Algorithms & Data Structures I
CSC 225

Ali Mashreghi

Fall 2018

Department of Computer Science, University of Victoria

Introduction

$$\begin{tikzcd}
\textbf{al-Khwarizmi} \arrow[d] \\
\textbf{algebra} \arrow[d] \\
\textbf{algorithm}
\end{tikzcd}$$

What is an algorithm?

- An **algorithm** is an **unambiguous** procedure that takes the input and produces the output.

What is an algorithm?

```
MAKETEA:
1. Fill the kettle with water
2. Boil the water
3. Pour boiled water into the cup
4. Put the tea bag inside the cup
5. Terminate
```

Input: Kettle, tea bag, empty cup

Algorithm

Output: A cup of tea

Sorting

- Sorting is a very fundamental problem with numerous applications:

1. Sorting your files in a directory (e.g. by name, size)
2. Sorting data in Excel
3. Sorting applicants for a job based on GPA

…..

Sorting

Sorting Problem

Input: A sequence of numbers $< a_1, a_2, \ldots, a_n >$

Output: A permutation (reordering) $< a'_1, a'_2, \ldots, a'_n >$ of the input where $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Example input: $< 7, 1, 2, 5, 6, 8, 2, 9 >$

Example output: $< 1, 2, 2, 5, 6, 7, 8, 9 >$

Designing an algorithm

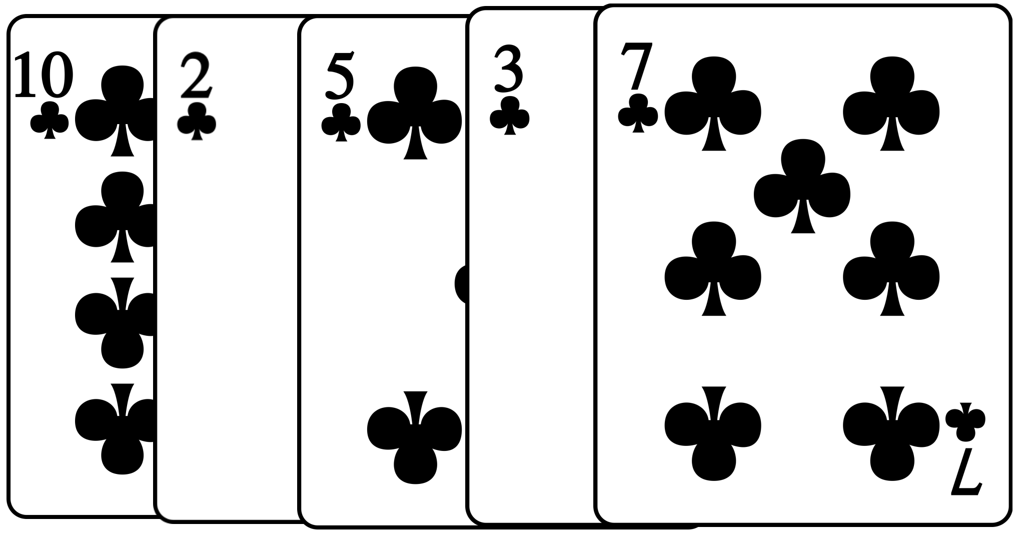
- The algorithm should be **correct**!

- An algorithm is correct if for **all inputs**, it **terminates** and produces the **correct output**.

- So, an algorithm that works on **some inputs** or **sometimes terminates** is **not correct**.

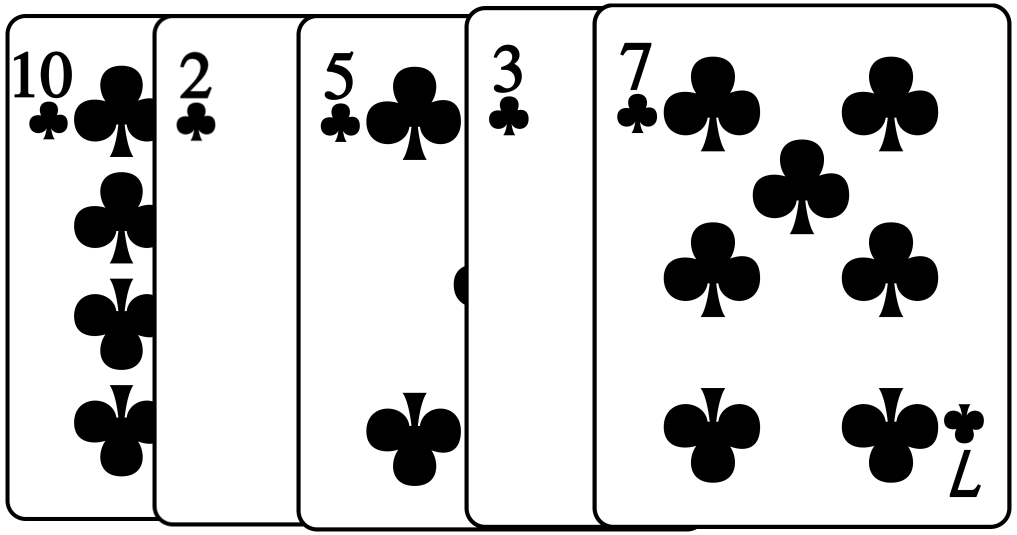
A simple algorithm

- Say we want to sort a set of cards



A simple algorithm

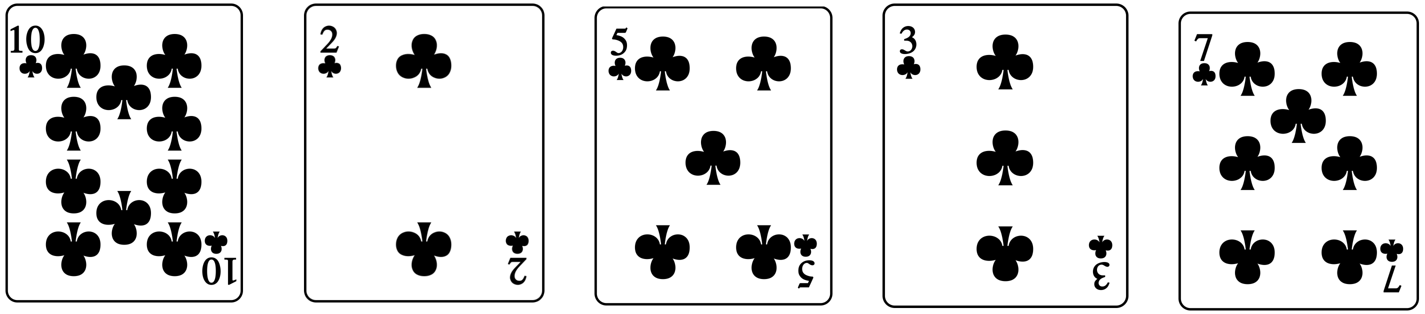
```
RANDOM-TRIAL:
1.  $i = 1$
2.  **while** $i < 1000$ **and** the cards are not sorted
3.      Throw all cards on the floor
4.      Pick them one by one randomly
5.      $i = i + 1$
6.  **terminate**
```



A simple algorithm

```
RANDOM-TRIAL:
1. $i = 1$
2. while $i < 1000$ and the cards are not sorted
3.     Throw all cards on the floor
4.     Pick them one by one randomly
5.     $i = i + 1$
6. terminate
```

A sample trial



Correctness

- **Question:** Is this a correct algorithm?

Correctness

- **Question:** Is this a correct algorithm?
- **Answer:** No, because it may never get the right order.

A correct algorithm

```
INSERTION-SORT($A$)
for $j = 2$ to $A.length$
    $key = A[j]$
    // Insert $A[j]$ into the sorted sequence $A[1 \mathinner{.\,.} j-1]$.
    $i = j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = key$
```

A correct algorithm

```
INSERTION-SORT($A$)
for $j = 2$ to $A.length$
    $key = A[j]$
    // Insert $A[j]$ into the sorted sequence $A[1 \, . . \, j-1]$.
    $i = j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = key$
```

pseudocode

- Pseudocode is a fancy word for structured English :)

Pseudocode

- A **pseudocode** is a **notation** that describes an algorithm clearly and briefly
- Pseudocode is intended for **human reading** and not for machines to execute
- Our pseudocodes omits the details that are specific to a programming language, and also error-handling and other software engineering issues

Some pseudocode conventions

procedure name

parameter(s)

length of the array is available

starting index is 1

comments using //

assignment

condition (short-circuiting)

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

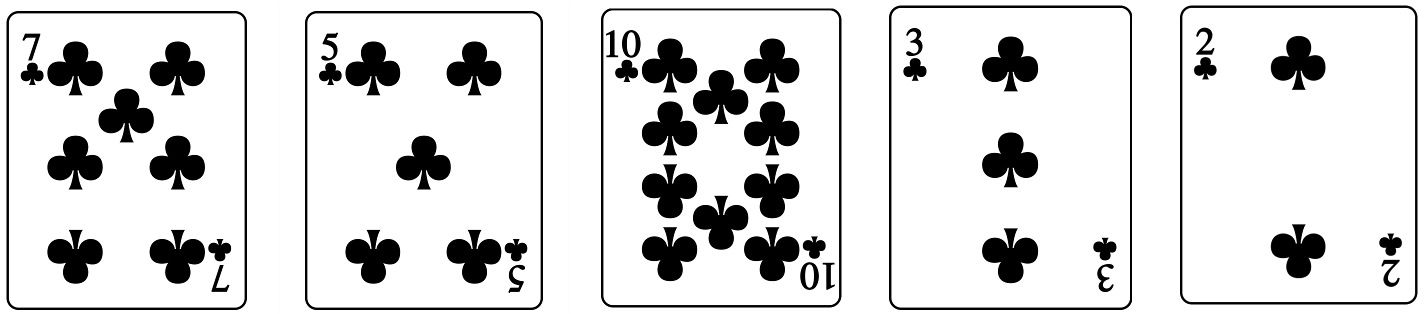
Some pseudocode conventions

- We pass parameters to a procedure ***by value***: the called procedure receives its own copy of the parameters, and if it assigns a value to a parameter, the change is *not* seen by the calling procedure. When objects are passed, the pointer to the data representing the object is copied, but the object's attributes are not. This is how passing parameters is done in Java.

- To see all conventions see CLRS page 20.

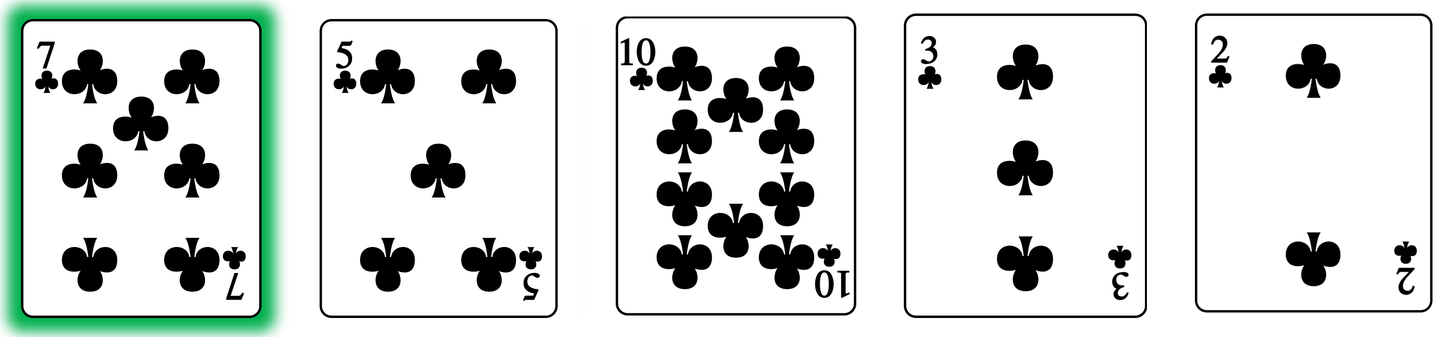
A correct algorithm

```
INSERTION-SORT($A$)
for $j = 2$ to $A.length$
    $key = A[j]$
    // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.
    $i = j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = key$
```



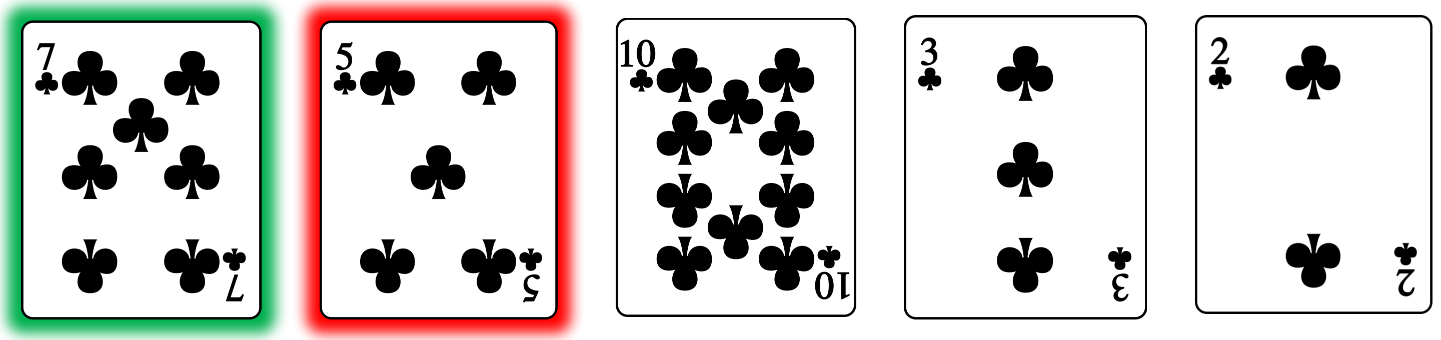
A correct algorithm

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



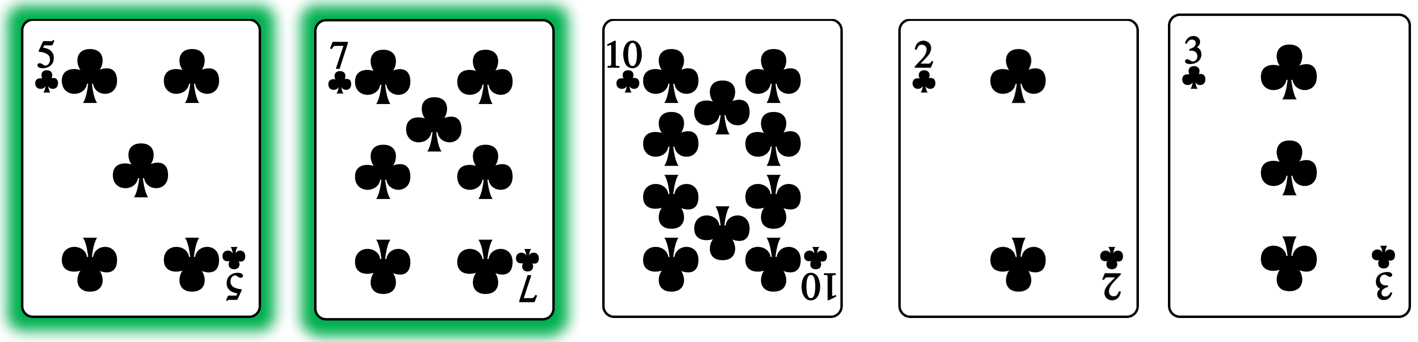
A correct algorithm

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



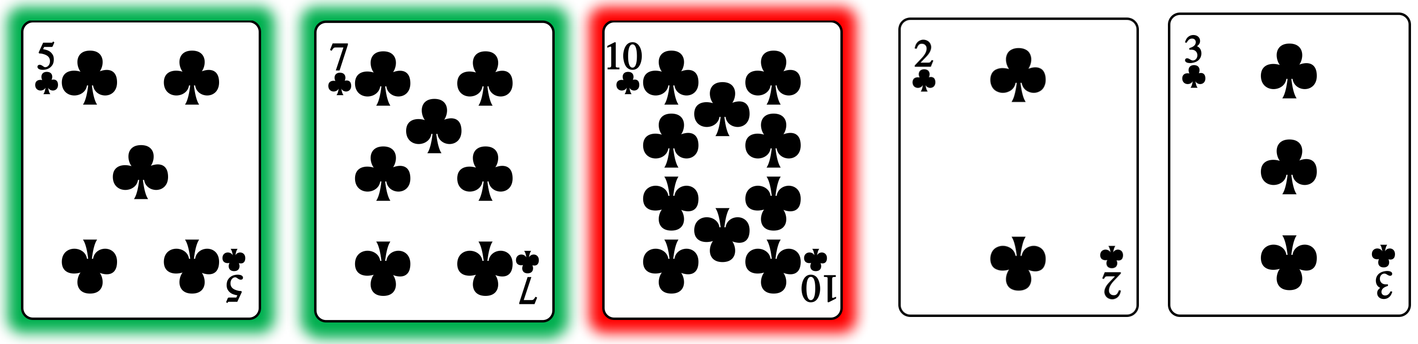
A correct algorithm

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



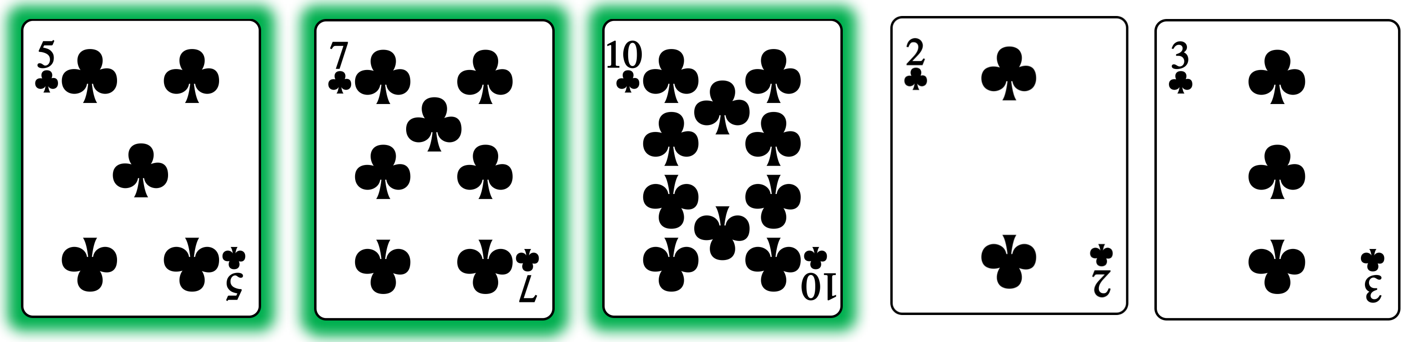
A correct algorithm

```
INSERTION-SORT(A)
1  for $j = 2$ to $A.length$
2      $key = A[j]$
3      // Insert $A[j]$ into the sorted sequence $A[1 \mathinner{.\,.} j-1]$.
4      $i = j - 1$
5      while $i > 0$ and $A[i] > key$
6          $A[i+1] = A[i]$
7          $i = i - 1$
8      $A[i+1] = key$
```



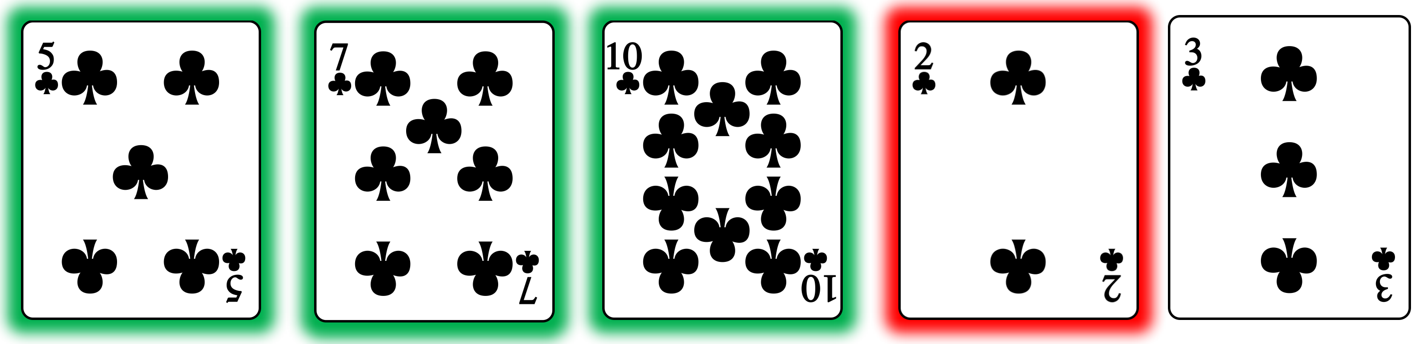
A correct algorithm

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



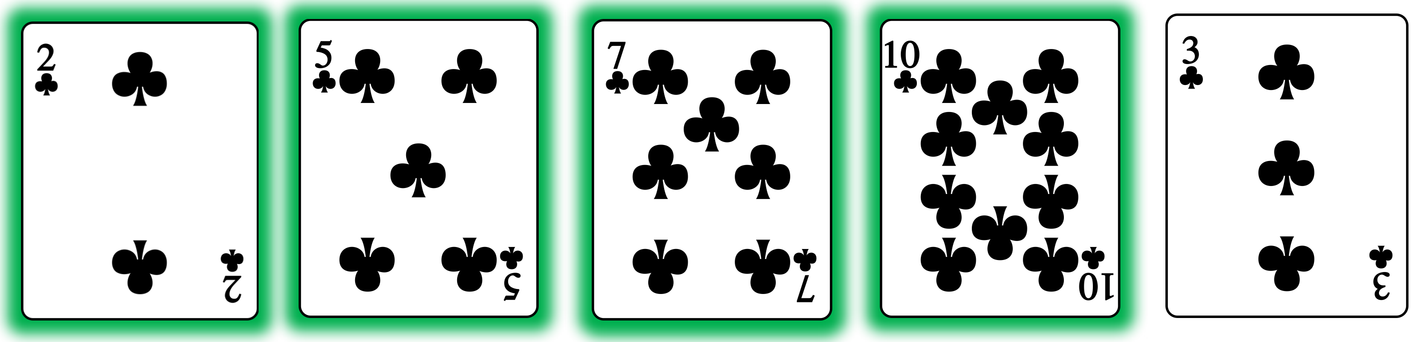
A correct algorithm

```
INSERTION-SORT($A$)
for $j = 2$ to $A.length$
    $key = A[j]$
    // Insert $A[j]$ into the sorted sequence $A[1 \mathinner{.\,.} j-1]$.
    $i = j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = key$
```



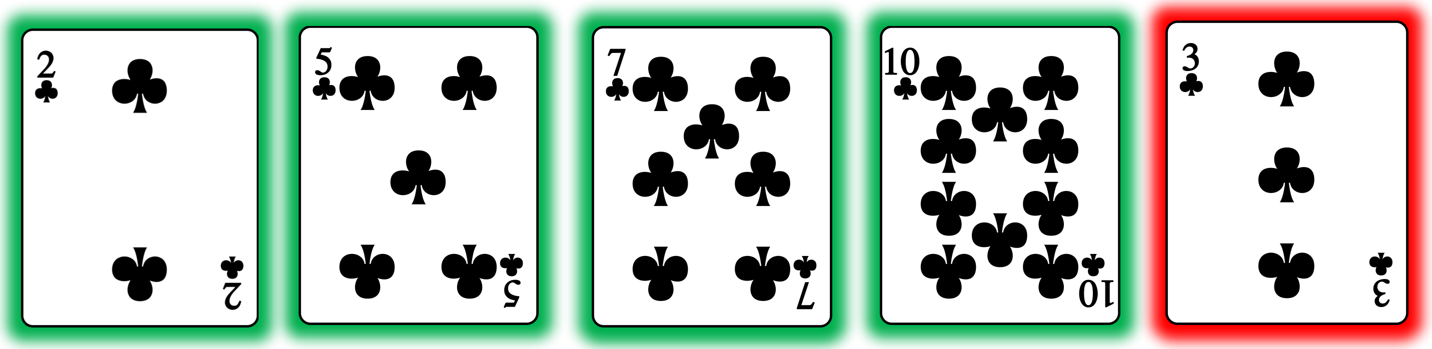
A correct algorithm

```
INSERTION-SORT($A$)
for $j = 2$ to $A.length$
    $key = A[j]$
    // Insert $A[j]$ into the sorted sequence $A[1 \mathinner{..} j-1]$.
    $i = j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = key$
```



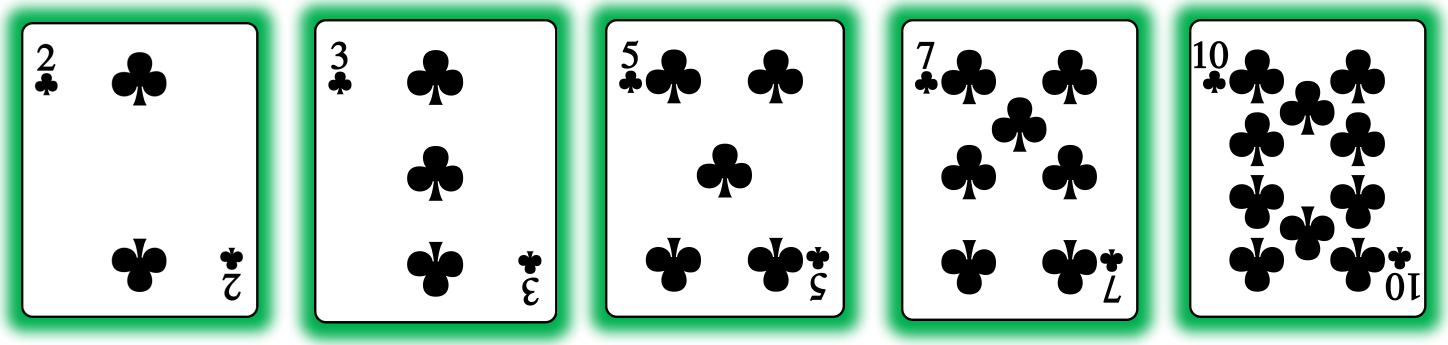
A correct algorithm

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



A correct algorithm

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



Correctness of Insertion-Sort algorithm

- We use a proof technique called ***loop invariant.***
- A loop invariant **is some property p.**
- The hard part is to find the right invariant.

Correctness of INSERTION-SORT algorithm

- We must show three things about the loop invariant:
    - **Initialization:** p is true prior to the first iteration
    - **Maintenance:** If p is true before an iteration, it remains true before the next
    - **Termination:** When the loop terminates correctness of p helps us show that the algorithm is correct

- This process is very similar to proof by induction!

Correctness of Insertion-Sort algorithm

Loop invariant:

At the start of each iteration of the **for loop** for some value of j, the subarray $A[1\,..\,j-1]$ consists of all of the elements of $A[1\,..\,j-1]$ in sorted order.