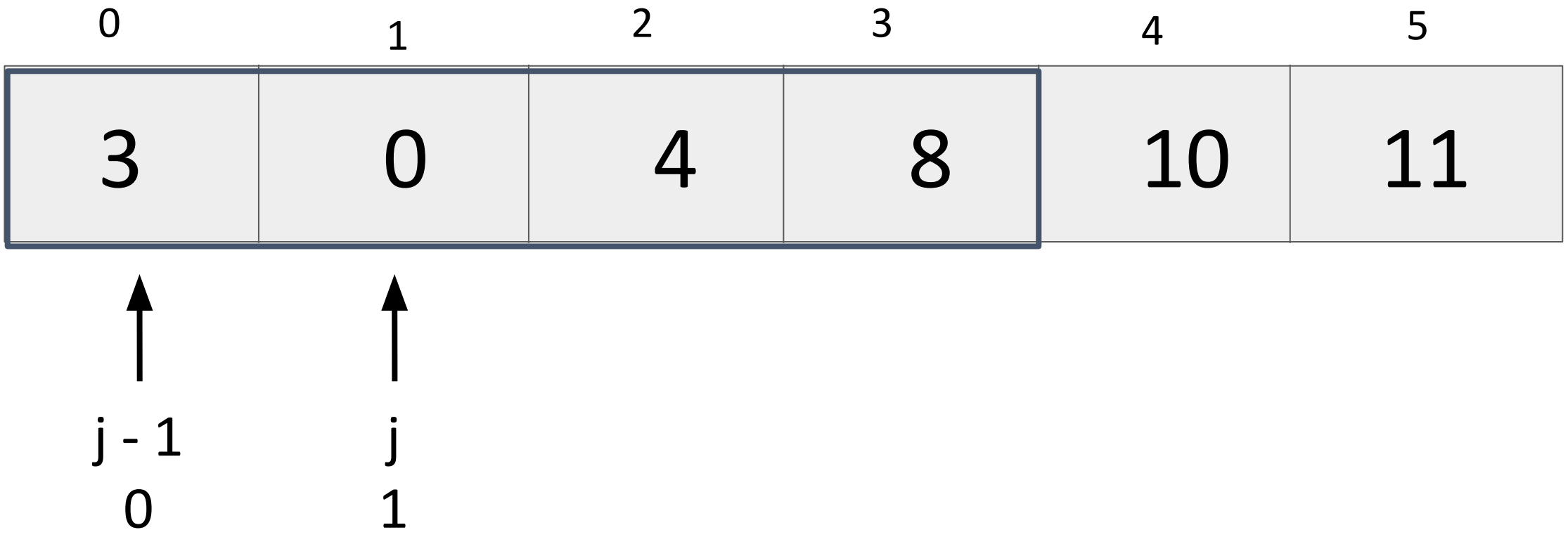


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 4

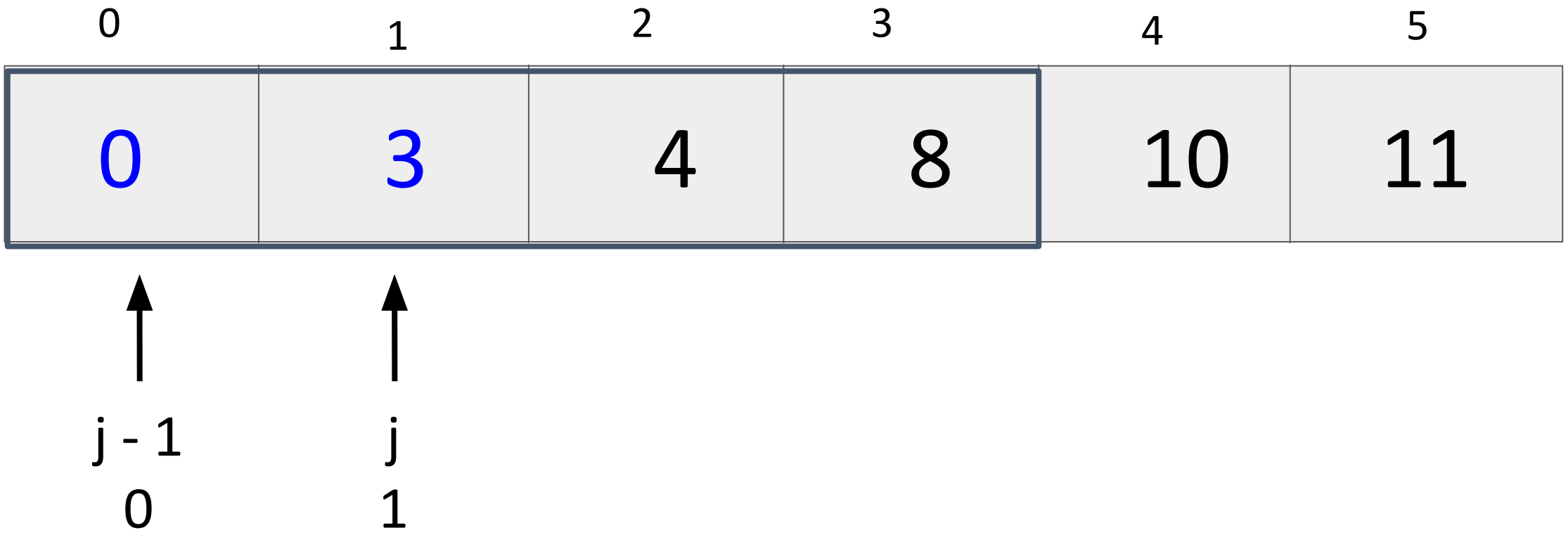


Reset and check pairs with shorter list

What next?

Bubble Sort

len = 4

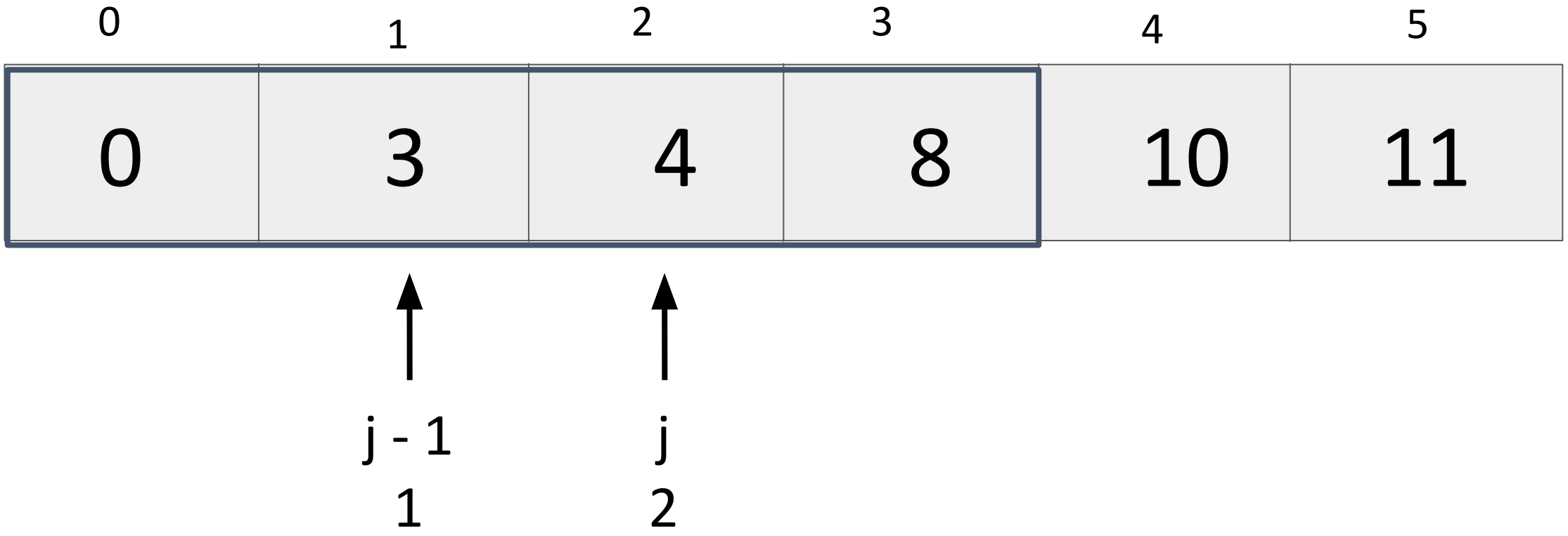


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 4

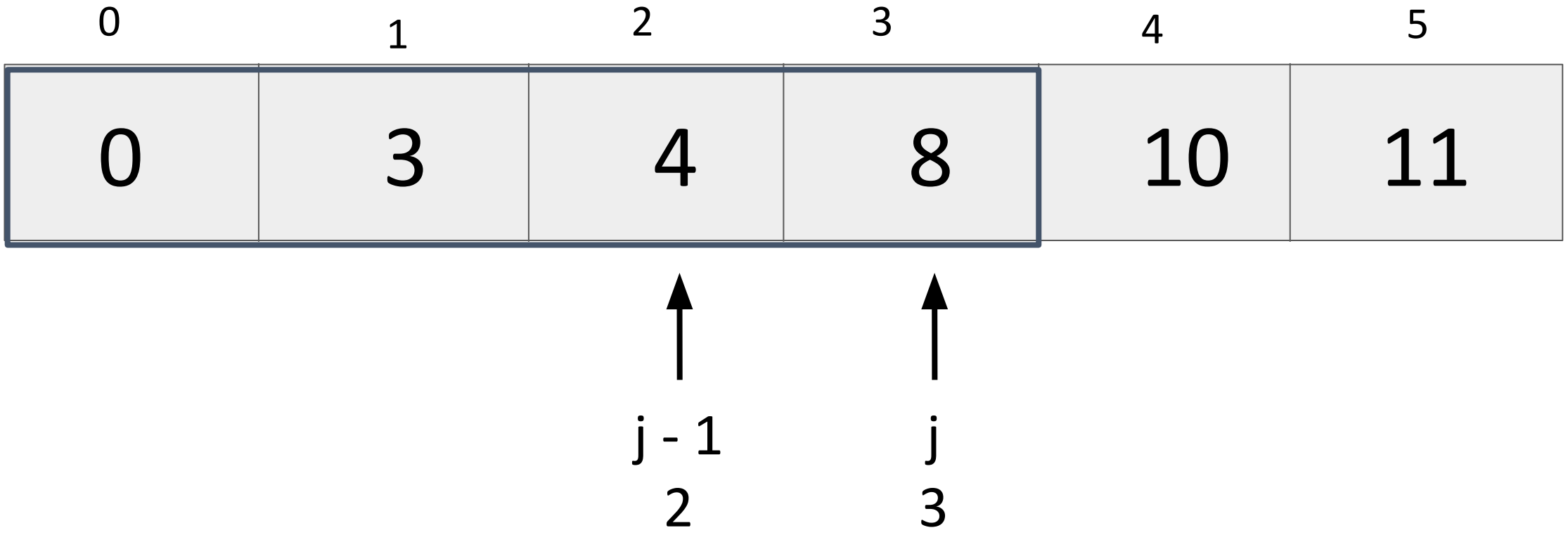


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 4

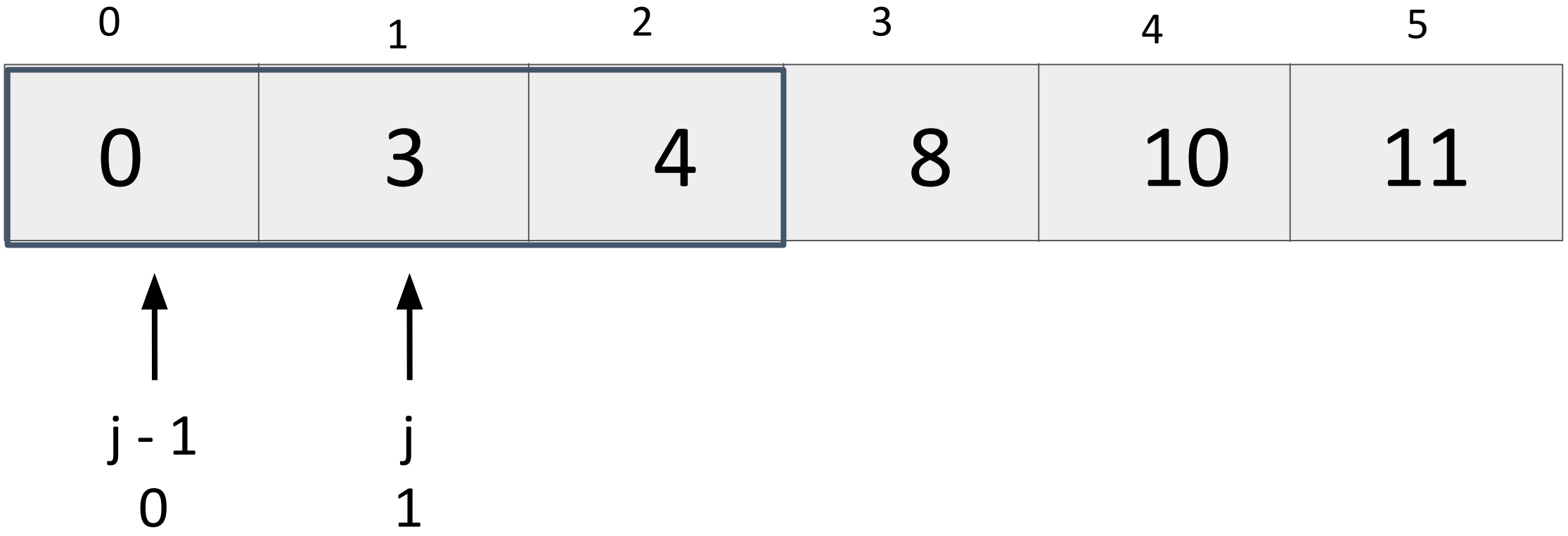


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 3

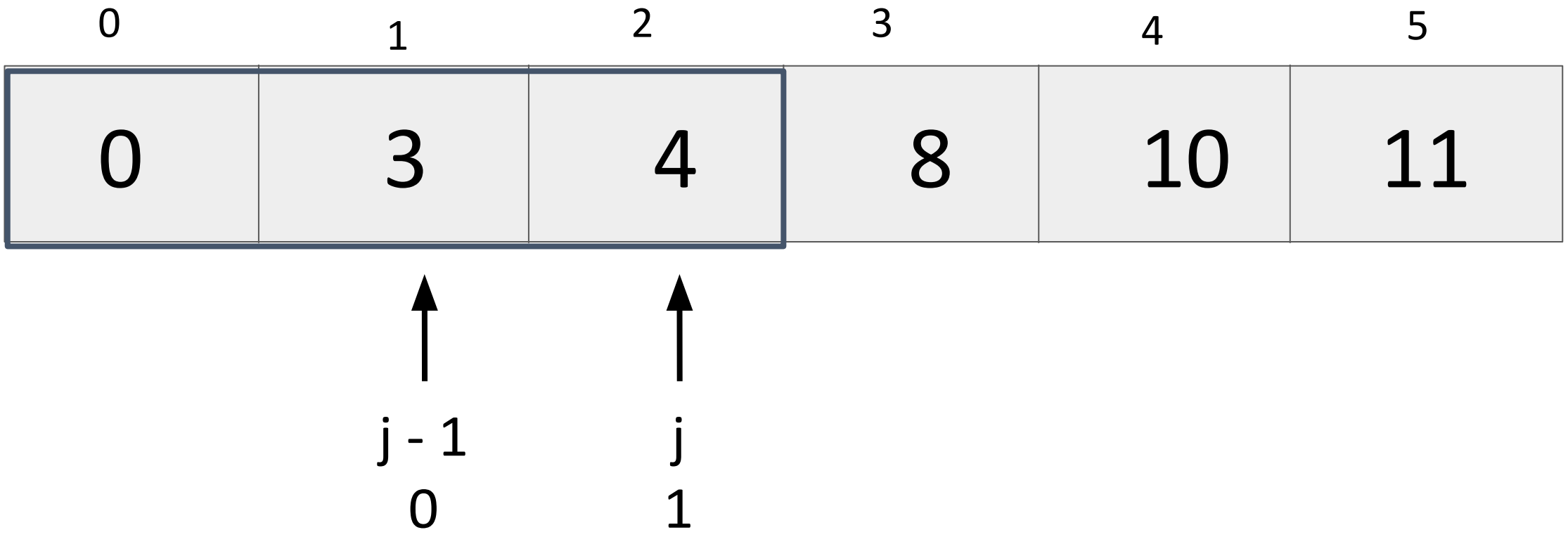


Reset; Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 3

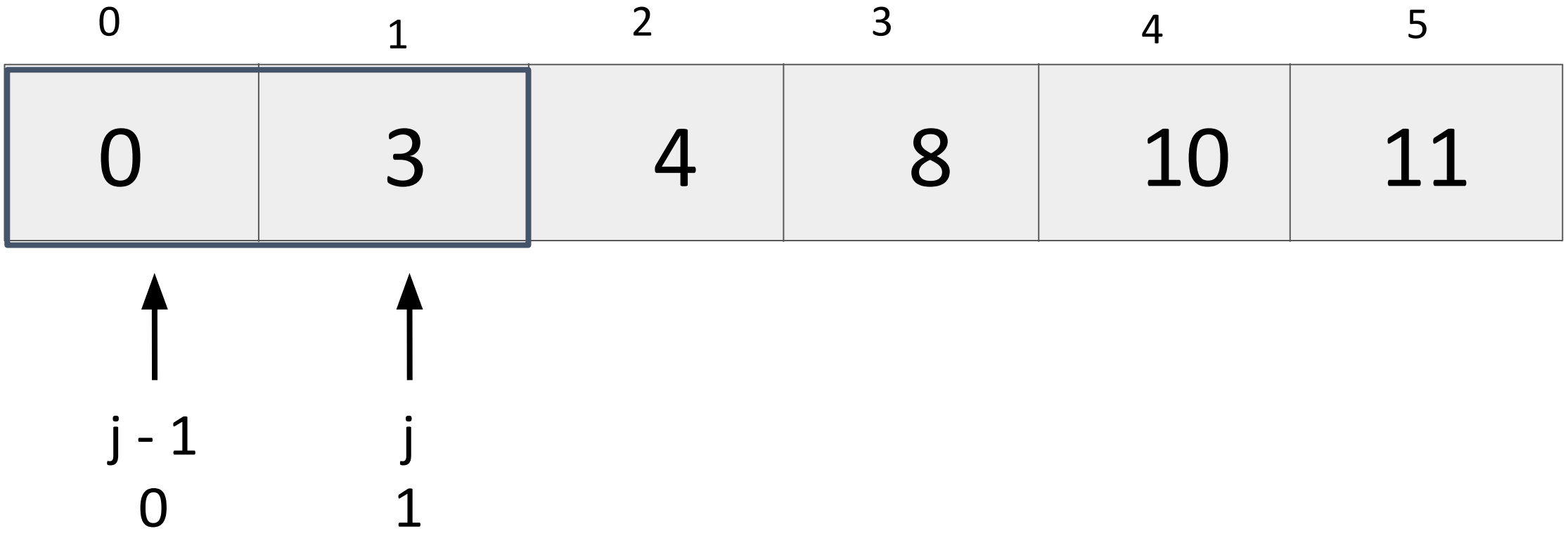


Reset; Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 2



Reset; Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort Implementation

Runtime Complexity

- Linear search grows linearly
- Binary search grows logarithmically

- Complexity of bubble sort?
  - How many comparisons do we make if there are 5 elements?
  - What about 10 elements?
  - Grows quadratically!

Selection Sort

Selection Sort

In place sorting algorithm

1. Separate the array into “sorted” and “unsorted”
   a. sorted starts empty
2. Find the min element in the unsorted array
3. Swap min with the first element in unsorted
4. repeat

Selection Sort Implementation

Runtime Complexity

- Linear search grows linearly
- Binary search grows logarithmically
- Bubble sort grows quadratically

- Complexity of selection sort?

Selection sort and Bubble sort are $O(N^2)$