SORTING

CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

13: Insertion Sort

Alexandra Papoutsaki
she/her/hers

Lecture 13: Insertion Sort

- Insertion sort

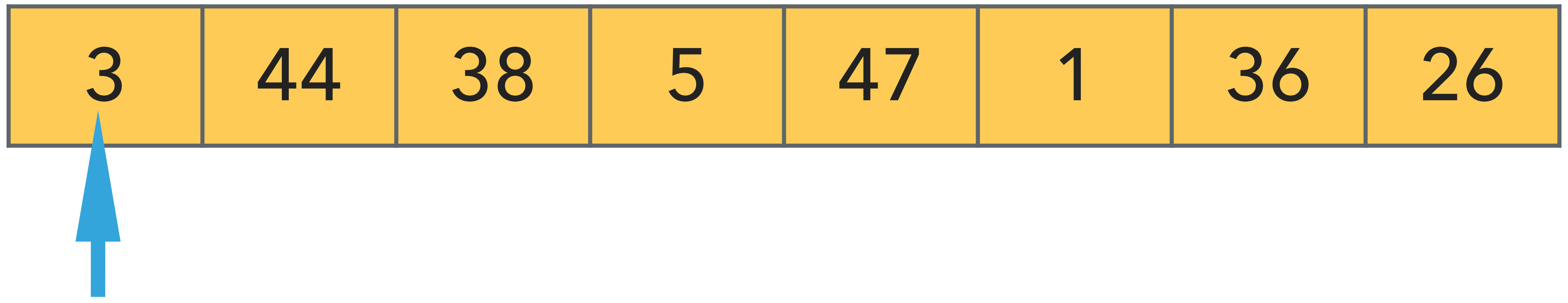
Some slides adopted from Algorithms 4th Edition or COS226

Insertion sort

3	44	38	5	47	1	36	26

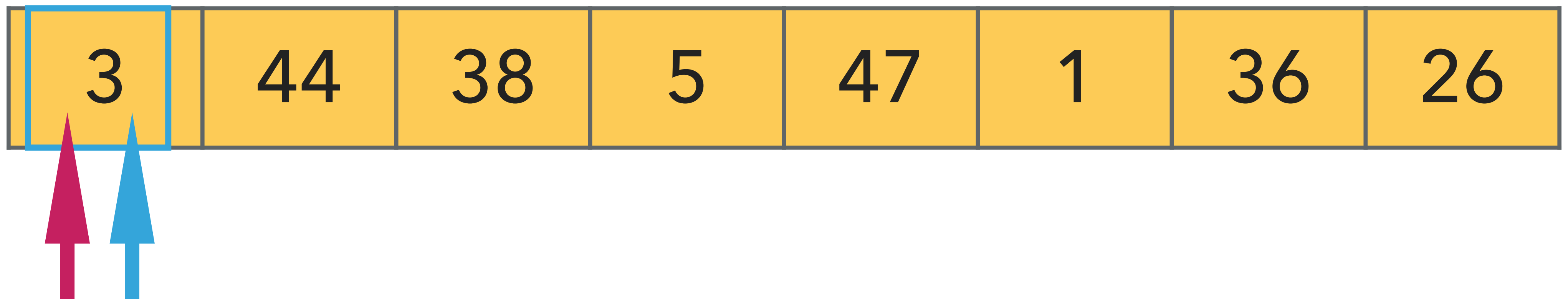
- Keep a *partially* sorted subarray on the left and an unsorted subarray on the right.
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



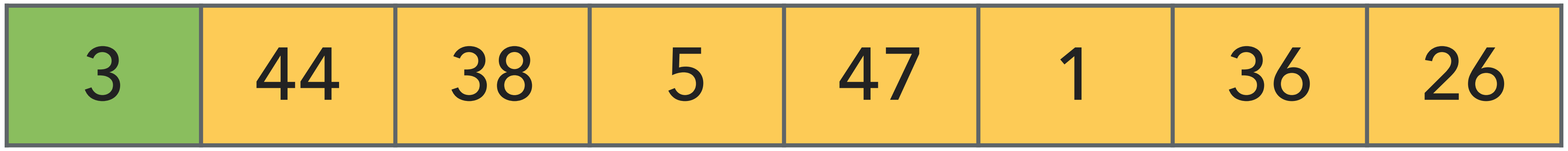
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



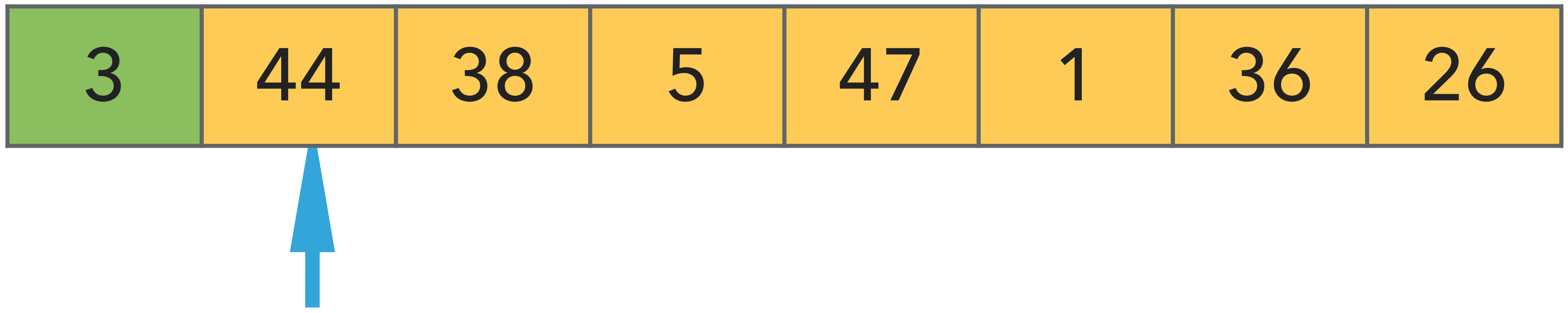
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



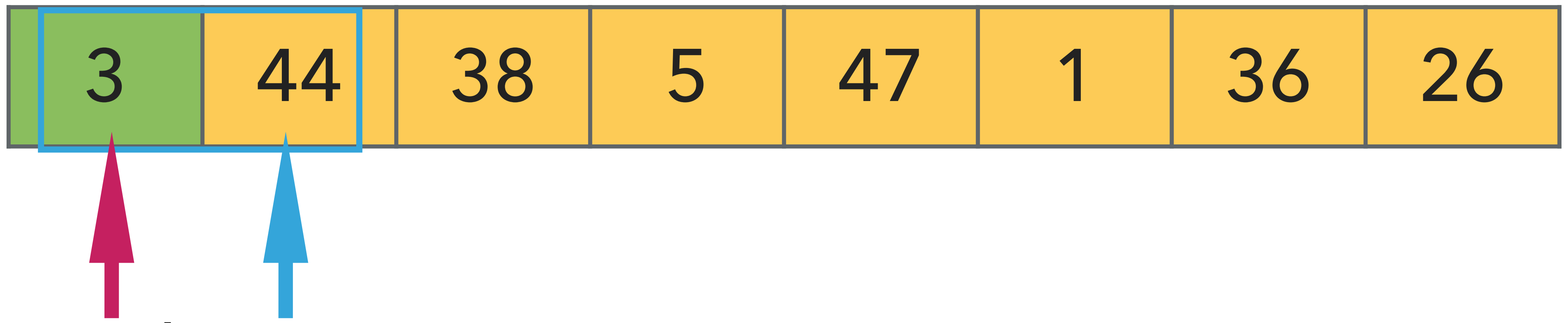
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



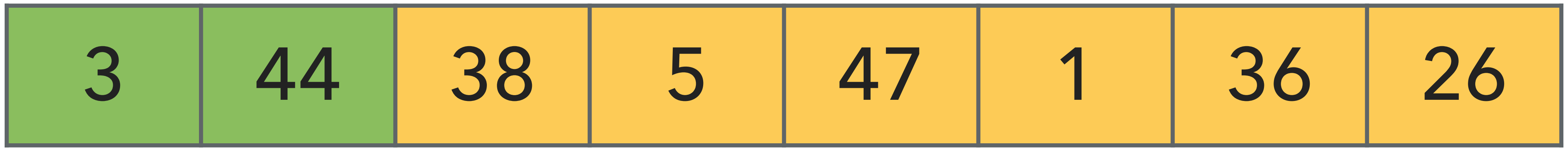
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



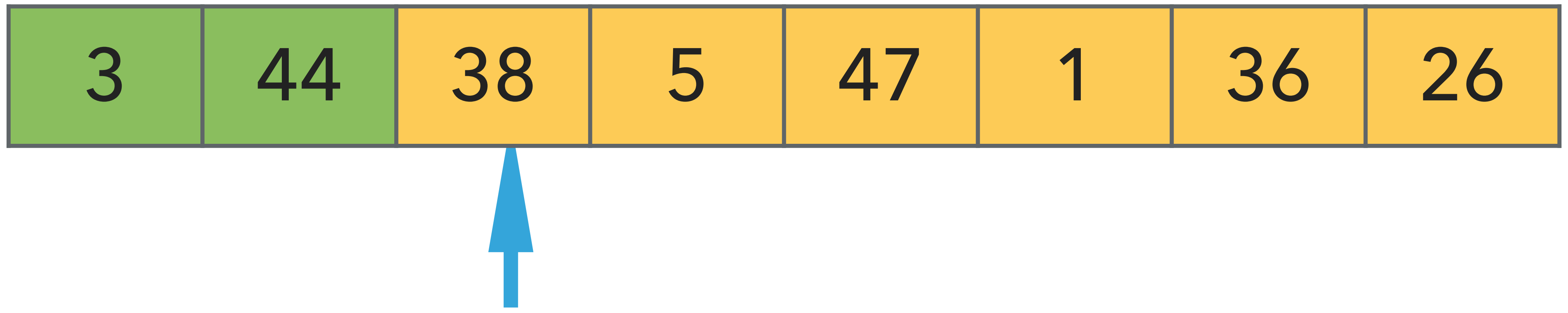
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



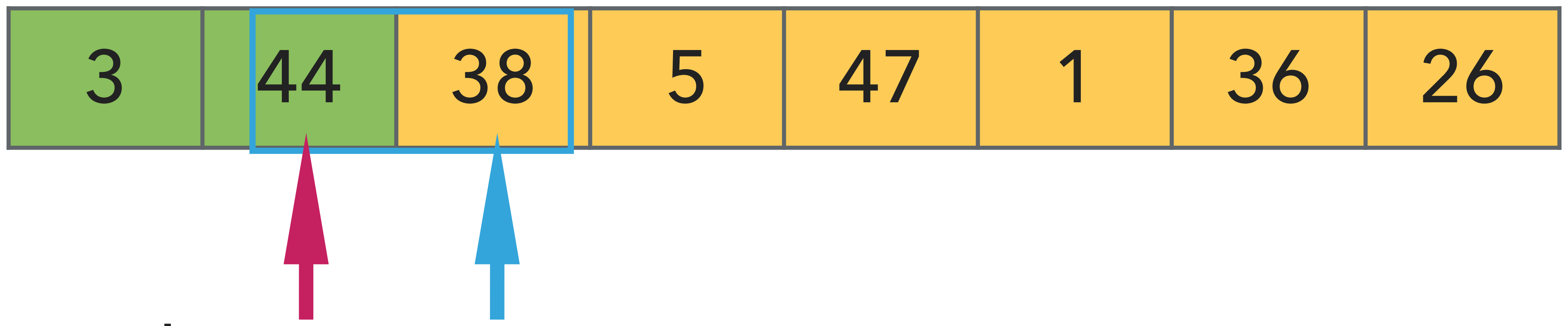
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



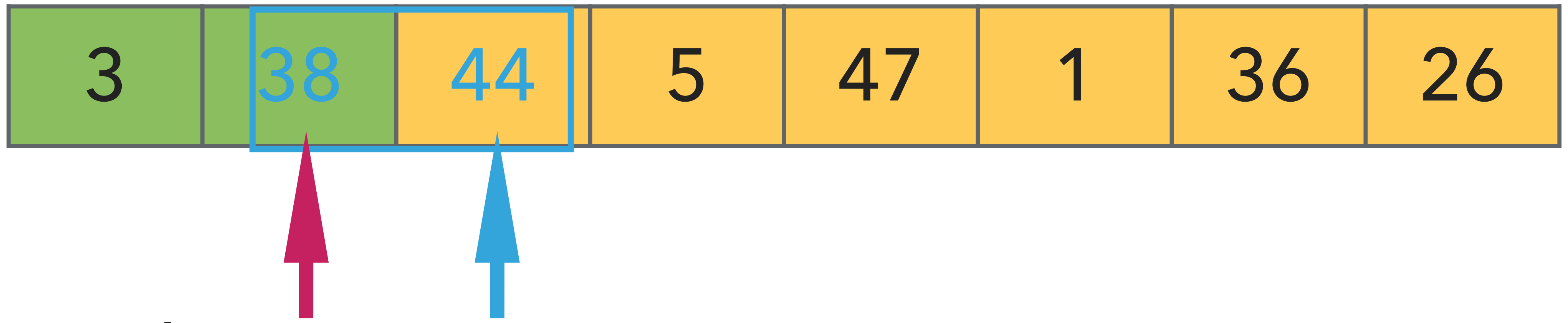
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



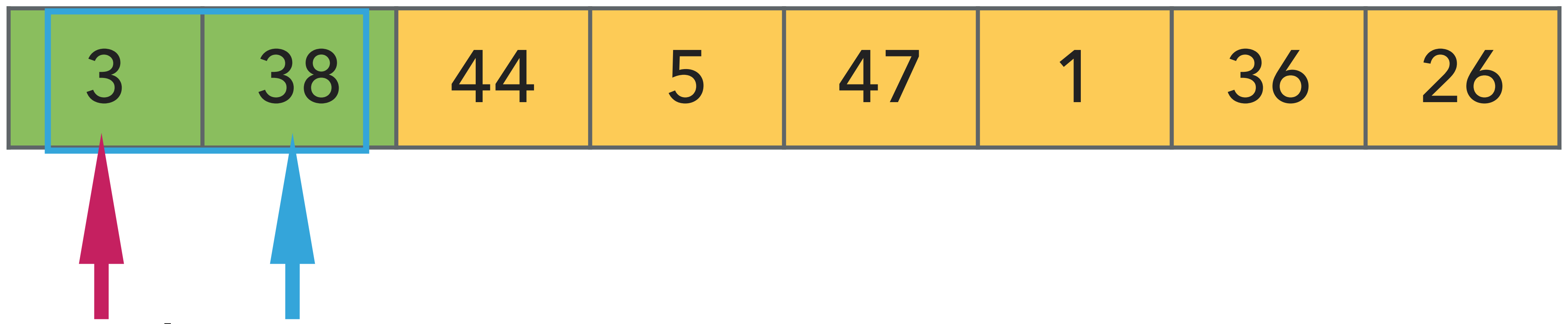
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



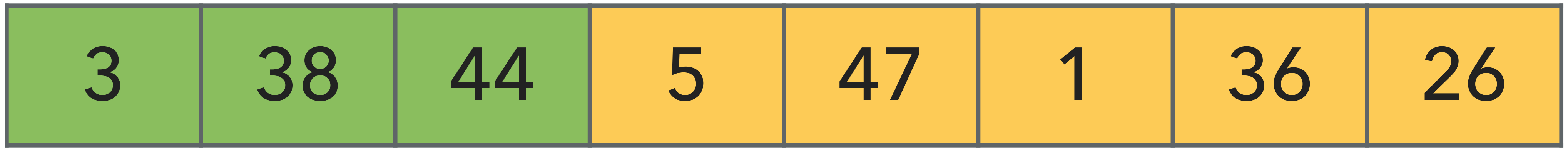
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



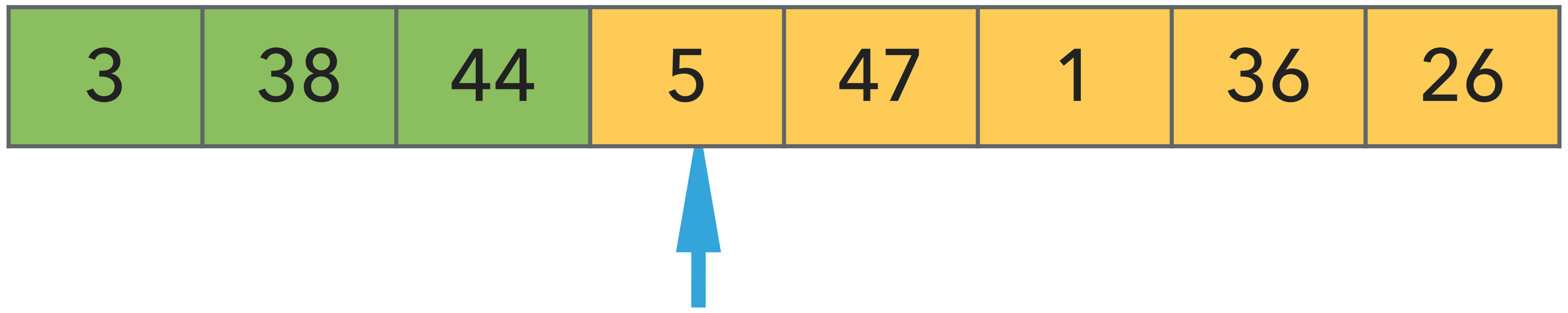
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



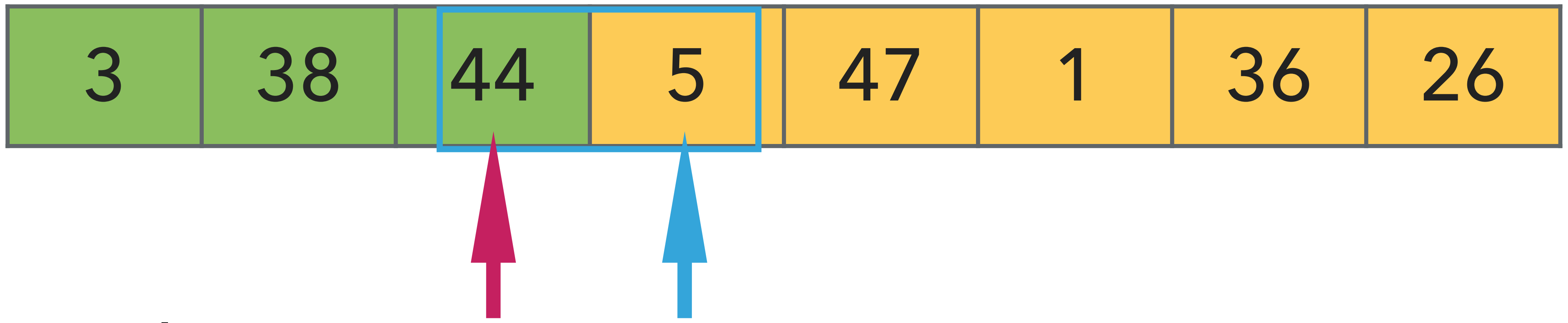
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



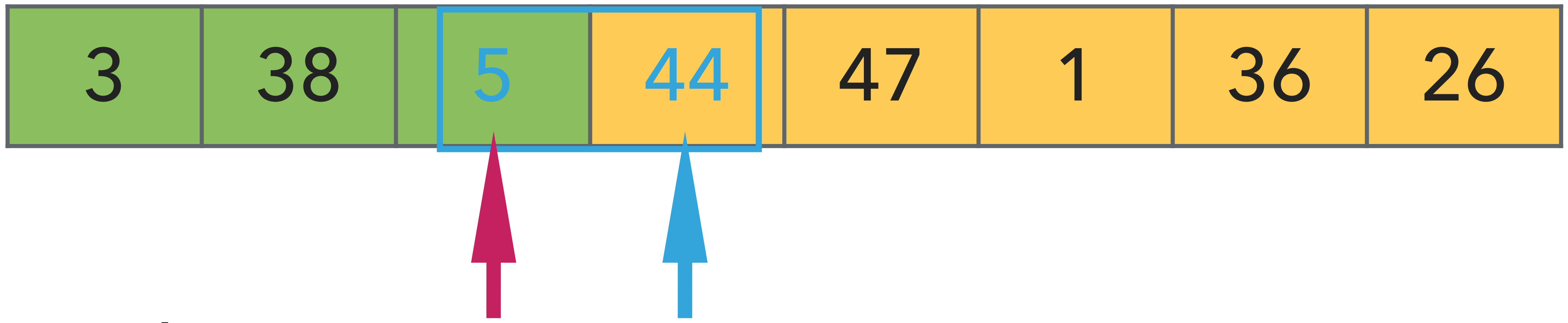
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



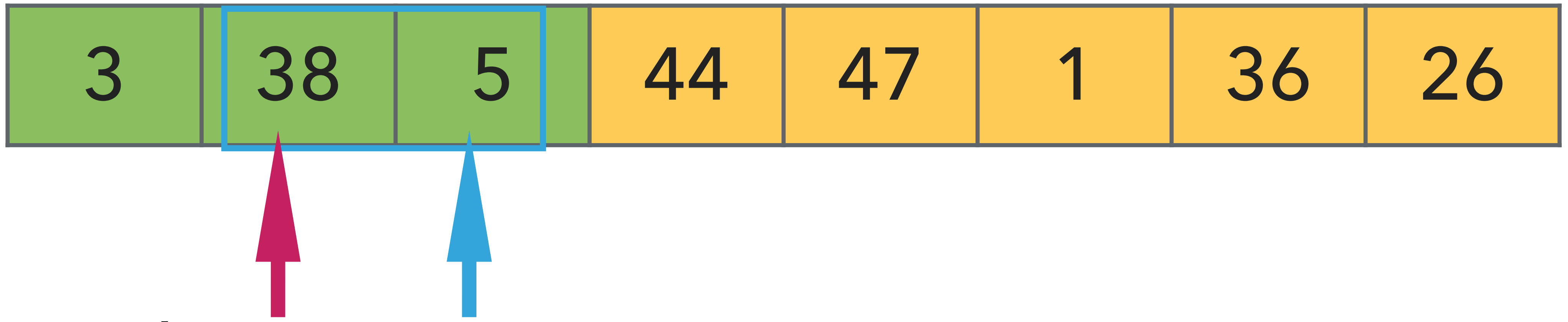
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



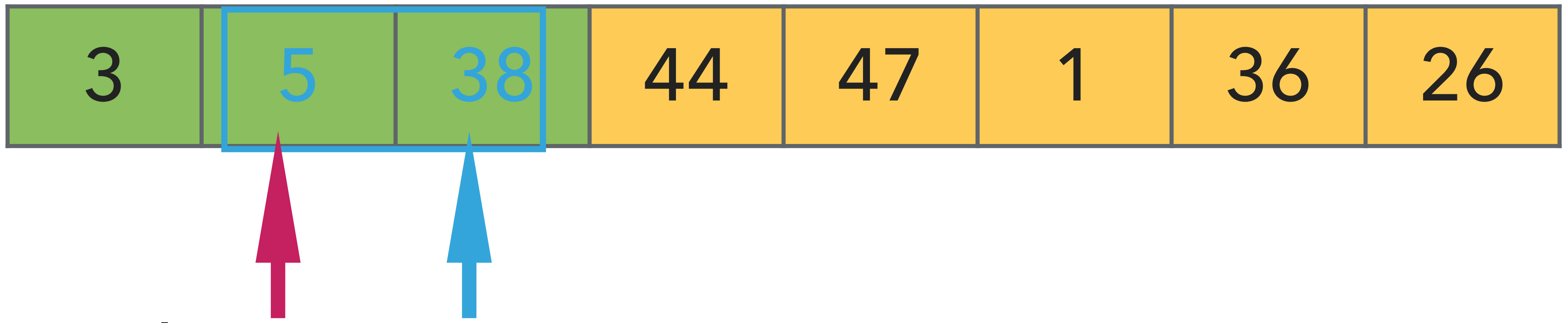
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



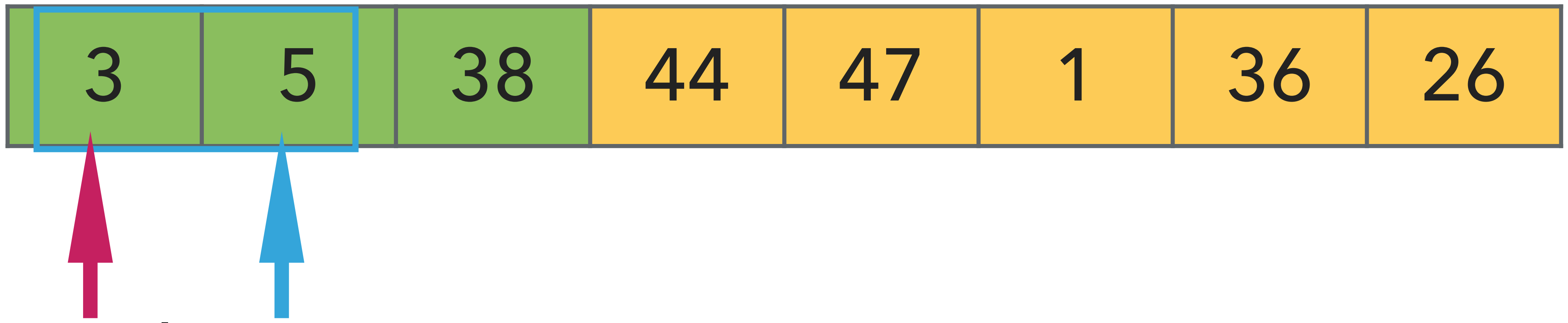
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



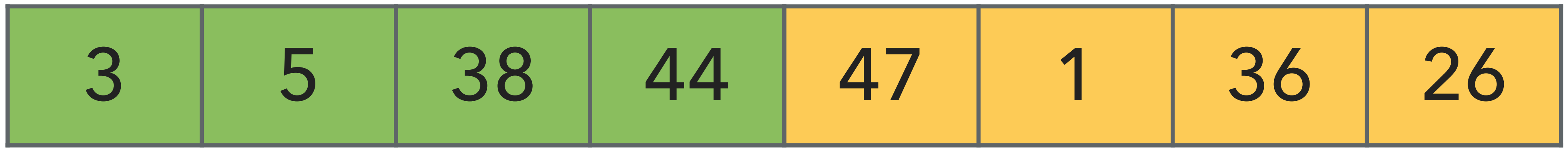
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



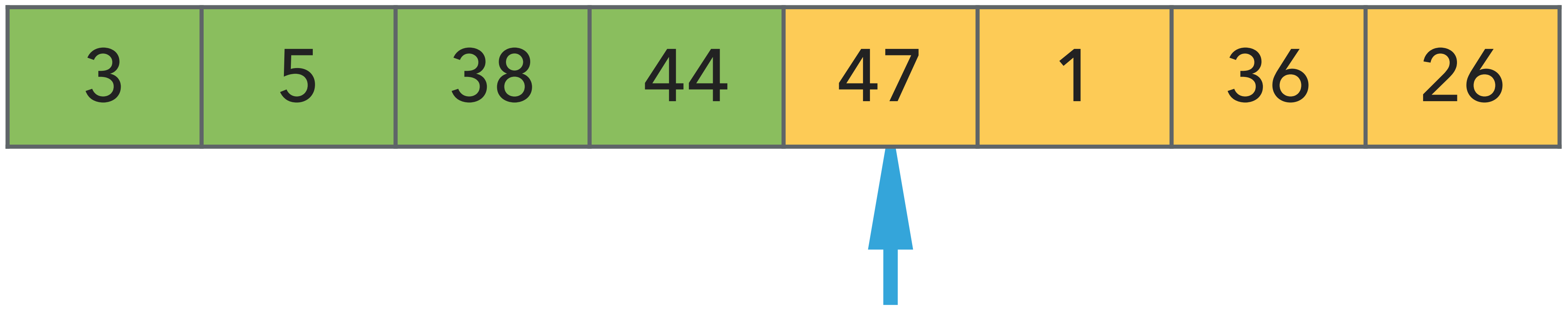
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



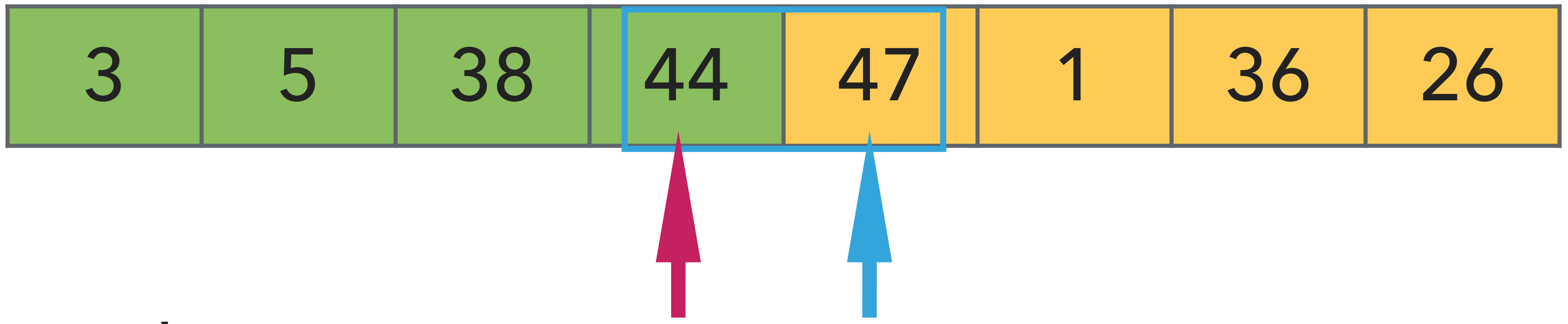
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



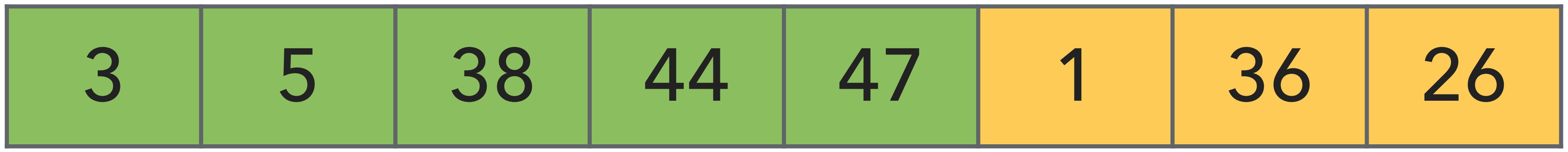
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



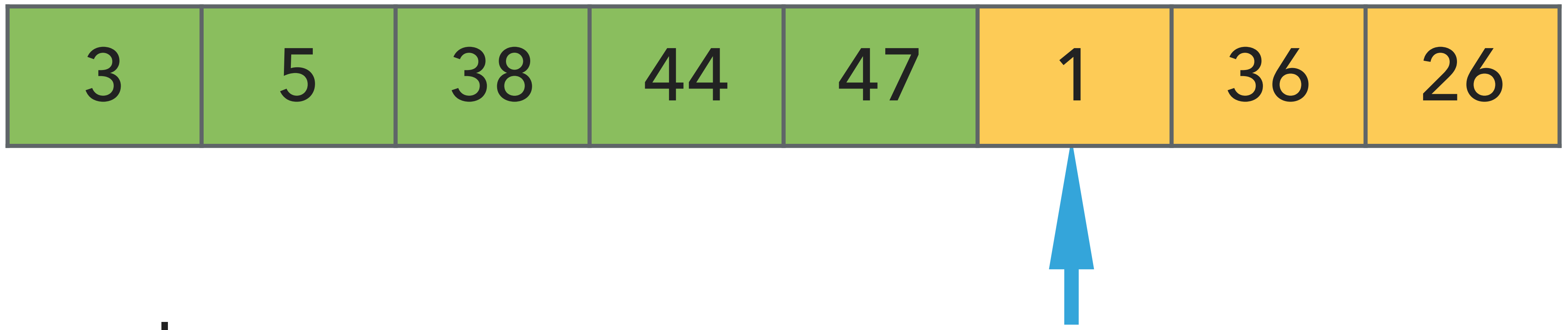
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



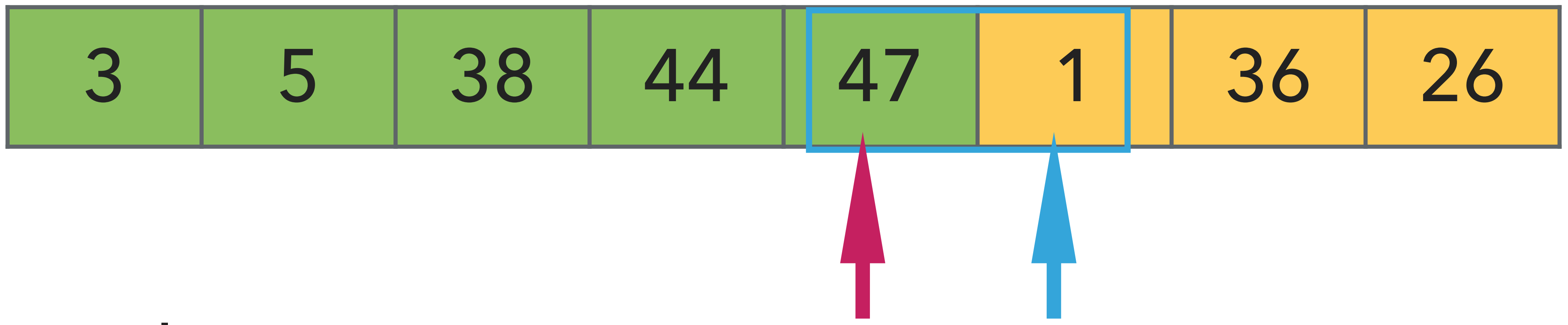
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



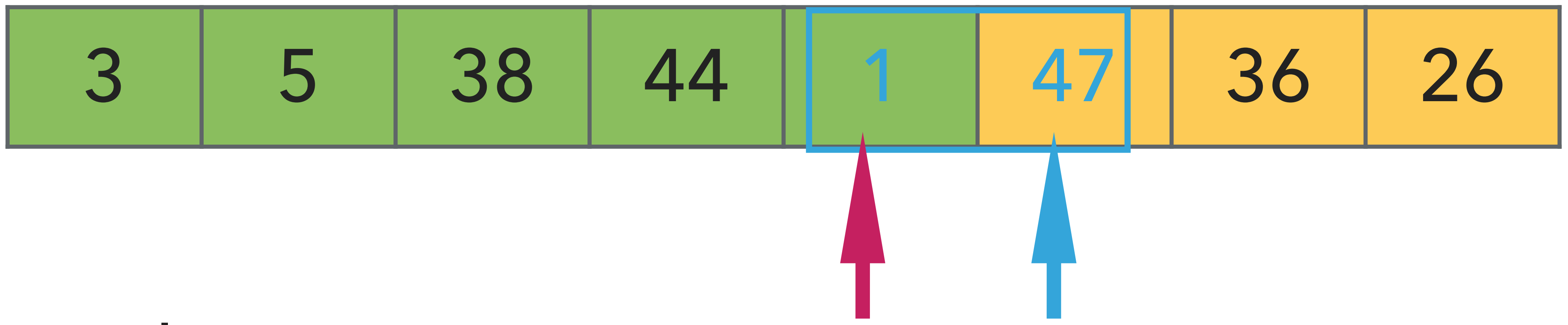
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



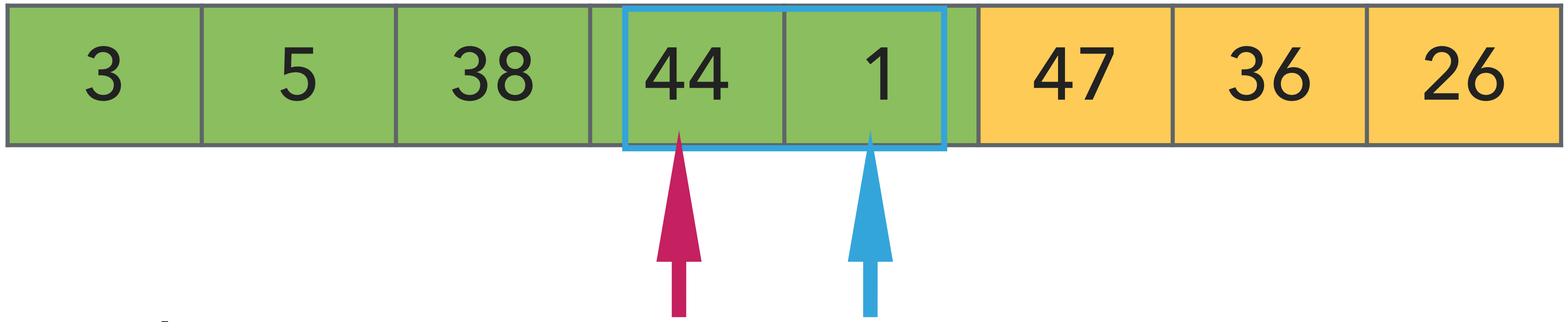
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



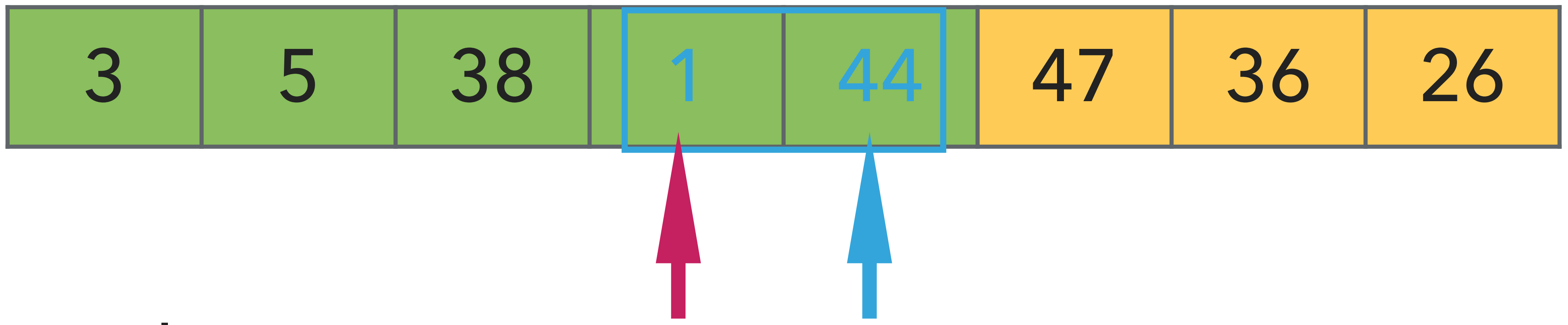
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



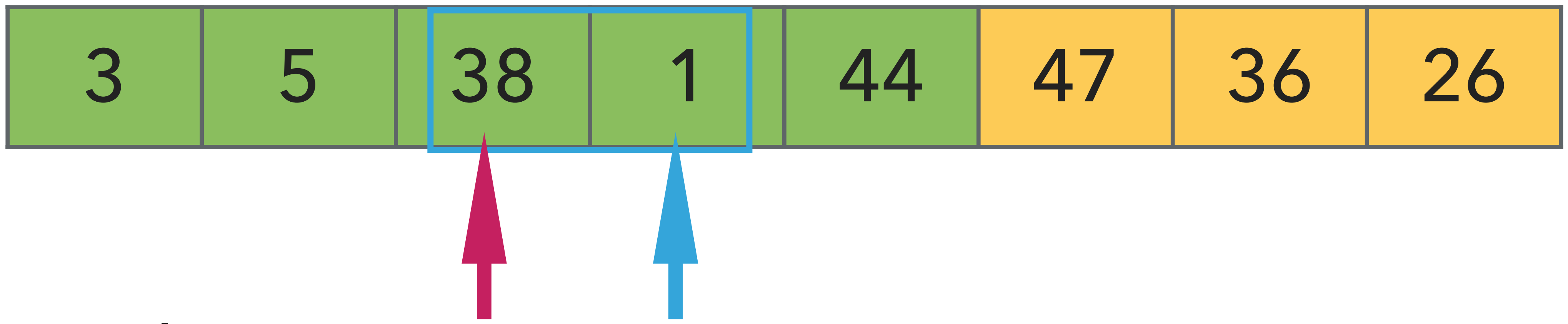
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



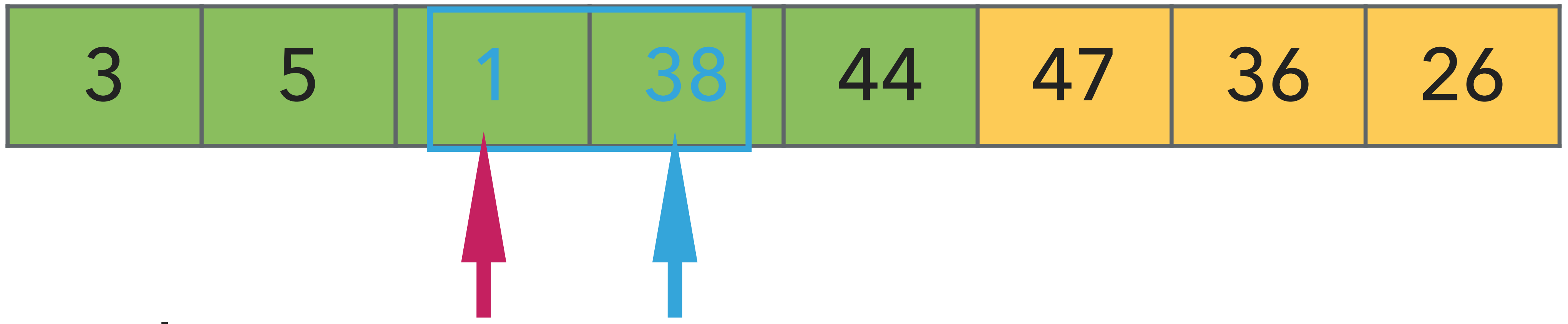
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



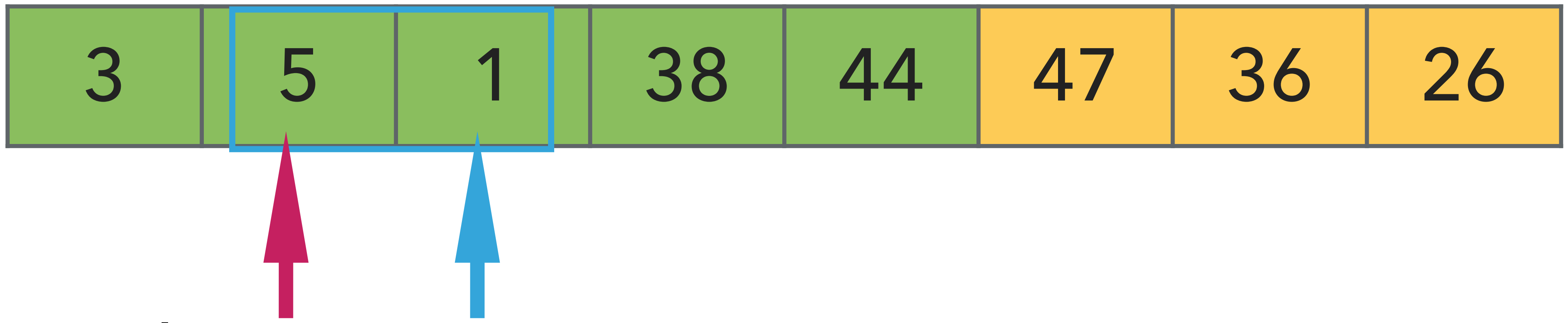
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



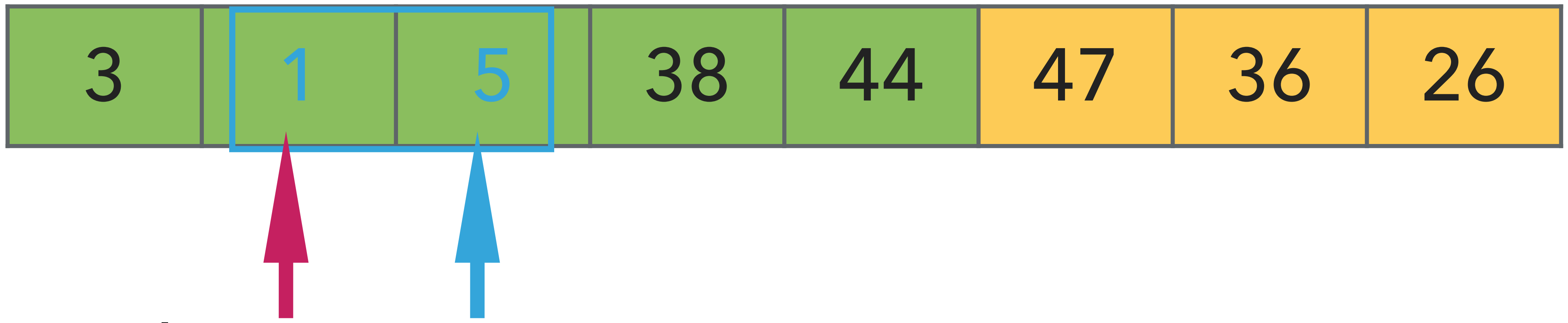
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



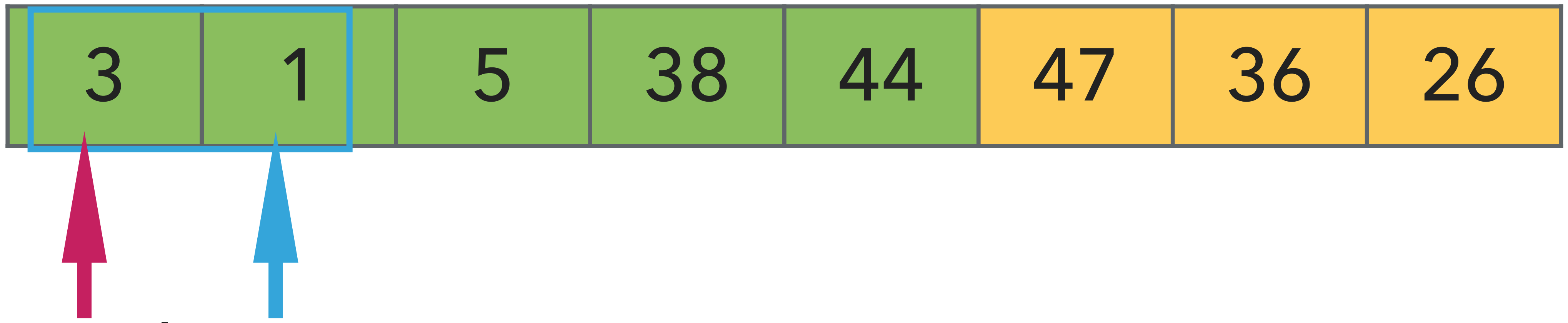
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



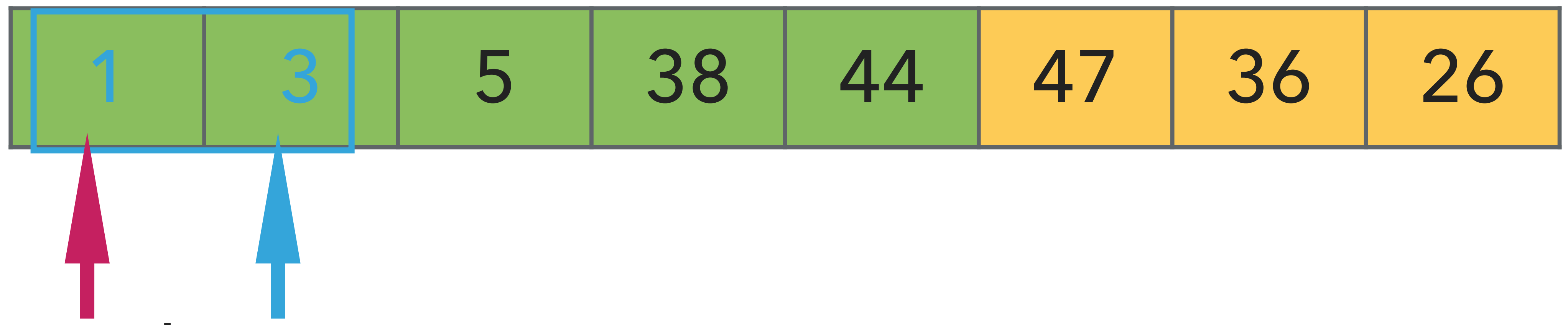
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



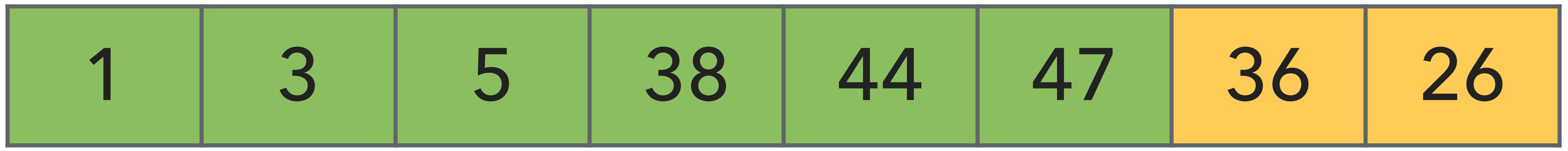
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



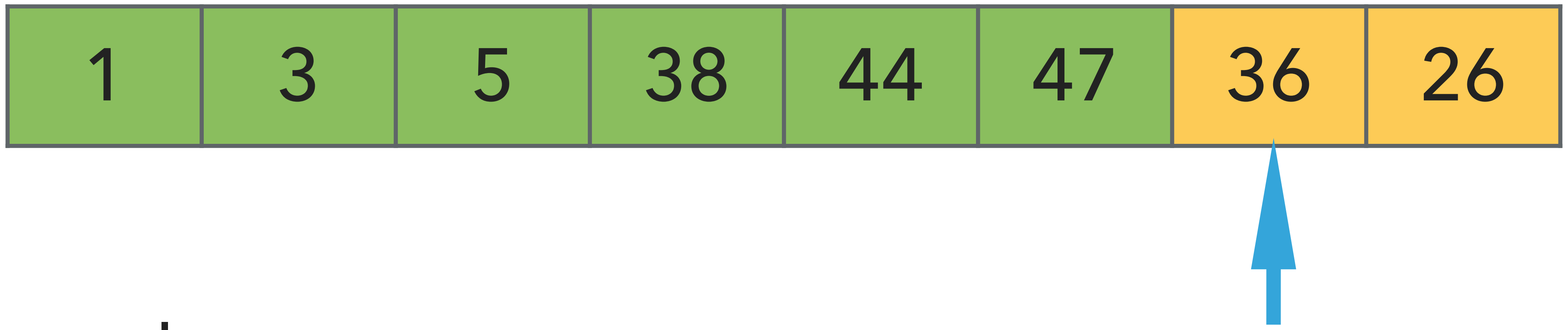
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



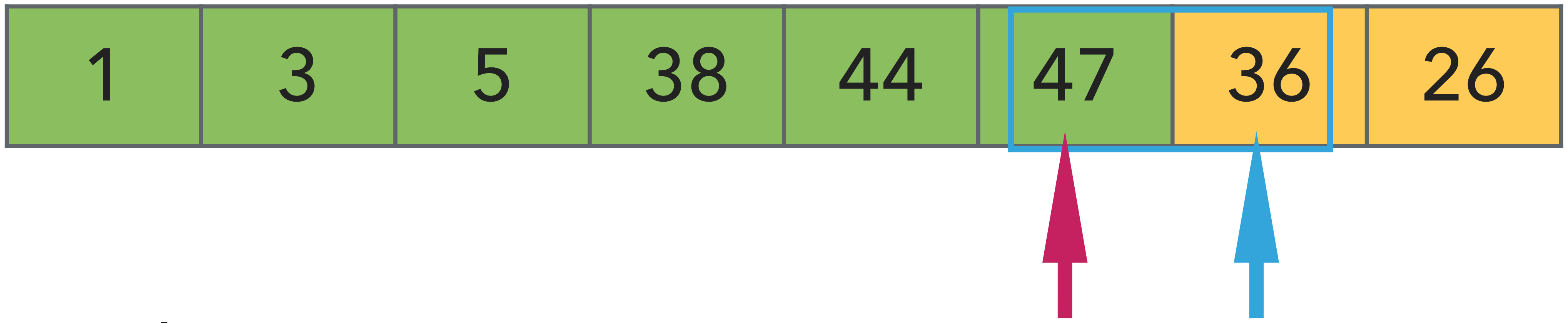
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



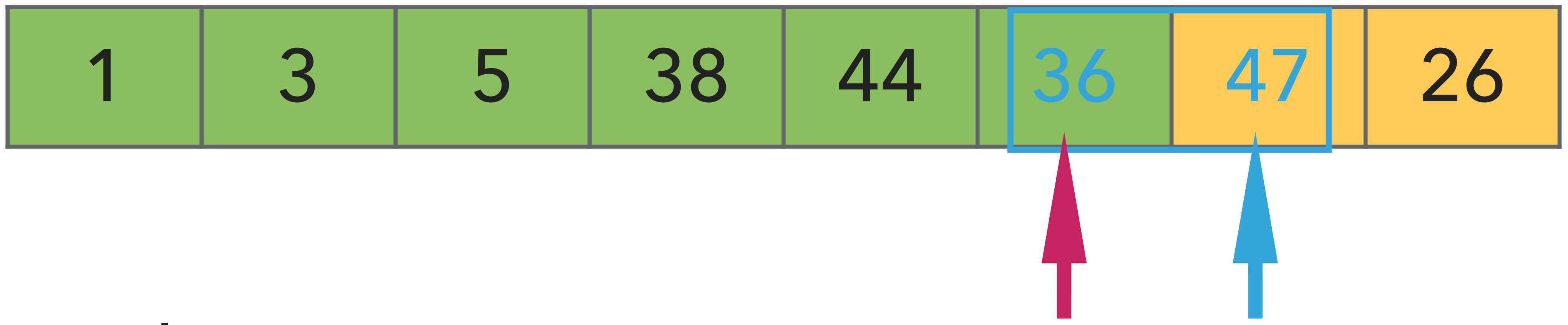
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



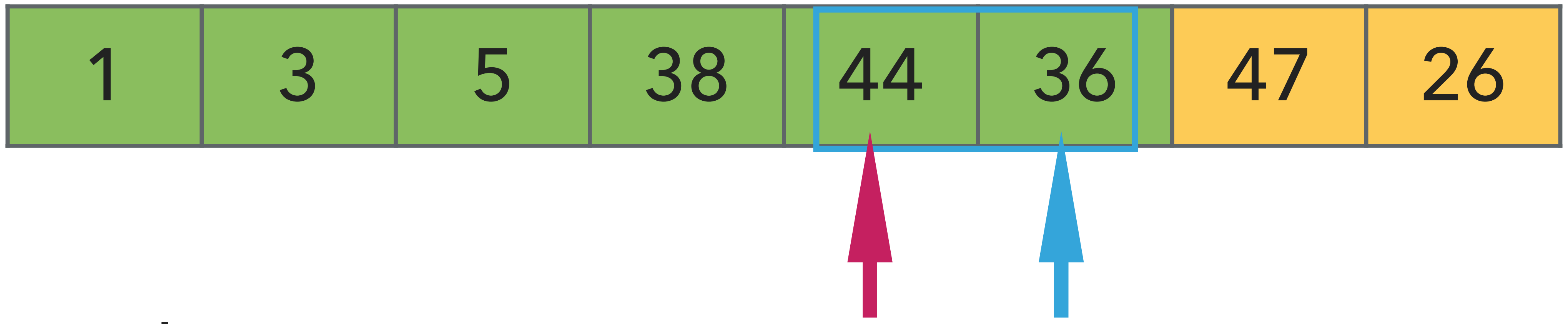
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



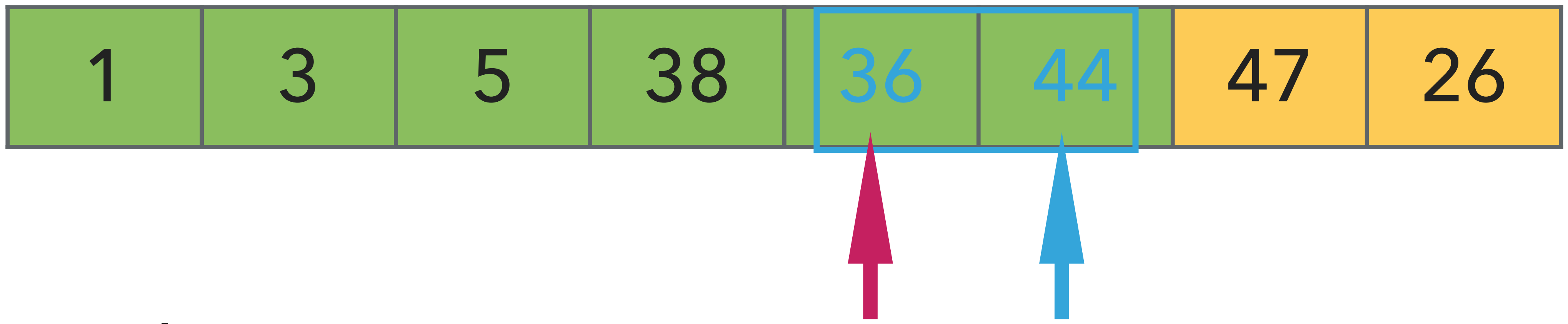
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



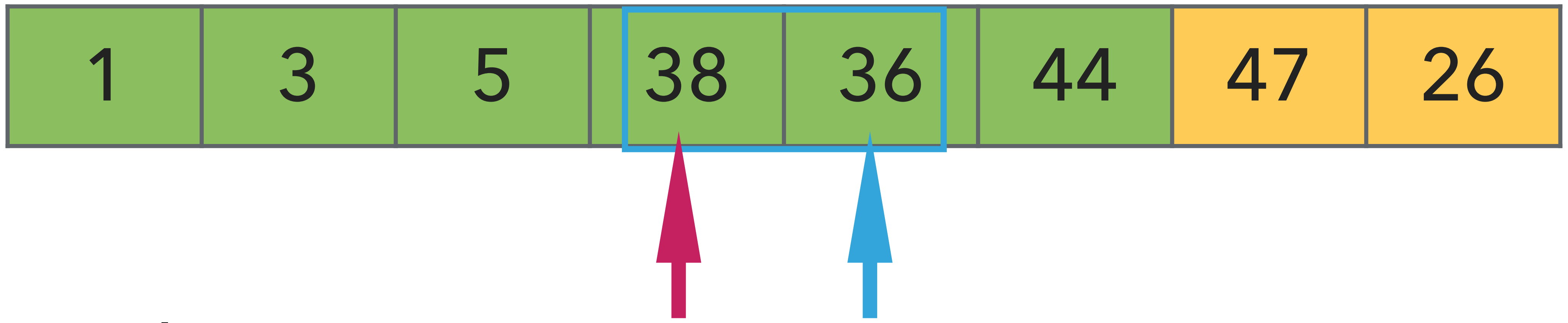
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



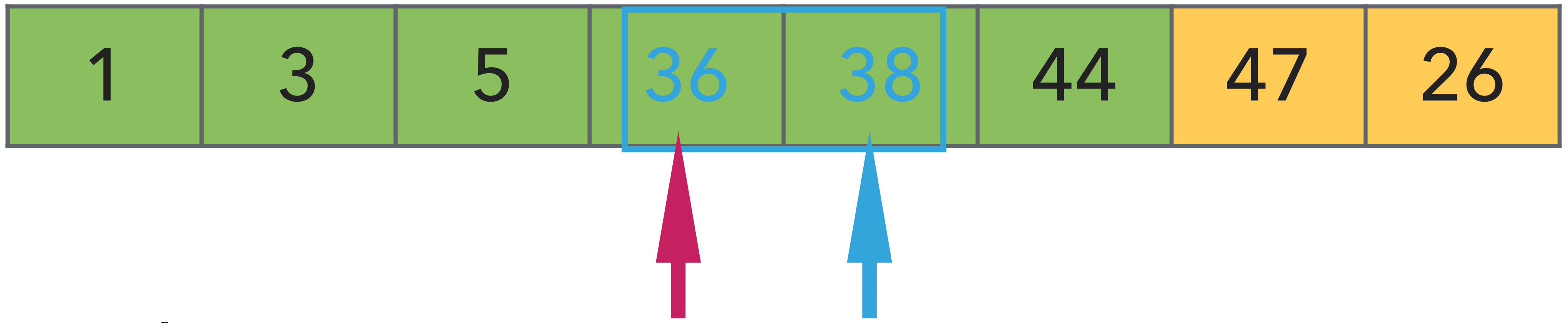
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



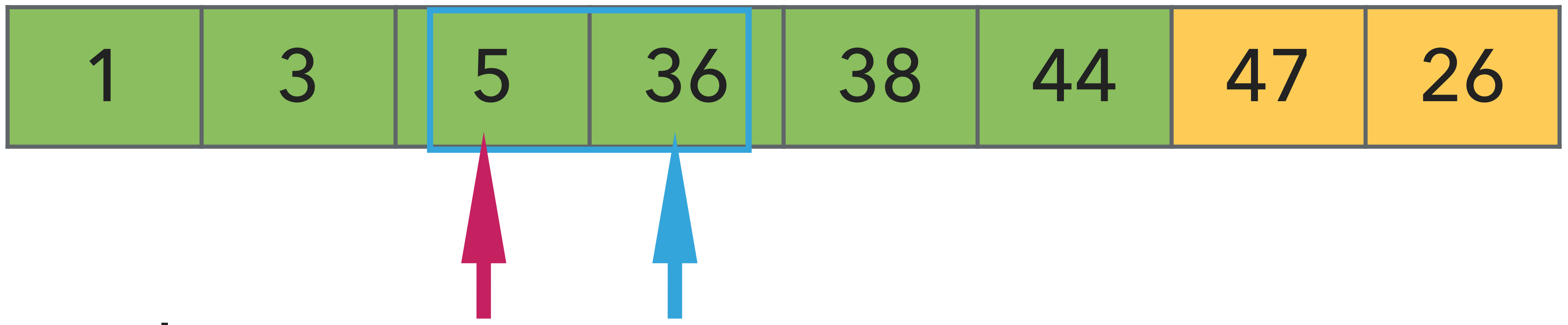
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



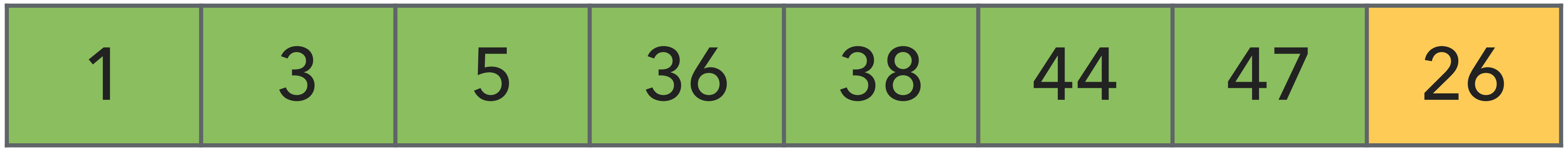
- Repeat:
    - Examine the next element in the unsorted subarray.
    - Find the location it belongs within the sorted subarray and insert it there.
    - Move subarray boundaries one element to the right.

Insertion sort



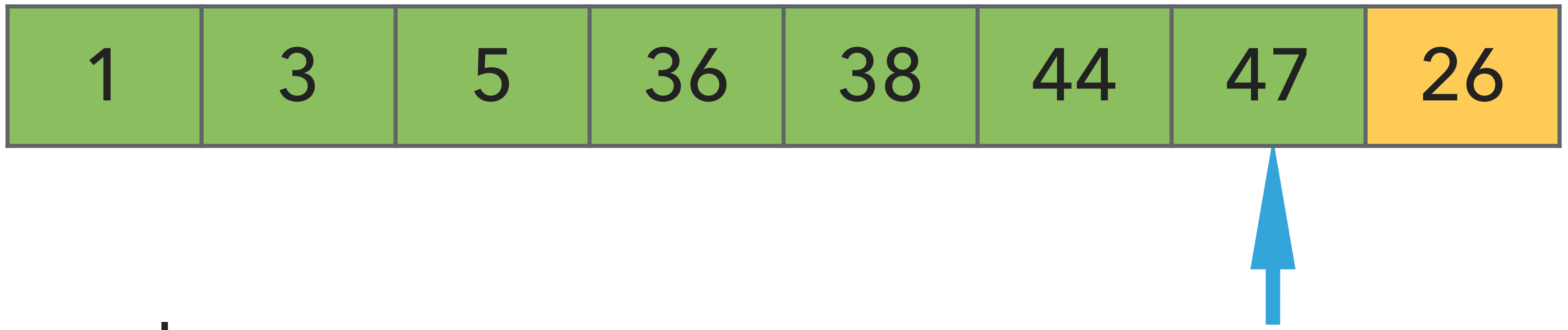
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



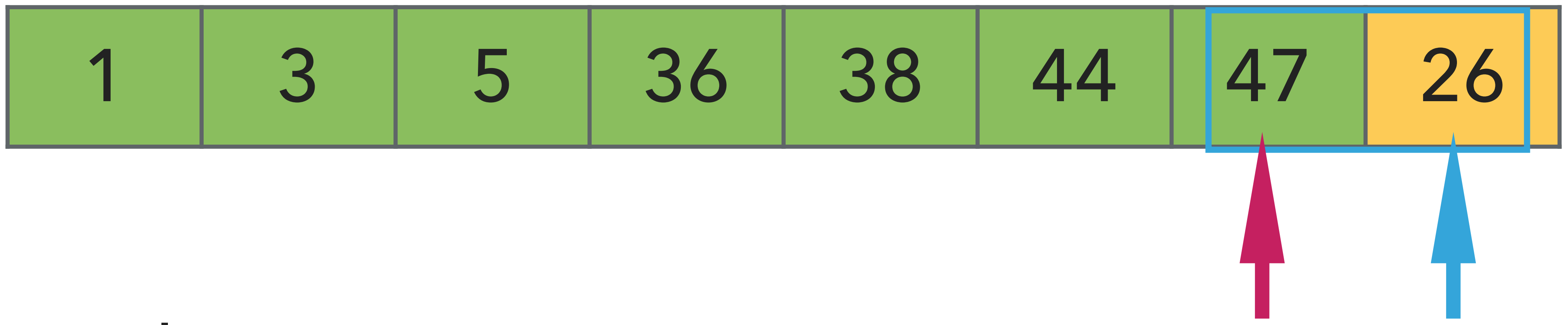
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



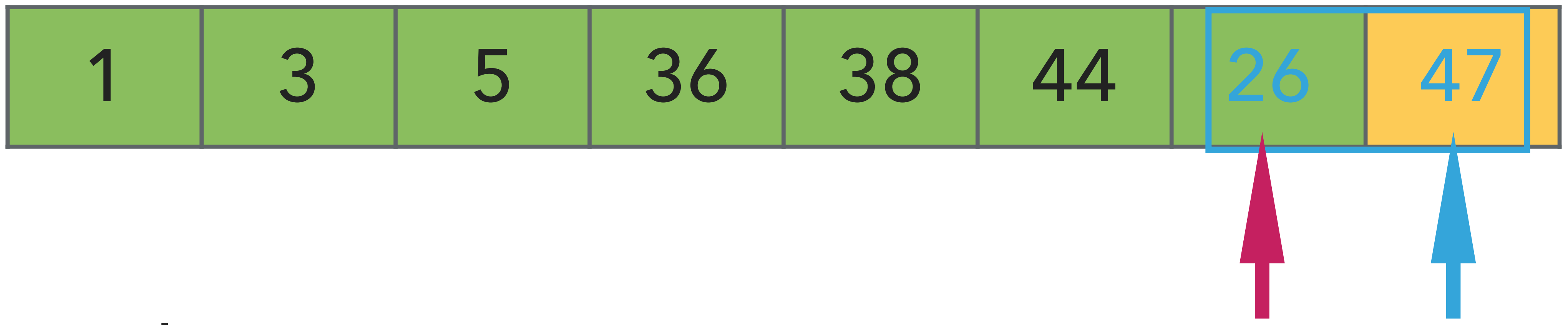
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



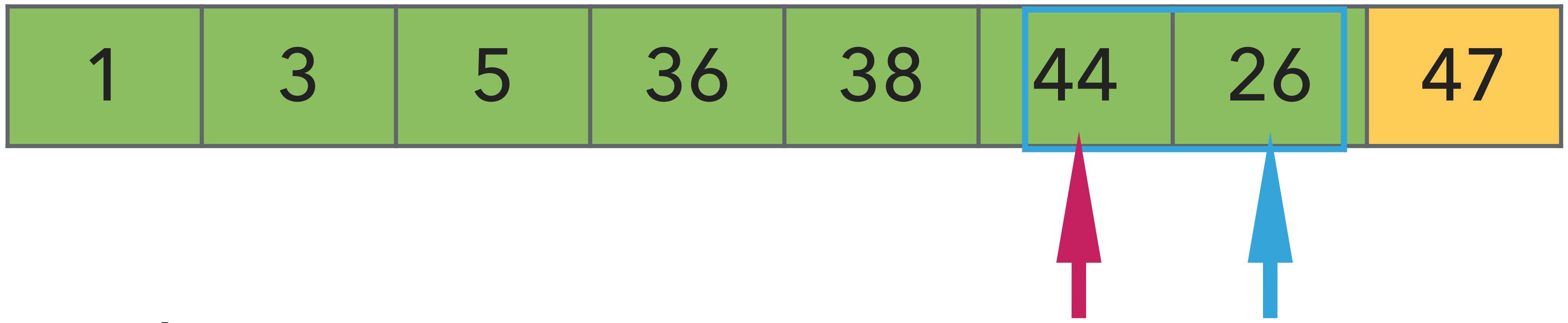
- Repeat:
    - Examine the next element in the unsorted subarray.
    - Find the location it belongs within the sorted subarray and insert it there.
    - Move subarray boundaries one element to the right.

Insertion sort



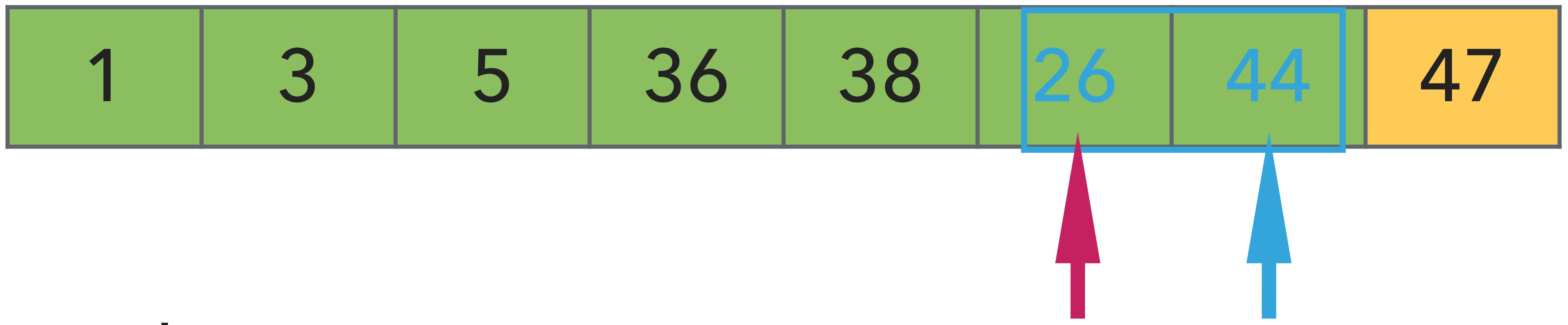
- Repeat:
    - Examine the next element in the unsorted subarray.
    - Find the location it belongs within the sorted subarray and insert it there.
    - Move subarray boundaries one element to the right.

Insertion sort



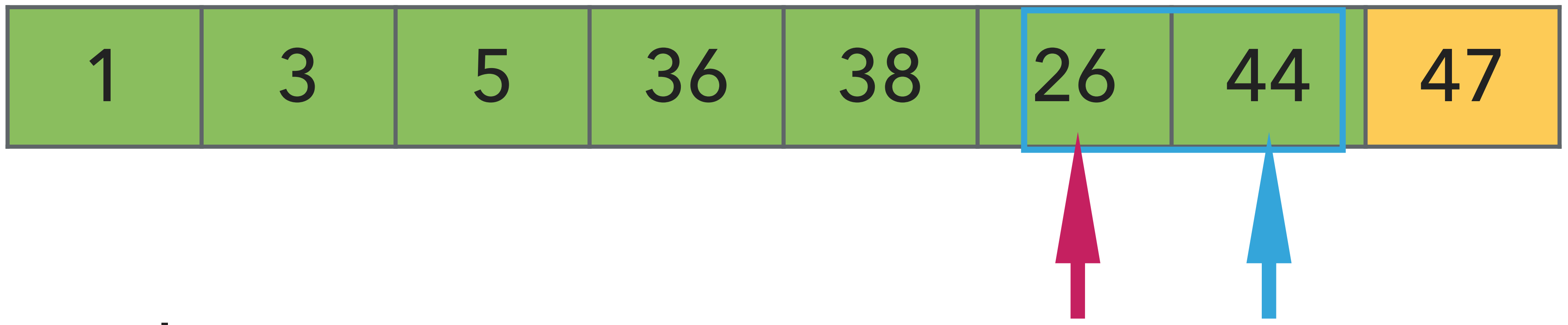
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



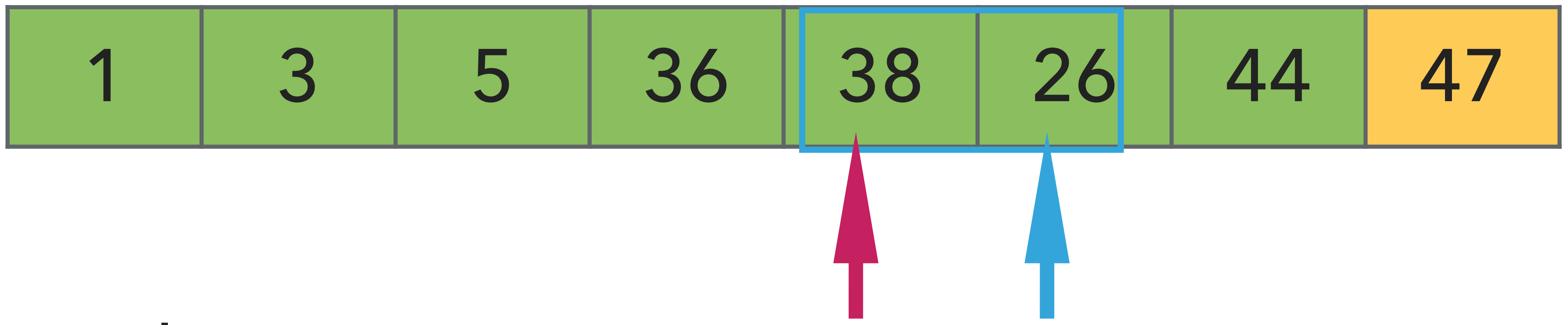
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



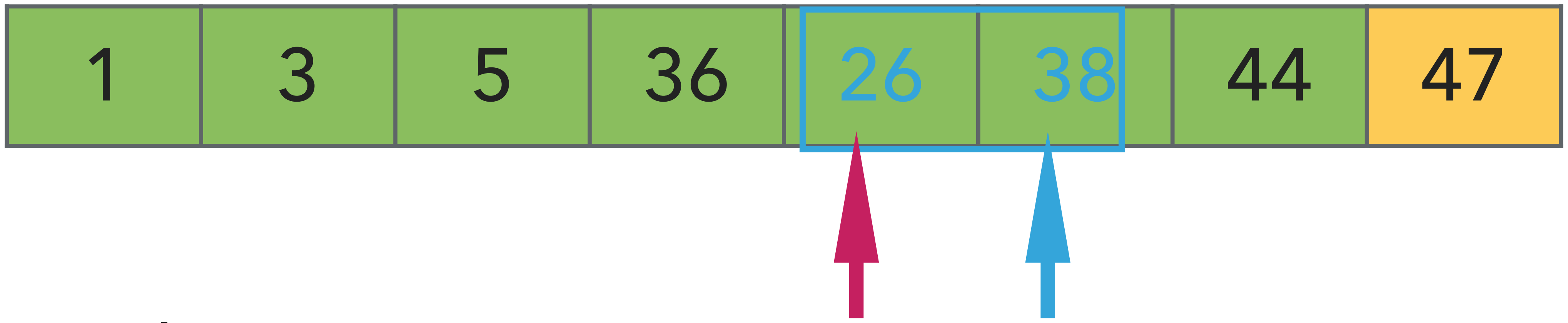
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



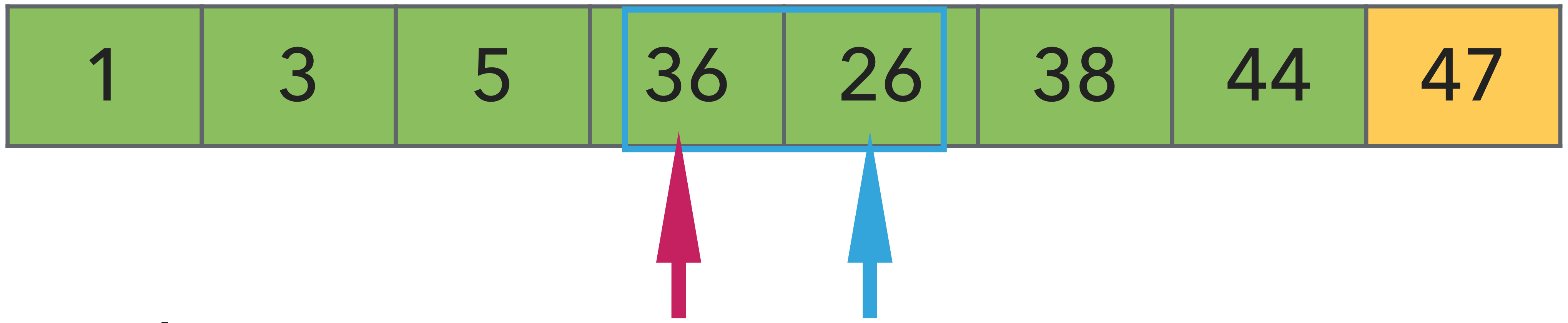
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



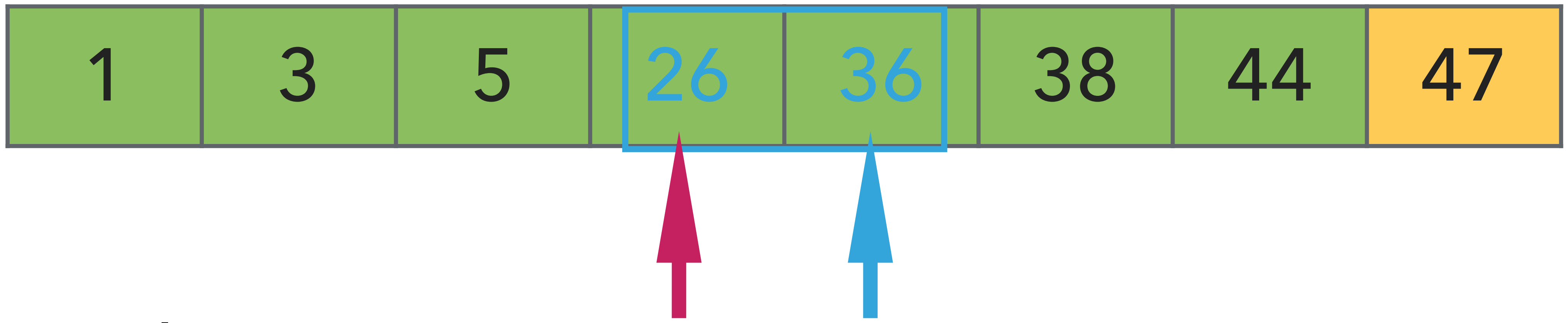
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



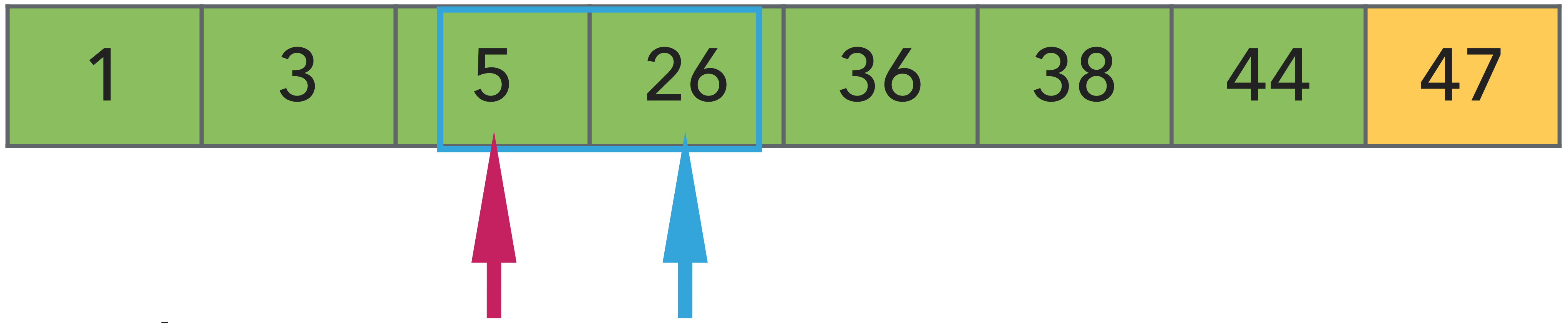
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



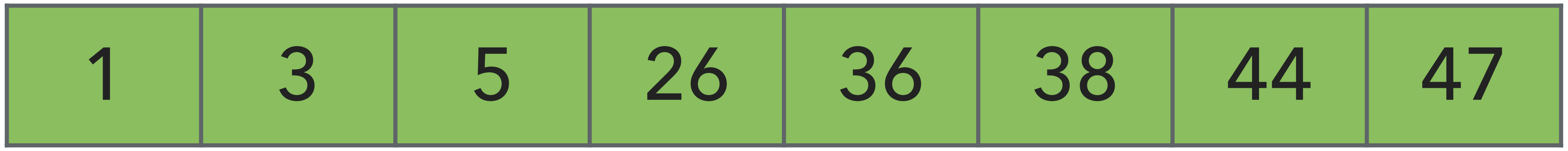
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

http://algs4.cs.princeton.edu

2.1 INSERTION SORT DEMO

https://algs4.cs.princeton.edu/lectures/demo/21DemoInsertionSort.mov

In case you didn't get this…

- https://www.youtube.com/watch?v=ROalU379l3U

PRACTICE TIME - Implement insertion sort

```java
public static <E extends Comparable<E>> void insertionSort(E[] a) {









 }
```

Insertion sort

```
public static <E extends Comparable<E>> void insertionSort(E[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
            for (int j = i; j > 0; j--) {
                if(a[j].compareTo(a[j-1])<0){
                    E temp = a[j];
                    a[j]=a[j-1];
                    a[j-1]=temp;
                }
                else{
                    break;
                }
            }
        }
 }
```

- Invariants: At the end of each iteration `i`:
    - the array *a* is sorted in ascending order for the first `i+1` elements `a[0…i]`

Insertion sort: mathematical analysis for worst-case

```
public static <E extends Comparable<E>> void insertionSort(E[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
           for (int j = i; j > 0; j--) {
               if(a[j].compareTo(a[j-1])<0){
                   E temp = a[j];
                   a[j]=a[j-1];
                   a[j-1]=temp;
               }
               else{
                  break;
               }
           }
       }
 }
```

- Comparisons: $0 + 1 + 2 + \ldots + (n-2) + (n-1) \sim n^2/2$, that is $O(n^2)$.
- Exchanges: $0 + 1 + 2 + \ldots + (n-2) + (n-1) \sim n^2/2$, that is $O(n^2)$.
- Worst-case running time is quadratic.
- In-place, requires almost no additional memory.
- Stable

Insertion sort: average and best case

```
public static <E extends Comparable<E>> void insertionSort(E[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
           for (int j = i; j > 0; j--) {
               if(a[j].compareTo(a[j-1])<0){
                   E temp = a[j];
                   a[j]=a[j-1];
                   a[j-1]=temp;
               }
               else{
                  break;
               }
           }
       }
   }
```

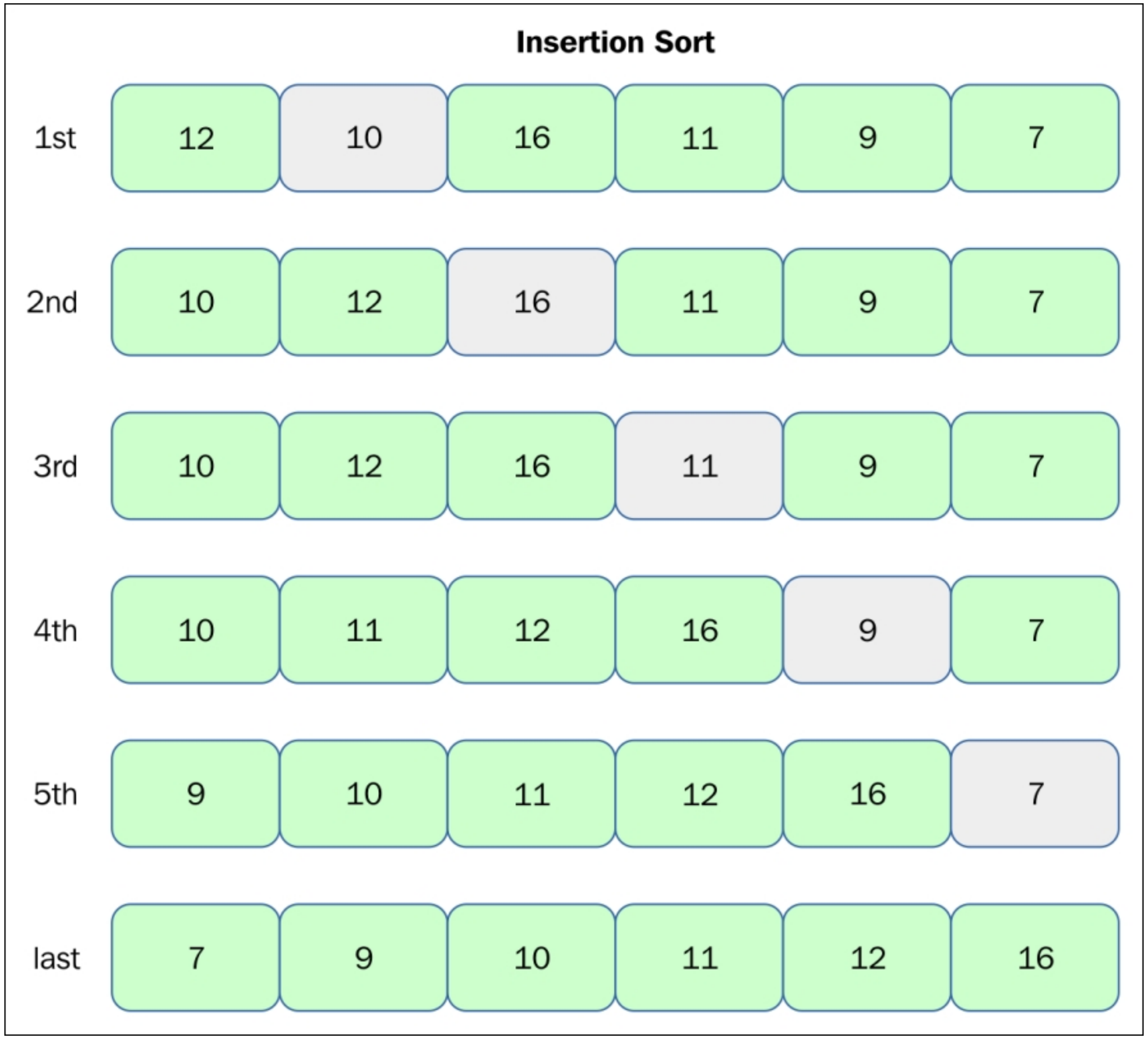
- Best case: $n - 1$ comparisons and 0 exchanges for an already sorted array.
- Average case: quadratic for both comparisons and exchanges $\sim n^2/4$ when sorting a randomly ordered array.

https://www.toptal.com/developers/sorting-algorithms/insertion-sort

Practice Time - Worksheet

- Using insertion sort, sort the array with elements [12,10,16,11,9,7].
- Visualize your work for every iteration of the algorithm.

Answer



https://subscription.packtpub.com/book/application_development/9781785888731/13/ch13lvl1sec90/insertion-sort?query=insertion%20sort

Lecture 13: Insertion Sort

- Insertion sort

Readings:

- Recommended Textbook:
  - Chapter 2.1 (pages 244–262)
  - Chapter 2.5 (Pages 338-339)
- Recommended Textbook Website:
  - Elementary sorts: https://algs4.cs.princeton.edu/21elementary/

Code

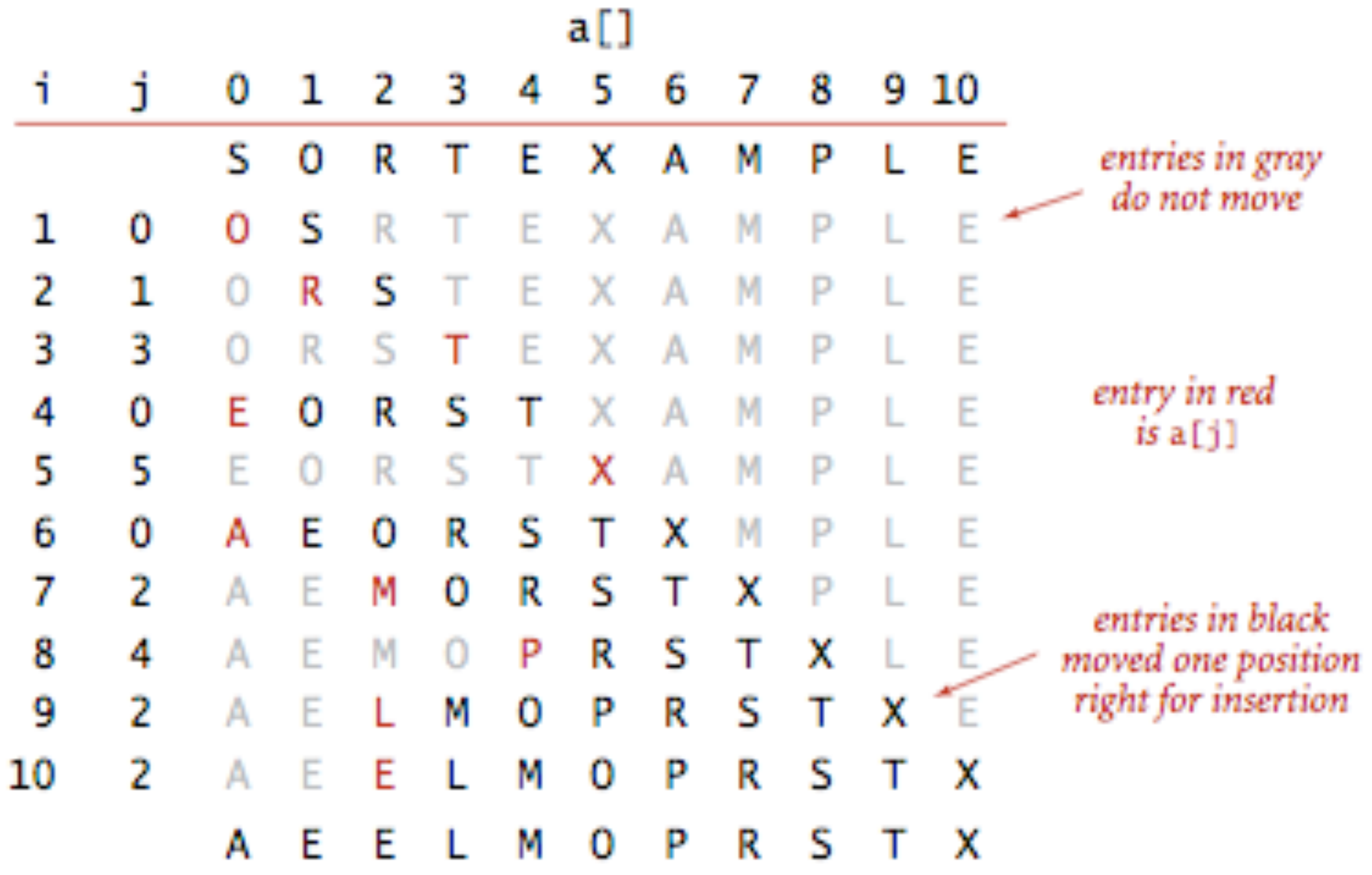
- Lecture 13 code

Worksheet

- Lecture 13 worksheet

Practice Problem 1 - Recommended textbook 2.1.4

- Show all the steps of how insertion sort would sort [E, A, S, Y, Q, U, E, S, T, I, O, N] in the style of the following trace which visualizes the array contents just after each insertion.



		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
1	0	O	S	R	T	E	X	A	M	P	L	E
2	1	O	R	S	T	E	X	A	M	P	L	E
3	3	O	R	S	T	E	X	A	M	P	L	E
4	0	E	O	R	S	T	X	A	M	P	L	E
5	5	E	O	R	S	T	X	A	M	P	L	E
6	0	A	E	O	R	S	T	X	M	P	L	E
7	2	A	E	M	O	R	S	T	X	P	L	E
8	4	A	E	M	O	P	R	S	T	X	L	E
9	2	A	E	L	M	O	P	R	S	T	X	E
10	2	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

entries in gray do not move
entry in red is a[j]
entries in black moved one position right for insertion

Trace of insertion sort (array contents just after each insertion)

Practice Problem 2

- Describe an array of n elements where the if statement in the inner loop is always false and the loop terminates. Now describe an array of n elements where the if statement is always satisfied.

Practice Problem 3 - Recommended textbook 2.1.6

- Which method runs faster for an array with all keys identical, selection sort or insertion sort?

Practice Problem 4 - Recommended textbook 2.1.7

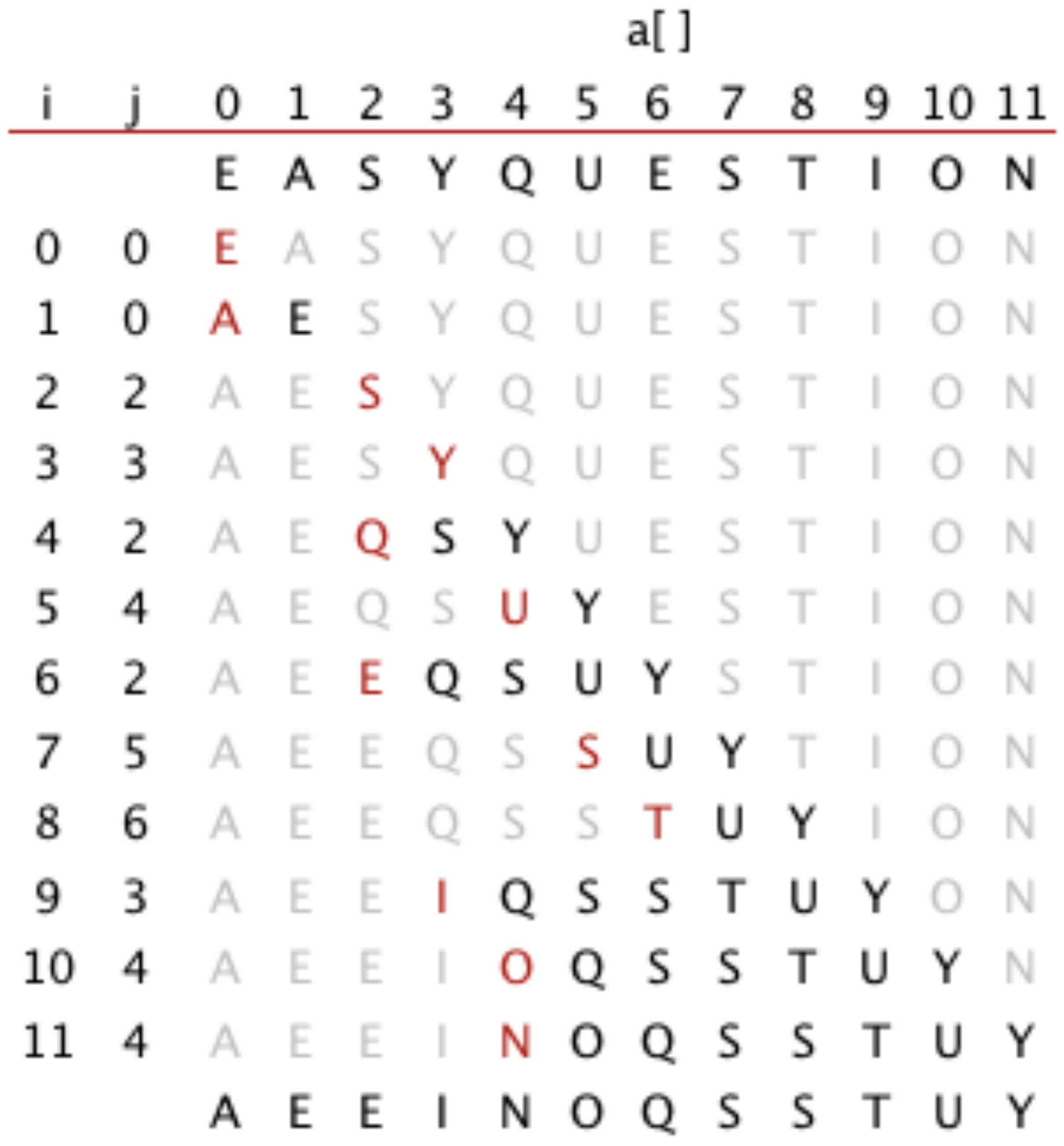
- Which method runs faster for an array in reverse order, selection sort or insertion sort?

Practice Problem 5 - Recommended textbook 2.1.8

- Suppose that we use insertion sort on a randomly ordered array where items have only one of three values. Is the running time linear, quadratic, or something in between?

ANSWER 1

- Show all the steps of how insertion sort would sort [E, A, S, Y, Q, U, E, S, T, I, O, N] in the style of the following trace which visualizes the array contents just after each insertion.



		a[ ]											
i	j	0	1	2	3	4	5	6	7	8	9	10	11
		E	A	S	Y	Q	U	E	S	T	I	O	N
0	0	E	A	S	Y	Q	U	E	S	T	I	O	N
1	0	A	E	S	Y	Q	U	E	S	T	I	O	N
2	2	A	E	S	Y	Q	U	E	S	T	I	O	N
3	3	A	E	S	Y	Q	U	E	S	T	I	O	N
4	2	A	E	Q	S	Y	U	E	S	T	I	O	N
5	4	A	E	Q	S	U	Y	E	S	T	I	O	N
6	2	A	E	E	Q	S	U	Y	S	T	I	O	N
7	5	A	E	E	Q	S	S	U	Y	T	I	O	N
8	6	A	E	E	Q	S	S	T	U	Y	I	O	N
9	3	A	E	E	I	Q	S	S	T	U	Y	O	N
10	4	A	E	E	I	O	Q	S	S	T	U	Y	N
11	4	A	E	E	I	N	O	Q	S	S	T	U	Y
		A	E	E	I	N	O	Q	S	S	T	U	Y

ANSWER 2

- Describe an array of n elements where the if statement in the inner loop is always false and the loop terminates. Now describe an array of n elements where the if statement is always satisfied.

- if statement always false when the array is already sorted, e.g., [1, 2, 3, 4]
- if statement always true when the array is in reverse order, e.g., [4, 3, 2, 1].

ANSWER 3

- Which method runs faster for an array with all keys identical, selection sort or insertion sort?
- Insertion sort is faster because it will only make one comparison per element (i.e., is linear) and will not need to exchange any elements. Instead, selection sort will still run in quadratic time.

ANSWER 4

- Which method runs faster for an array in reverse order, selection sort or insertion sort?
- Selection sort. Big O says both are quadratic, but selection sort needs only n exchanges, while insertion sort $n^2/2$ exchanges

ANSWER 5

- Suppose that we use insertion sort on a randomly ordered array where items have only one of three values. Is the running time linear, quadratic, or something in between?
- Quadratic. Insertion sort's running time is linear when the array is already sorted or all elements are equal. With three possible values the running time is quadratic.