Introduction to Algorithms

Email: grad.saeed@gmail.com

Github: github.com/saeedgrad

The problem of sorting

Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Example:

Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

Keys, satellite data, record

- The numbers to be sorted are also known as the keys. Although the problem is conceptually about sorting a sequence, the input comes in the form of an array with n elements.
- When we want to sort numbers, it's often because they are the keys associated with other data, which we call satellite data.
- Together, a key and satellite data form a record.

Example

Initial Array Representation

```
+-----+-------+-----------+
| ID  | Name  | Price     | Description
+-----+-------+-----------+
| 1   | Prod1 | P1 (500)  | Desc1
| 2   | Prod2 | P2 (300)  | Desc2
| 3   | Prod3 | P3 (450)  | Desc3
| 4   | Prod4 | P4 (150)  | Desc4
```

Example

cont'd

After Full Sort

```
+-----+-------+-----------+
| ID  | Name  | Price     | Description
+-----+-------+-----------+
| 4   | Prod4 | P4 (150)  | Desc4
| 2   | Prod2 | P2 (300)  | Desc2
| 3   | Prod3 | P3 (450)  | Desc3
| 1   | Prod1 | P1 (500)  | Desc1
```

Pseudocode

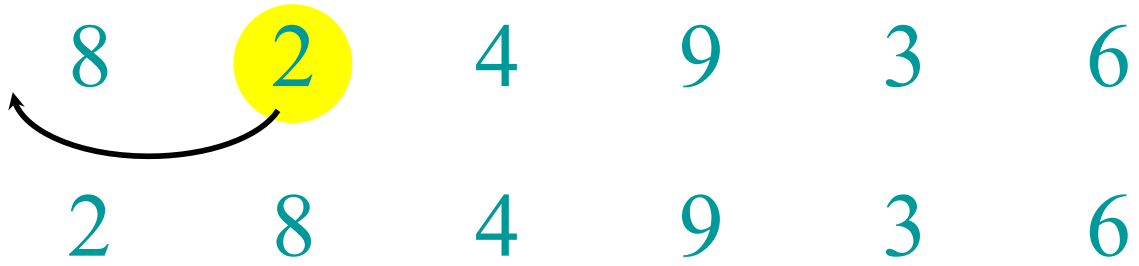
- we'll typically describe algorithms as procedures written in a pseudocode.
- In pseudocode, we employ whatever expressive method is most clear and concise to specify a given algorithm.
- Another difference between pseudocode and real code is that pseudocode often ignores aspects of software engineering.

Example of insertion sort

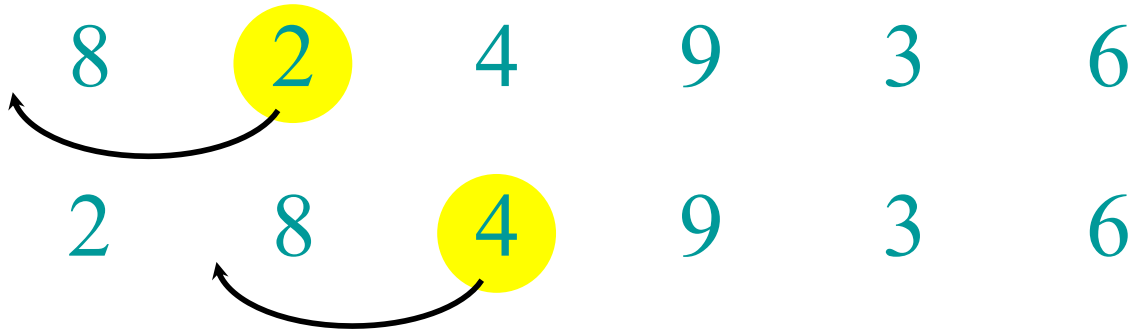
8 2 4 9 3 6

Example of insertion sort

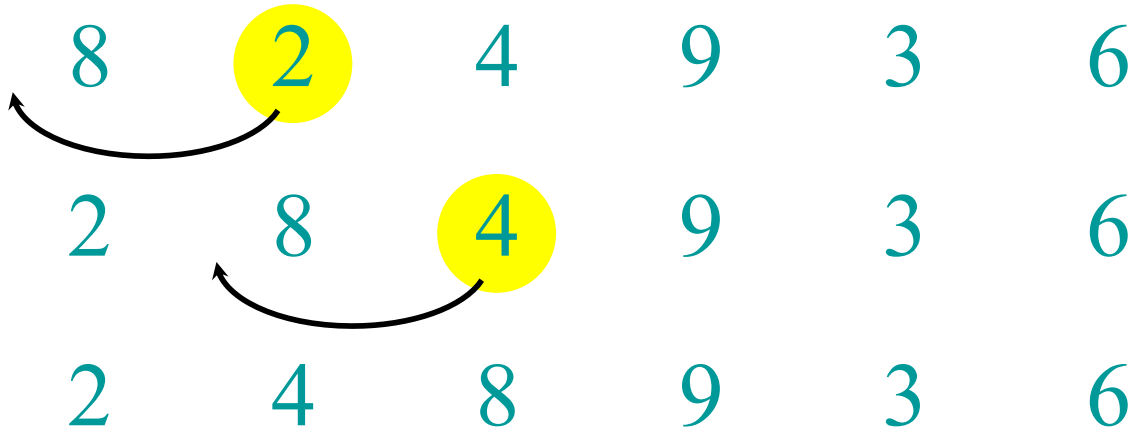
Example of insertion sort



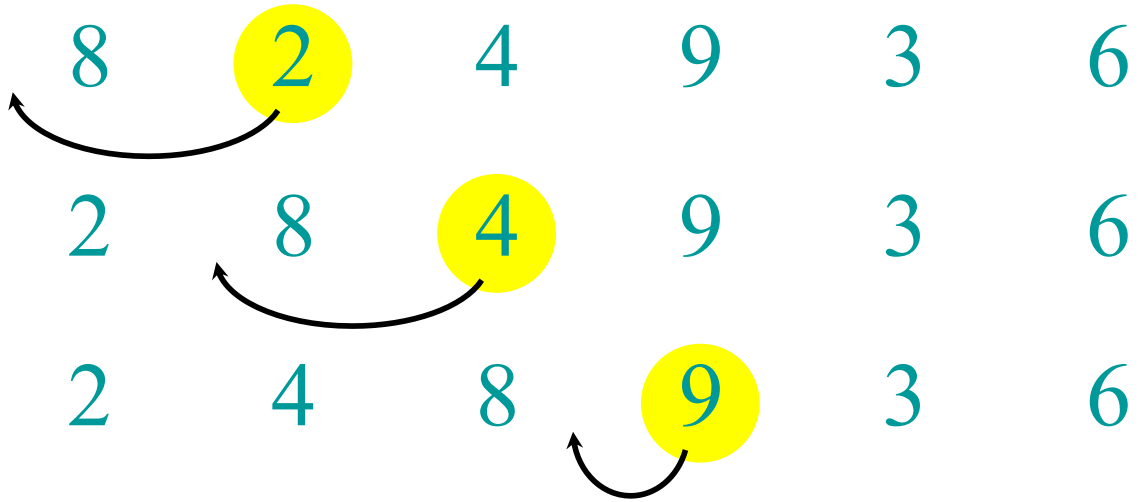
Example of insertion sort



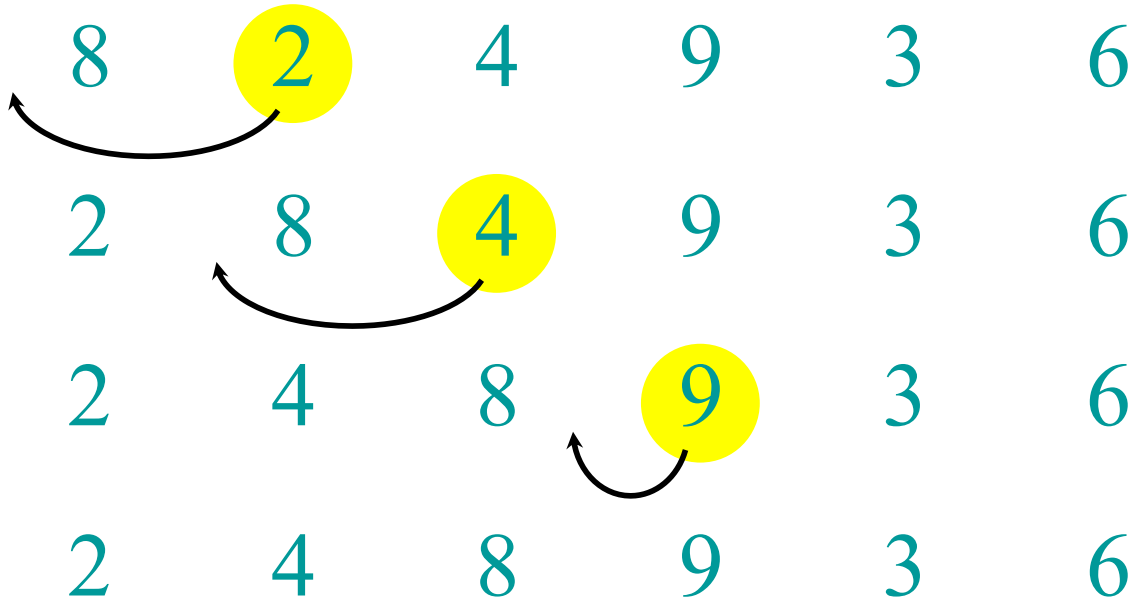
Example of insertion sort



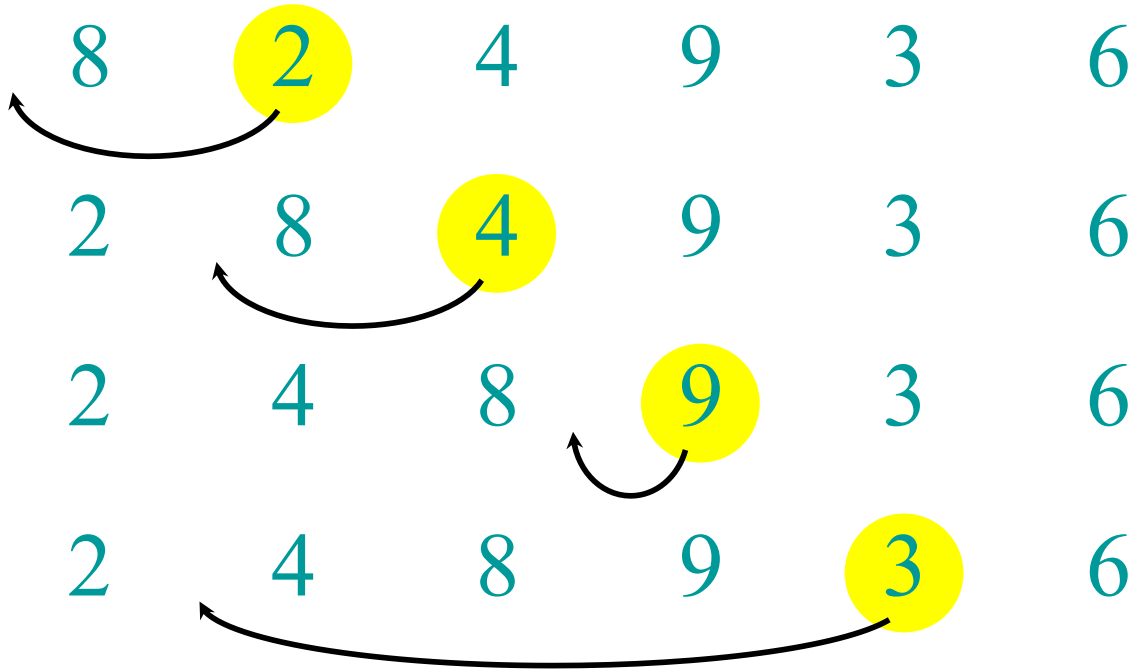
Example of insertion sort



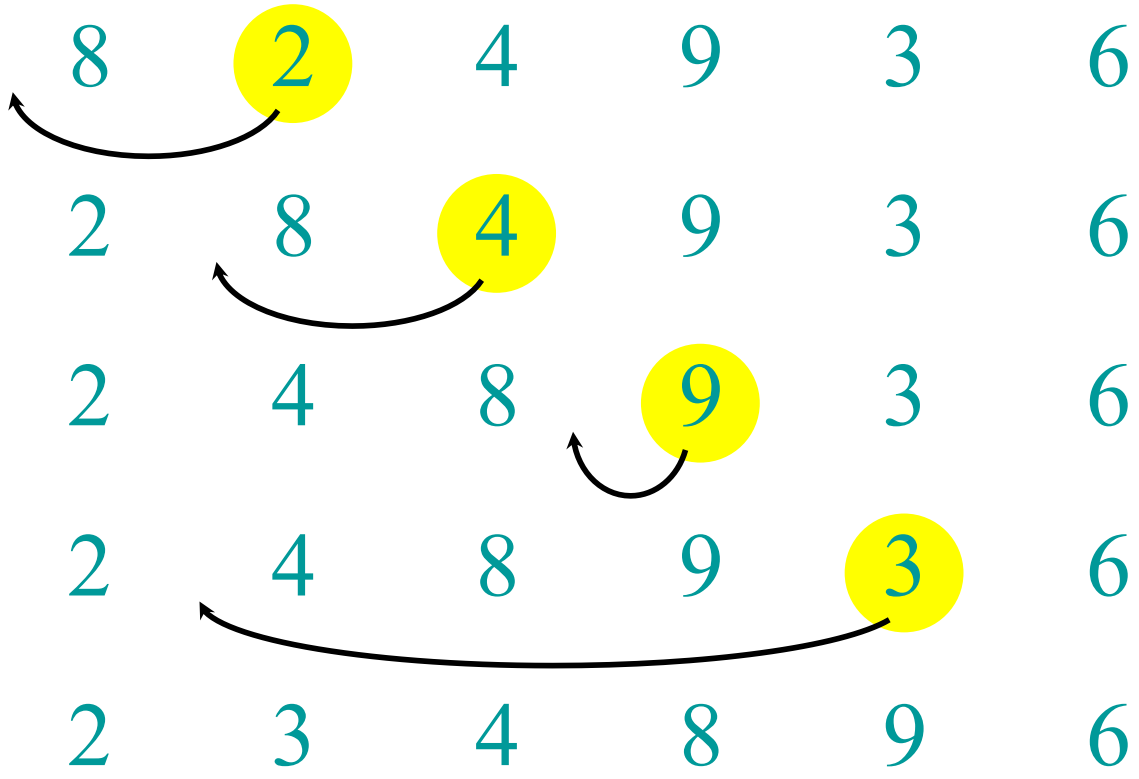
Example of insertion sort



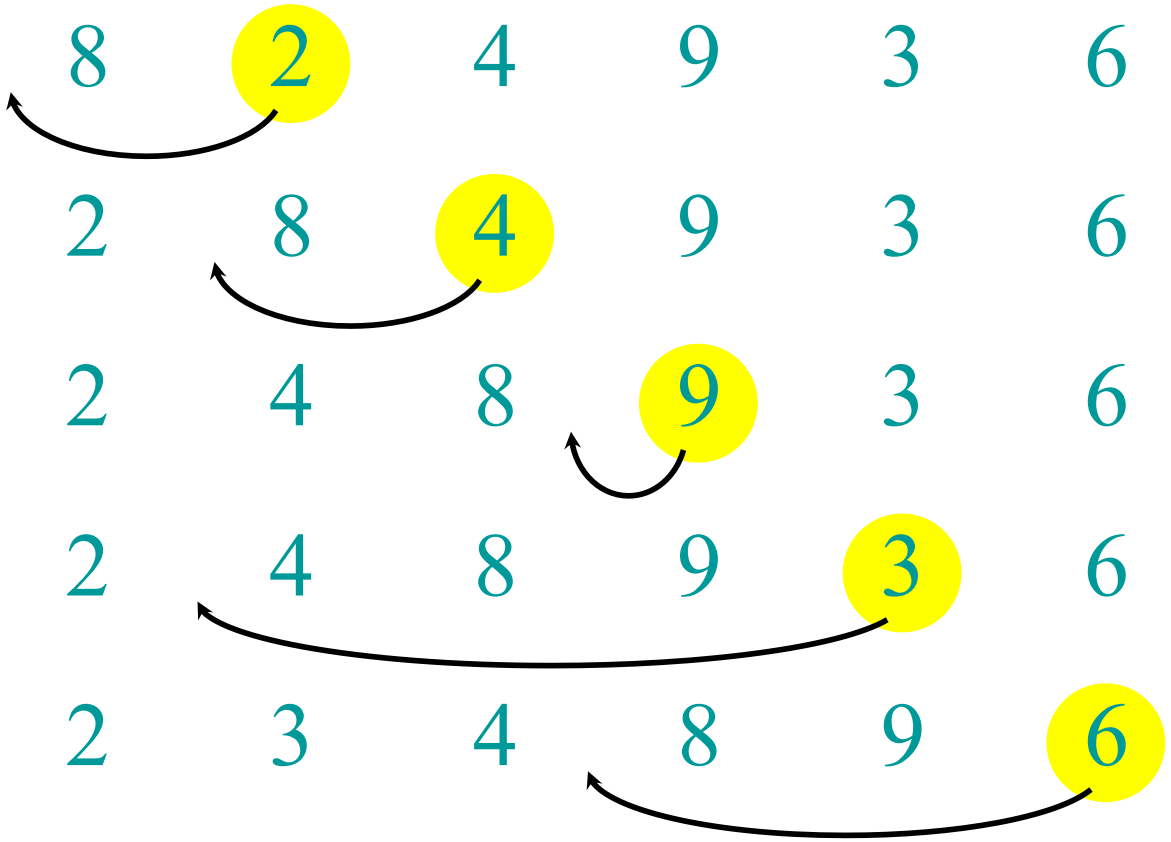
Example of insertion sort



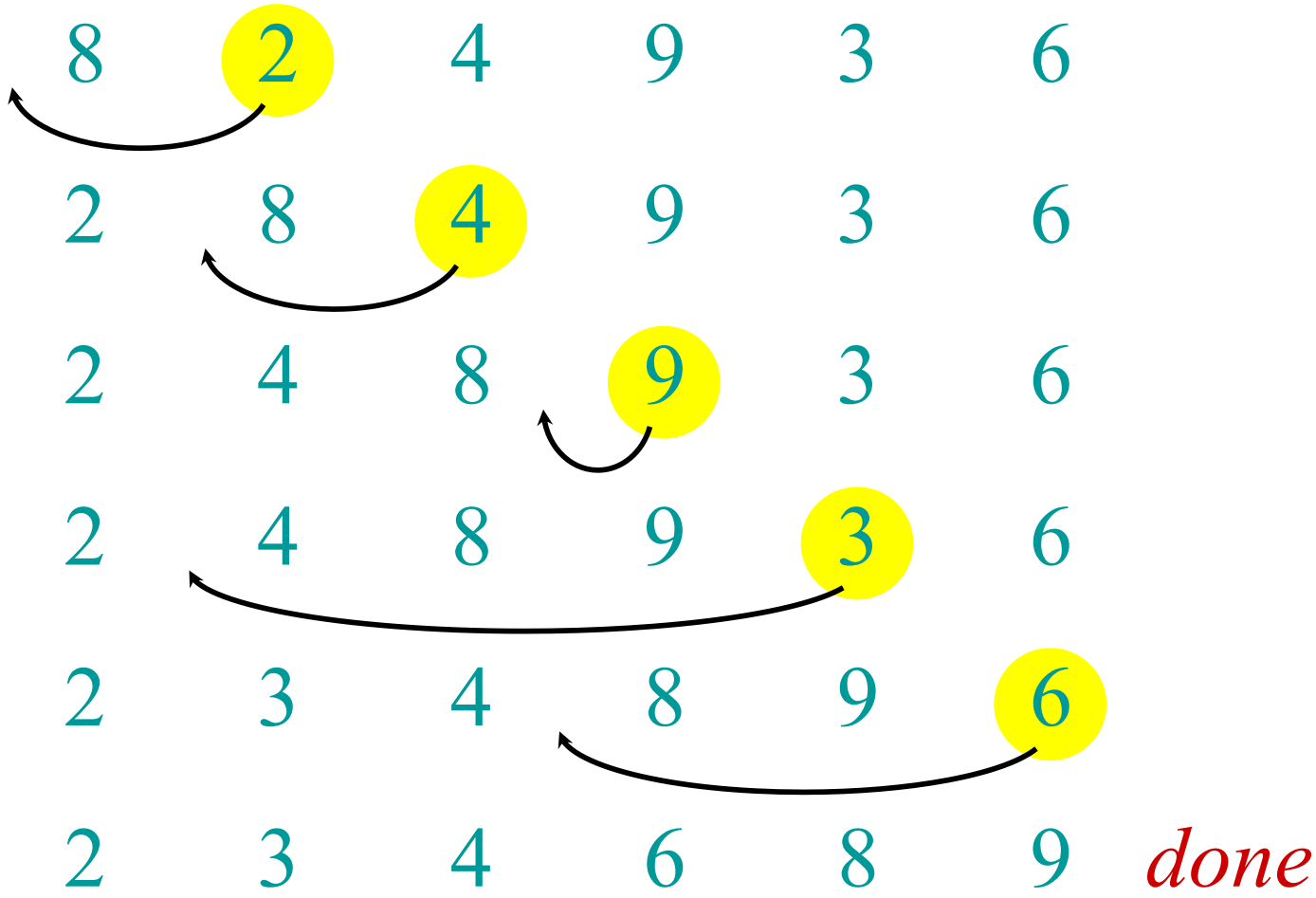
Example of insertion sort



Example of insertion sort

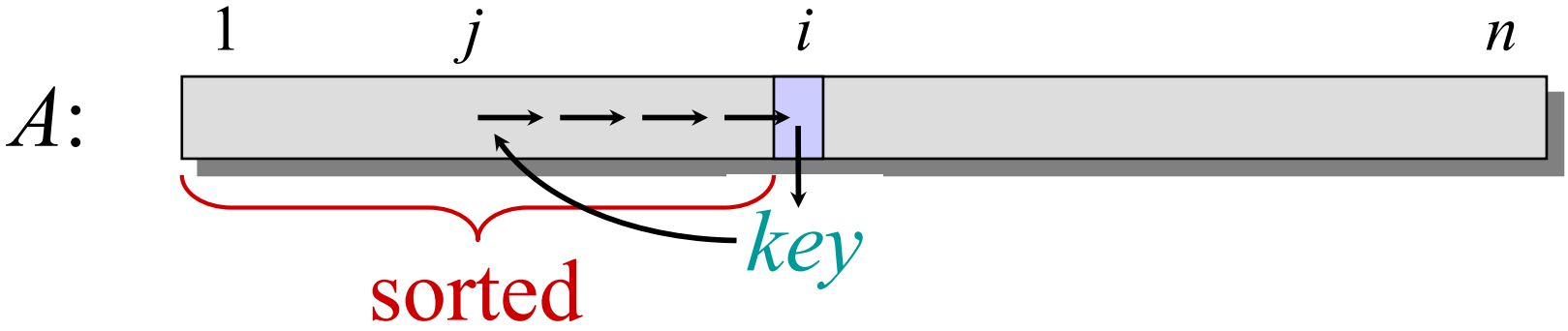


Example of insertion sort



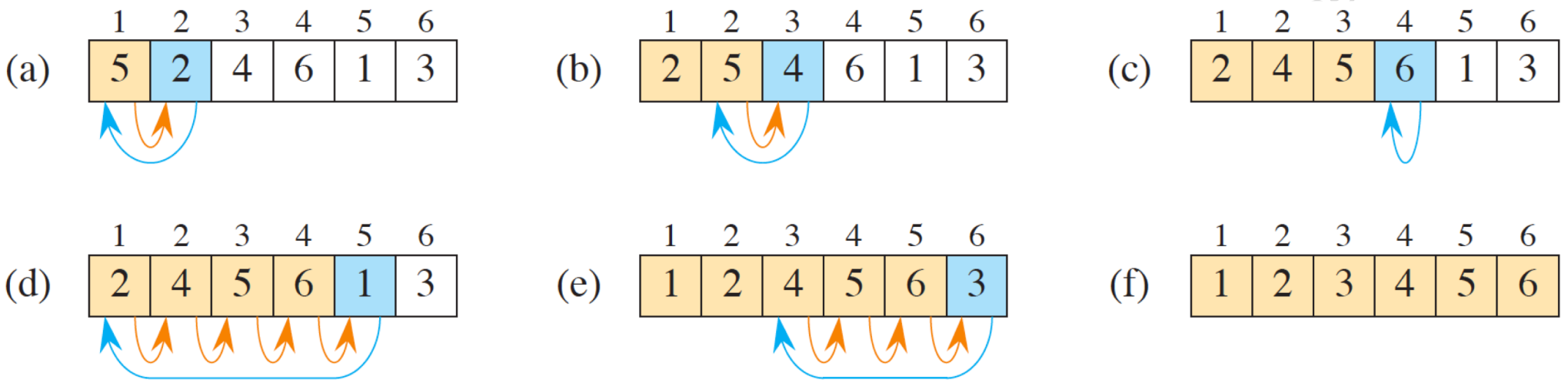
Insertion sort

```
INSERTION-SORT(A, n)
1   for i = 2 to n
2       key = A[i]
3       // Insert A[i] into the sorted subarray A[1 : i − 1].
4       j = i − 1
5       while j > 0 and A[j] > key
6           A[j + 1] = A[j]
7           j = j − 1
8       A[j + 1] = key
```



Example INSERTION-SORT(A,n)

sequence $\langle 5, 2, 4, 6, 1, 3 \rangle$



Pseudocode conventions

- Indentation indicates block structure.
- The looping constructs **while**, **for**, and **repeat-until** and the **if-else** conditional construct have interpretations similar to those in C, Python. we use the keyword **downto** when a for loop decrements its loop counter.
- The symbol // indicates that the remainder of the line is a comment.
- Variables (such as *i* , *j* , and *key*) are local to the given procedure.

Pseudocode conventions

- We access array elements by specifying the array name followed by the index in square brackets.
- Arrays in this book use 1-origin indexing.
- The notation ":" denotes a subarray. Thus, $A[i:j]$ indicates the subarray of A consisting of the elements $A[i], A[i+1], \ldots, A[j]$.

Loop invariants and the correctness of insertion sort

- **Initialization**: It is true prior to the first iteration of the loop.
- **Maintenance**: If it is true before an iteration of the loop, it remains true before the next iteration.
- **Termination**: The loop terminates, and when it terminates, the invariant-usually along with the reason that the loop terminated-gives us a useful property that helps show that the algorithm is correct.

Correctness of insertion sort

Initialization: We start by showing that the loop invariant holds before the first loop iteration, when $i = 2$.[2] The subarray $A[1 : i - 1]$ consists of just the single element $A[1]$, which is in fact the original element in $A[1]$. Moreover, this subarray is sorted (after all, how could a subarray with just one value not be sorted?), which shows that the loop invariant holds prior to the first iteration of the loop.

Correctness of insertion sort

Maintenance: Next, we tackle the second property: showing that each iteration maintains the loop invariant. Informally, the body of the **for** loop works by moving the values in $A[i-1]$, $A[i-2]$, $A[i-3]$, and so on by one position to the right until it finds the proper position for $A[i]$ (lines 4–7), at which point it inserts the value of $A[i]$ (line 8). The subarray $A[1:i]$ then consists of the elements originally in $A[1:i]$, but in sorted order. ***Incrementing*** i (increasing its value by 1) for the next iteration of the **for** loop then preserves the loop invariant.

Correctness of insertion sort

Termination: Finally, we examine loop termination. The loop variable i starts at 2 and increases by 1 in each iteration. Once i's value exceeds n in line 1, the loop terminates. That is, the loop terminates once i equals $n + 1$. Substituting $n + 1$ for i in the wording of the loop invariant yields that the subarray $A[1:n]$ consists of the elements originally in $A[1:n]$, but in sorted order. Hence, the algorithm is correct.