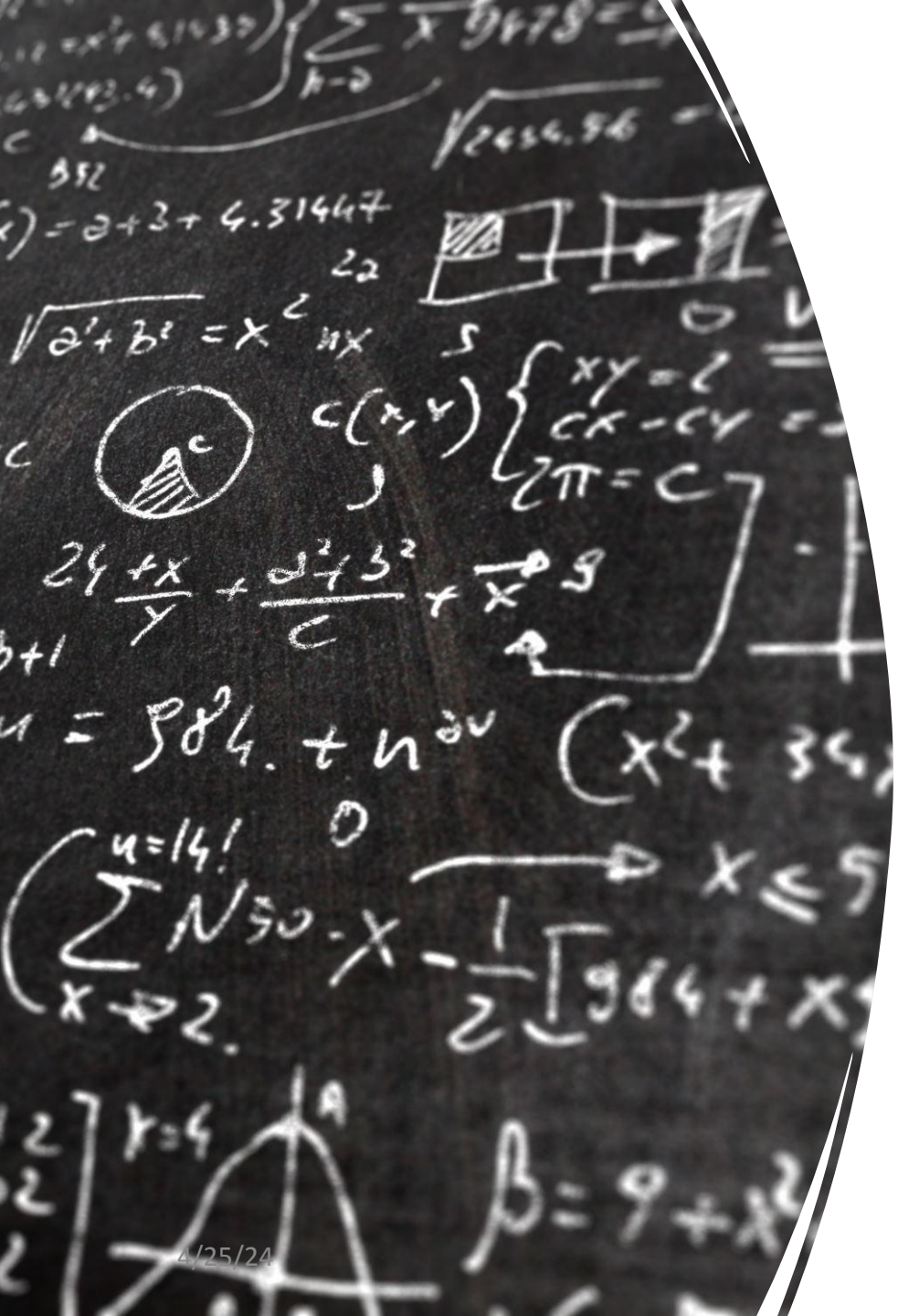


CS 113 – Computer Science I

Lecture 25 – Runtime

Thursday 4/25/2024

Announcements

HW10:

- Released tonight
- Due May 2nd?

Timing Programs

Two ways:

1. Unix time (command line)
2. In Java

Unix time

```
time java YourProgram
```

The `time` command in Unix is used to measure the elapsed time of commands and programs. When you execute a command or program with `time`, it will output three different times:

1. **Real Time**: actual elapsed time from the start of the command until it ends.
2. **User Time**: amount of CPU time spent executing the command in **user mode.**
3. **System Time**: amount of CPU time spent by the kernel on behalf of the command in **kernel mode.**

Java Timing

```
long startTime = System.currentTimeMillis();
//code to time
long endTime = System.currentTimeMillis();
long elapsed = endTime - startTime;
```

- Can get more fine grained timing on different sections of your code
- Measured in milliseconds

Runtime Analysis: Big O Notation

- Mathematical notation used to describe the performance or complexity of an algorithm.
- Hardware independent
- Represents the upper bound of the time complexity in the worst-case scenario.
- Helps us understand how the runtime of an algorithm grows ***as the input size increases.***

Constant Time Operations

What are some operations that are independent of the input size?

- Assignments
- Declarations
- Accessing an index in an array
- arr.length;
- mul, divide, sub, add, mod, etc
- greater than, less than, etc
- printing
- if (x > 100)

Big-O Example 1

```
int n = Integer.parseInt(args[0]);
int sum = 0;
int i = 0;

while (i < n) {
   sum = sum + i;
   i++;
}

System.out.println(sum);
```

How does the runtime grow as a function of the input size?

Linearly!

O(n)

Big-O Example 2

```
int n = Integer.parseInt(args[0]);
int tot = 0;
int i = 0;

while (i < n) {
   tot = tot * i;
   i++;
}

System.out.println(tot);
```

How does the runtime grow as a function of the input size?

Linearly!

O(n)

Big-O Example 3

```
int n = Integer.parseInt(args[0]);
int tot = 0;
int sum = 0;
int i = 0;

while (i < n) {
   tot = tot * i;
   i++;
}

while (i < n) {
   sum = sum + i;
   i++;
}
```

How does the runtime grow as a function of the input size?

Linearly!

O(n + n) = O(2n) = O(n)

We care about the asymptotic case... The n factor dominates.

Big-O Example 4

```
int n = Integer.parseInt(args[0]);

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
       System.out.println(i, j);
    }
}
```

How does the runtime grow as a function of the input size?

Quadratically!

O(n^2)

We do n operations n times

Big-O Example 5

```
int[] lst =
   {1,2,3,5,7,12,19,34,55,67,99,101};

int n = lst.length;
int mid = floor(n/2);
System.out.println(lst[mid]);
```

How does the runtime grow as a function of the size of lst?

Constant! The runtime is not affected by the number of elements in `lst`

O(1)

Big-O Example 6

```
int n = Integer.parseInt(args[0]);
while (n > 1) {
    println(n);
    n = n/2;
}
```

How does the runtime grow as a function of the size of n?

O(logn)

Big-O Example 7

```
int n = Integer.parseInt(args[0]);
for (int i = 0; i < n; i++) {
    for (int j = 0; j < 10; j++) {
        println(i, j);
    }
}
```

How does the runtime grow as a function of the size of n?

O(n*10) = O(n)

n term dominates as we approach infinity

Big-O Example 8

```
int n = Integer.parseInt(args[0]);
for (int i = 0; i < n; i++) {
   k = n;
   while (k > 1) {
       System.out.println(i, k);
       k = k/2;
   }
}
```

How does the runtime grow as a function of the size of n?

O(n*logn) = O(n)

n term dominates as we approach infinity

Big-O Example 9

```
File f = new File("input.txt");
Scanner sc = new Scanner(f);

while (sc.hasNextLine()) {
    String line = sc.nextLine();
    String[] entries = line.split(" ");

   ...
}
```

How does the runtime grow as a function of the size of input.txt?

O(n) where n is number of lines

Searching

Big-O Example 10: Linear Search

```
public static int search(int x, int[] L) {
    int index = -1;
    for (int i = 0; i < L.length; i++) {
        if (L[i] == x) {
            index = i;
        }
    }
    return index;
}
```

How does the runtime grow as a function of the size of L?

O(n)

Big-O Example 11: Binary Search

```
int binarySearch(int[] arr, int target, int lo, int hi) {
    if (hi < lo) return -1;

    int mid = (hi+lo)/2;

    if (arr[mid] > target) {
        return binarySearch(arr, target, lo, mid-1);
    } else if (arr[mid] < target) {
        return binarySearch(arr, target, mid+1, hi);
    } else {
        return mid;
    }
}
```

How does the runtime grow as a function of the size of arr?

O(log n)

Linear Search vs Binary Search

Search.java

Sorting

For binary search, our array first needs to be sorted

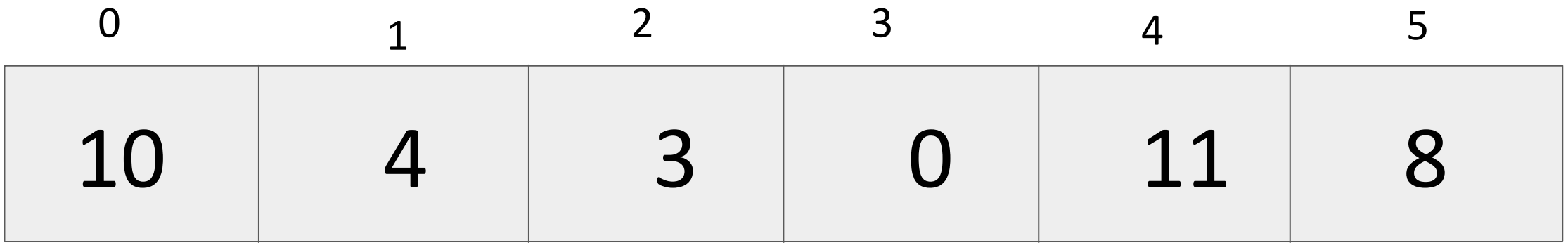
Bubble Sort

- 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

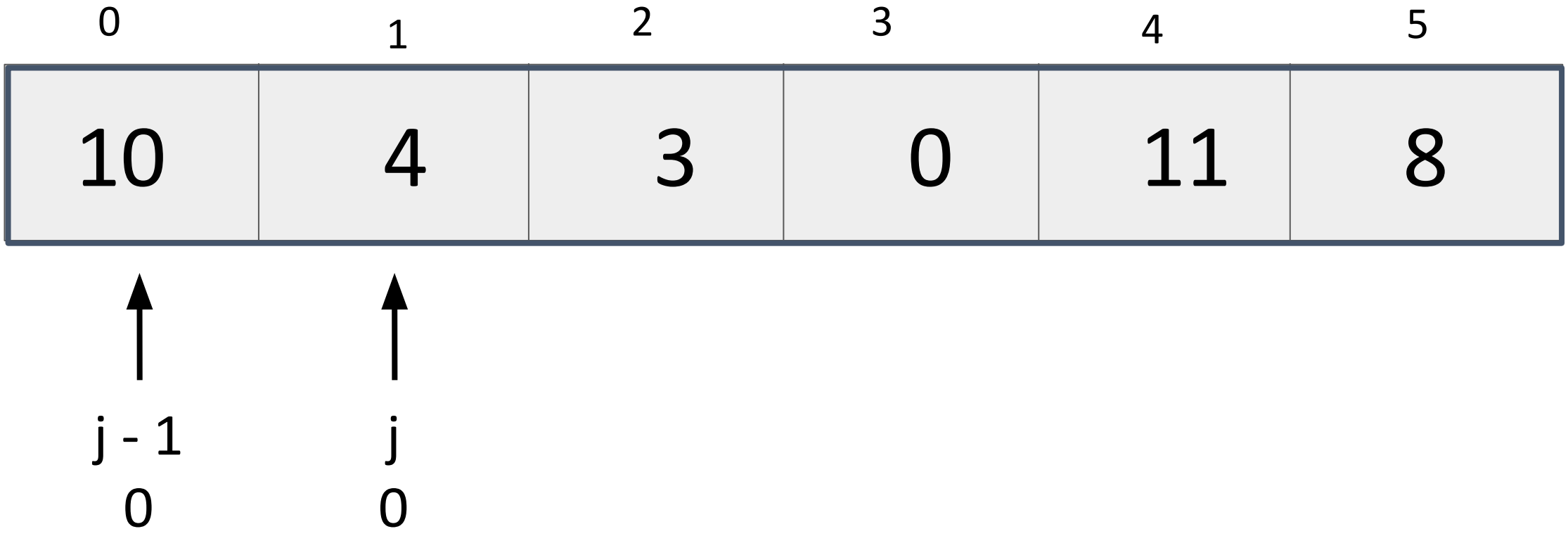
Bubble Sort



What do we do first?

Bubble Sort

len = 6

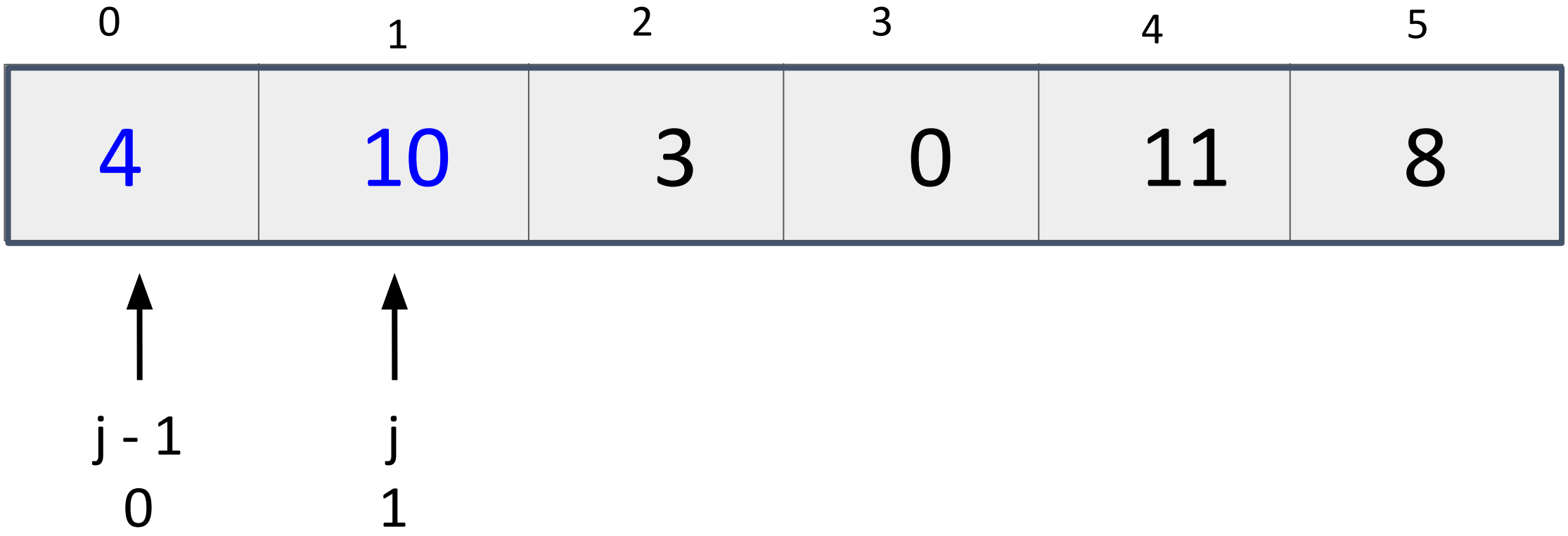


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

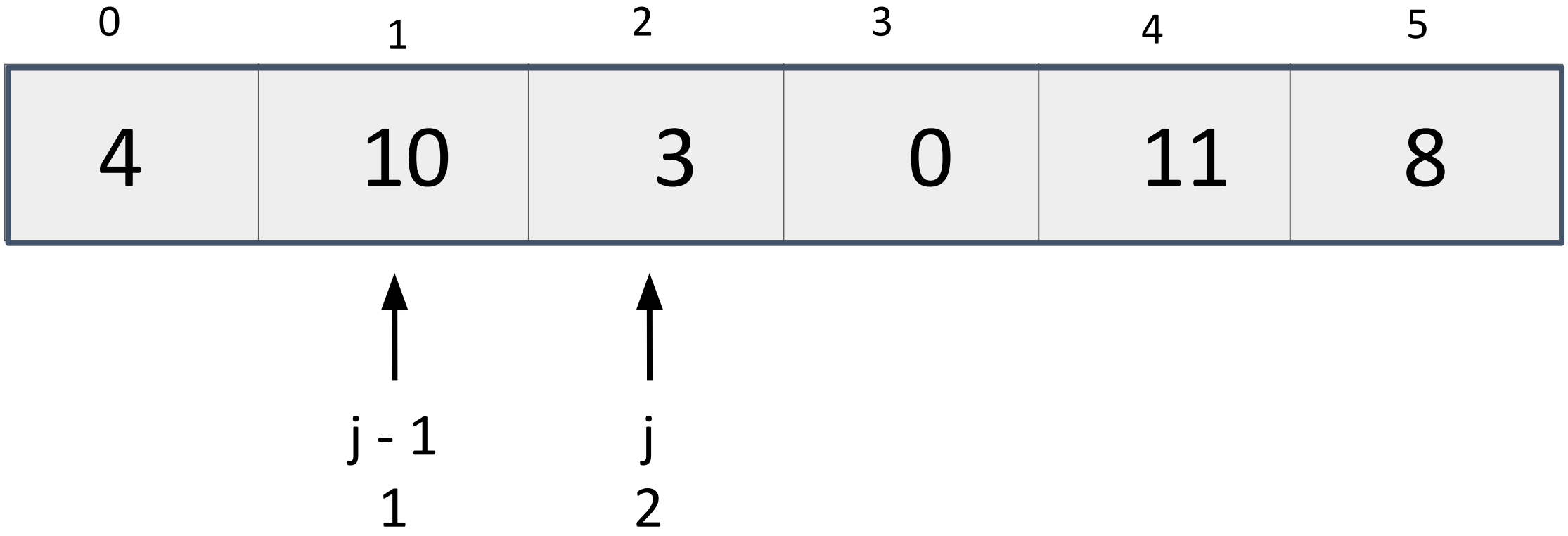


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

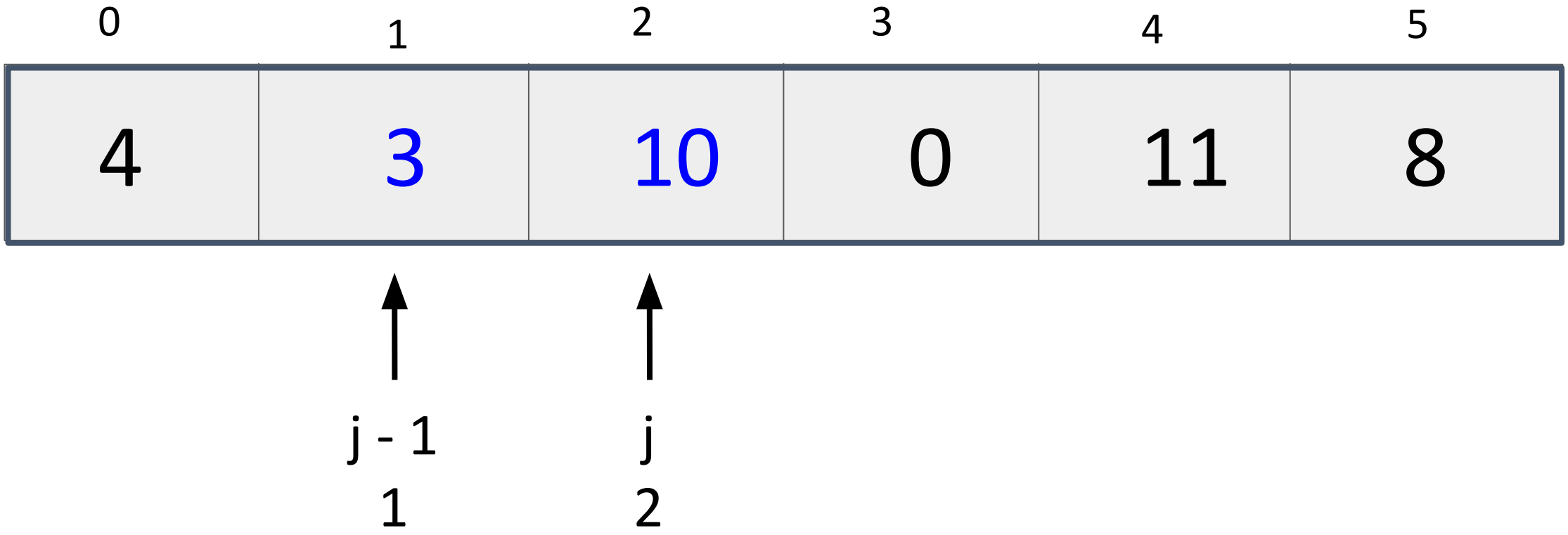


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

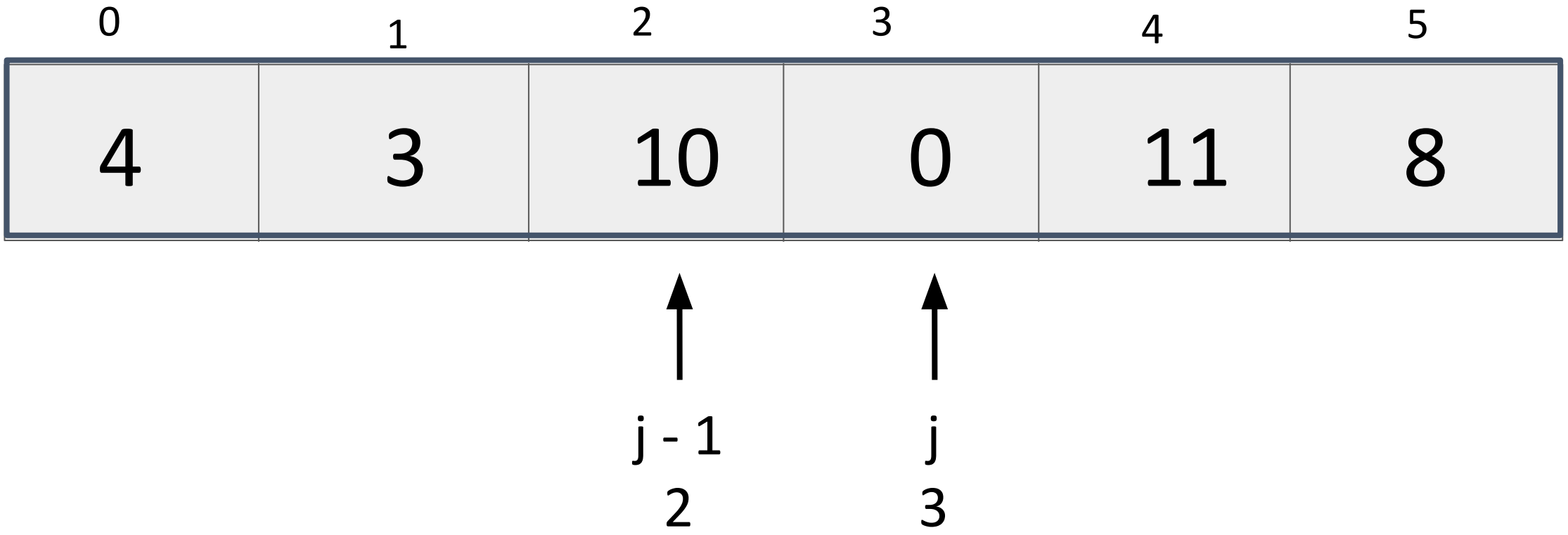


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

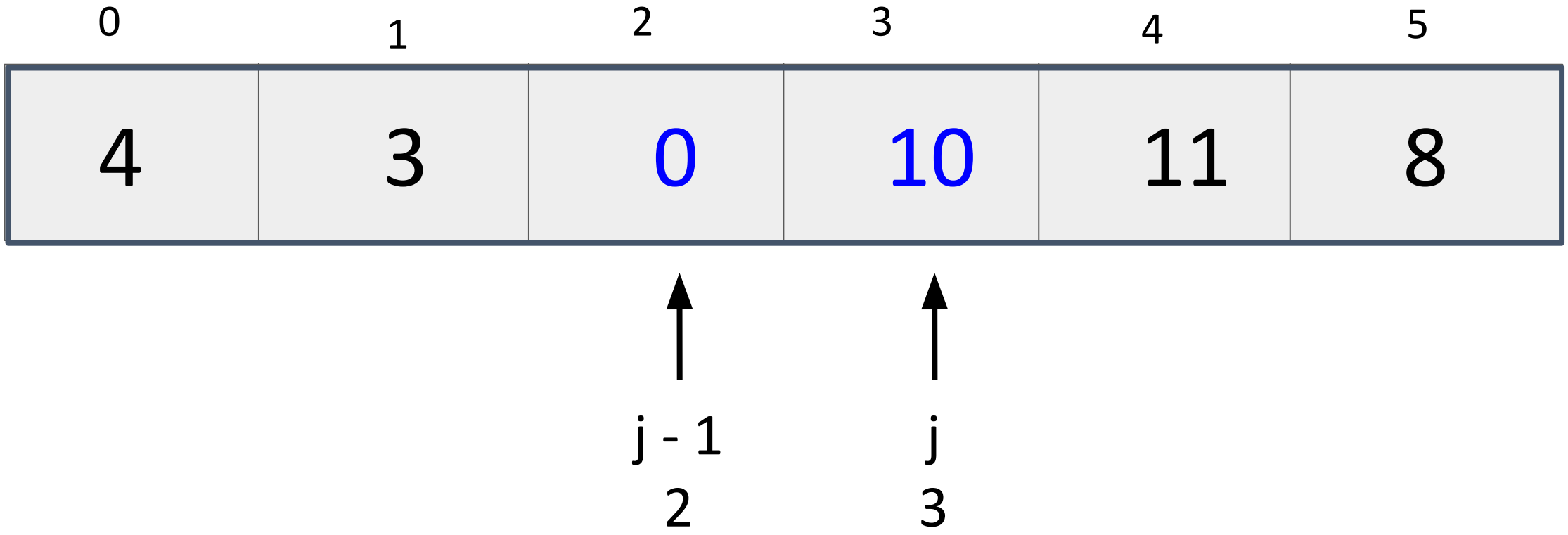


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

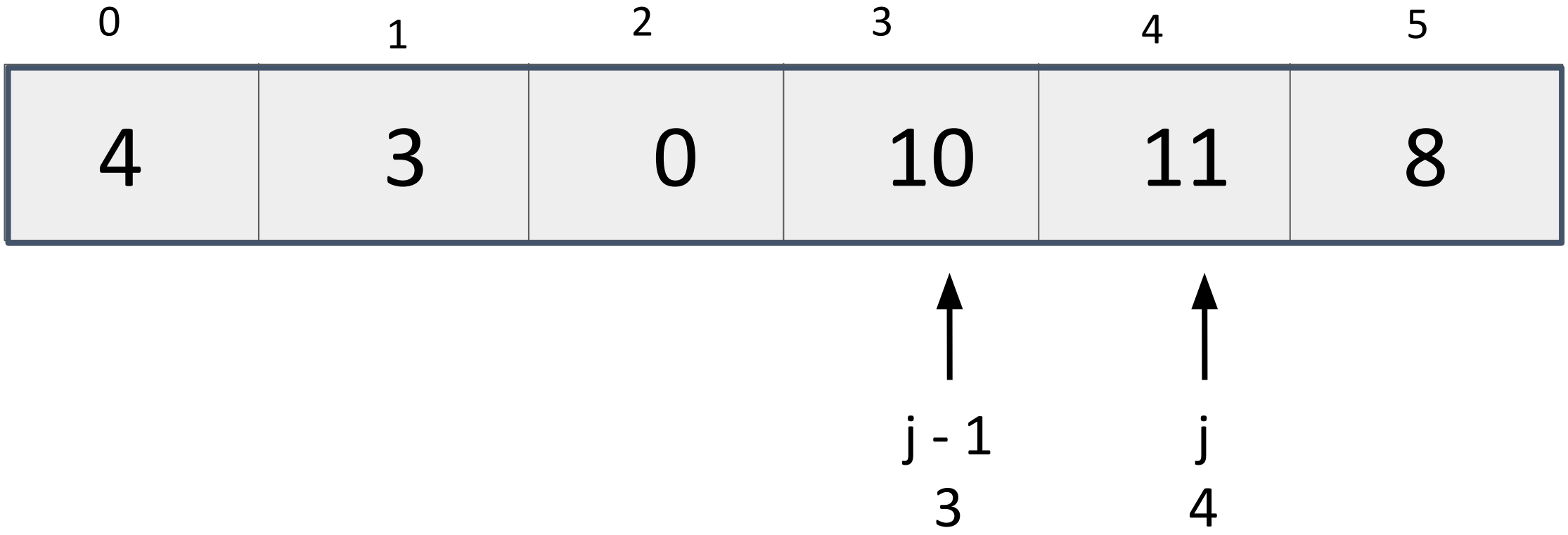


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

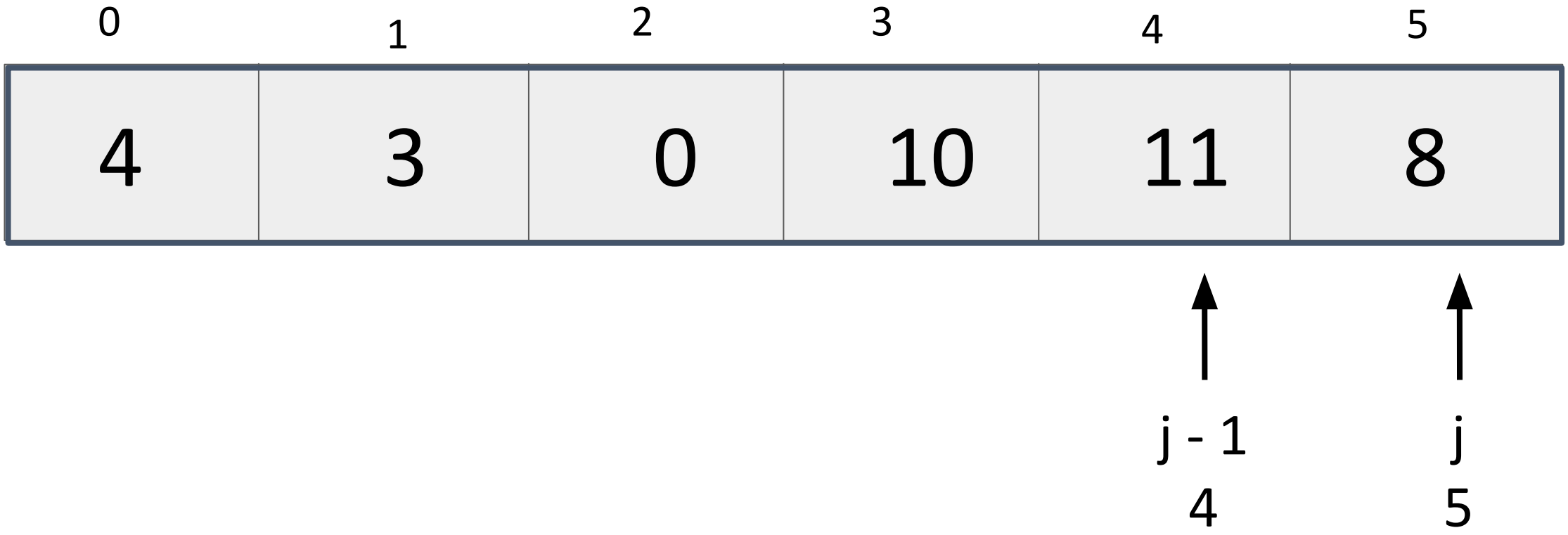


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

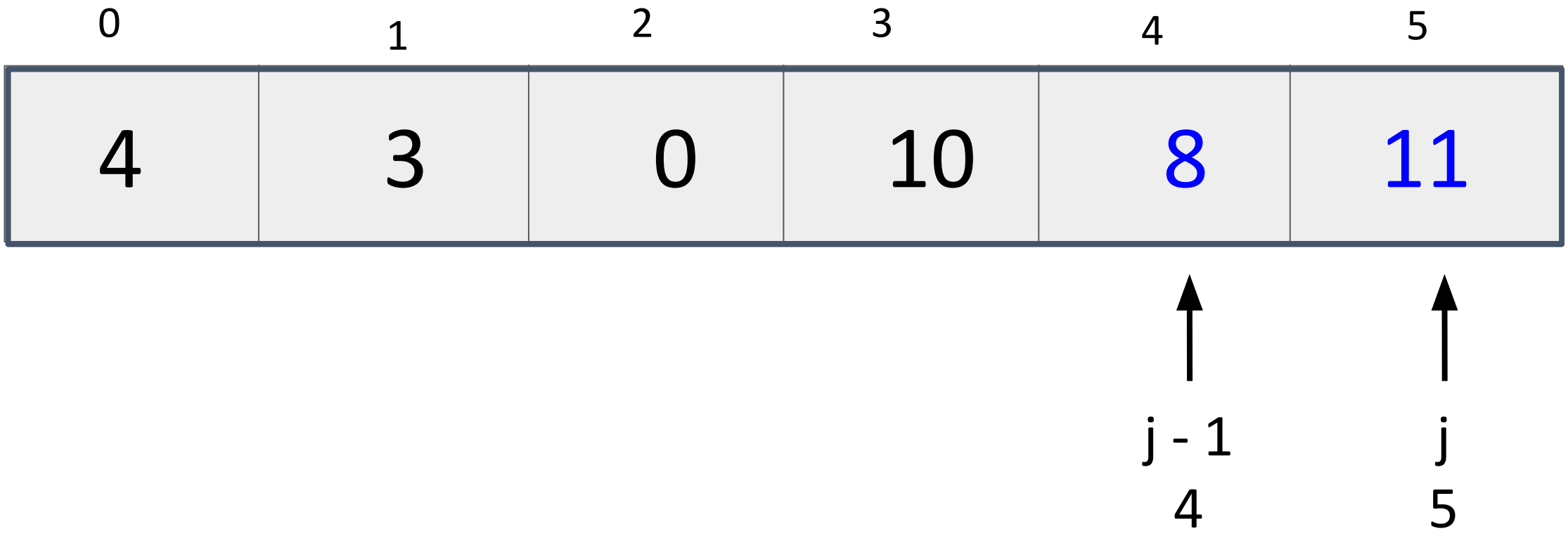


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 6

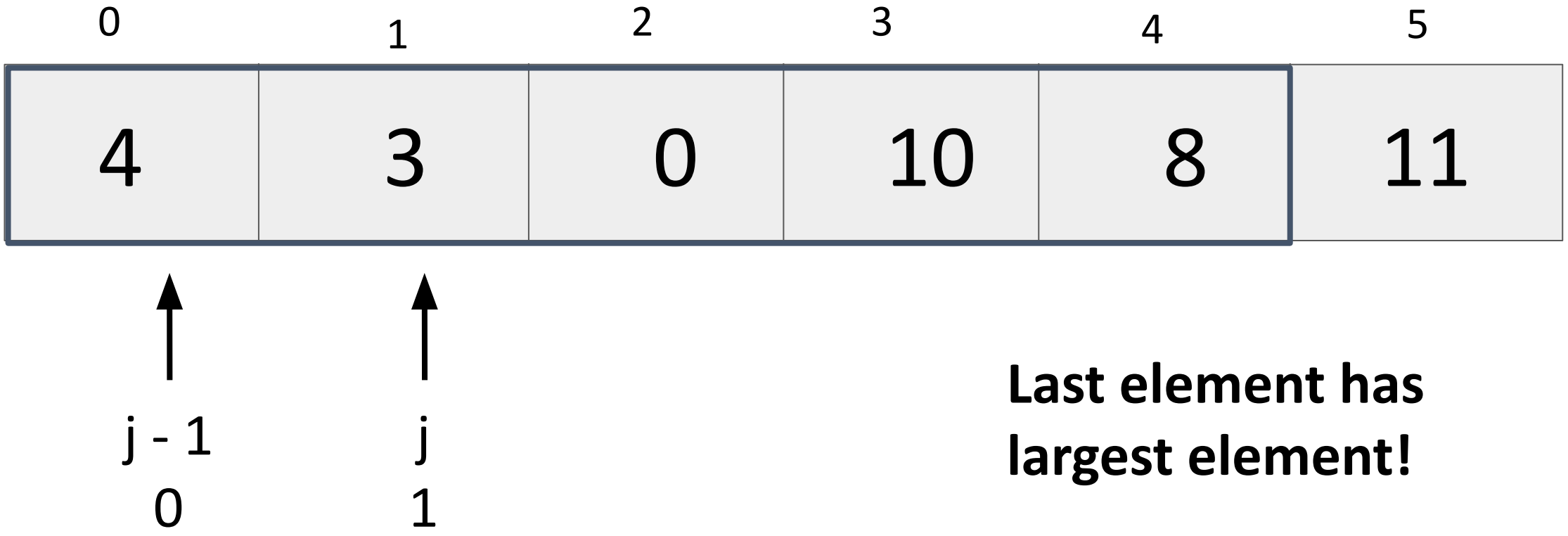


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5



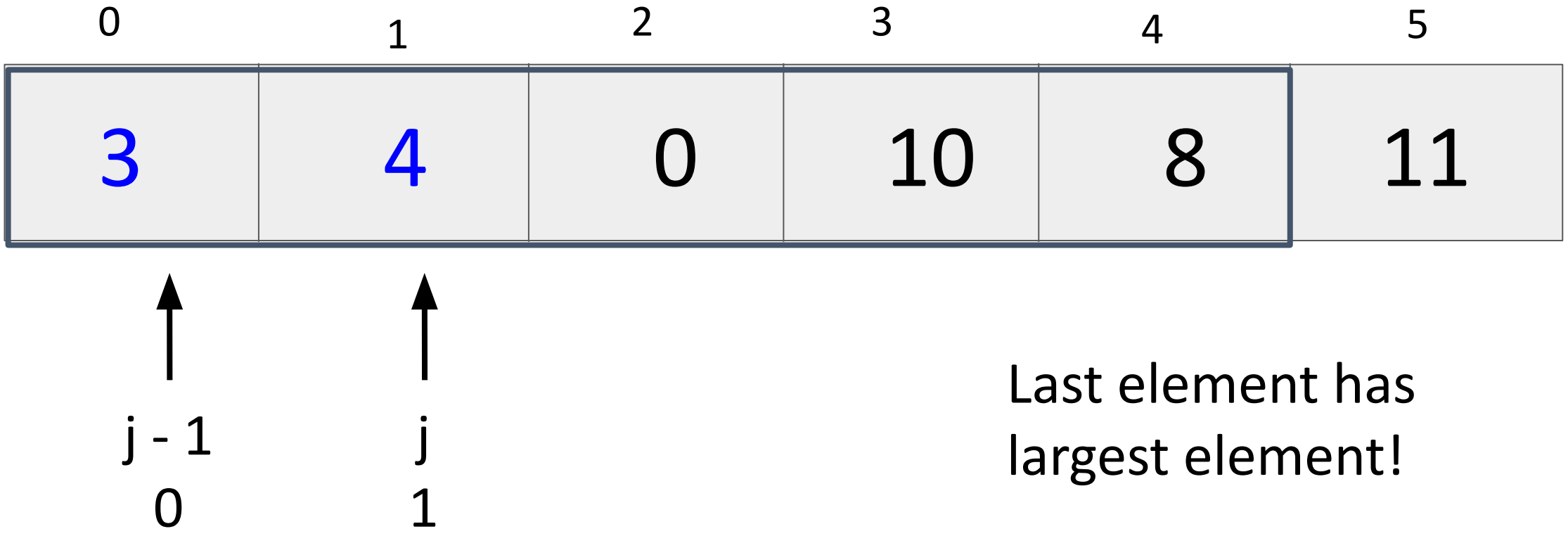
Last element has largest element!

Reset and compare pairs with shorter list!

What next?

Bubble Sort

len = 5



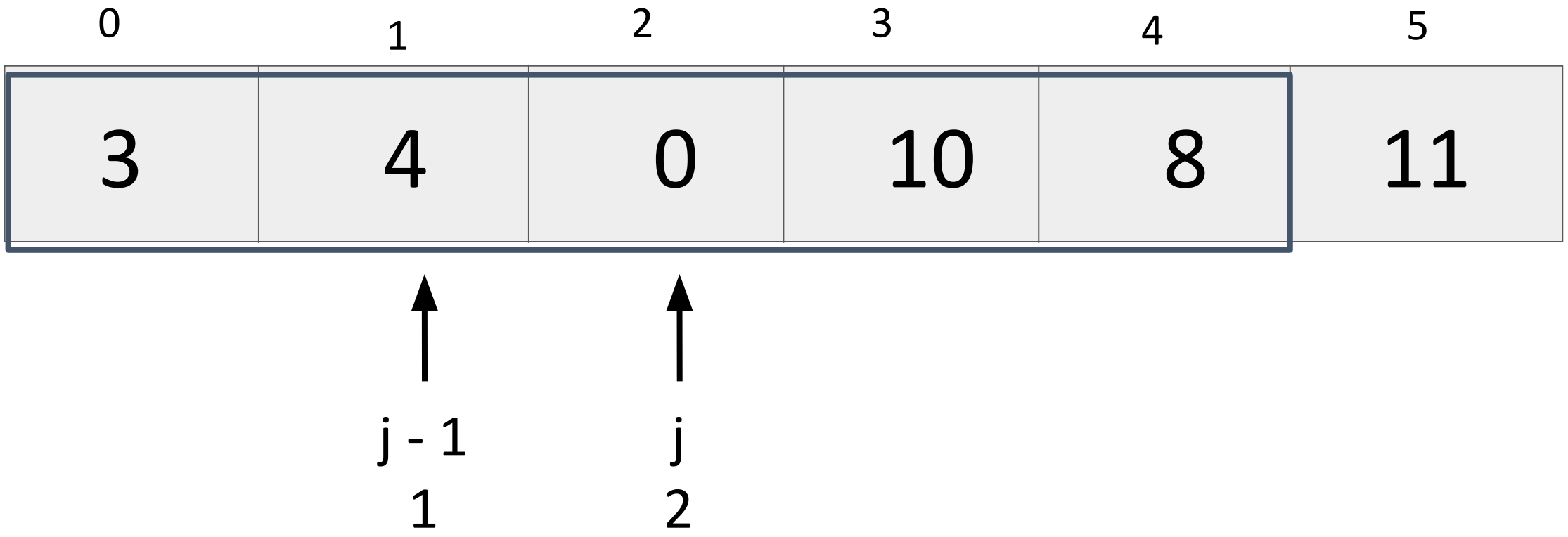
Last element has largest element!

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

What next?

Bubble Sort

len = 5

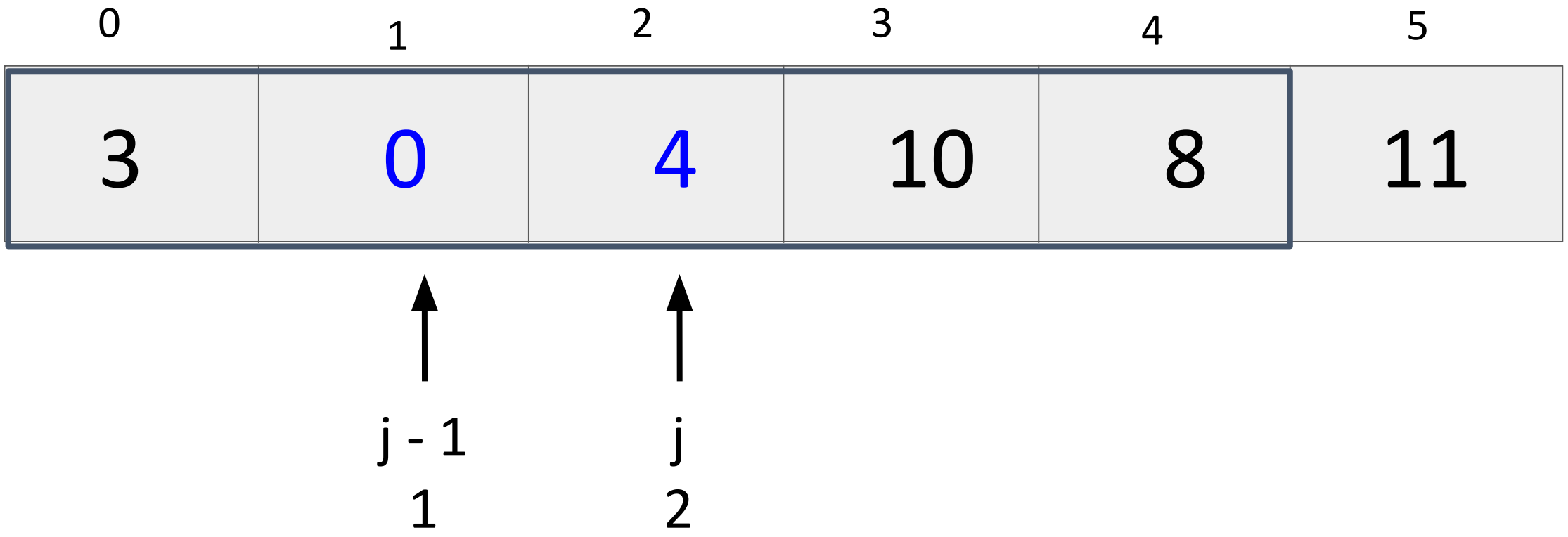


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5

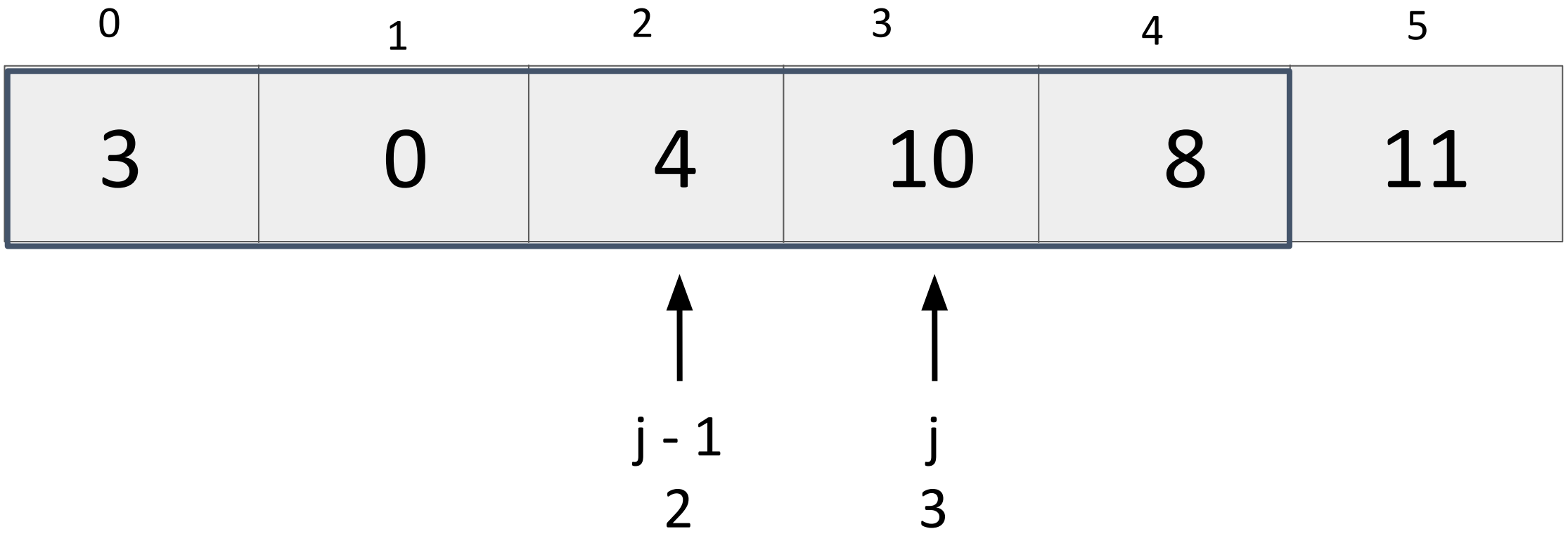


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5

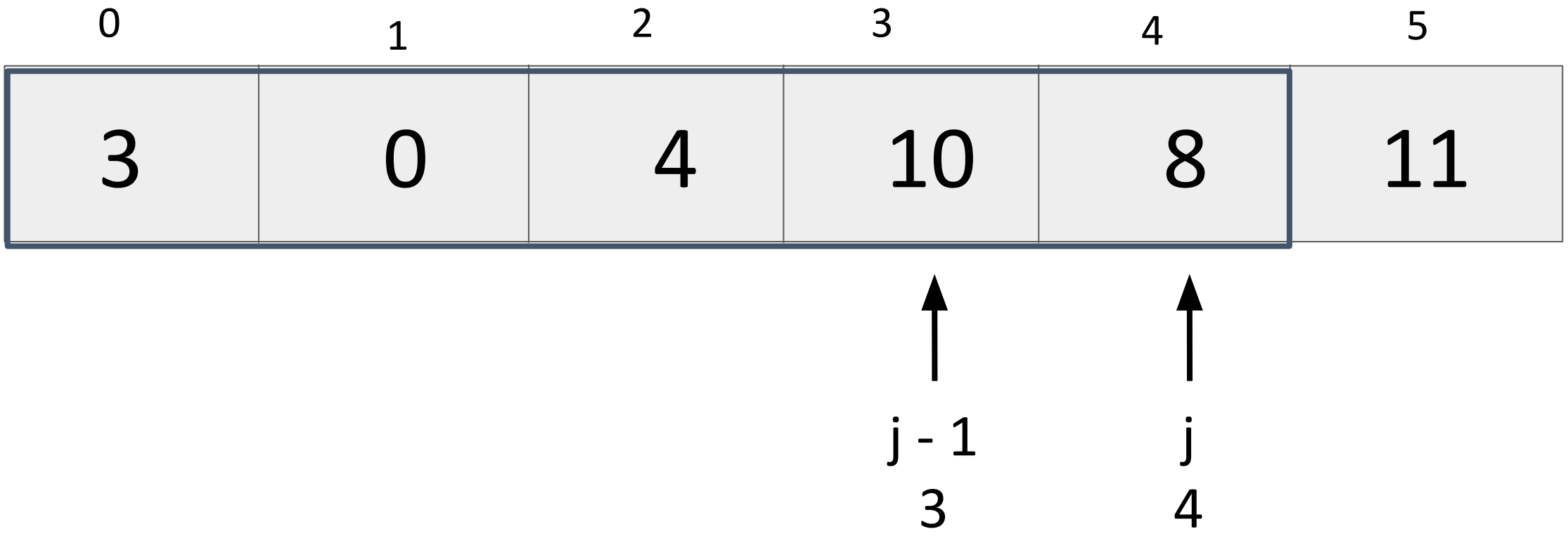


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5

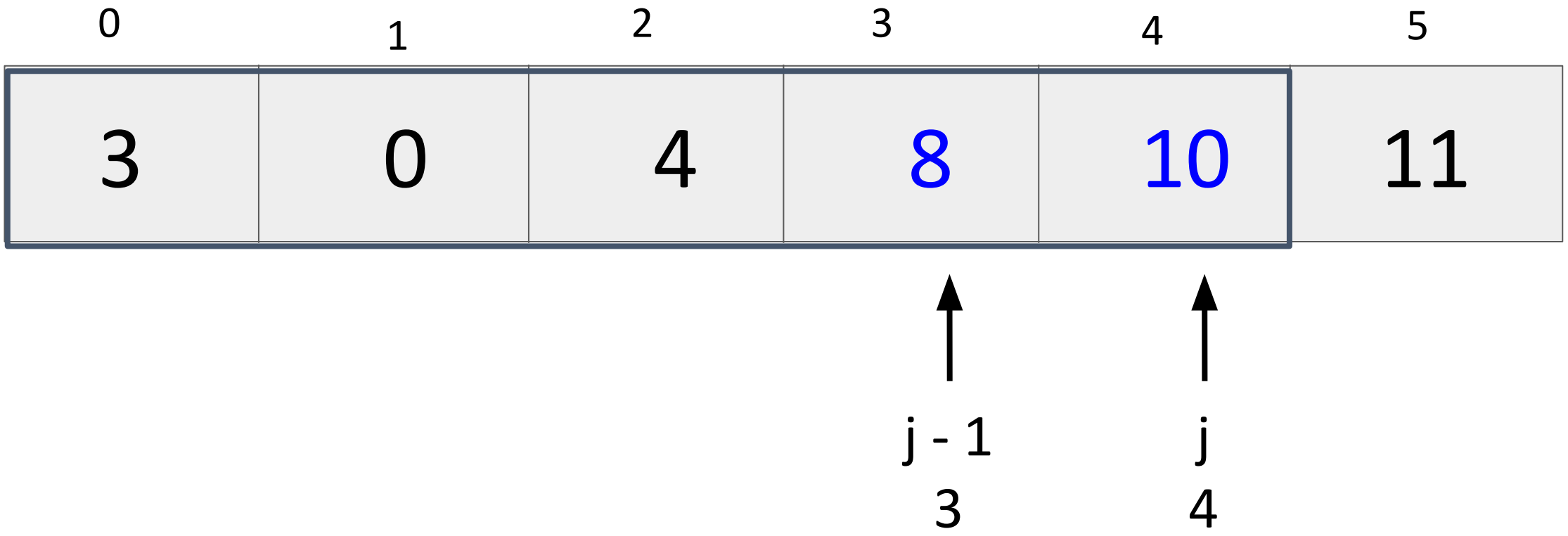


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 5

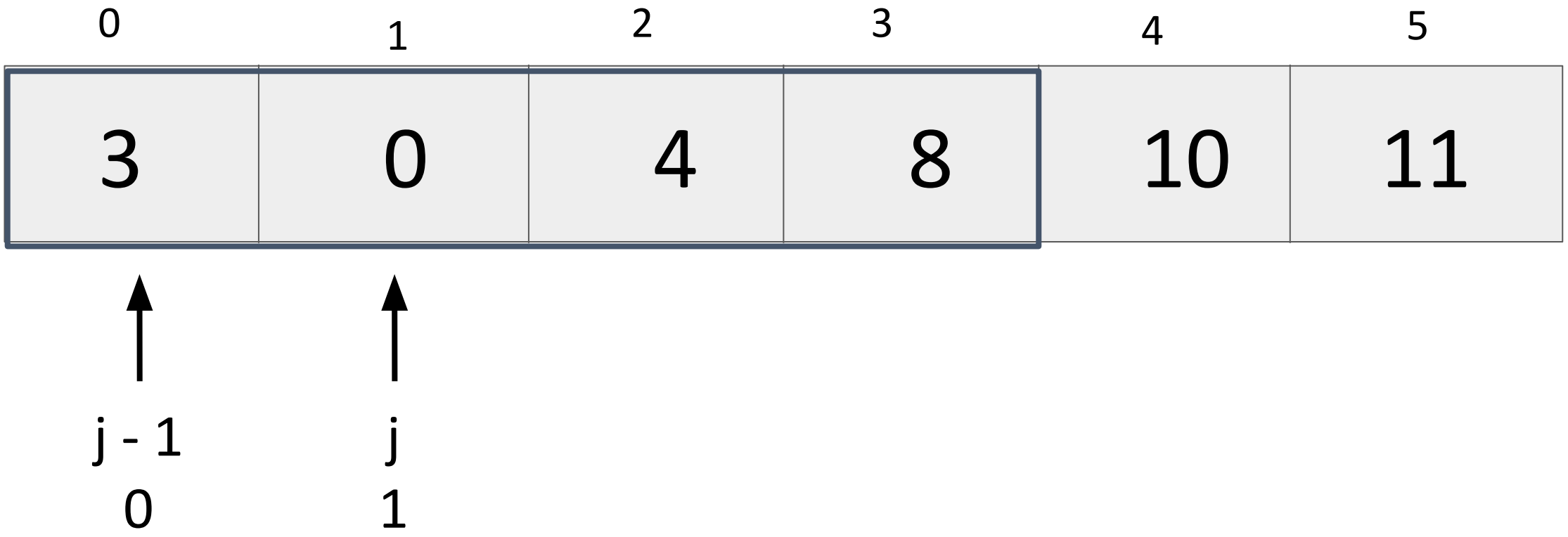


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 4

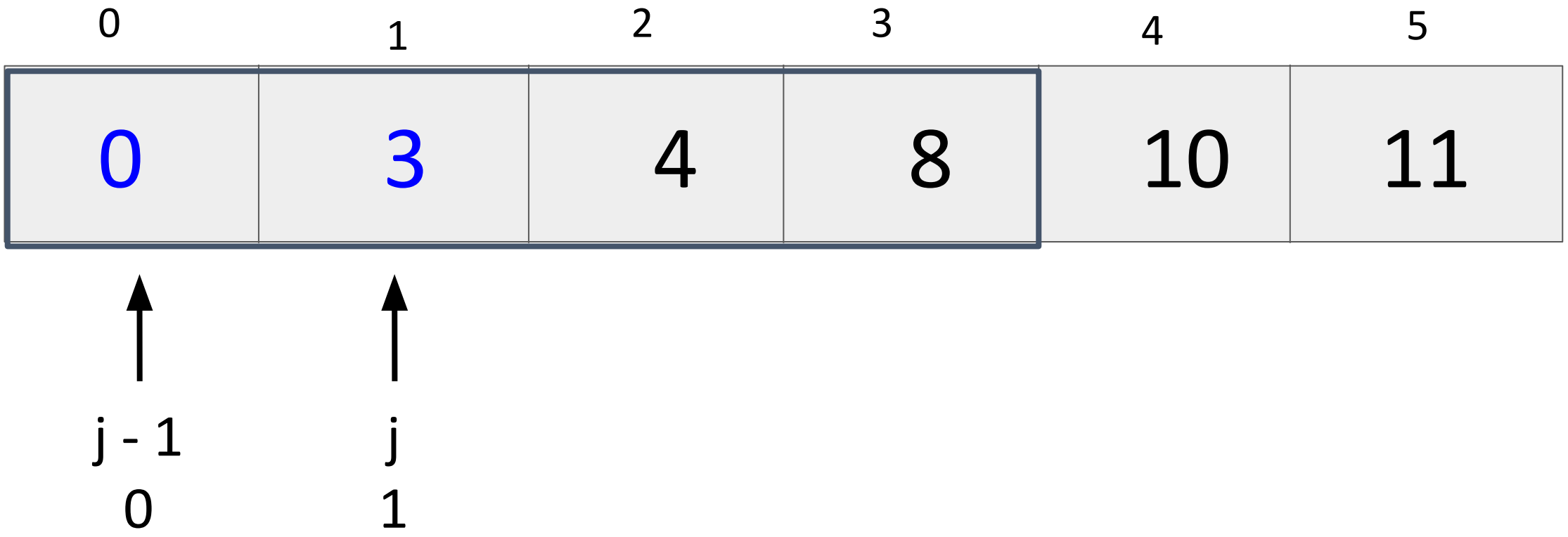


Reset and check pairs with shorter list

What next?

Bubble Sort

len = 4

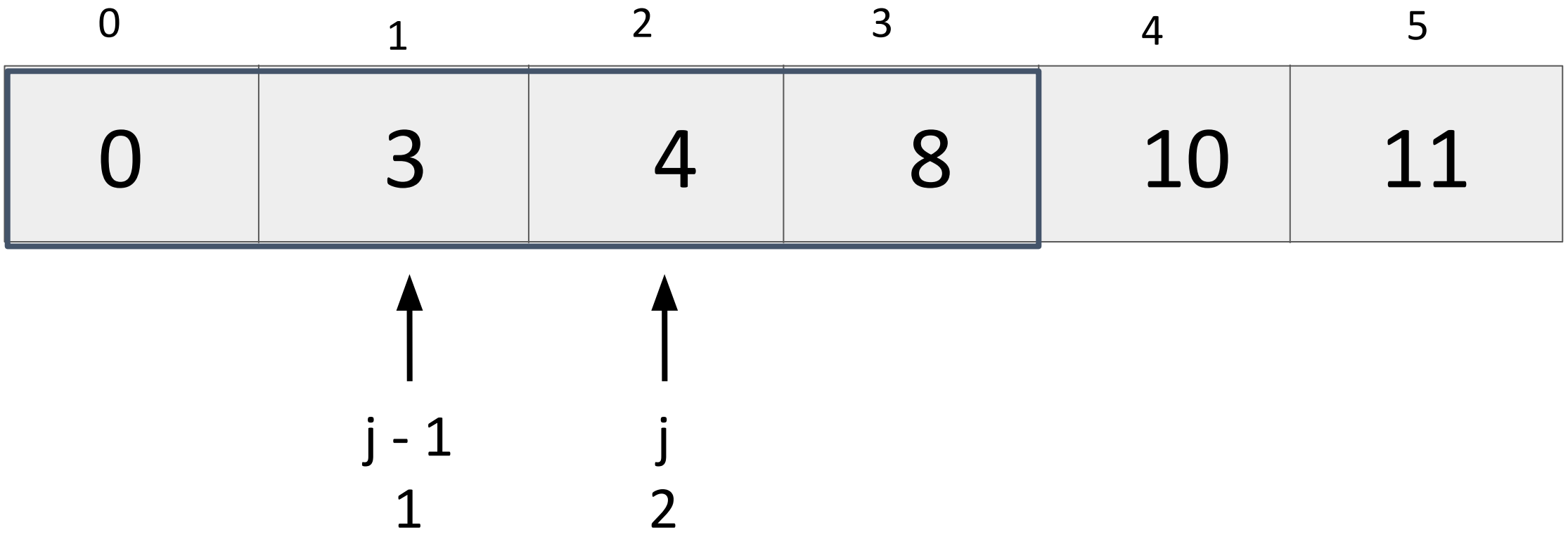


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 4

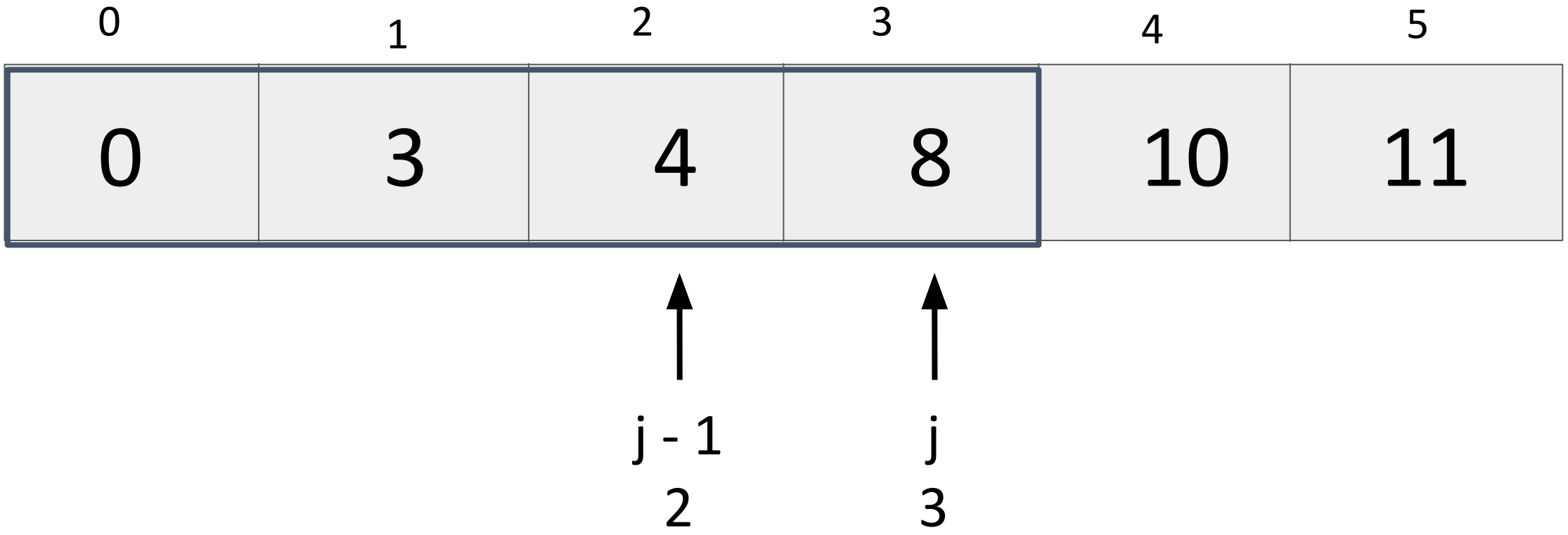


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 4

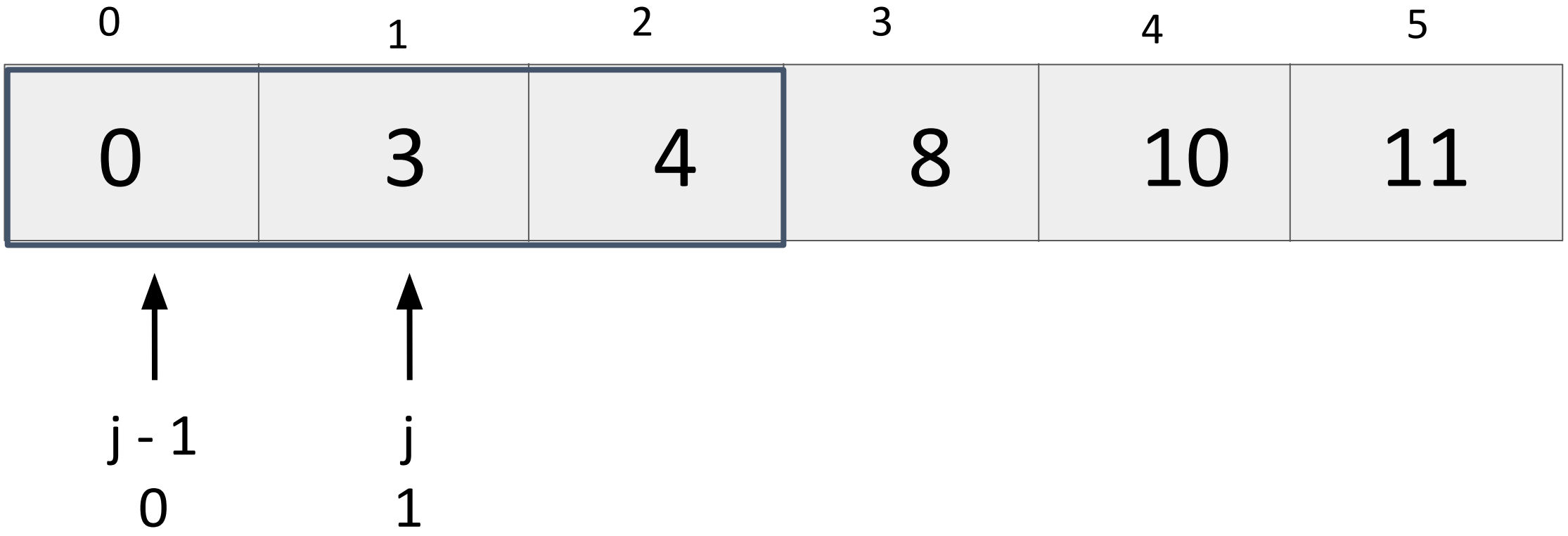


Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 3

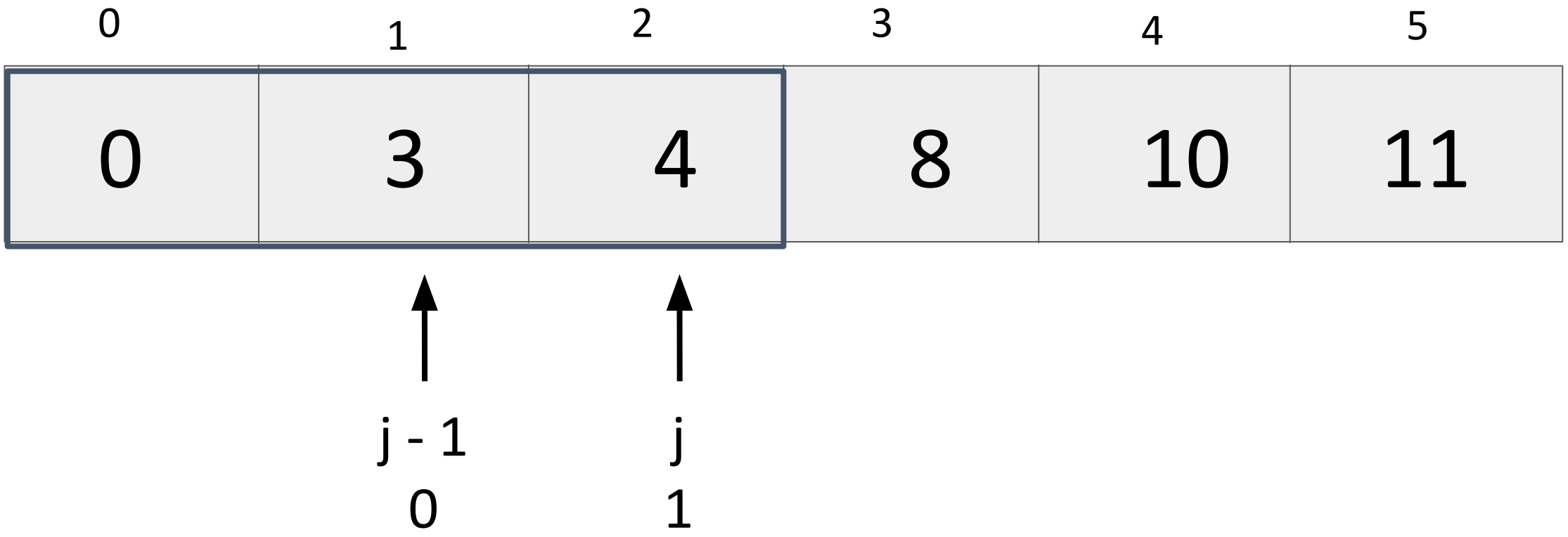


Reset; Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 3

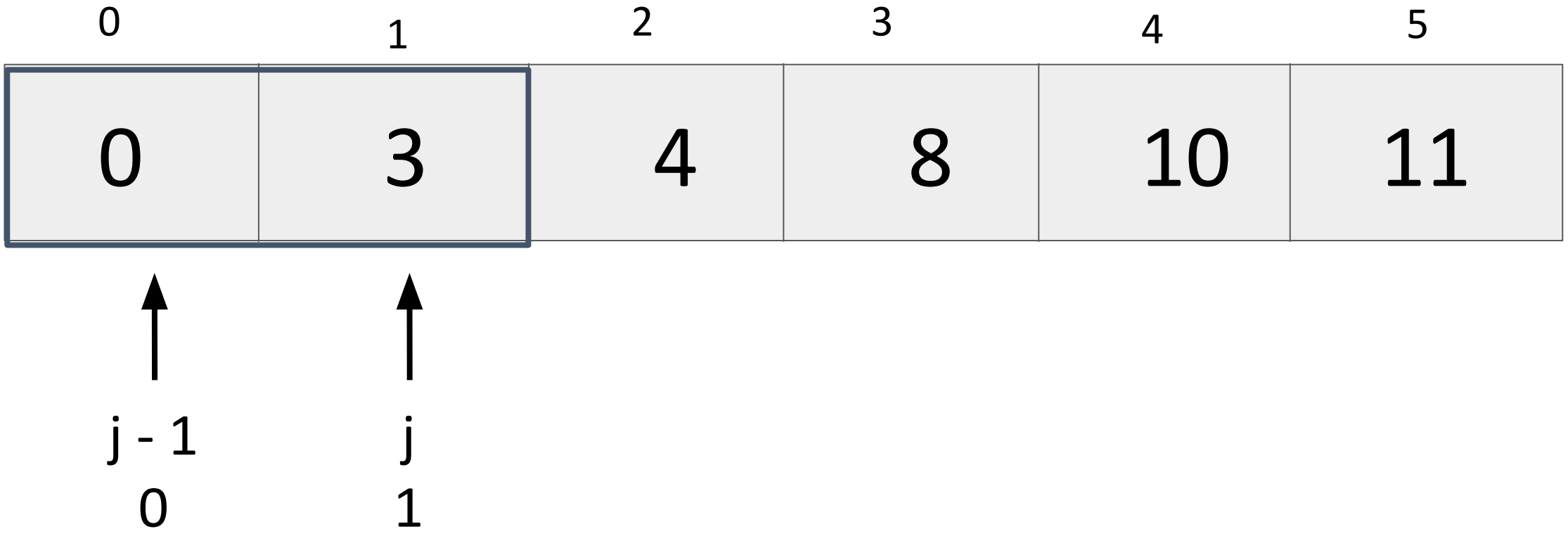


Reset; Compare j-1 and j; Swap if L[j-1] > L[j]

What next?

Bubble Sort

len = 2



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

What next?

Big-O Example 12: Bubble Sort

```
public static void bubbleSort(int[] L)  {

  for (int n = 0; n < L.length; n++){
    for (int j = 1; j < L.length-n; j++){
       if (L[j-1] > L[j]) {
          swap(j-1, j, L);
       }
    }
  }
}
```

How does the runtime grow as a function of the size of L?

O(n^2)

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

Big-O Example 13: Selection Sort

```
void selectionSort(int[] L){
   for (int i = 0; i < L.length; i++) {
      int minIdx = i;
      for (int j = i+1; j < L.length; j++) {
        if (L[j] < L[minIdx]) {
          minIdx = j;
        }
      }
      swap(i, minIdx, L);
   }
}
```

How does the runtime grow as a function of the size of L?

O(n^2)

Sort + Search Comparison

Which will be the fastest?

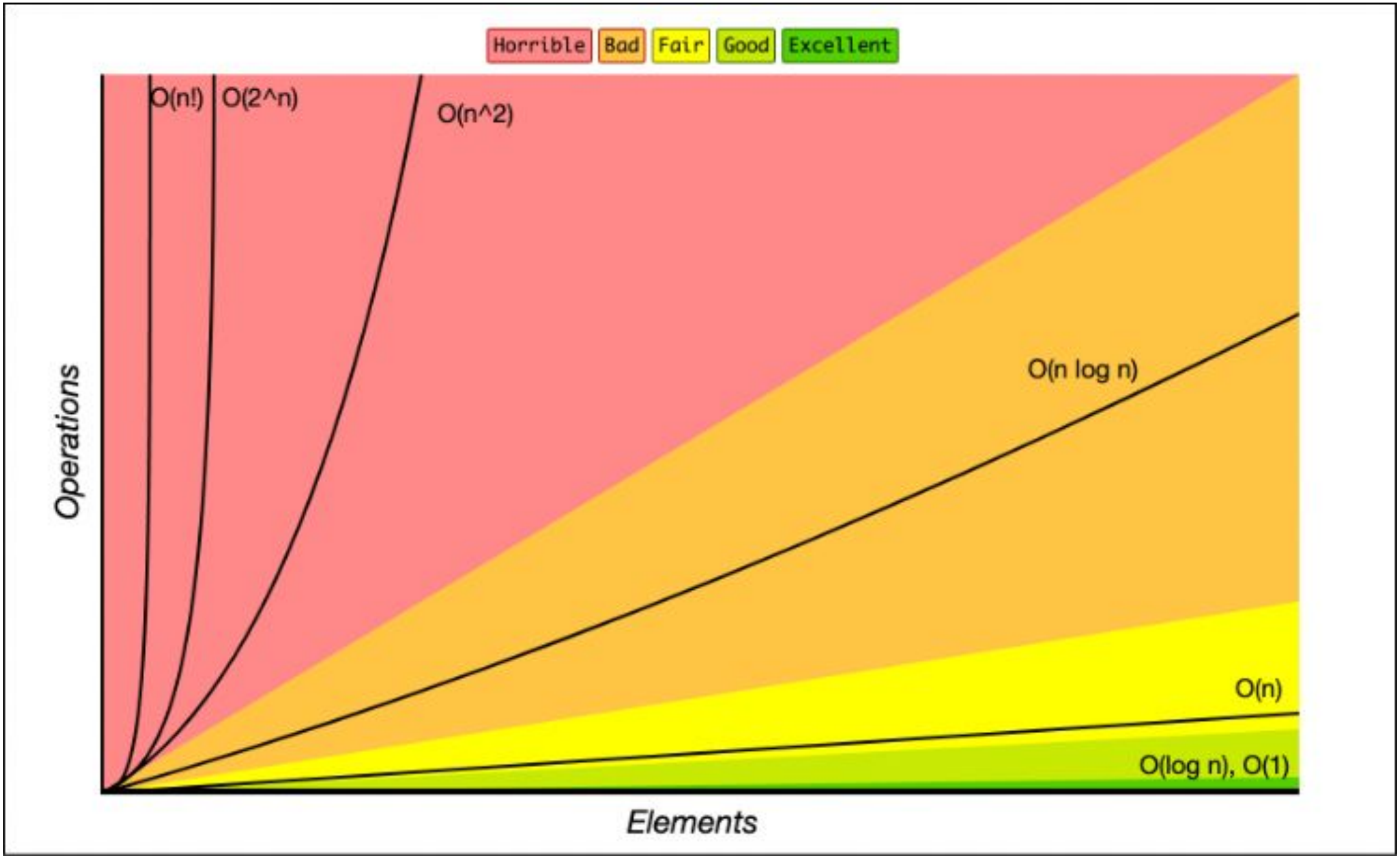
1) Linear Search

2) Bubble Sort + Binary Search

3) Selection Sort + Binary Search

4) Bubble Sort + Linear Search

5) Selection Sort + Linear Search


Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements