CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

10: Finish Queues, Sorting Fundamentals

Tom Yeh
he/him/his

Textbook implementation of queues

- ResizingArrayQueue.java: for implementation of queues with ArrayLists.
- LinkedQueue.java: for implementation of queues with singly linked lists.

Stacks, Queues, and Iterators

- Stacks
- Queues
- Applications
- Java Collections
- Iterators

Stack applications

- Java Virtual Machine.
- Basic mechanisms in compilers, interpreters (see CS101).
- Back button in browser.
- Undo in word processor.
- Infix expression evaluation (Dijskstra’s algorithm with two stacks).
- Postfix expression evaluation.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

Algorithms

FOURTH EDITION

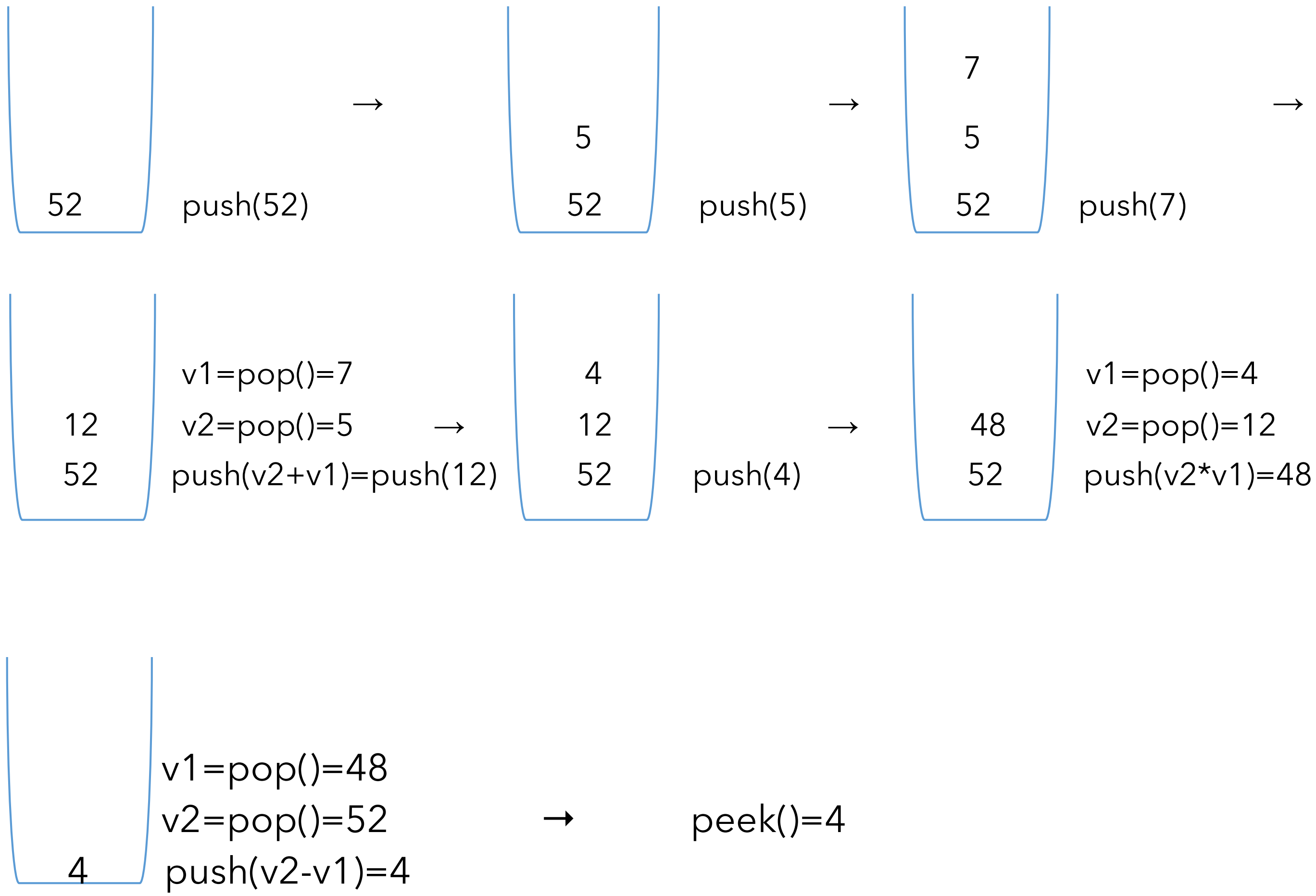
ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

1.3 DIJKSTRA'S 2-STACK DEMO

Postfix expression evaluation example (Calculator Assignment)

Example: (52 - ((5 + 7) * 4) ⇒ 52 5 7 + 4 * -



52 push(52) → 5 52 push(5) → 7 5 52 push(7) →

12 52 v1=pop()=7 v2=pop()=5 push(v2+v1)=push(12) → 4 12 52 push(4) → 48 52 v1=pop()=4 v2=pop()=12 push(v2*v1)=48

4 v1=pop()=48 v2=pop()=52 push(v2-v1)=4 → peek()=4

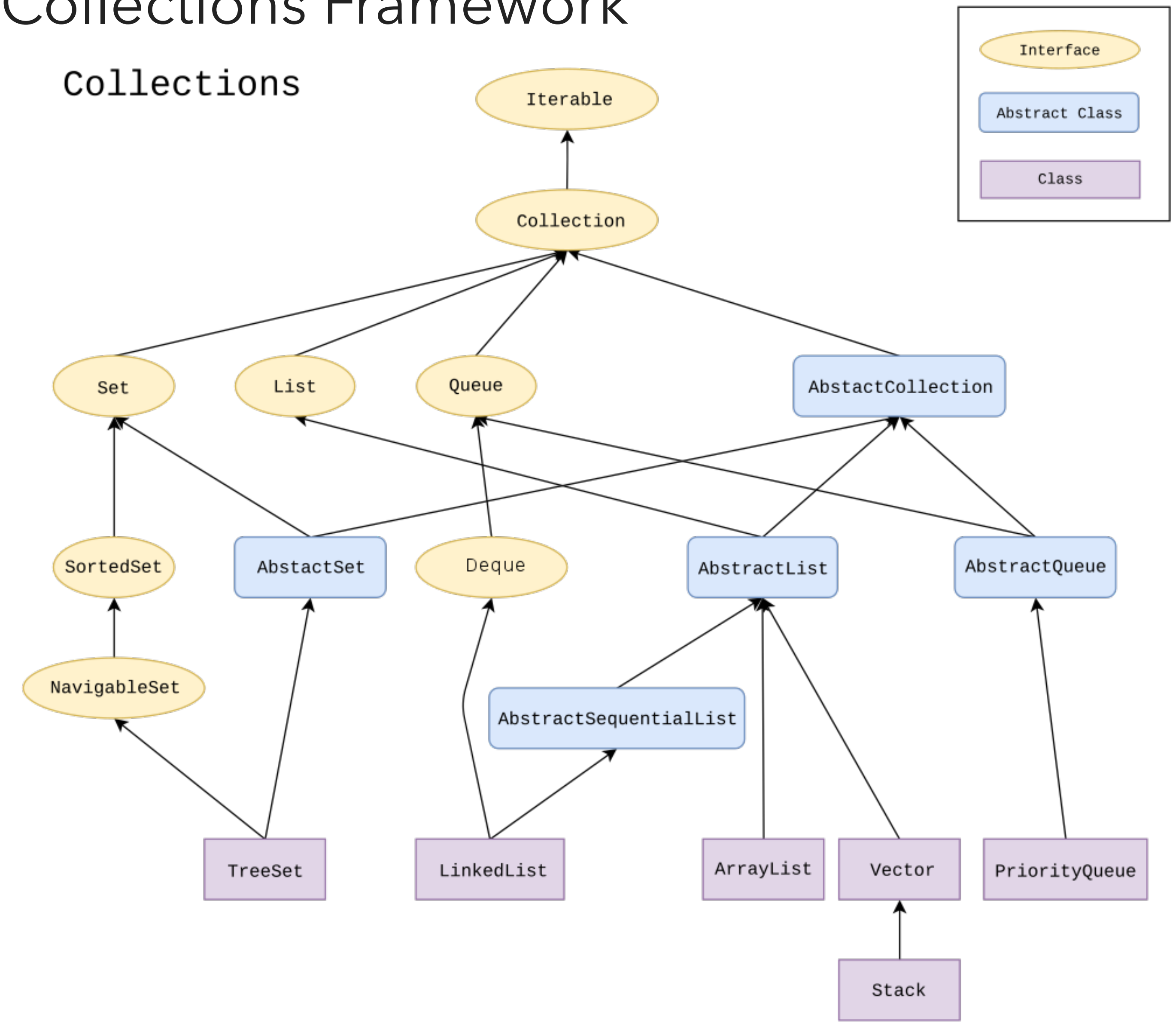
Queue applications

- Spotify playlist.
- Data buffers (netflix, Hulu, etc.).
- Asynchronous data transfer (file I/O, sockets).
- Requests in shared resources (printers).
- Traffic analysis.
- Waiting times at calling center.

Lecture 9: Stacks, Queues, and Iterators

- Stacks
- Queues
- Applications
- Java Collections
- Iterators

The Java Collections Framework



https://en.wikipedia.org/wiki/Java_collections_framework

Deque in Java Collections

- Do not use Stack.
- Queue is an interface…
- It's recommended to use Deque instead.
- Double-ended queue (can add and remove from either end).

```
java.util.Deque;
```

```
public interface Deque<E> extends Queue<E>
```

- You can choose between LinkedList and ArrayDeque implementations.
    - `Deque deque = new ArrayDeque(); //preferable`

https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html

Lecture 9: Stacks, Queues, and Iterators

- Stacks
- Queues
- Applications
- Java Collections
- Iterators

Iterator Interface

- Interface that allows us to traverse a collection one element at a time.

```
public interface Iterator<E> {
  //returns true if the iteration has more elements
  //that is if next() would return an element instead of throwing an exception
  boolean hasNext();

  //returns the next element in the iteration
  //post: advances the iterator to the next value
  E next();

  //removes the last element that was returned by next
  default void remove(); //optional, better avoid it altogether
}
```

https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

Iterator Example

```
List<String> myList = new ArrayList<String>();
//… operations on myList

Iterator listIterator = myList.iterator();

while(listIterator.hasNext()){
  String elt = listIterator.next();
  System.out.println(elt);
}
```

Iterable Interface

- Interface that allows an object to be the target of a for-each loop:

```
for(String elt: myList){
  System.out.println(elt);
}
```

```
interface Iterable<E>{
  //returns an iterator over elements of type E
  Iterator<E> iterator();


}
```

https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html

How to make your data structures iterable?

1. Implement `Iterable` interface.
2. Make a private class that implements the `Iterator` interface.
3. Override `iterator()` method to return an instance of the private class.

Example: making ArrayList iterable

```
public class ArrayList<Item> implements Iterable<Item> {
     //…
    public Iterator<Item> iterator() {

        return new ArrayListIterator();
     }


    private class ArrayListIterator implements Iterator<Item> {

        private int i = 0;

        public boolean hasNext() {
               return i < n;

        }

        public Item next() {

            return a[i++];

        }

        public void remove() {
                throw new UnsupportedOperationException();

        }
```

Traversing `ArrayList`

- All valid ways to traverse ArrayList and print its elements one by one.

```
for(String elt:a1) {
    System.out.println(elt);
}

a1.forEach(System.out::println);
a1.forEach(elt->{System.out.println(elt);});

a1.iterator().forEachRemaining(System.out::println);
a1.iterator().forEachRemaining(elt->{System.out.println(elt);});
```

Lecture 9: Stacks, Queues, and Iterators

- Stacks
- Queues
- Applications
- Java Collections
- Iterators

Readings:

- Oracle's guides:
  - Collections: https://docs.oracle.com/javase/tutorial/collections/intro/index.html
  - Deque: https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html
  - Iterator: https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html
  - Iterable: https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html
- Textbook:
  - Chapter 1.3 (Page 126–157)
- Website:
  - Stacks and Queues: https://algs4.cs.princeton.edu/13stacks/

Practice Problems:

- 1.3.2–1.3.8, 1.3.32–1.3.33

TEXT

Lecture 12: Sorting Fundamentals

- Introduction
- Selection sort
- Insertion sort

Some slides adopted from Algorithms 4th Edition or COS226

Why study sorting?

- It’s more common than you think: e.g., sorting flights by price, contacts by last name, files by size, emails by day sent, neighborhoods by zipcode, etc.
- Good example of how to compare the performance of different algorithms for the same problem.
- Some sorting algorithms relate to data structures.
- Sorting your data will often be a good starting point when solving other problems (keep that in mind for interviews).

Definitions

- Sorting: the process of arranging n items of a collection in non-decreasing order (e.g., numerically or alphabetically).
- Key: assuming that an item consists of multiple components, the key is the property based on which we sort items.
    - Examples: items could be books and potential keys are the title or the author which can be sorted alphabetically or the ISBN which can be sorted numerically.

Total order

- Sorting is well defined if and only if there is total order.
- Total order: a binary relation $\leq$ on a set C that satisfies the following statements for all v, w, and x in C:
    - Connexity: $v \leq w$ or $w \leq v$.
    - Transitivity: for all v, w, x, if $v \leq w$ and $w \leq x$ then $v \leq x$.
    - Antisymmetry: if both $v \leq w$ and $w \leq v$, then $v = w$.

How many different algorithms for sorting can there be?

- Adaptive heapsort
- Bitonic sorter
- Block sort
- Bubble sort
- Bucket sort
- Cascade mergesort
- Cocktail sort
- Comb sort
- Flashsort
- Gnome sort
- **Heapsort**
- **Insertion sort**
- Library sort
- **Mergesort**
- Odd-even sort
- Pancake sort
- **Quicksort**
- Radixsort
- **Selection sort**
- Shell sort
- Spaghetti sort
- Treesort
- …

Rules of the game - Comparing

- We will be sorting arrays of n items, where each item contains a key. In Java, objects are responsible in telling us how to *naturally* compare their keys.
- Let's say we want to sort an array of objects of type T.
- Our class T should implement the `Comparable` interface (more on this in a few lectures). We will need to implement the `compareTo` method to satisfy a total order.

https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

Rules of the game - Comparing

- `public int compareTo(T that)`
- Implement it so that `v.compareTo(w)`:
  - Returns >0 if v is greater than `w`.
  - Returns <0 if v is smaller than `w`.
  - Returns 0 if v is equal to `w`.
- Java classes such as `Integer`, `Double`, `String`, `File` all implement `Comparable`.

https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

Two useful abstractions

- We will refer to data only through comparisons and exchanges.
- Less: Is v less than w?

```
private static boolean less(Comparable v, Comparable w) {
     return v.compareTo(w) < 0;
}
```

- Exchange: swap item in array a[] at index i with the one at index j.

```
private static void exch(Comparable[] a, int i, int j) {
      Comparable swap = a[i];
      a[i]=a[j];

      a[j]=swap;
}
```

Rules of the game - Cost model

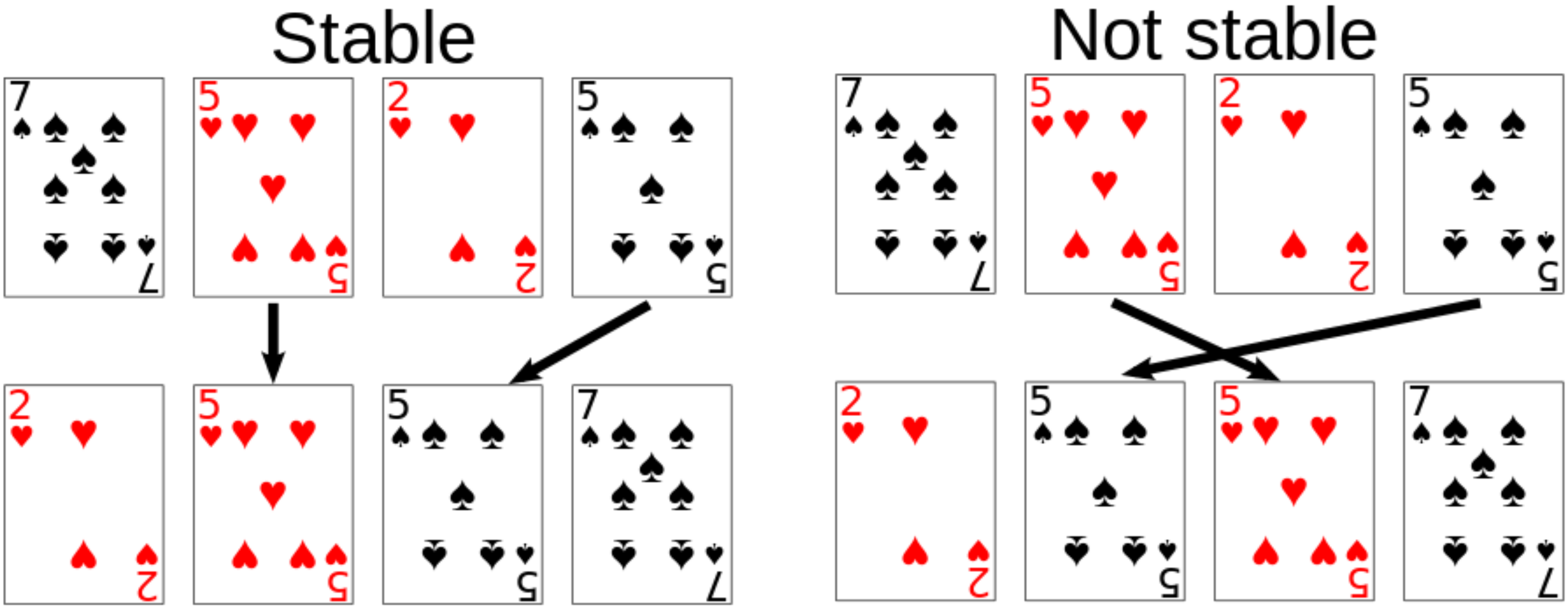
- Sorting cost model: we count compares and exchanges. If a sorting algorithm does not use exchanges, we count array accesses.
- There are other types of sorting algorithms where they are not based on comparisons (e.g., radixsort). We will not see these in CS62 but stay tuned for CS140.

Rules of the game - Memory usage

- Extra memory: often as important as running time. Sorting algorithms are divided into two categories:
  - In place: use constant or logarithmic extra memory, beyond the memory needed to store the items to be sorted.
  - Not in place: use linear auxiliary memory.

Rules of the game - Stability

- Stable: sorting algorithms that sort repeated elements in the same order that they appear in the input.



https://en.wikipedia.org/wiki/Sorting_algorithm#/media/File:Sorting_stability_playing_cards.svg

Lecture 12: Sorting Fundamentals

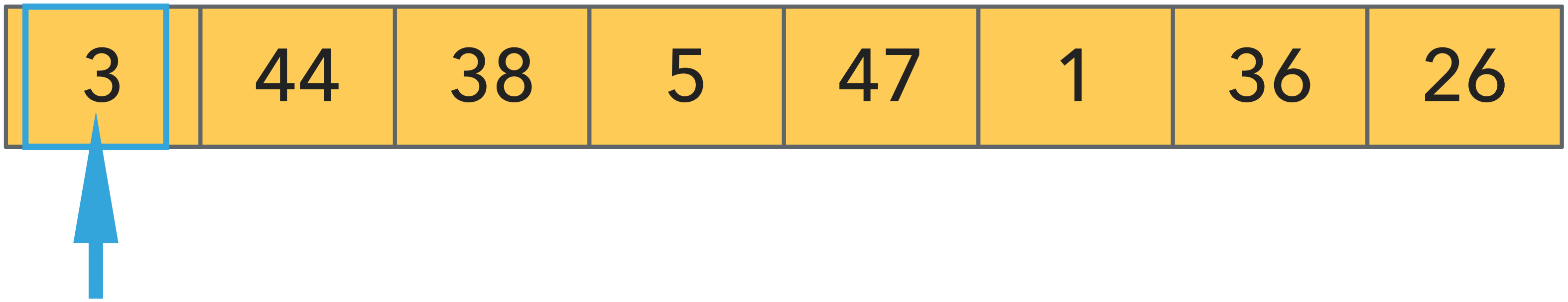
- Introduction
- Selection sort
- Insertion sort

Selection sort

3	44	38	5	47	1	36	26

- Divide the array in two parts: a sorted subarray on the left and an unsorted on the right.
- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

Selection sort



3	44	38	5	47	1	36	26

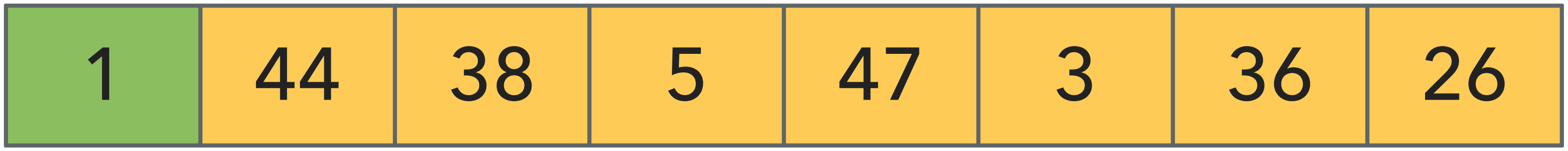
- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

Selection sort

1	44	38	5	47	3	36	26

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

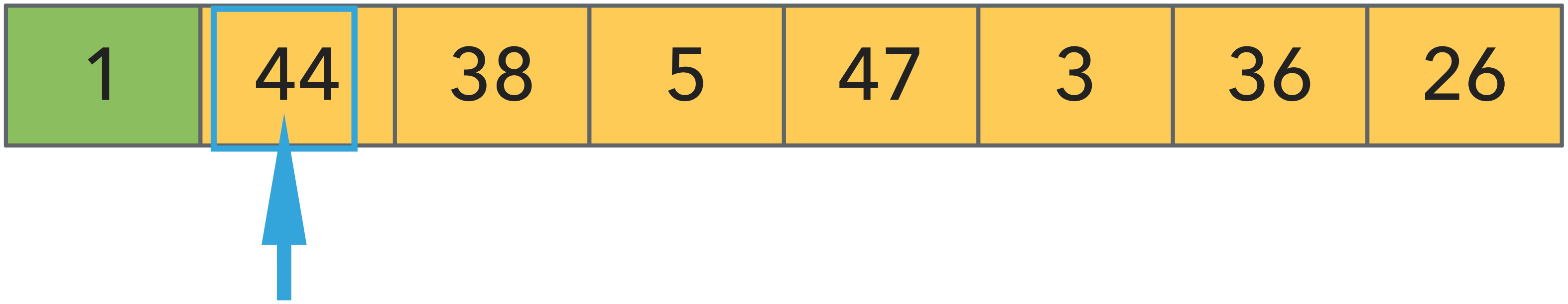
Selection sort



1	44	38	5	47	3	36	26

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

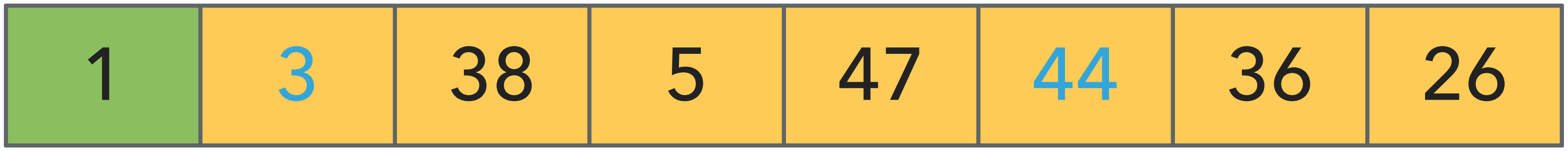
Selection sort



1	44	38	5	47	3	36	26

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

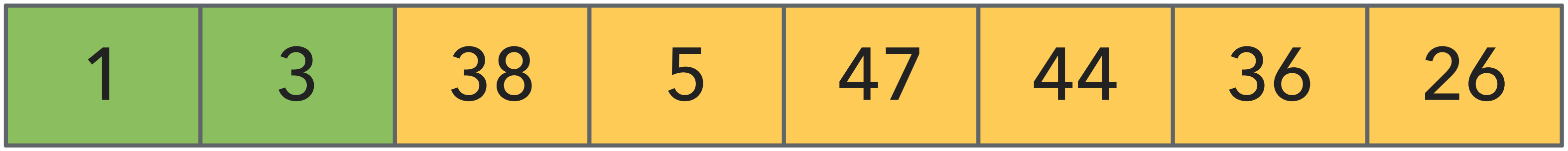
Selection sort



1	3	38	5	47	44	36	26

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

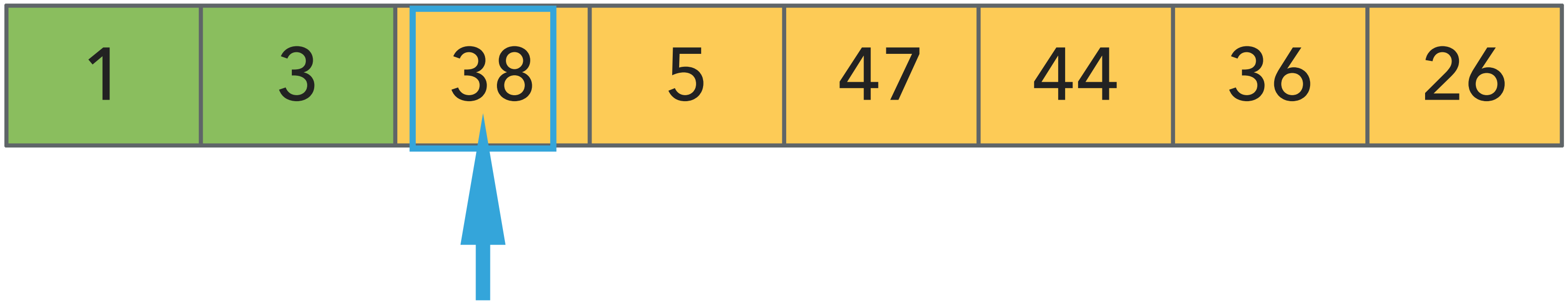
Selection sort



1	3	38	5	47	44	36	26

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

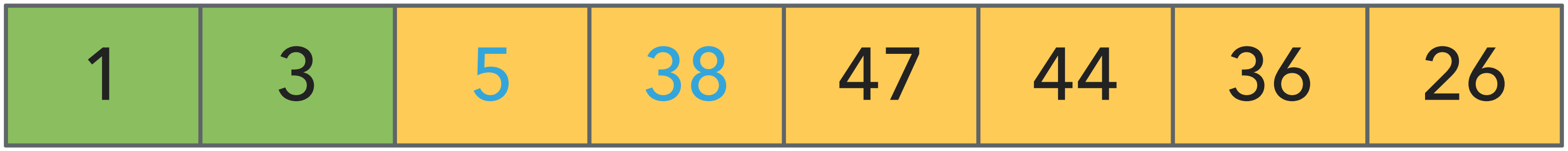
Selection sort



1	3	38	5	47	44	36	26

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

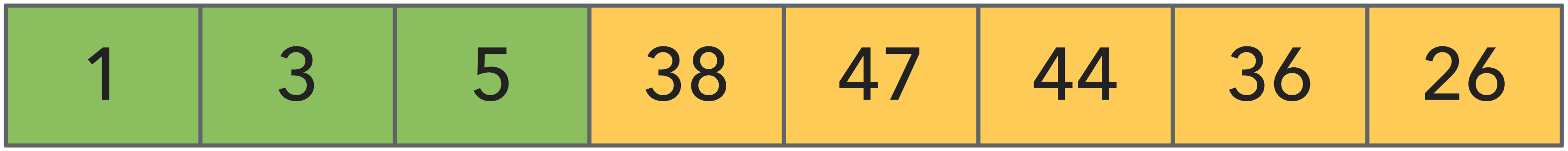
Selection sort



1	3	5	38	47	44	36	26

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

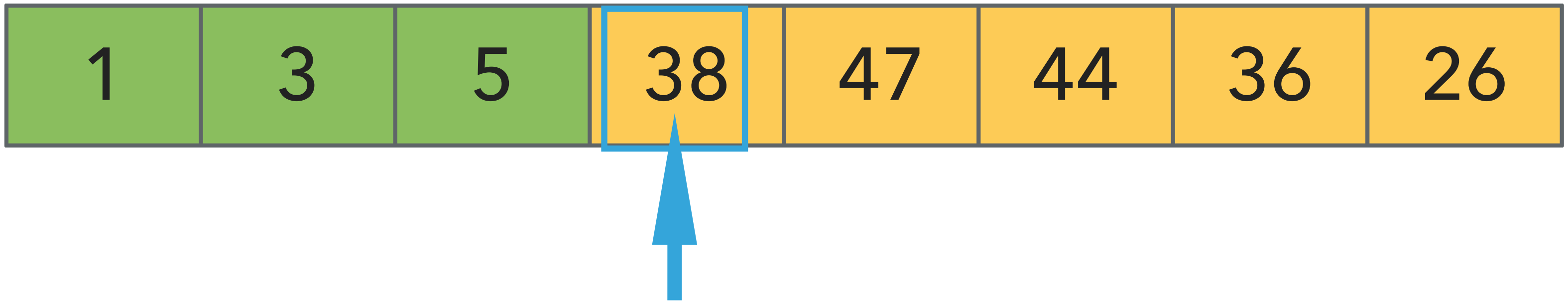
Selection sort



1	3	5	38	47	44	36	26

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

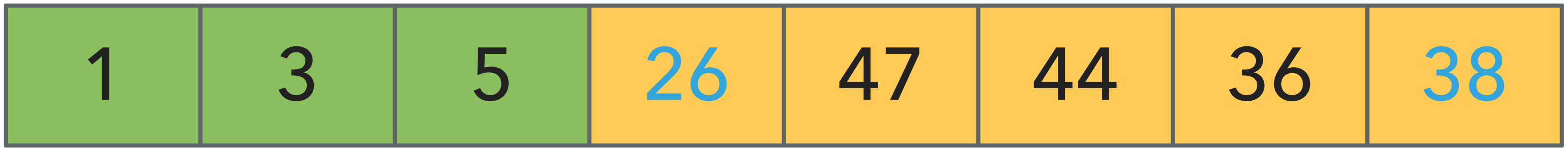
Selection sort



1	3	5	38	47	44	36	26

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

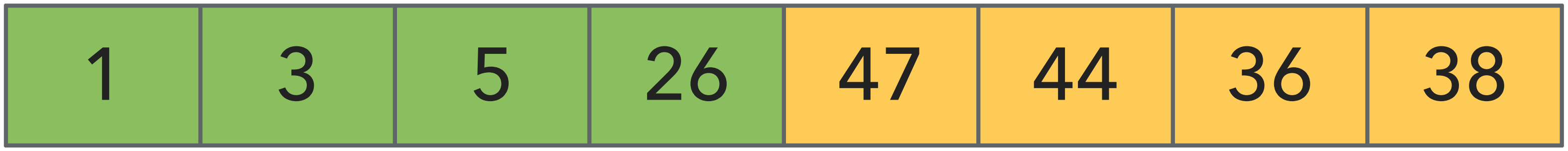
Selection sort



1	3	5	26	47	44	36	38

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

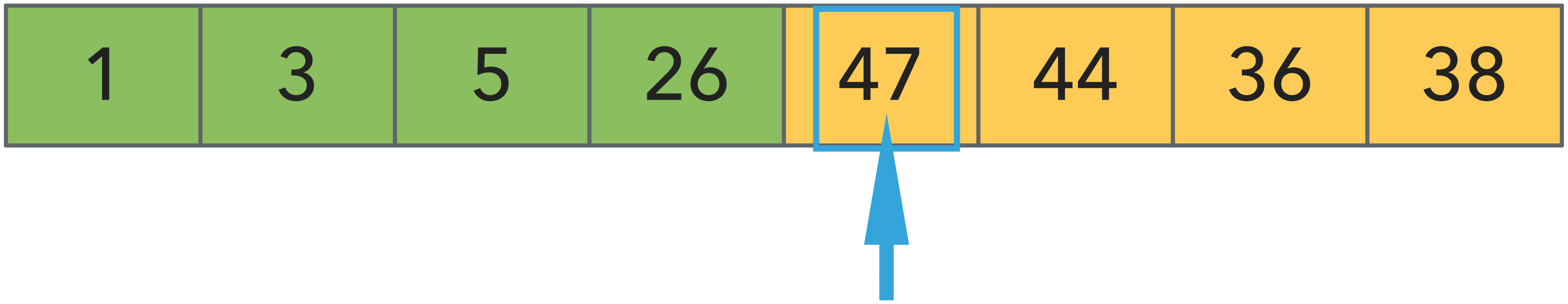
Selection sort



1	3	5	26	47	44	36	38

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

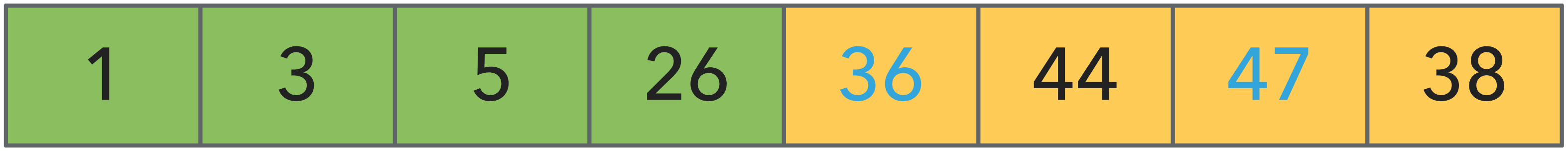
Selection sort



1	3	5	26	47	44	36	38

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

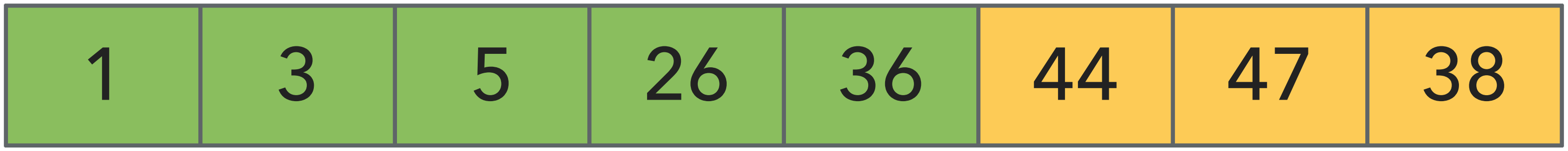
Selection sort



1	3	5	26	36	44	47	38

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

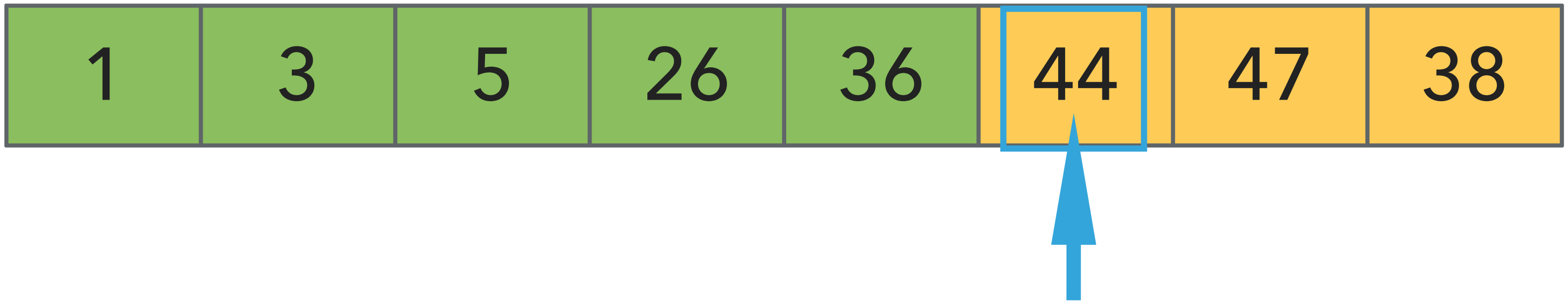
Selection sort



1	3	5	26	36	44	47	38

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

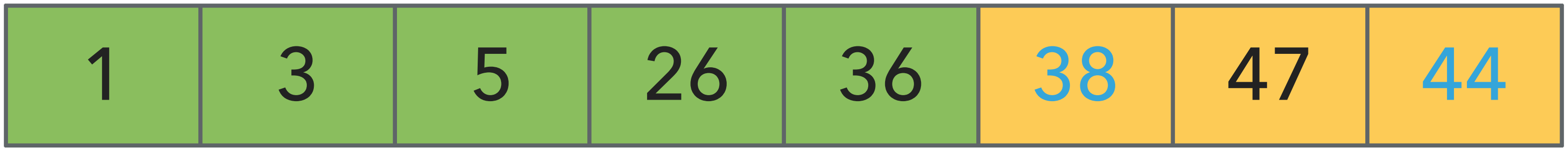
Selection sort



1	3	5	26	36	44	47	38

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

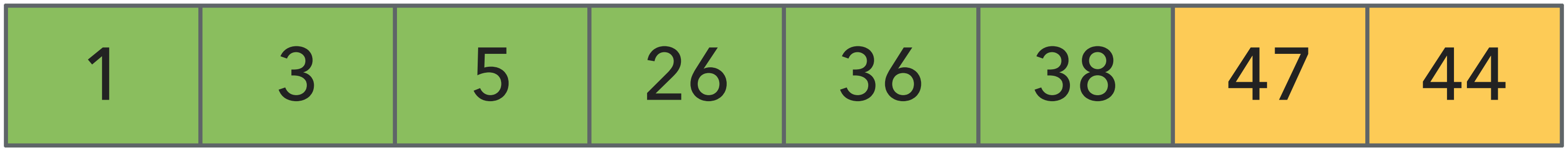
Selection sort



1	3	5	26	36	38	47	44

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

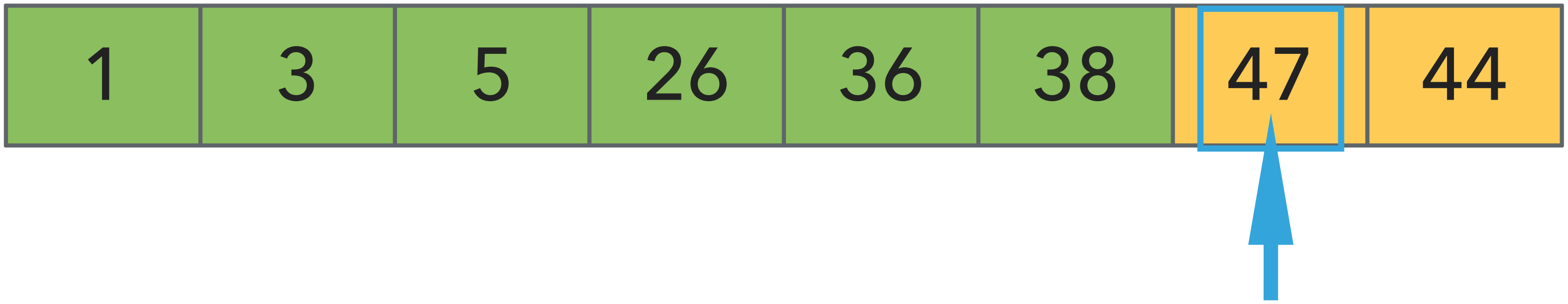
Selection sort



1	3	5	26	36	38	47	44

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

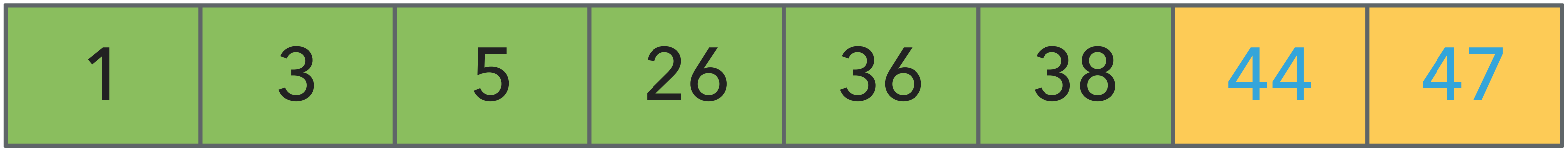
Selection sort



1	3	5	26	36	38	47	44

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

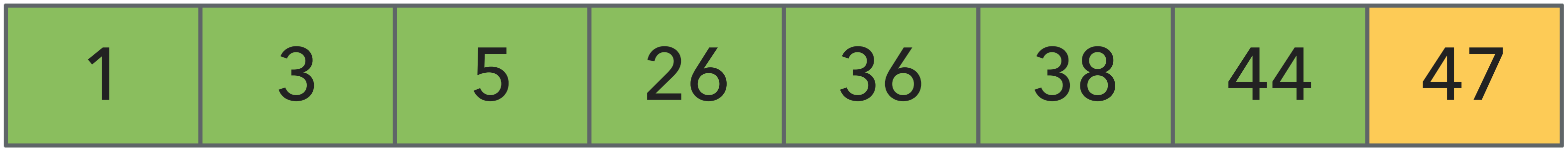
Selection sort



1	3	5	26	36	38	44	47

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

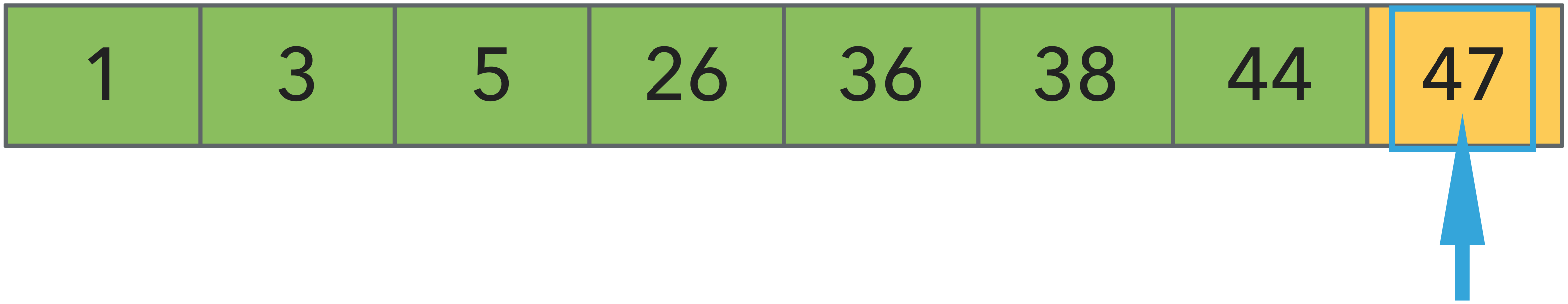
Selection sort



1	3	5	26	36	38	44	47

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

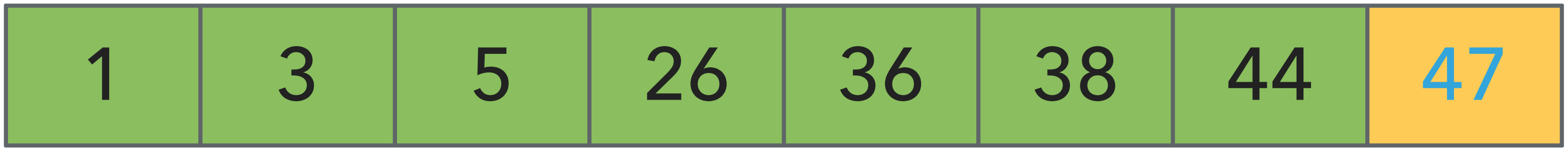
Selection sort



1	3	5	26	36	38	44	47

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

Selection sort



1	3	5	26	36	38	44	47

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

Selection sort

1	3	5	26	36	38	44	47

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.

Selection sort

```
public static void sort(Comparable[] a) {









   }
```

Selection sort

```
public static void sort(Comparable[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
            int min = i;
            for (int j = i+1; j < n; j++) {
                if (less(a[j], a[min]))
                    min = j;
            }
            exch(a, i, min);
       }
   }
```

← In iteration i

← Find the index min of the smallest remaining array

← swap a[i] and a[min]

- 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]
  - no entry in a[i+1…n-1] is smaller than any entry in a[0…i]

Selection sort: mathematical analysis for worst-case

```
public static void sort(Comparable[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
            int min = i;
            for (int j = i+1; j < n; j++) {
                if (less(a[j], a[min]))
                    min = j;
            }
            exch(a, i, min);
       }
   }
```

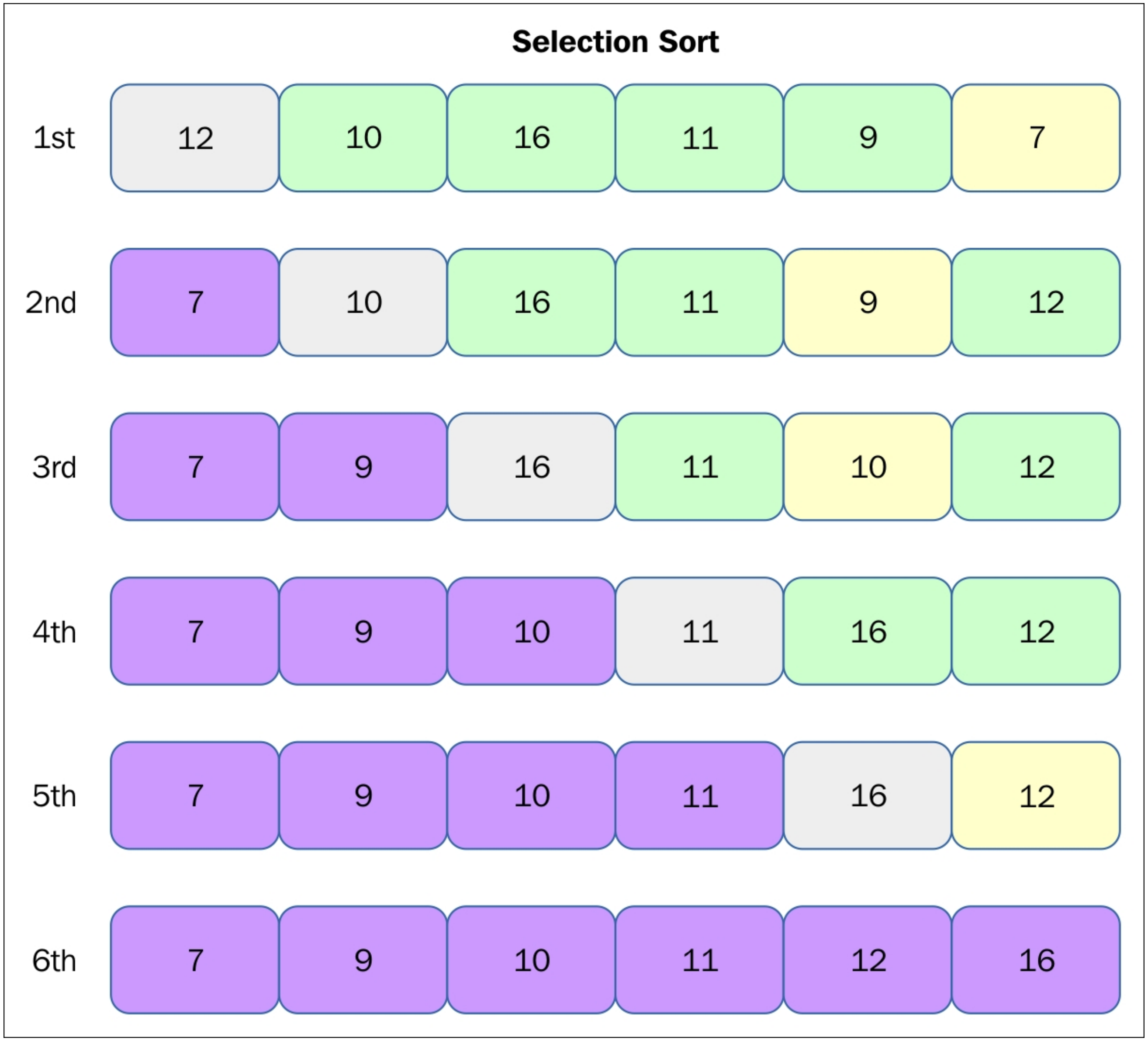
- Comparisons: $1 + 2 + \ldots + (n-2) + (n-1) \sim n^2/2$, that is $O(n^2)$.
- Exchanges: n or $O(n)$, making it useful when exchanges are expensive.
- Running time is quadratic, even if input is sorted.
- In-place, requires almost no additional memory.
- Not stable, think of the array [5_a, 3, 5_b, 1] which will end up as [1, 3, 5_b, 5_a].

Practice Time

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

SELECTION SORT

Answer



Selection Sort

1st	12	10	16	11	9	7
2nd	7	10	16	11	9	12
3rd	7	9	16	11	10	12
4th	7	9	10	11	16	12
5th	7	9	10	11	16	12
6th	7	9	10	11	12	16

https://subscription.packtpub.com/book/application_development/9781785888731/13/ch13lvl1sec89/selection-sort

Lecture 12: Sorting Fundamentals

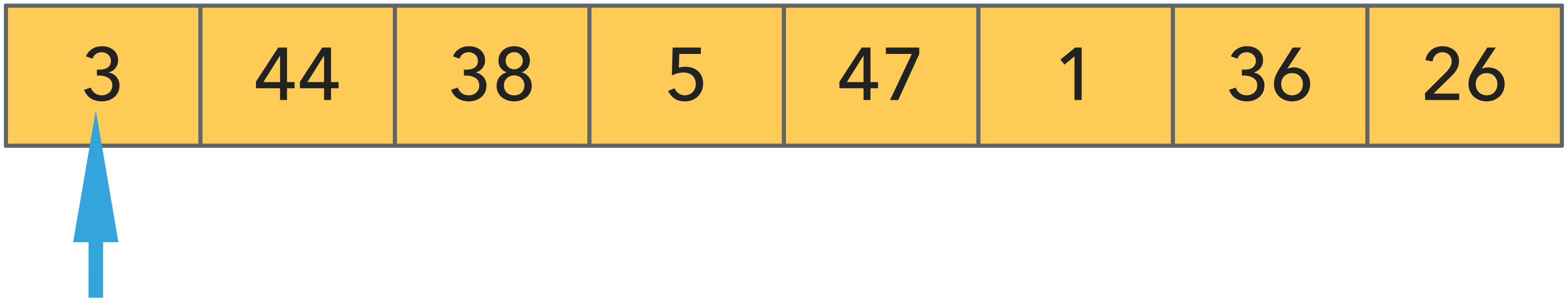
- Introduction
- Selection sort
- Insertion sort

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.

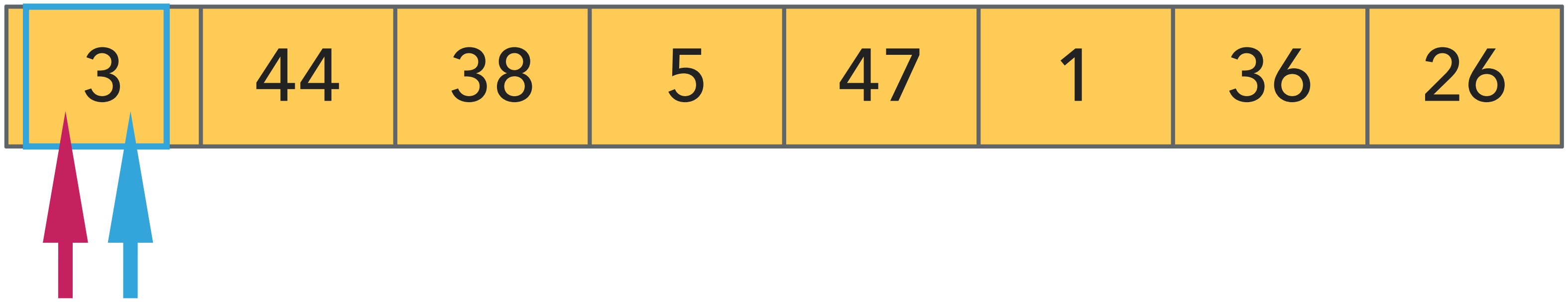
Insertion sort



3	44	38	5	47	1	36	26

- 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.

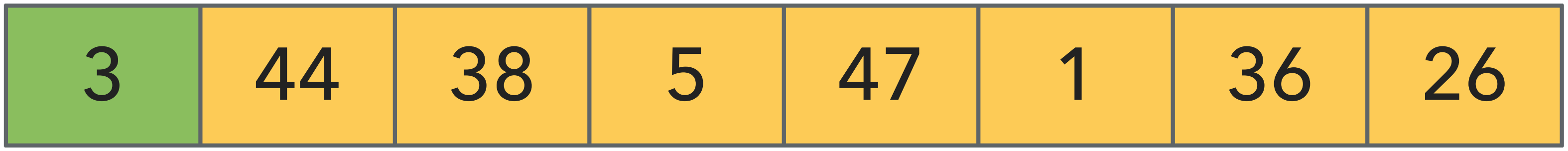
Insertion sort



3	44	38	5	47	1	36	26

- 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.

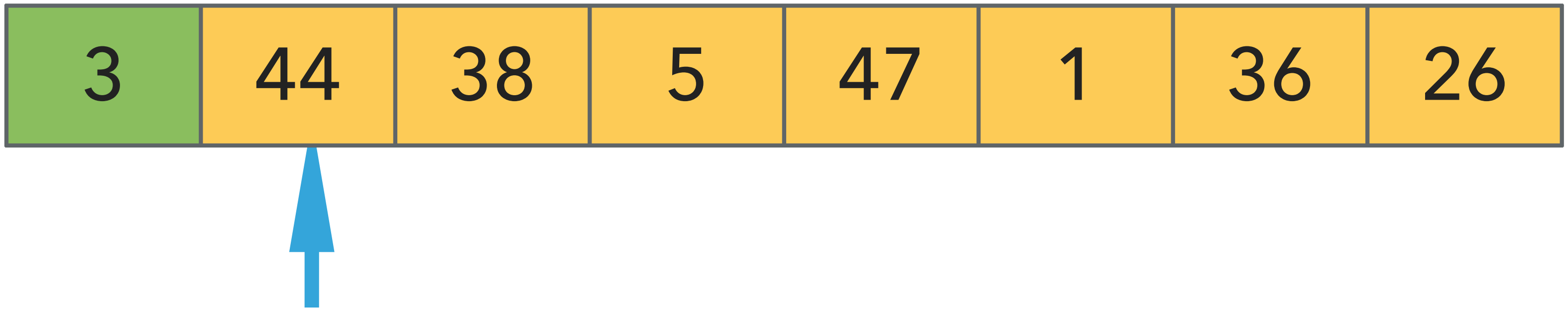
Insertion sort



3	44	38	5	47	1	36	26

- 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.

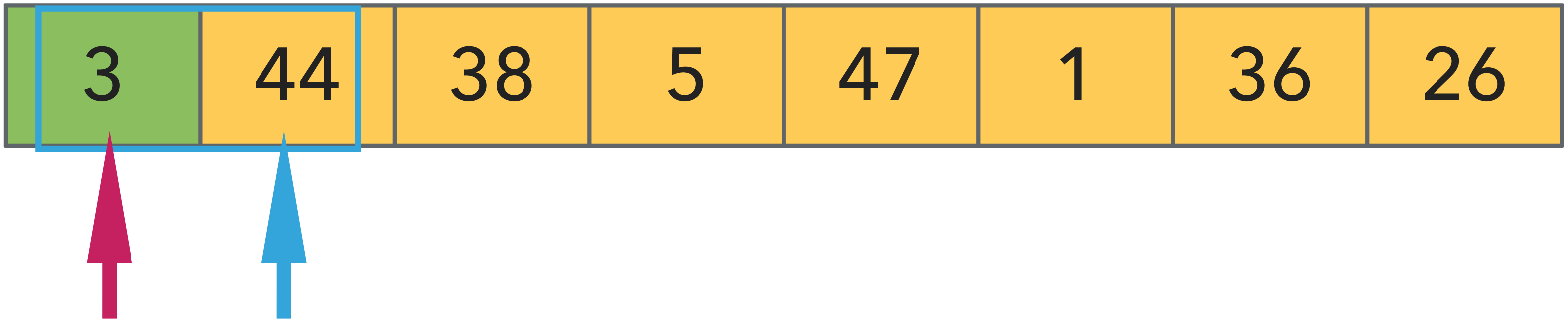
Insertion sort



3	44	38	5	47	1	36	26

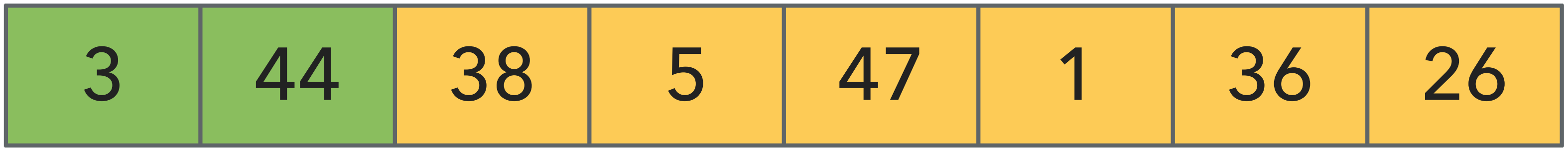
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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.

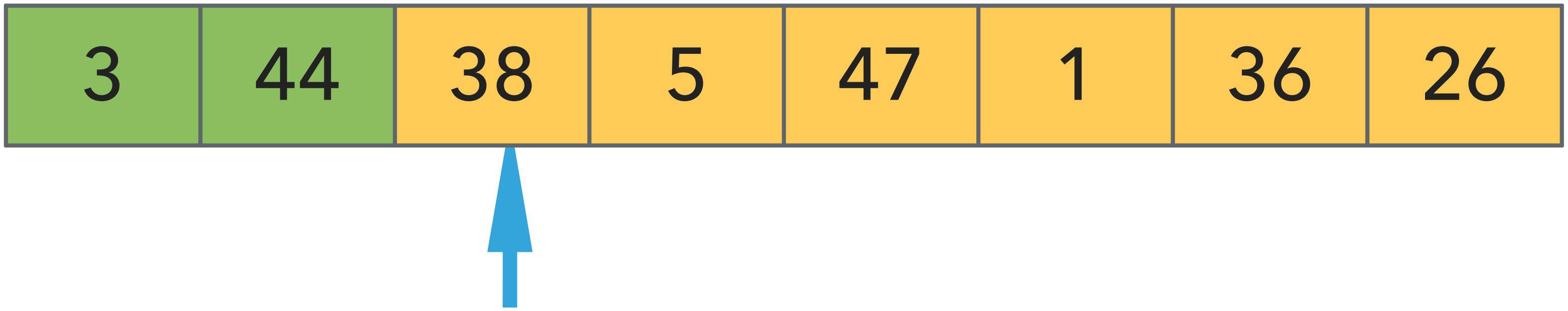
Insertion sort



3	44	38	5	47	1	36	26

- 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.

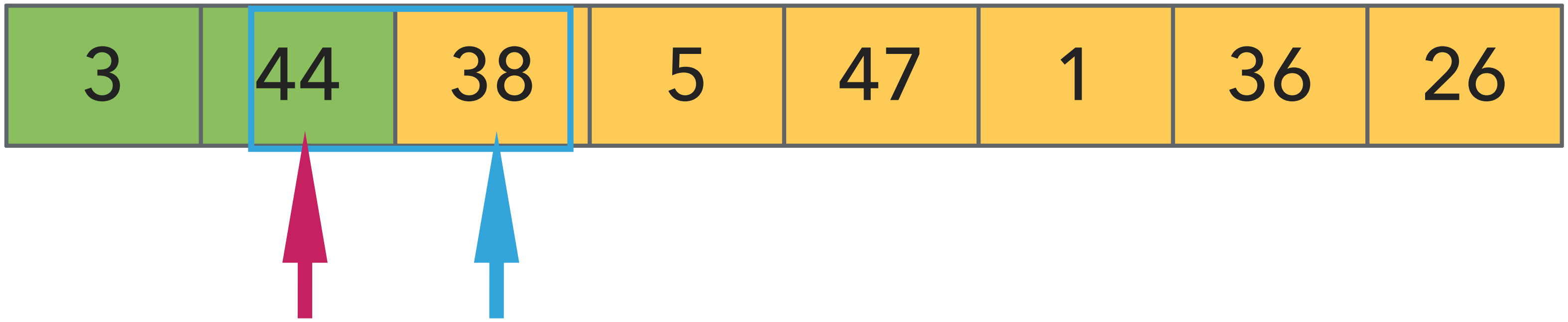
Insertion sort



3	44	38	5	47	1	36	26

- 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.

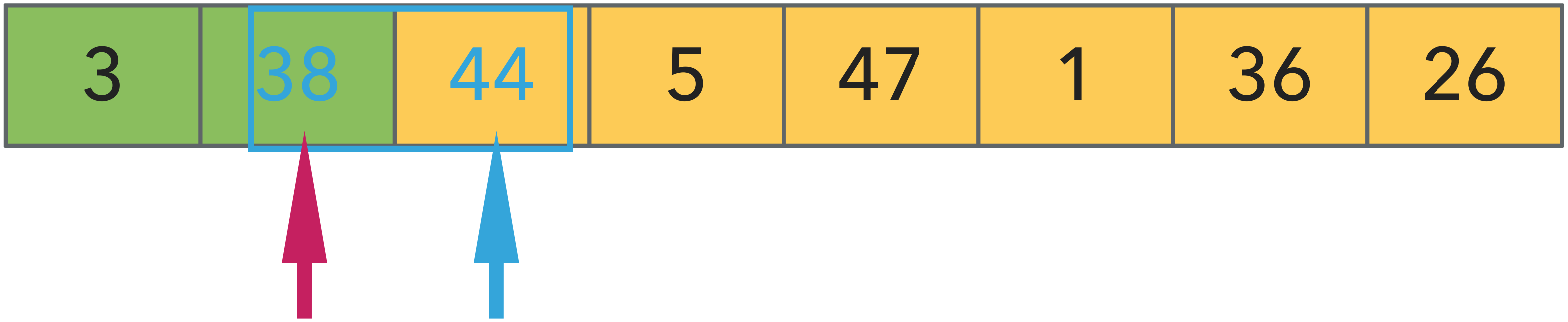
Insertion sort



3	44	38	5	47	1	36	26

- 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.

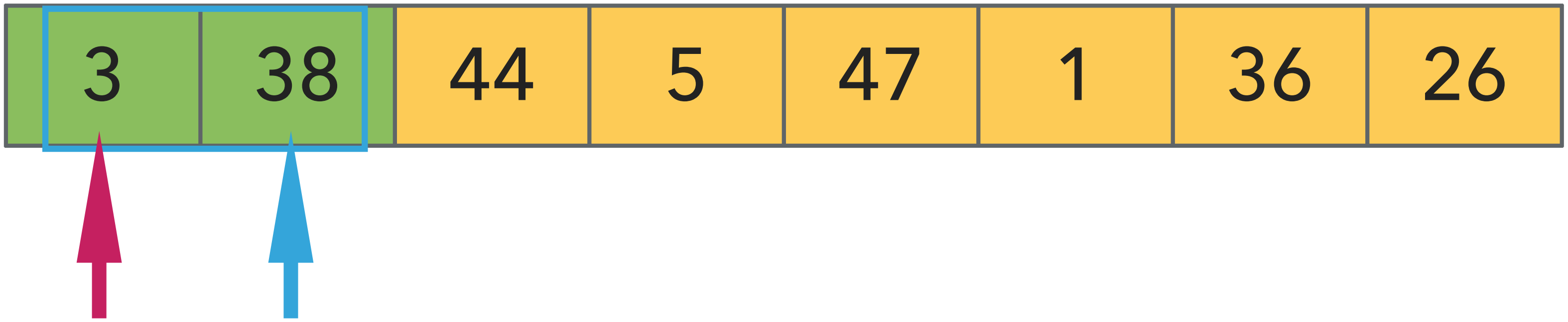
Insertion sort



3	38	44	5	47	1	36	26

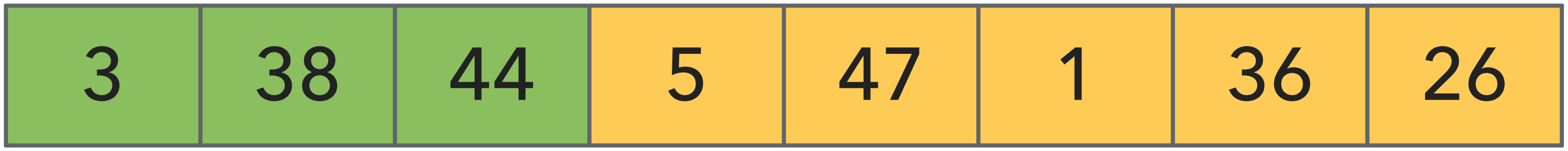
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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.

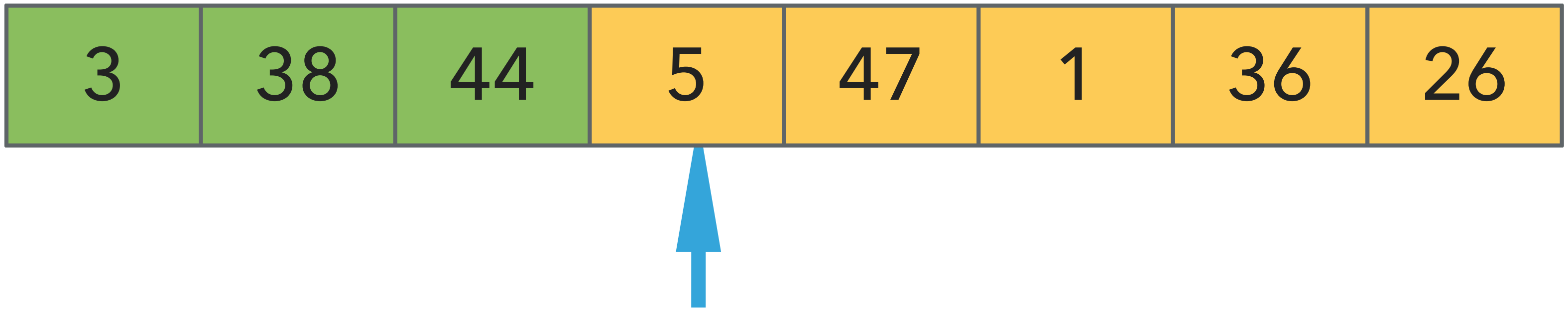
Insertion sort



3	38	44	5	47	1	36	26

- 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.

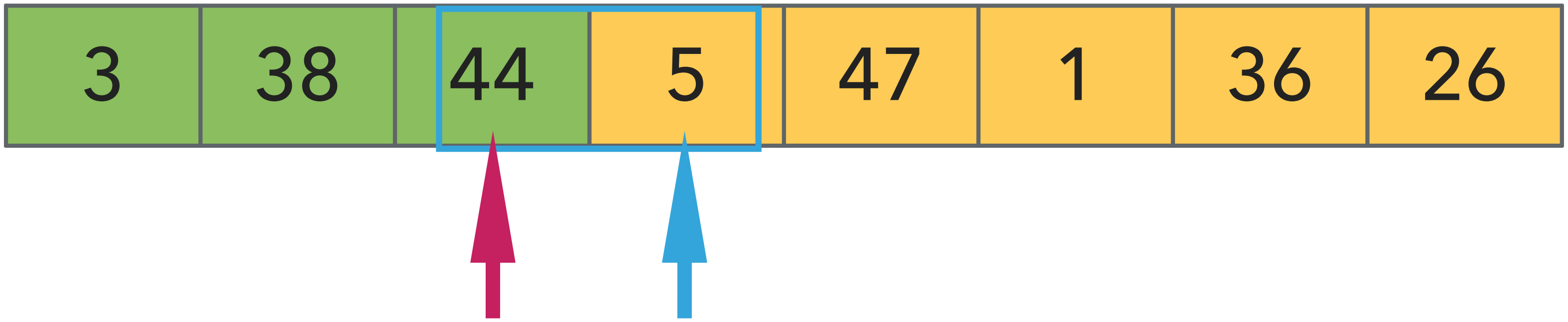
Insertion sort



3	38	44	5	47	1	36	26

- 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.

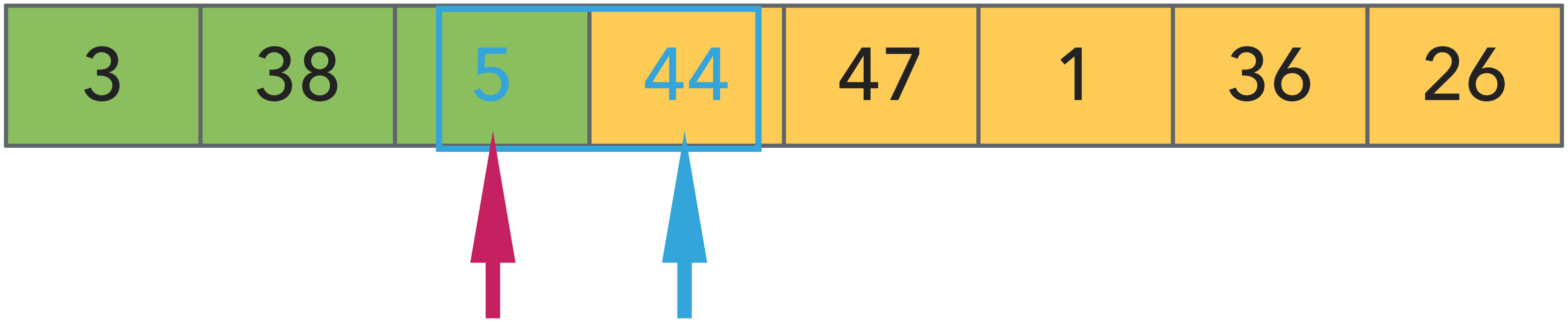
Insertion sort



3	38	44	5	47	1	36	26

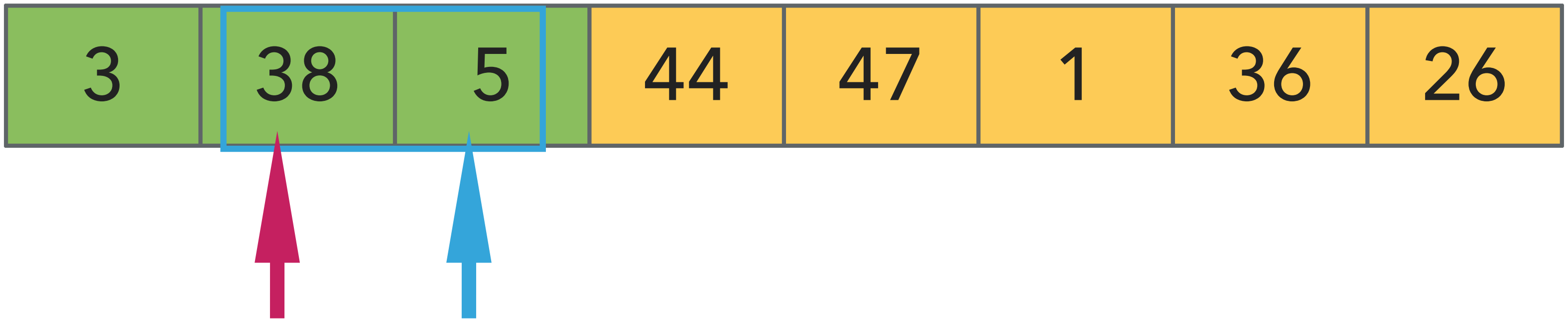
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



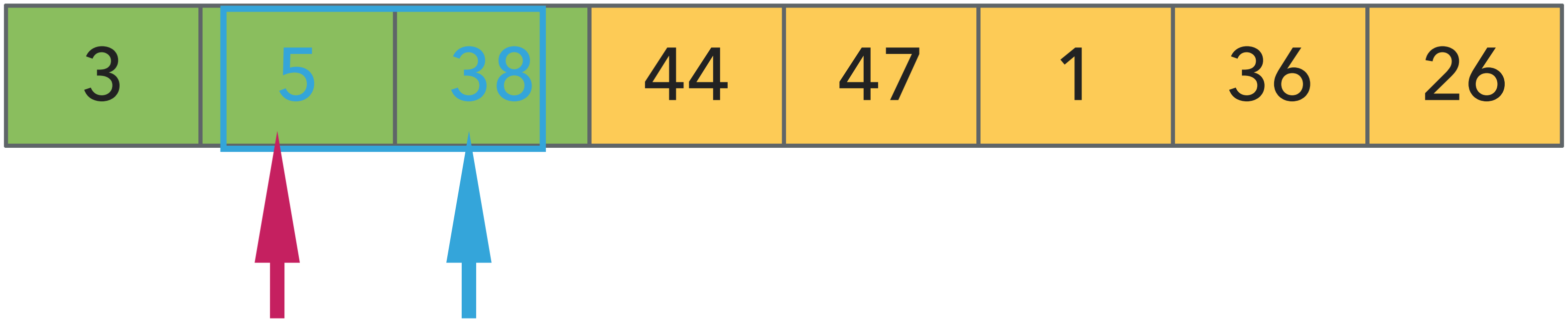
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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.

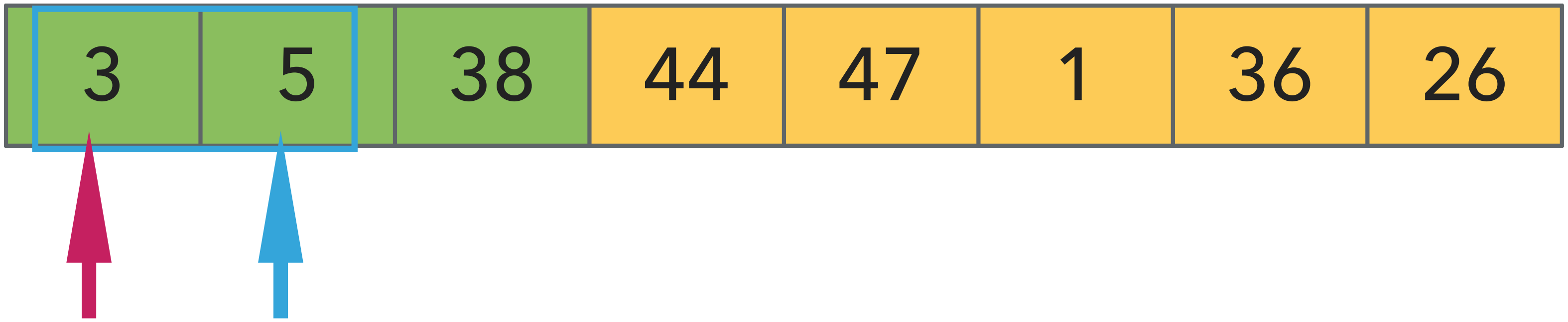
Insertion sort



3	5	38	44	47	1	36	26

- 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



3	5	38	44	47	1	36	26

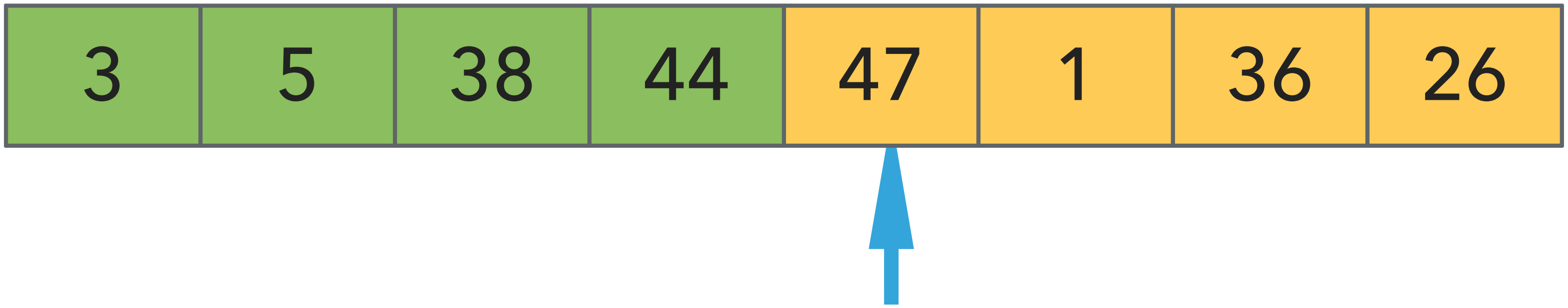
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort

3	5	38	44	47	1	36	26

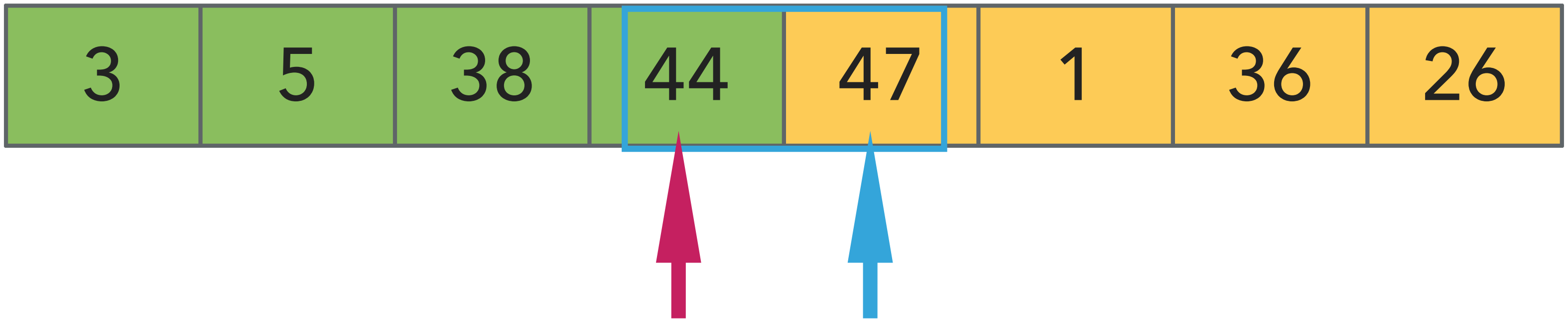
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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



3	5	38	44	47	1	36	26

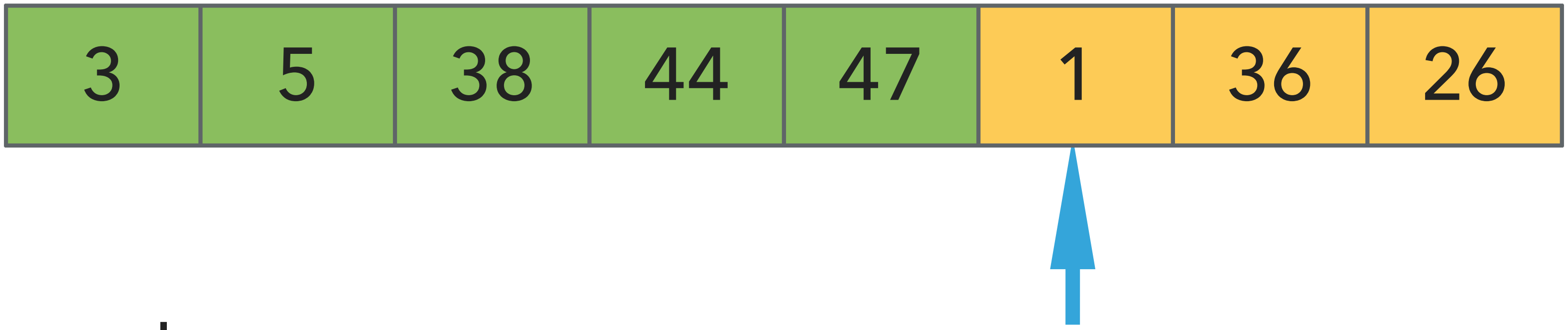
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort

3	5	38	44	47	1	36	26

- 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.

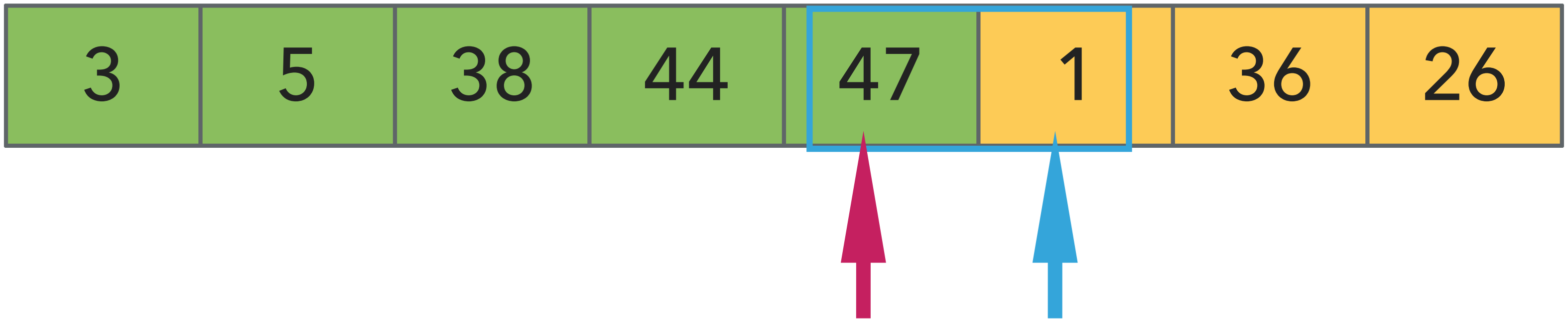
Insertion sort



3	5	38	44	47	1	36	26

- 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.

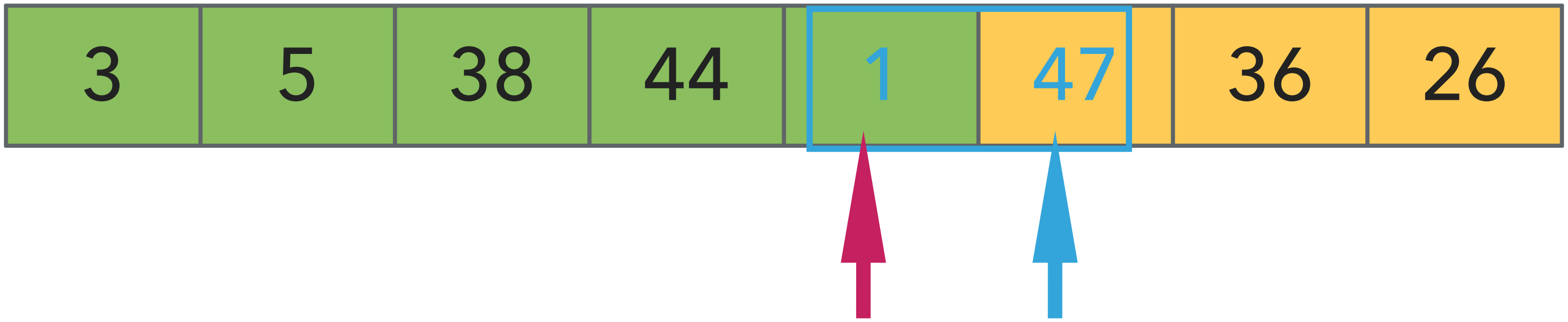
Insertion sort



3	5	38	44	47	1	36	26

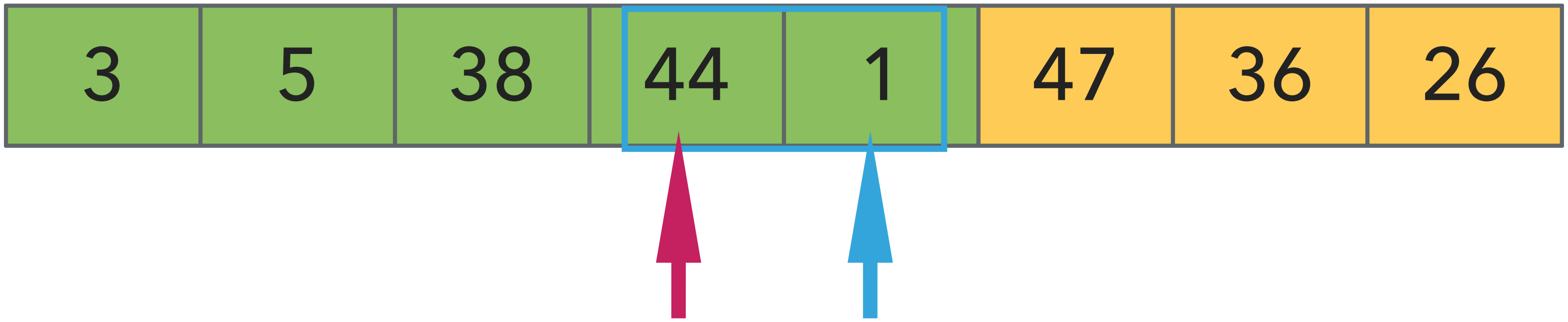
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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.

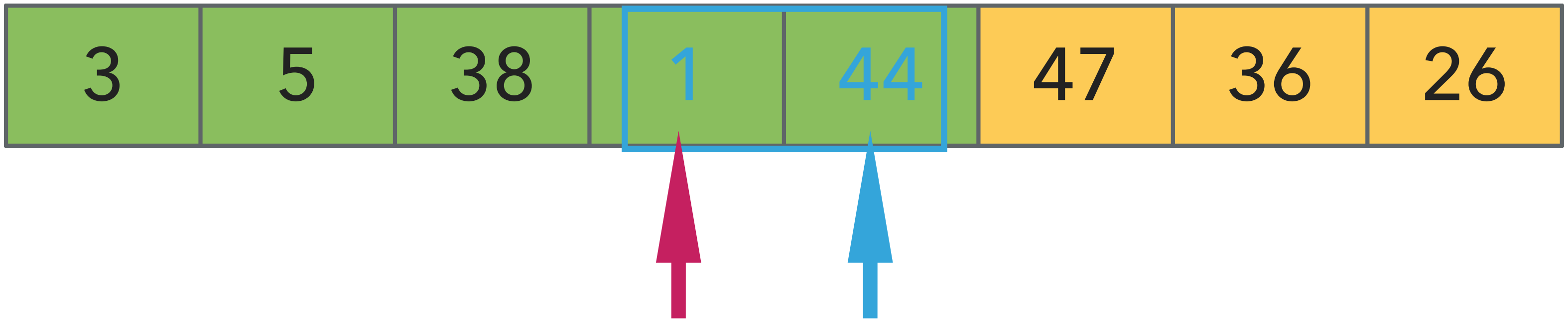
Insertion sort



3	5	38	44	1	47	36	26

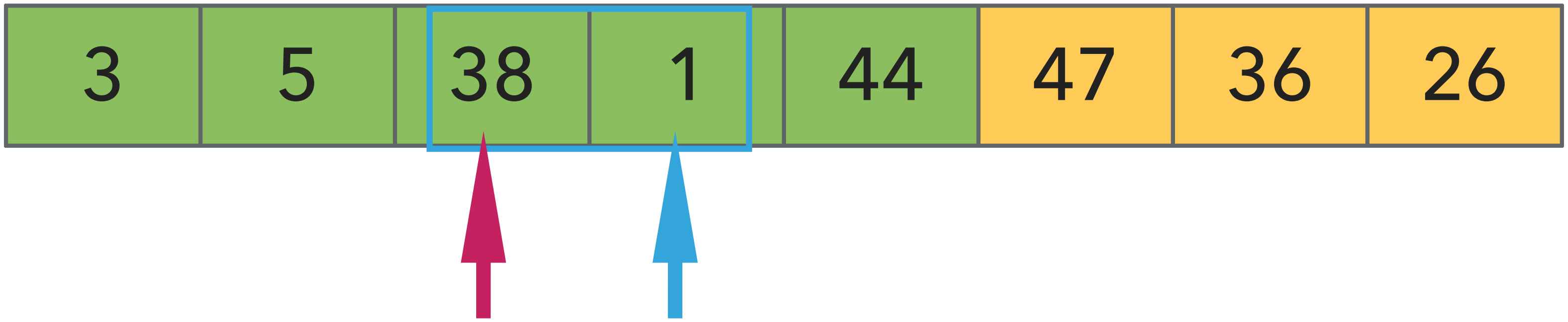
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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.

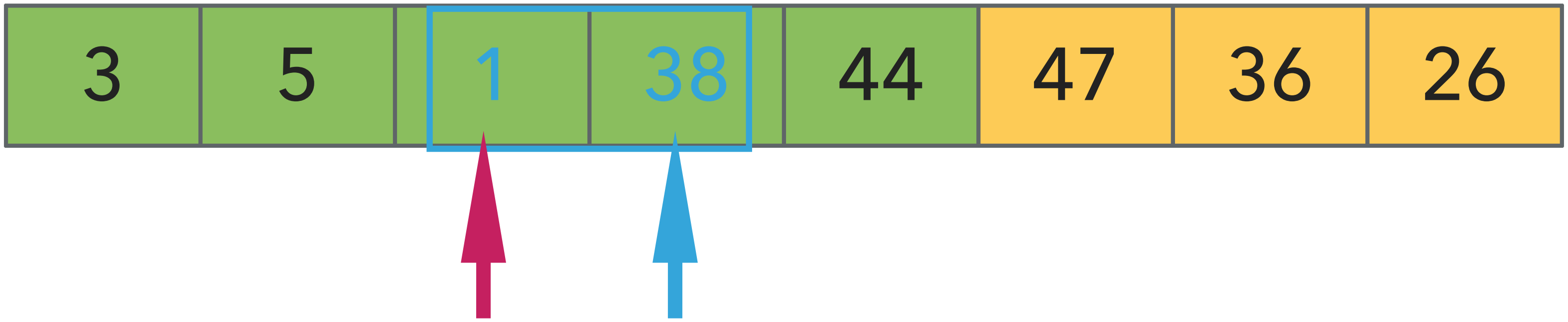
Insertion sort



3	5	38	1	44	47	36	26

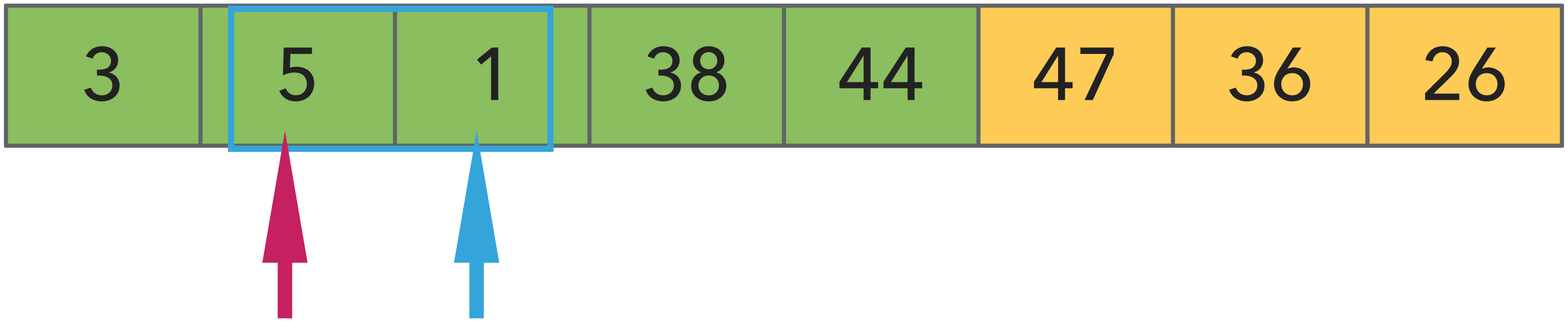
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



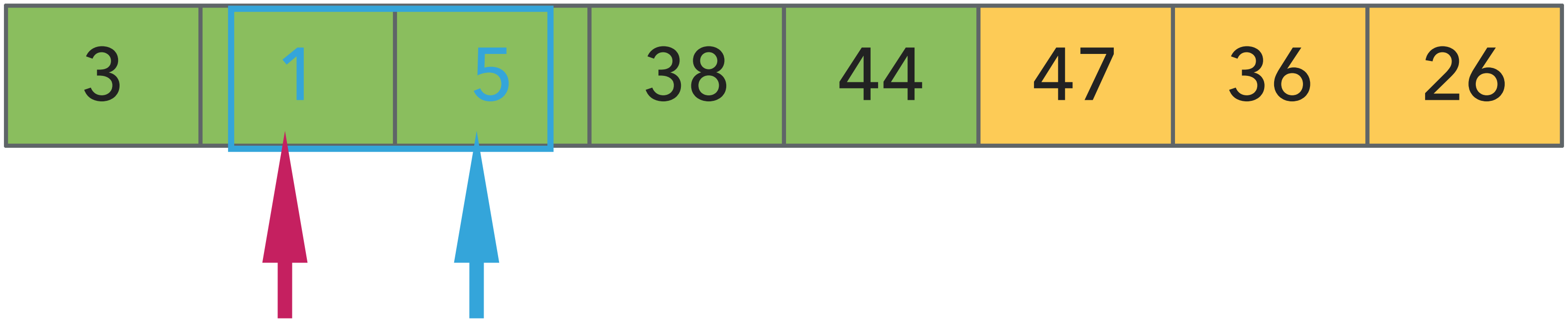
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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.

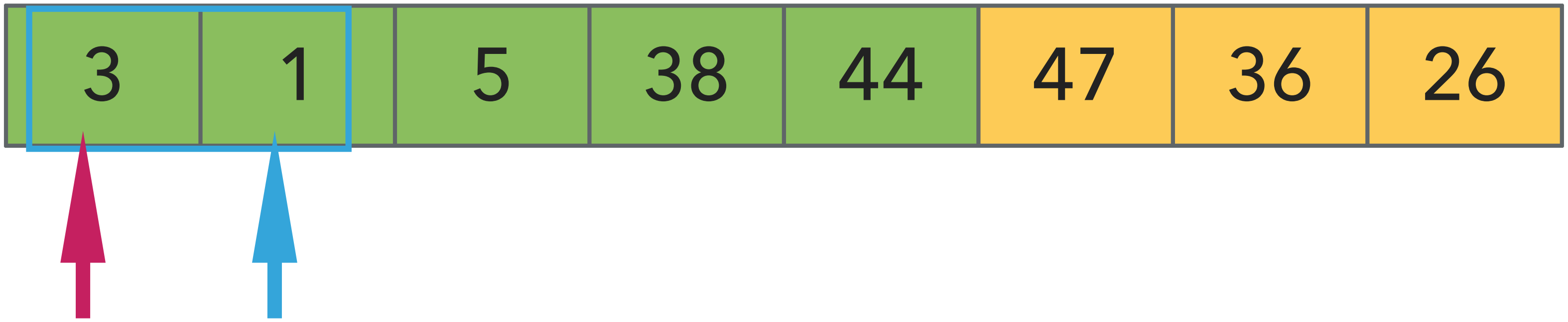
Insertion sort



3	1	5	38	44	47	36	26

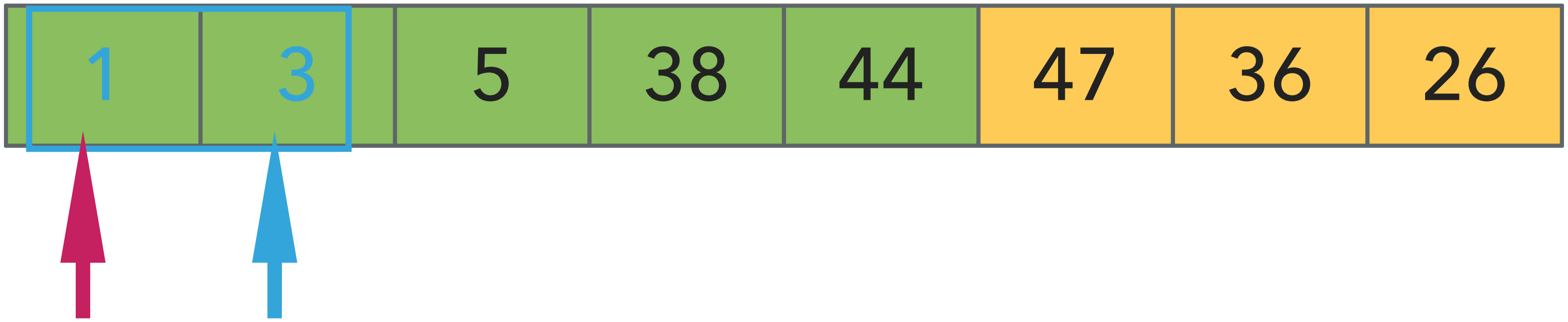
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



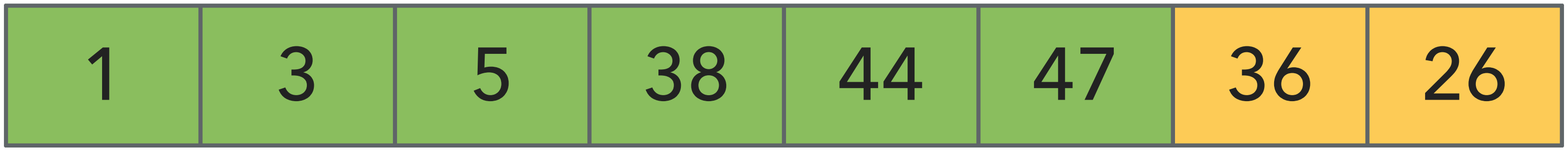
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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.

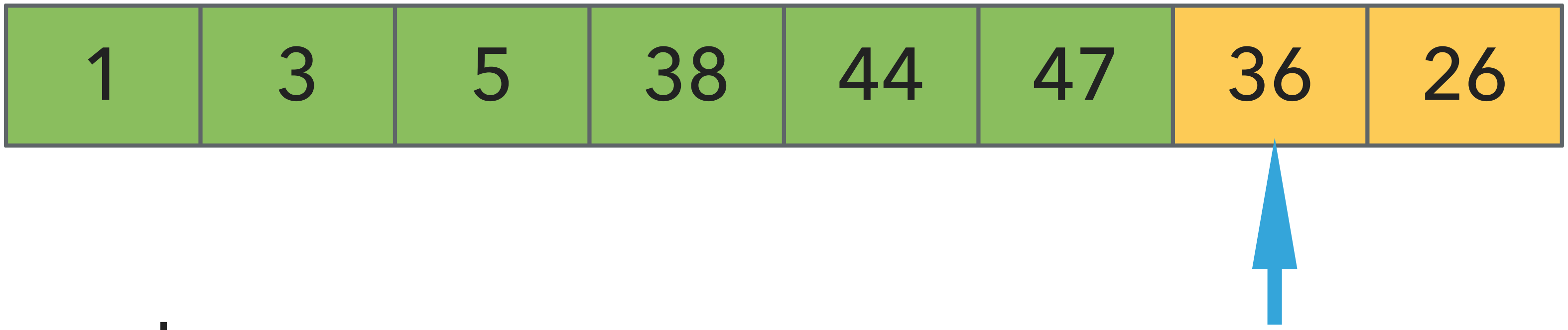
Insertion sort



1	3	5	38	44	47	36	26

- 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.

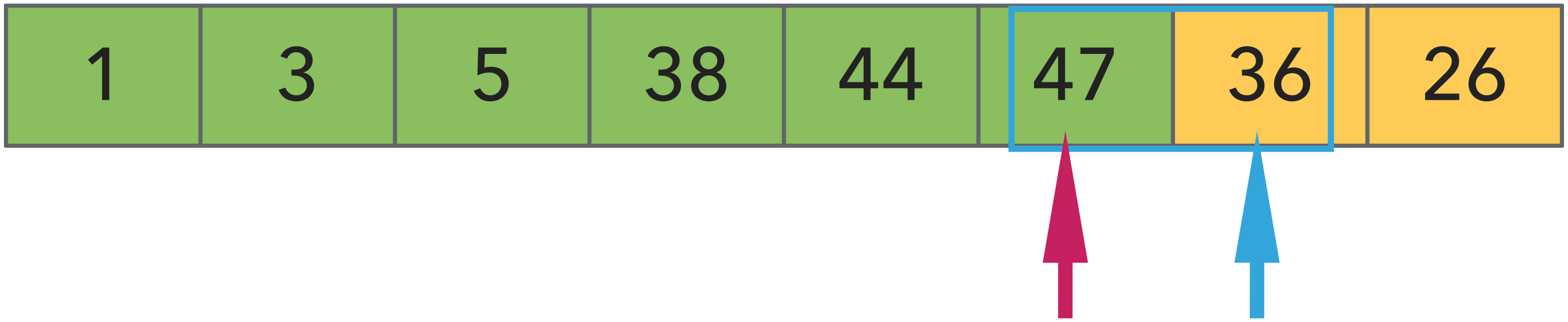
Insertion sort



1	3	5	38	44	47	36	26

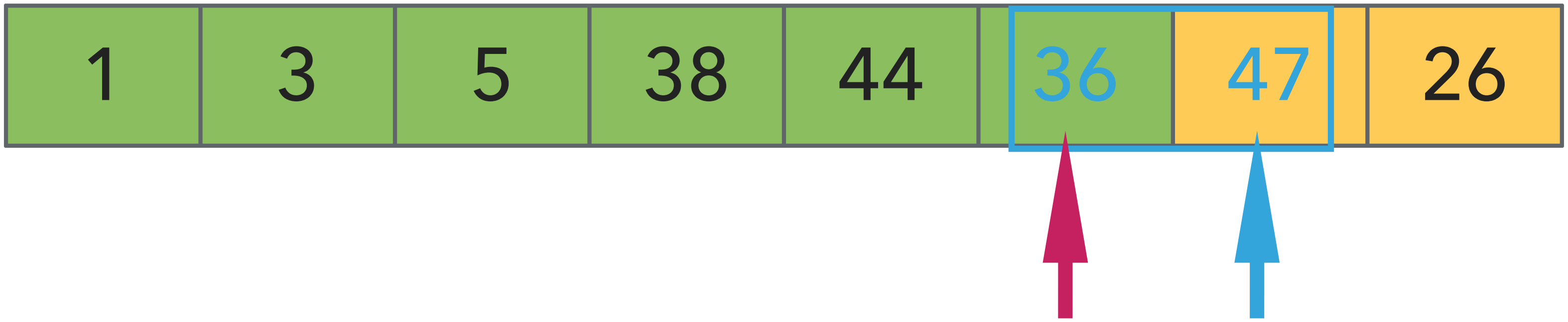
- Repeat:
  - **Examine the next element in the unsorted subarray.**
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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.

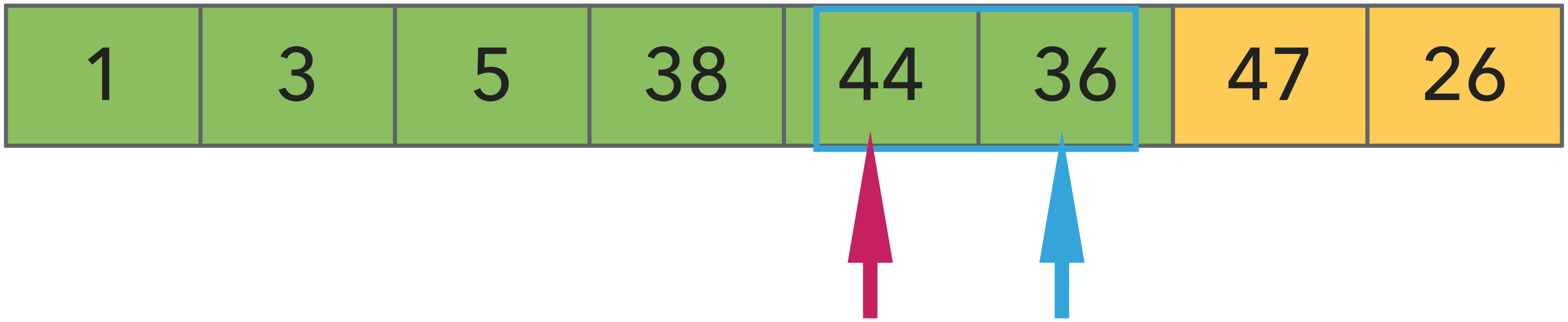
Insertion sort



1	3	5	38	44	36	47	26

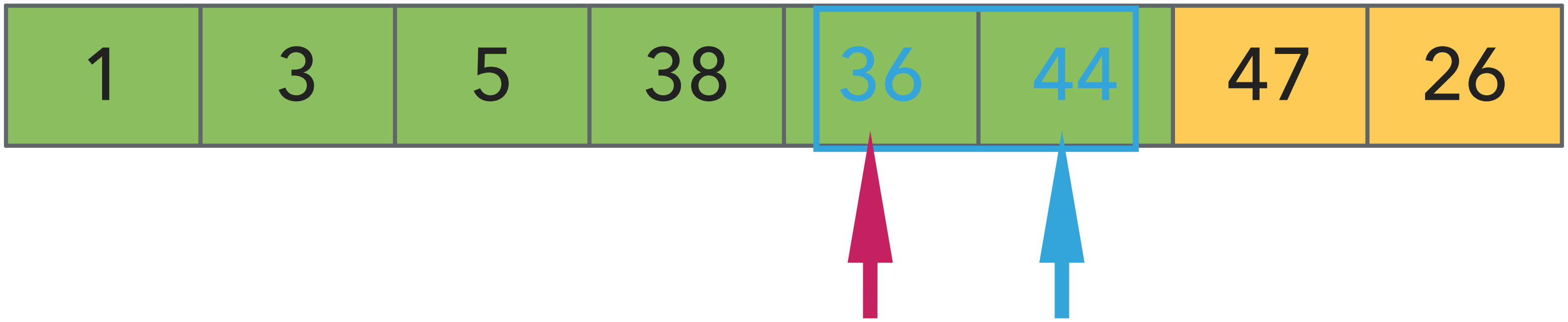
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



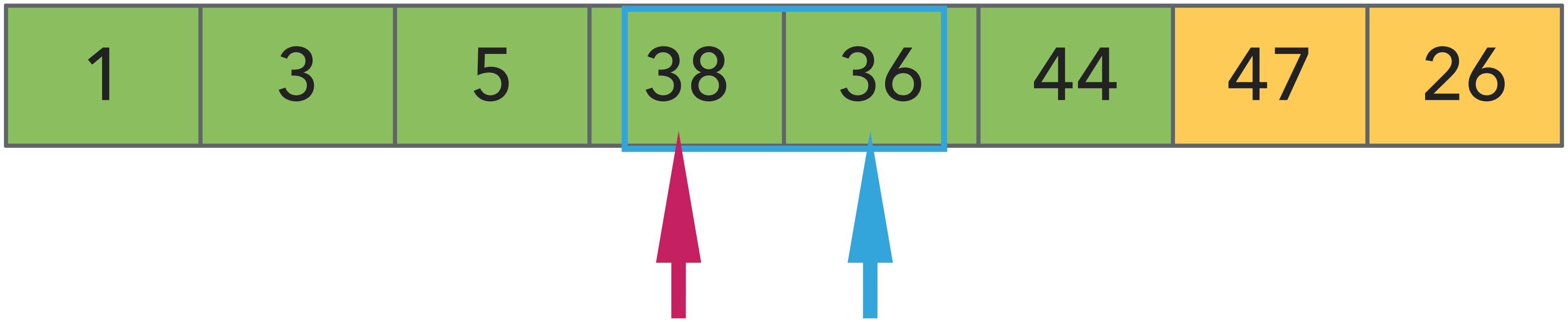
- Repeat:
    - Examine the next element in the unsorted subarray.
    - Find the location it belongs within the sorted subarray and insert it there.
    - Move subarray boundaries one element to the right.

Insertion sort



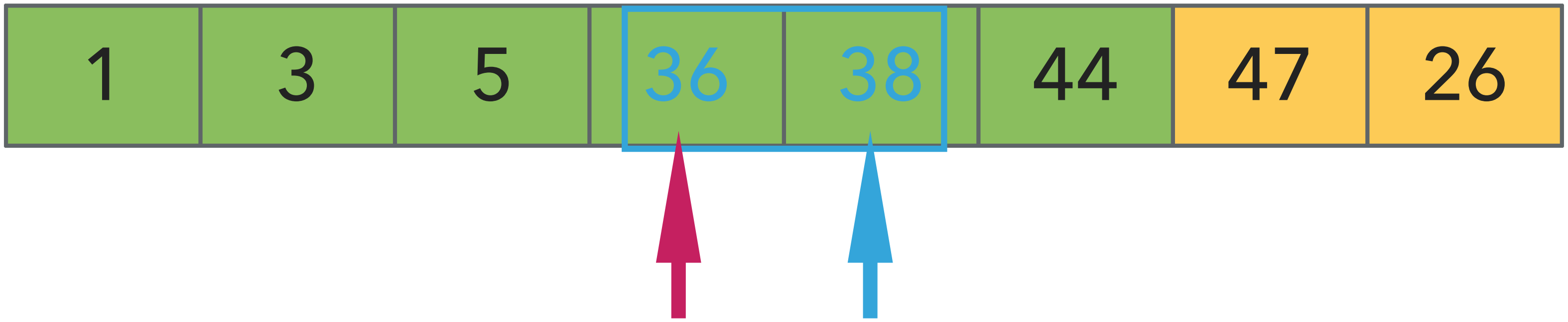
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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.

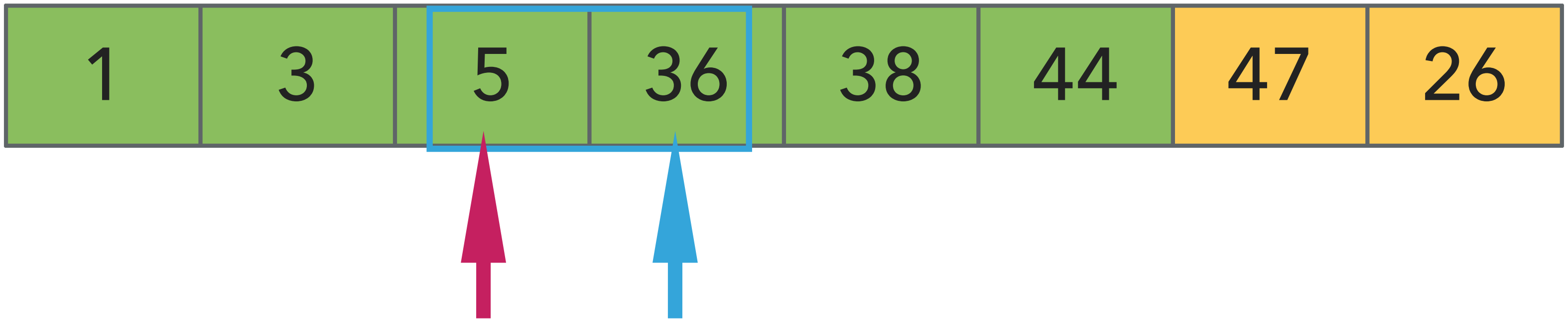
Insertion sort



1	3	5	36	38	44	47	26

- 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.

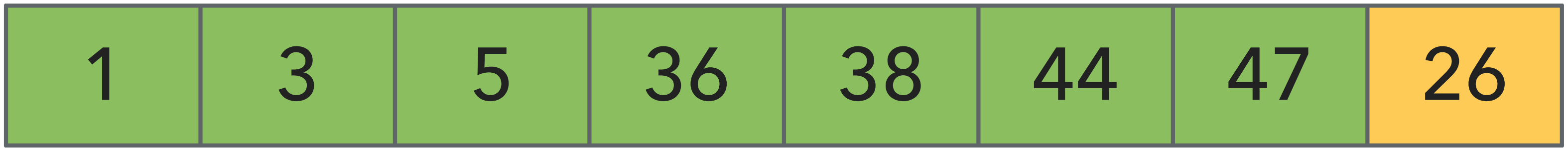
Insertion sort



1	3	5	36	38	44	47	26

- 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.

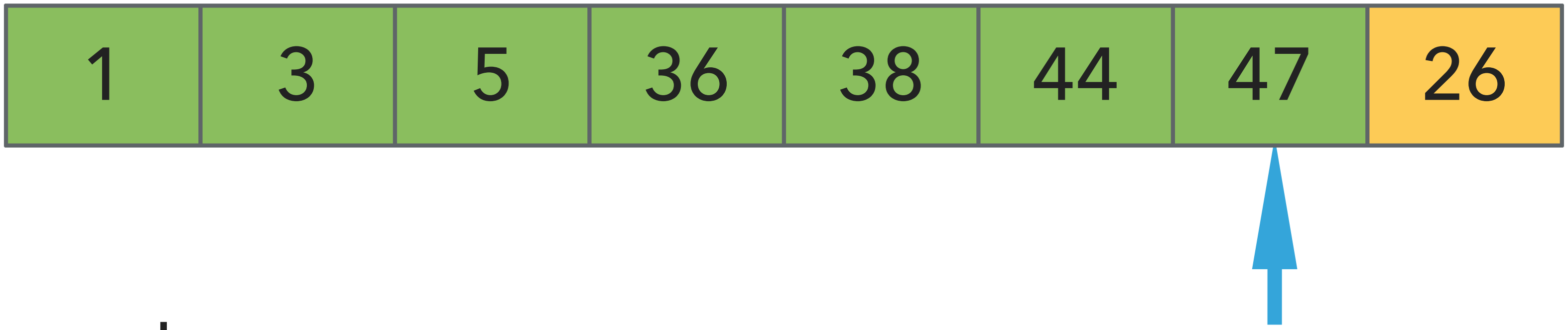
Insertion sort



1	3	5	36	38	44	47	26

- 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.

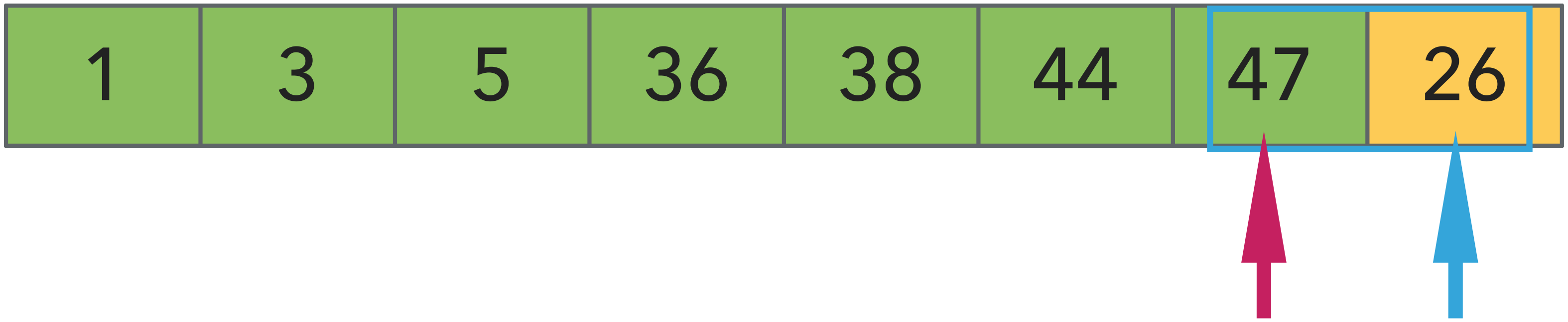
Insertion sort



1	3	5	36	38	44	47	26

- 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.

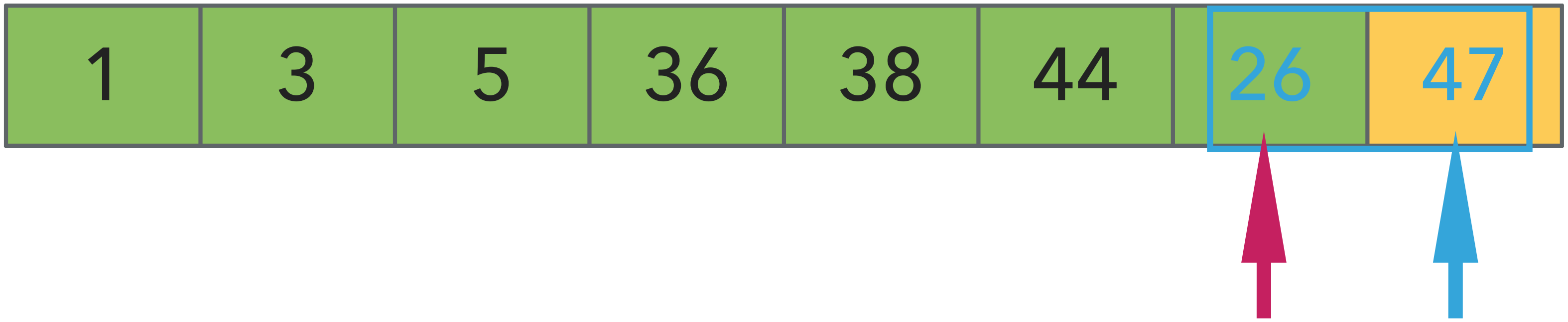
Insertion sort



1	3	5	36	38	44	47	26

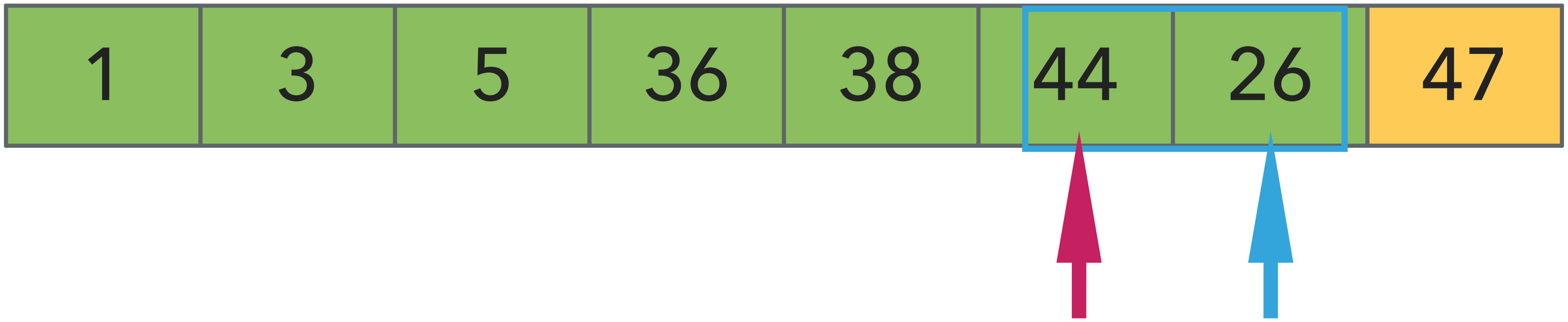
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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.

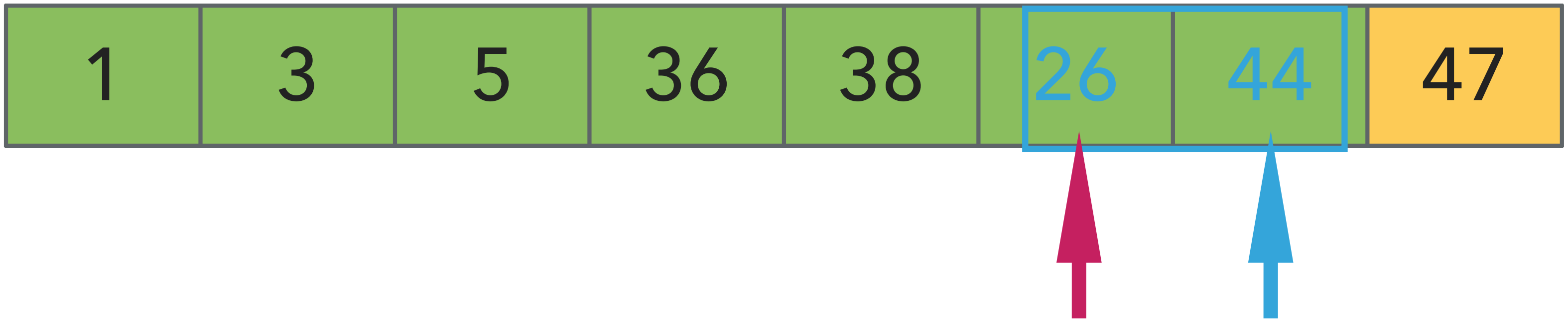
Insertion sort



1	3	5	36	38	44	26	47

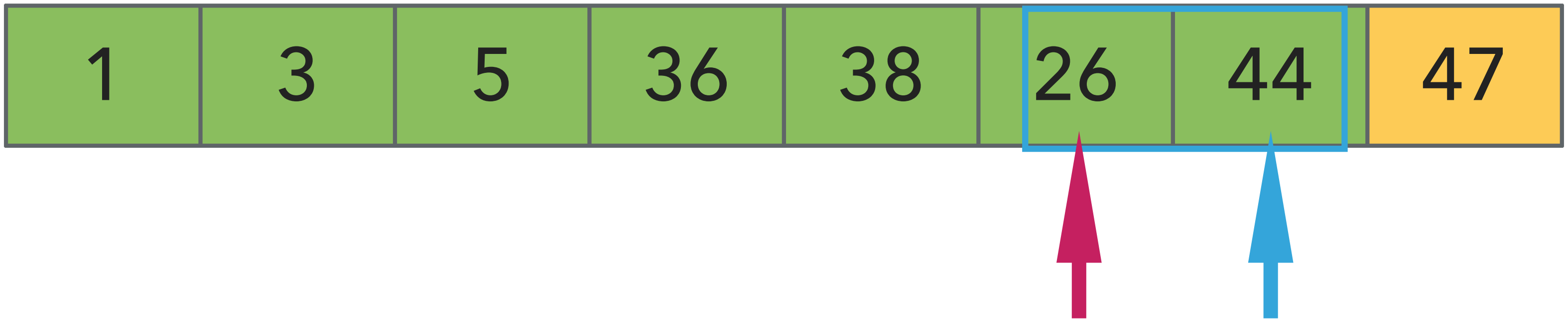
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



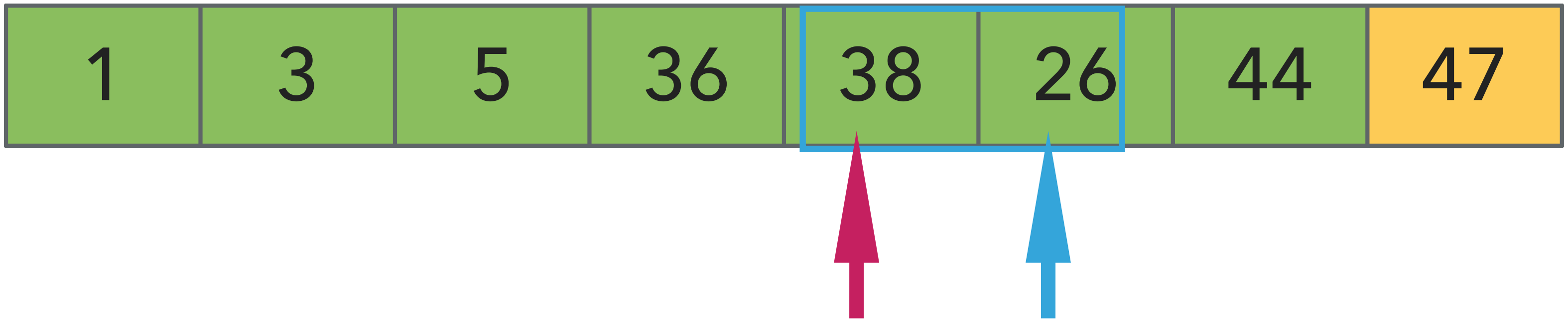
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



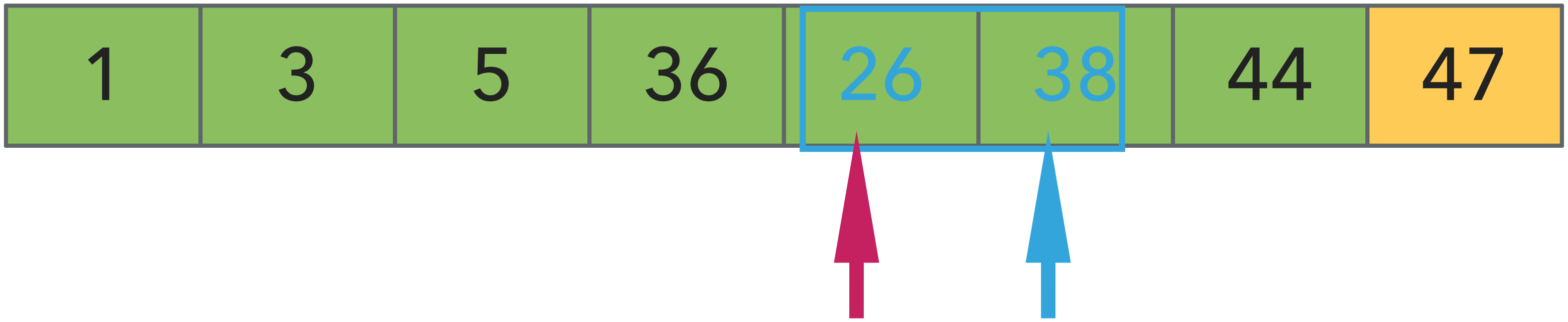
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



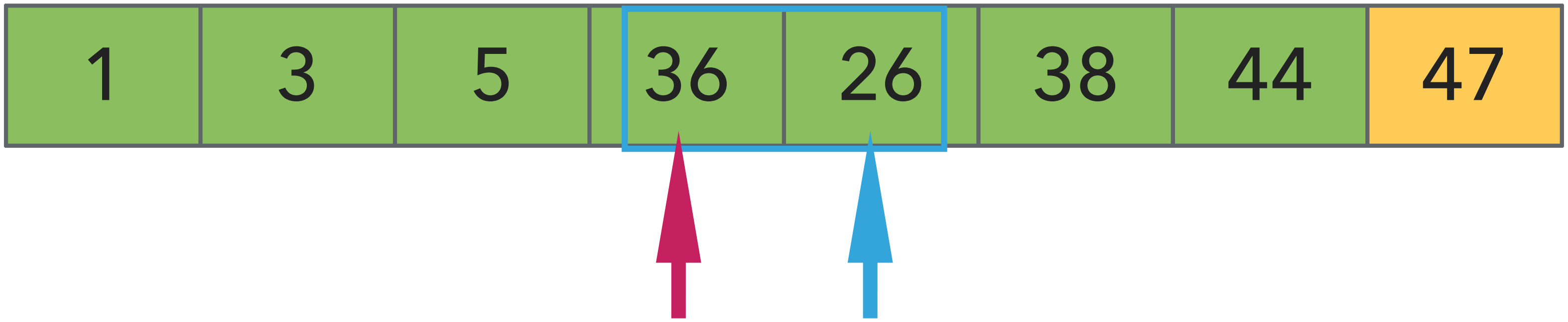
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



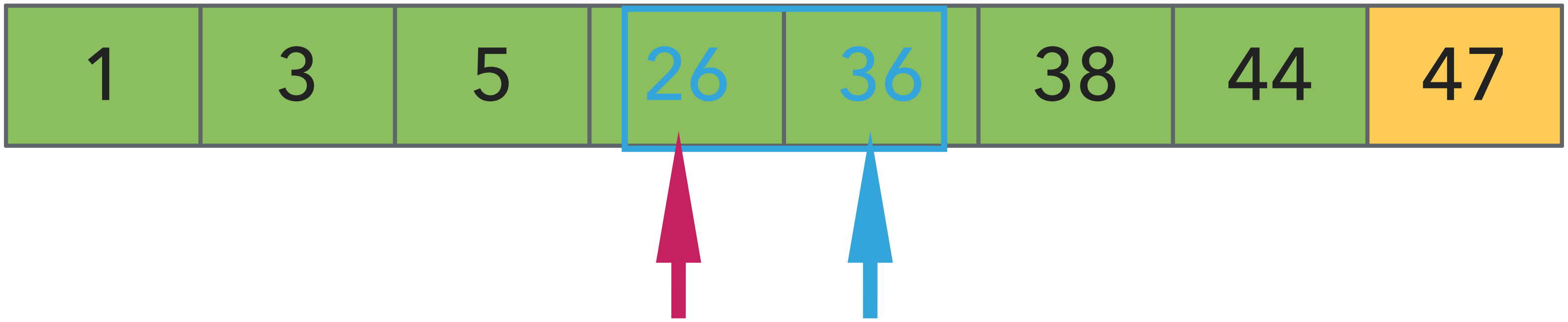
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

Insertion sort



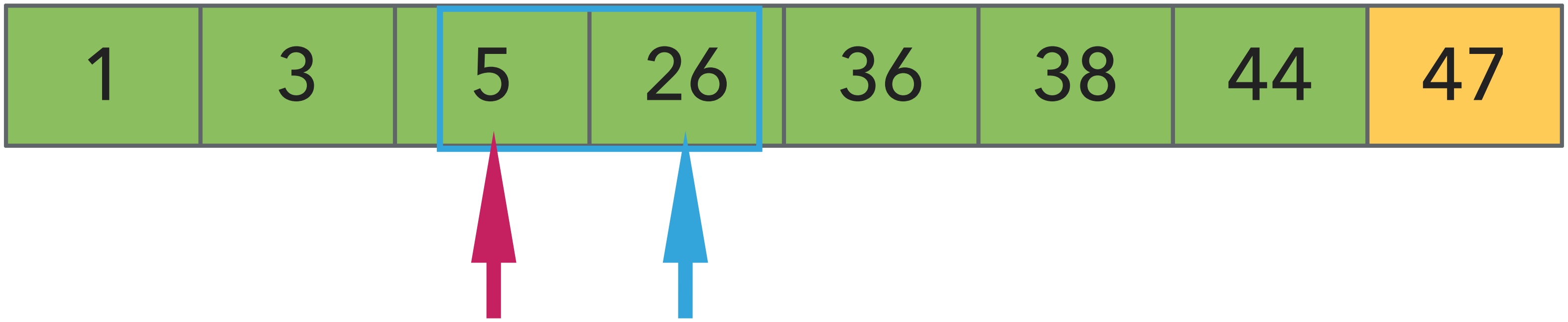
- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.

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



1	3	5	26	36	38	44	47

- 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

1	3	5	26	36	38	44	47

- 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.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

2.1 INSERTION SORT DEMO

INSERTION SORT

In case you didn’t get this…

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

Insertion sort

```
public static void sort(Comparable[] a) {



   }
```

Insertion sort

```
public static void sort(Comparable[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
           for (int j = i; j > 0; j--) {
               if(less(a[j], a[j-1]))
                  exch(a, j, j-1);
               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 void sort(Comparable[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
           for (int j = i; j > 0; j--) {
               if(less(a[j], a[j-1]))
                  exch(a, j, j-1);
               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 void sort(Comparable[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
           for (int j = i; j > 0; j--) {
               if(less(a[j], a[j-1]))
                  exch(a, j, j-1);
               else
                  break;
           }
       }
   }
```

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

- Best case: $n - 1$ comparisons and 0 exchanges for an already sorted array.

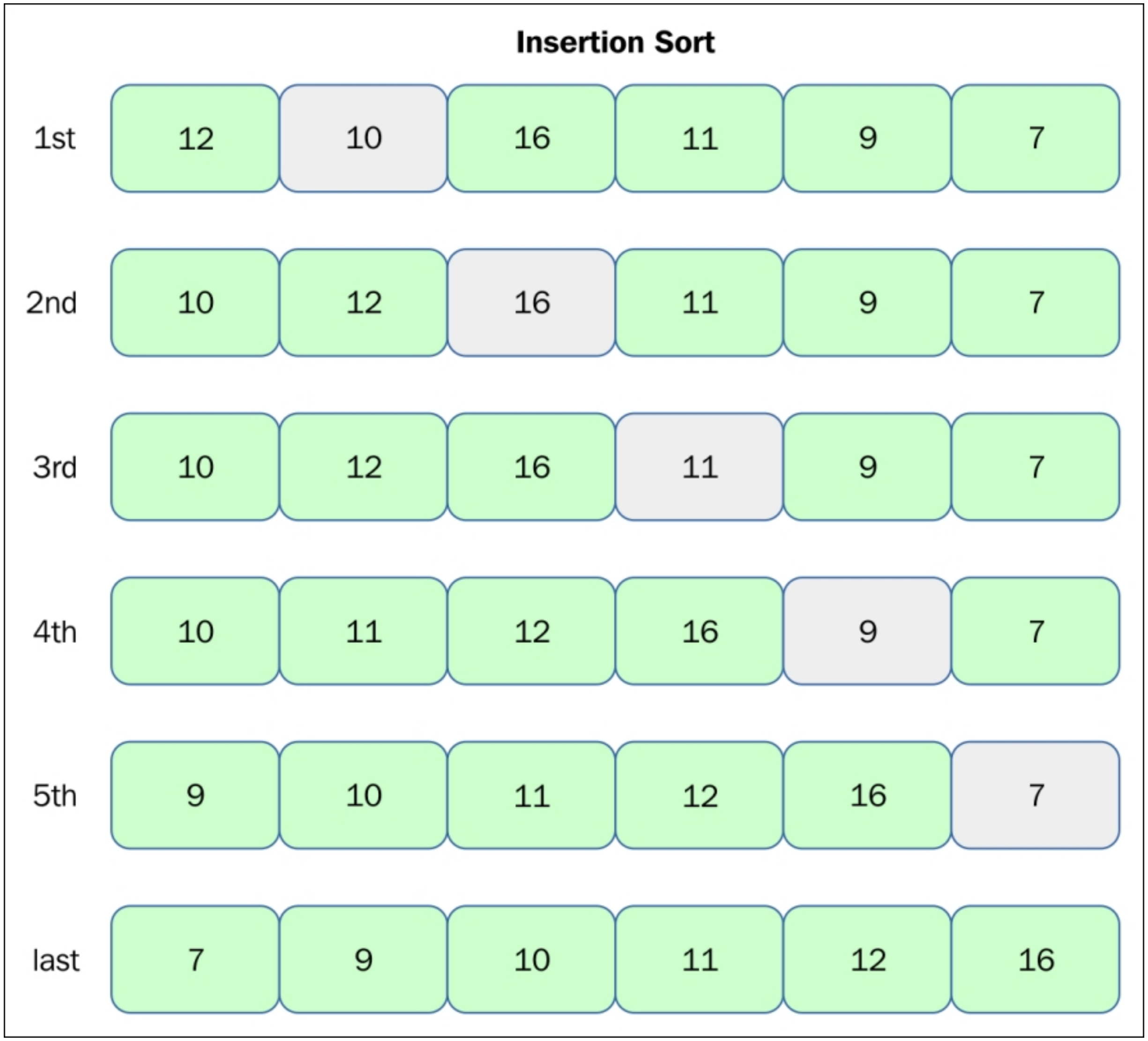
https://www.toptal.com/developers/sorting-algorithms/insertion-sort

Practice Time

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

INSERTION SORT

Answer



Insertion Sort

1st	12	10	16	11	9	7
2nd	10	12	16	11	9	7
3rd	10	12	16	11	9	7
4th	10	11	12	16	9	7
5th	9	10	11	12	16	7
last	7	9	10	11	12	16

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

Lecture 12: Sorting Fundamentals

- Introduction
- Selection sort
- Insertion sort

Readings:

- Textbook:
  - Chapter 2.1 (pages 244–262)
- Website:
  - Elementary sorts: https://algs4.cs.princeton.edu/21elementary/
  - Code: https://algs4.cs.princeton.edu/21elementary/Selection.java.html and https://algs4.cs.princeton.edu/21elementary/Insertion.java.html

Practice Problems:

- 2.1.1-2.1.8