AYUDANTÍA 1

INDUCCIÓN
NOTACIÓN ASINTÓTICA
CORRECTITUD
INTRO A SORTING

Inducción

PRINCIPIO DE INDUCCIÓN SIMPLE

Sea P una propiedad sobre los elementos de N. Si se cumple que:

BASE INDUCTIVA $P(n_0)$ es verdadero $n_0 \in N$ cumple con propiedad P

HIPÓTESIS TESIS

PASO INDUCTIVO Si $P(n)$, entonces $P(n+1)$

cada vez que n cumple con la propiedad n+1 también la cumple

Entonces todos los elementos de N a partir de n_0 cumplen con la propiedad

EJEMPLO

Demostrar que la afirmación P(n) -> $1 + 2 + ... + n = n(n+1)/2$
se cumple para todo natural n

1. Base inductiva: $n = 1$
2. Hipótesis inductiva: Asumimos que P(m) se cumple para un número m
3. Tesis inductiva: Demostramos que P(m+1) se cumple

EJEMPLO

1. Base inductiva: $n = 1$:
   a. $1 = 1 * (1+1) / 2$
2. Hipótesis inductiva: Asumimos que $P(m)$ se cumple para un número m:
   a. $1 + 2 + ... + m = m(m+1)/2$
3. Tesis inductiva: Demostramos que $P(m+1)$ se cumple:
   a. $1 + 2 + ... + m + (m + 1) = m(m+1)/2 + (m+1)$
   b. $1 + 2 + ... + m + (m + 1) = (m+1) * (m/2 + 1)$
   c. $1 + 2 + ... + m + (m + 1) = (m+1) * ((m+1) + 1) / 2$

Inducción

PRINCIPIO DE INDUCCIÓN FUERTE

Sea P una propiedad sobre elementos de N. Si se cumple que:

- $\forall n \in N$, $\forall k \in N$, $k < n$, $P(k)$ es verdadero $\Rightarrow$ $P(n)$ es verdadero

Entonces P es verdadero para todos los elementos de N.

En palabras simples “Suponemos que para todo número menor que n se cumple la propiedad P, si n también la cumple entonces se cumple para todos los naturales”

EJEMPLO

Demostrar que todo natural $n>1$ puede ser escrito como la multiplicación de uno o más números primos.

1. Base inductiva: $n = 2$
2. Hipótesis inductiva: Asumimos que se cumple para todo k natural tal que $k < m$.
3. Tesis inductiva: Demostramos que se cumple para m

EJEMPLO

1. Base inductiva: n = 2:
    a. 2 es primo
2. Hipótesis inductiva: Asumimos que se cumple para todo k ∈ N, tal que k < m
3. Tesis inductiva: Demostramos que se cumple para m
    a. Si m es primo -> trivial
    b. Si m no es primo:
        i. Existen 2 enteros j, l ∈ [2, k] tal que m = j*l
        ii. Por H.I. j y l pueden ser escritos como el producto de primos
        iii. m puede ser escrito como producto de primos

Notación Asintótica

NOS PERMITE:

1. Determinar tiempos de respuesta de nuestro algoritmo
2. Determinar recursos computacionales
3. Ver escalabilidad de nuestra función
4. Elegir algoritmos de forma eficiente

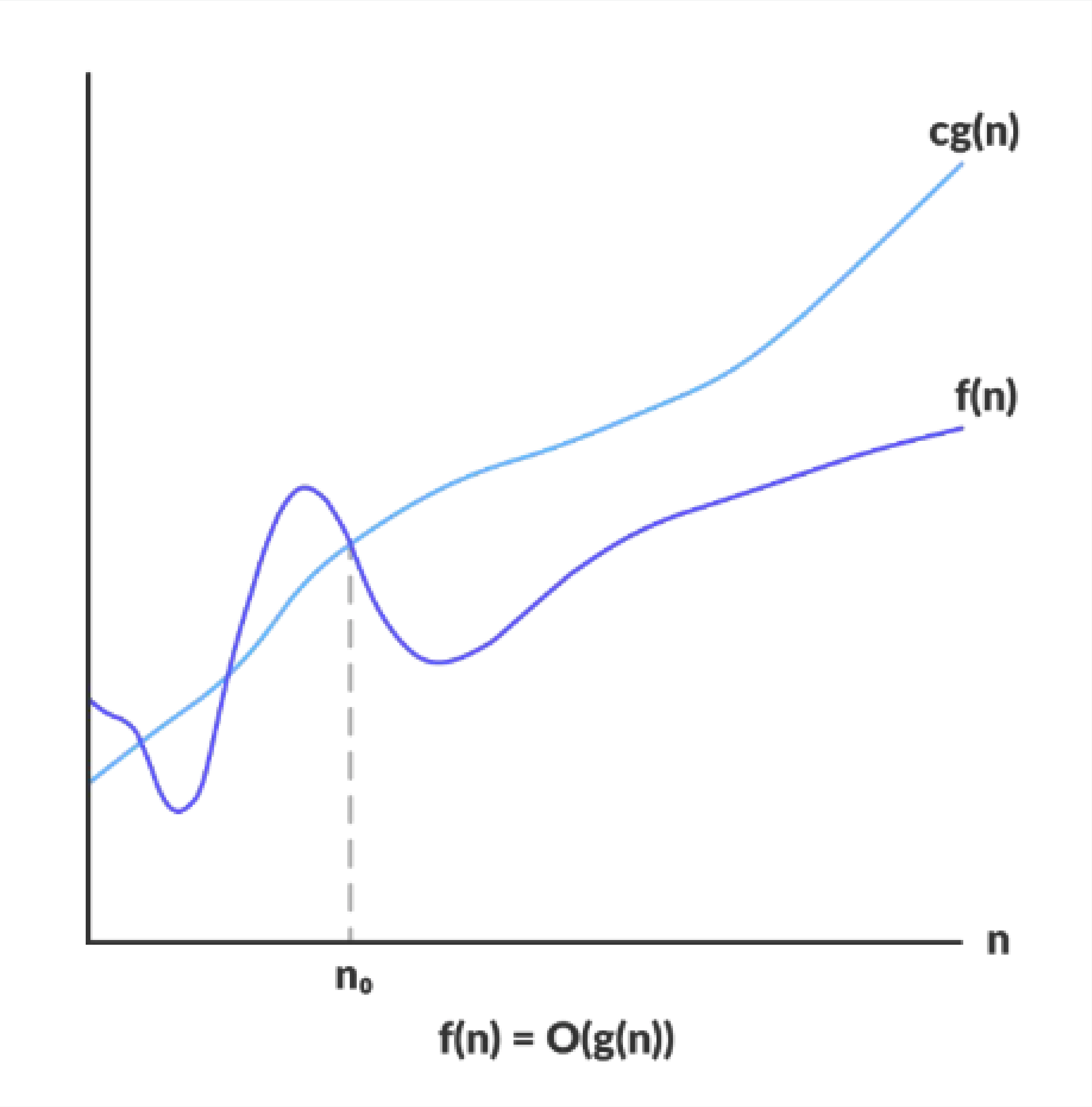
Notación O

Dada una función g(n), denotamos como O(g(n)) al conjunto de funciones tales que:

$$O(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^+ | \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : f(n) \leq cg(n), \forall n > n_0\}$$

*La notación O es una cota superior asintótica

Notación O



Si un tiempo de ejecución es de O(g(n)), entonces para n suficientemente grande, el tiempo de ejecución es a lo más c·g(n) para alguna constante c

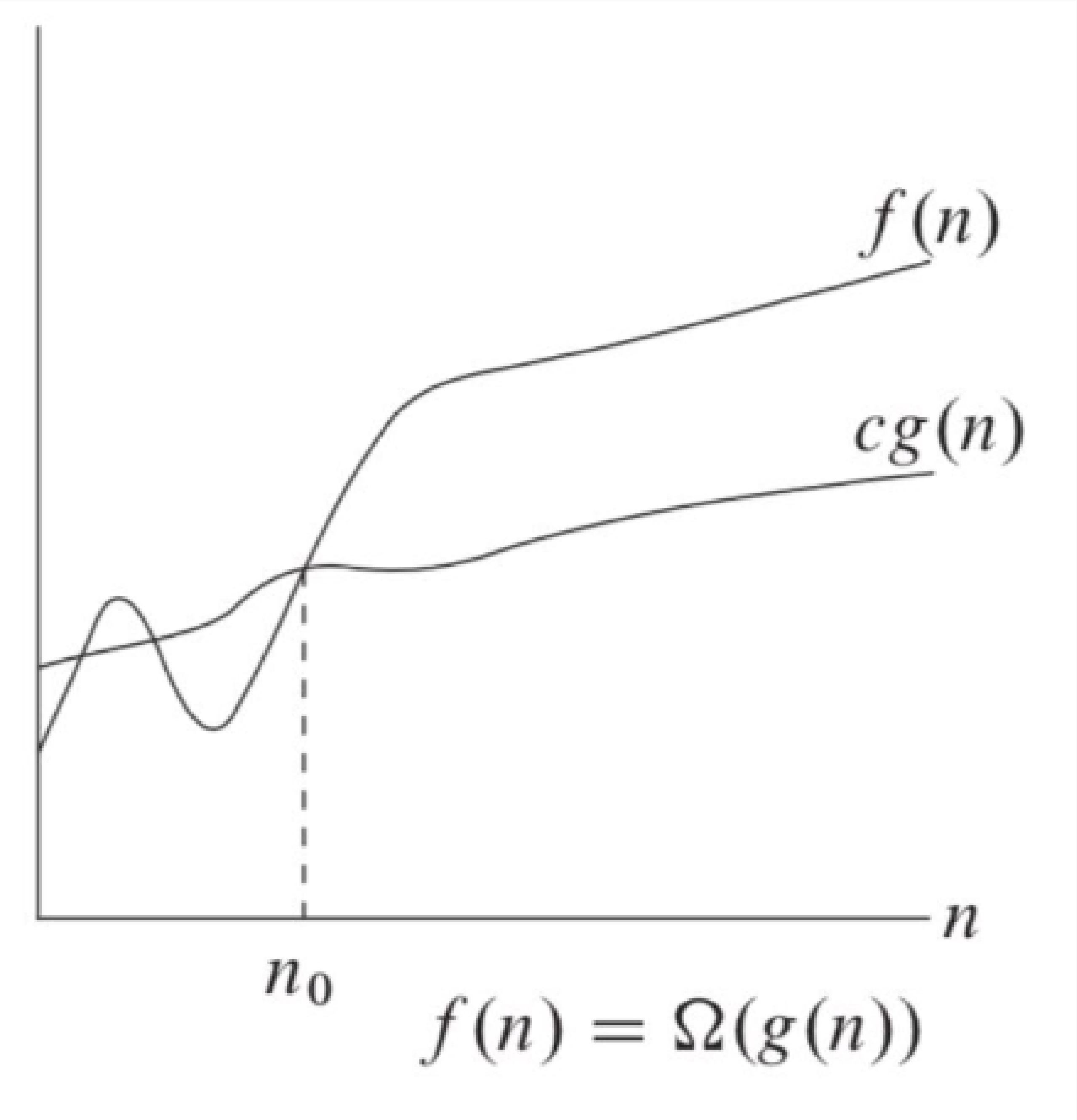
Notación Omega

Dada una función g(n), denotamos como $\Omega(g$(n)) al conjunto de funciones tales que:

$$\Omega(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^+ | \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : 0 < cg(n) \leq f(n), \forall n > n_0\}$$

*Para decir que un algoritmo toma por lo menos cierta cantidad de tiempo
*La notación Omega es una cota asintótica inferior

Notación Omega



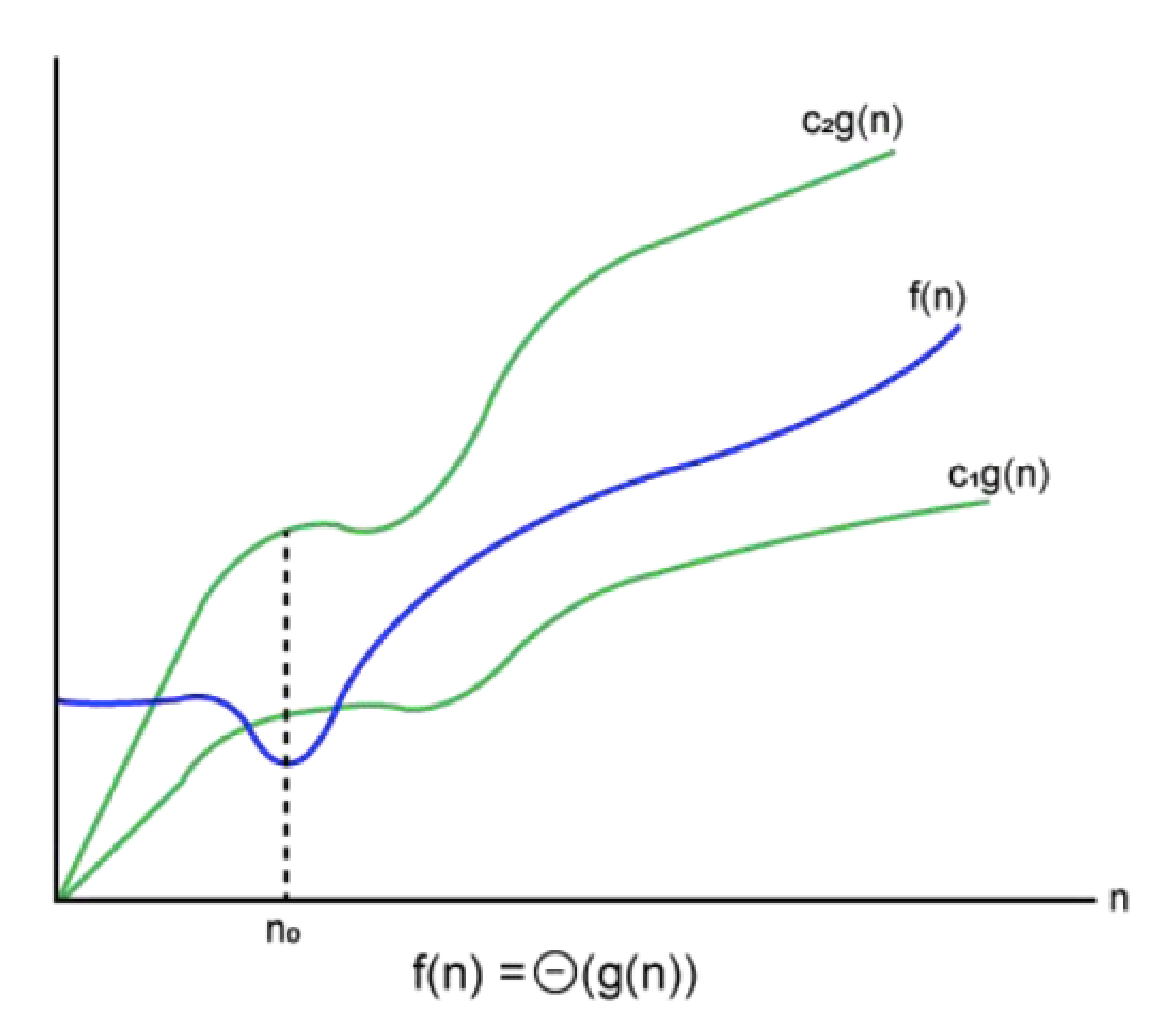
Si un tiempo de ejecución es de Omega(g(n)), entonces para n suficientemente grande, el tiempo de ejecución es por lo menos c·g(n) para alguna constante c

Notación Theta

Diremos que f (n) ϵ $\boldsymbol{\Theta}(g$(n)) si f (n) ϵ Ω(g(n)) y f (n) ϵO(g(n)), es decir:

$$\Theta(g) = \{f : \mathbb{N} \to \mathbb{R}^+ | \exists c_1, c_2 \in \mathbb{R}^+ : 0 < c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n > n_0\}$$

Notación Theta



Si un tiempo de ejecución es de $\Theta(g(n))$, implica $O(g(n))$ y $Omega(f(n))$

CORRECTITUD

Un algoritmo es correcto si siempre se cumple que:

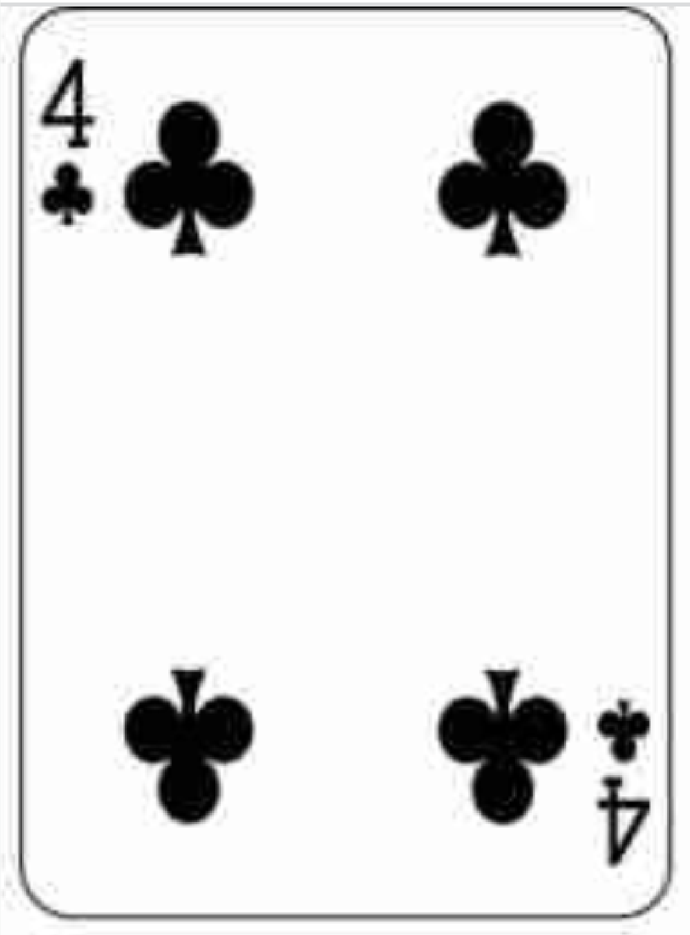
1. El algoritmo termina en tiempo finito
2. Se obtiene el resultado esperado (ej: ordenar)

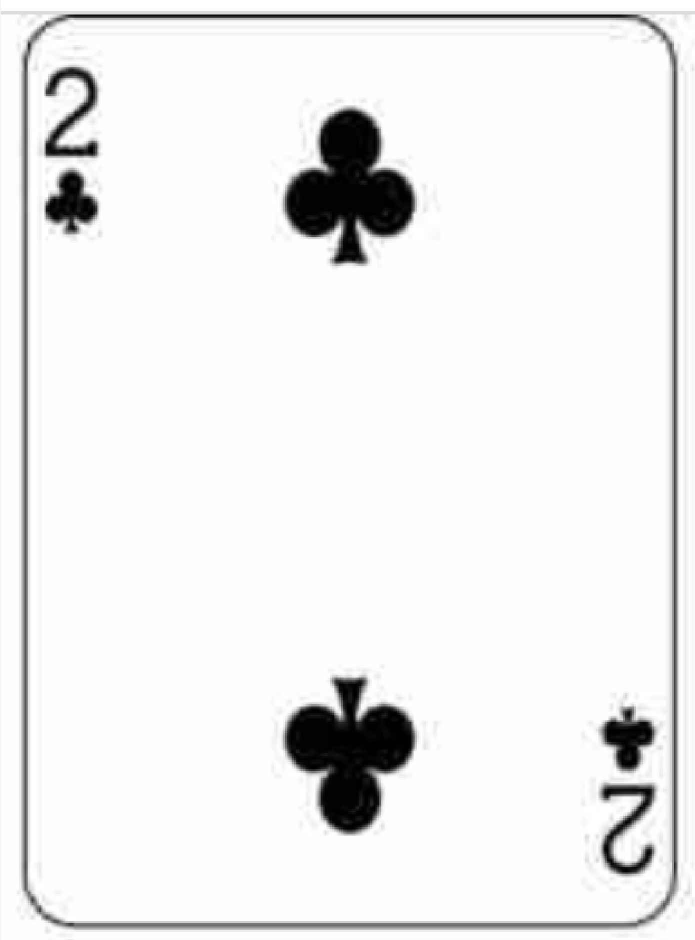
La correctitud se suele demostrar mediante inducción porque es compatible con problemas de tamaño crecientes

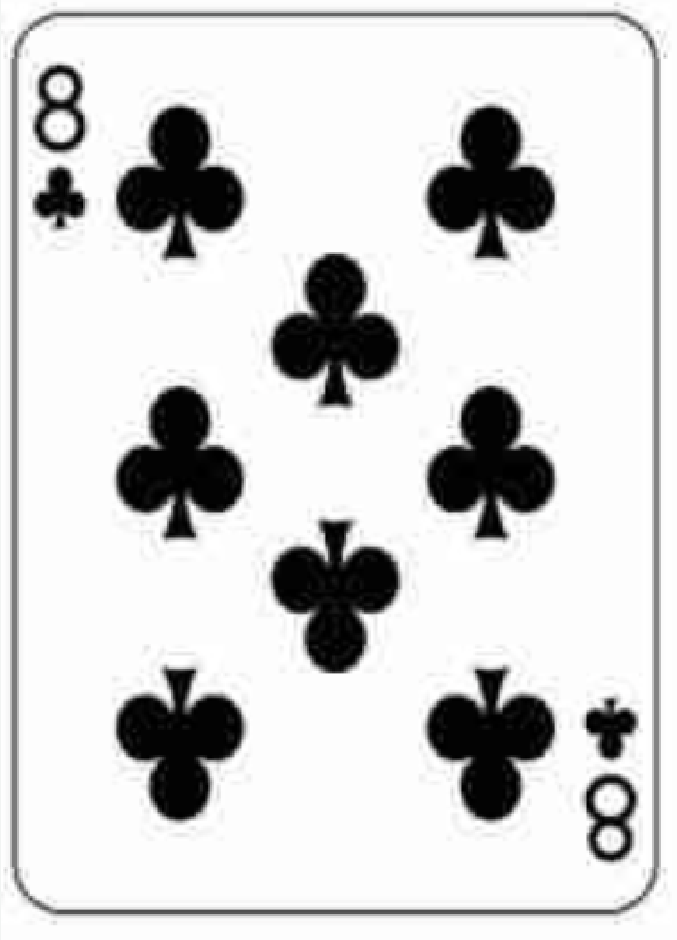
EJEMPLO: BUBBLE SORT

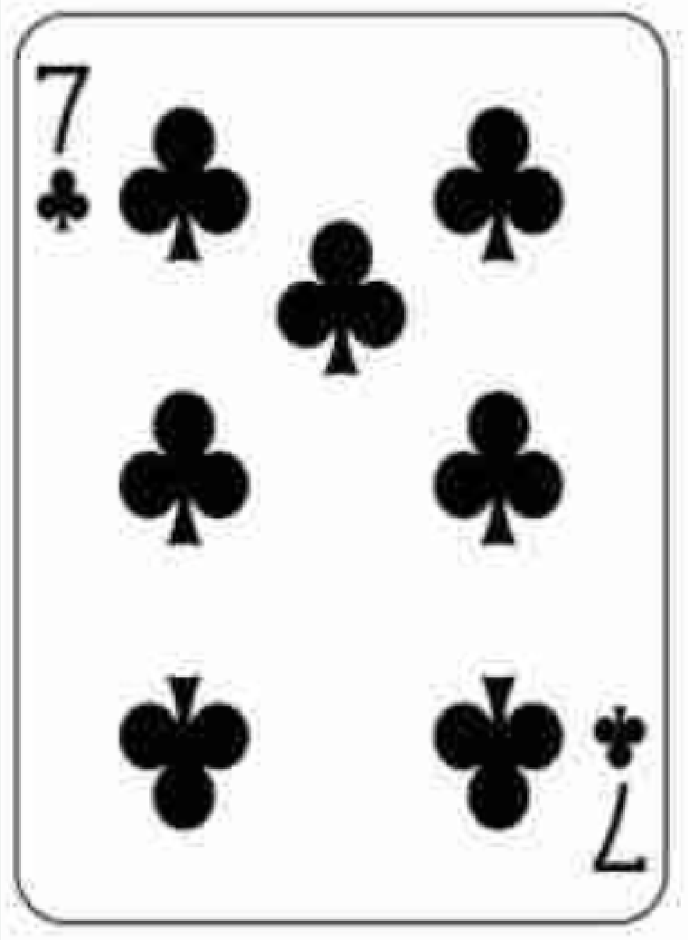
1. Tenemos una secuencia desordenada
2. Tomar el primer dato 'x' de la secuencia
3. Comparar con el elemento siguiente 'y'
4. Si y < x, se intercambian
5. Avanzar una posición y actualizar 'x'
6. Repetir 3 - 5 hasta el penúltimo elemento
7. Repetir todo hasta hacer una iteración sin intercambios

EJEMPLO: BUBBLE SORT



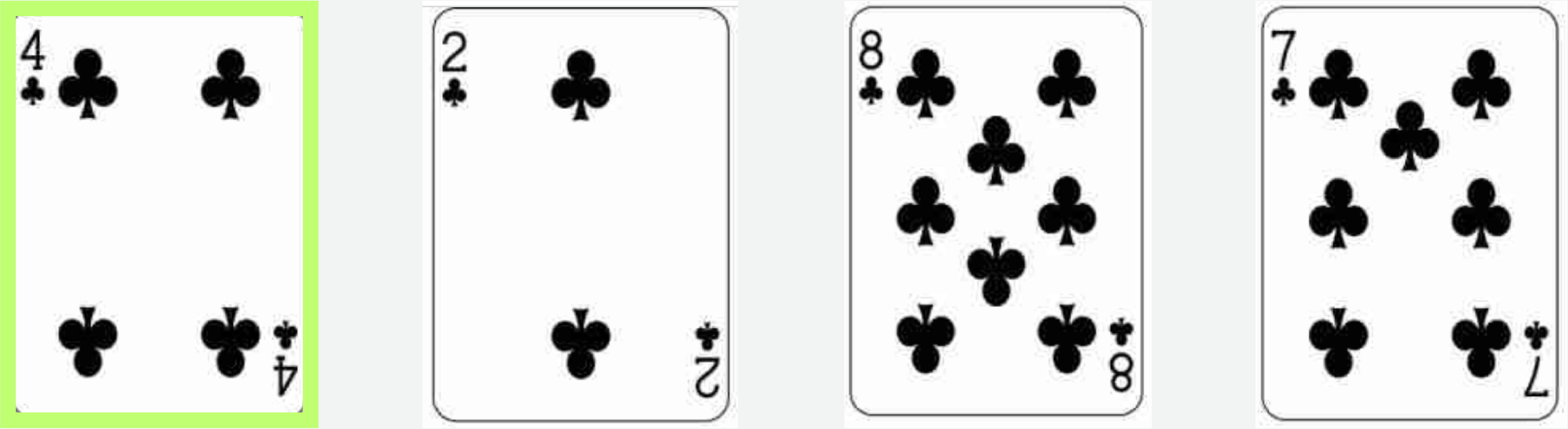






1º ITERACIÓN

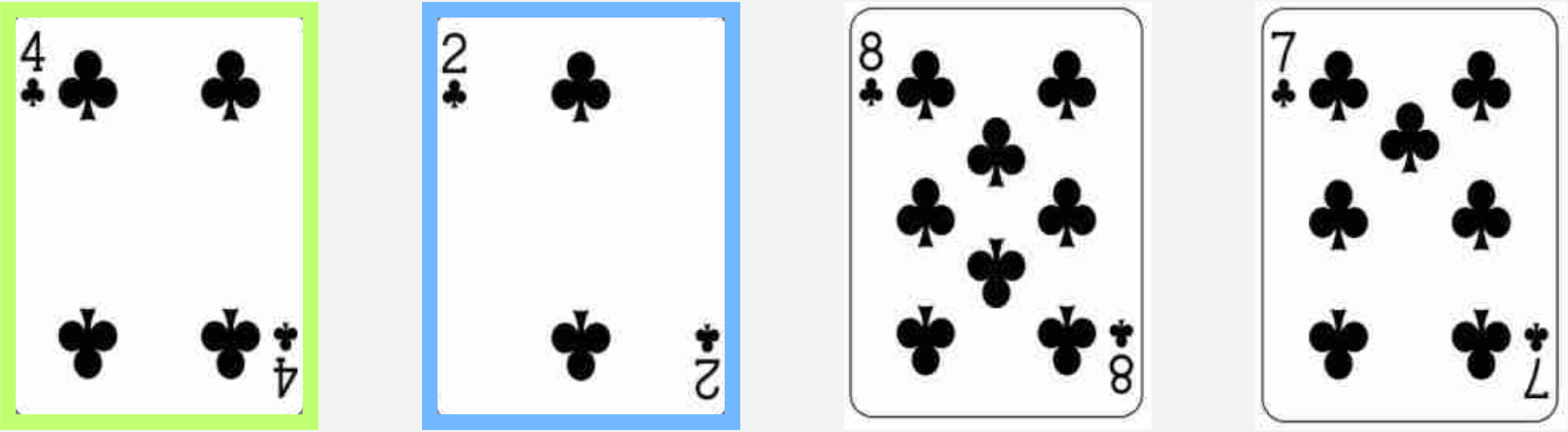
Intercambios = 0



Posición actual

1º ITERACIÓN

Intercambios = 0

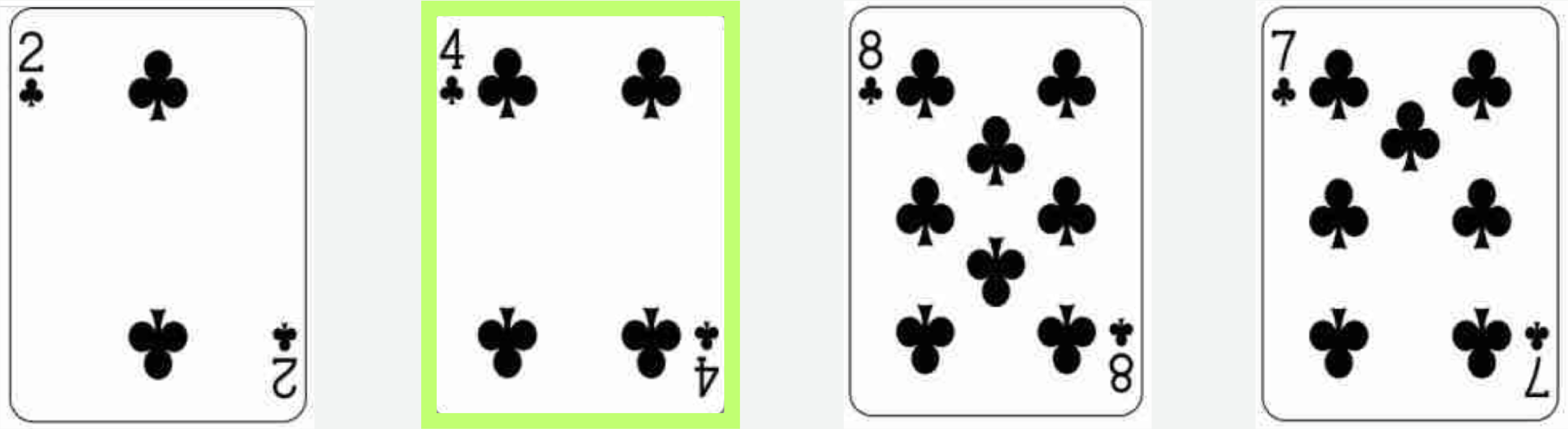


Posición actual

Posición siguiente

1º ITERACIÓN

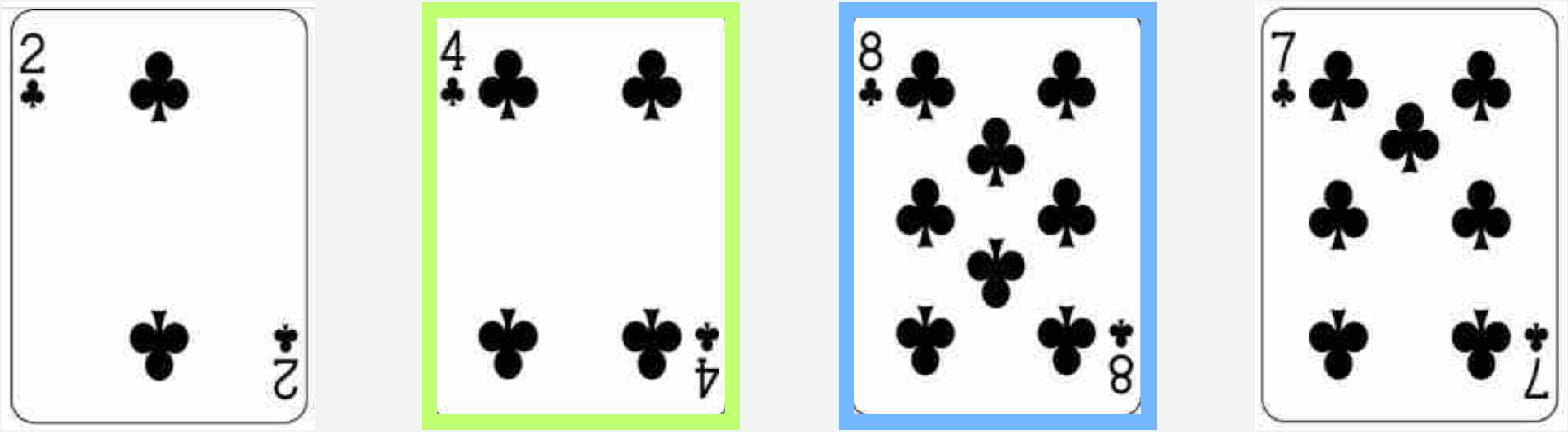
Intercambios = 1



Posición actual

1º ITERACIÓN

Intercambios = 1

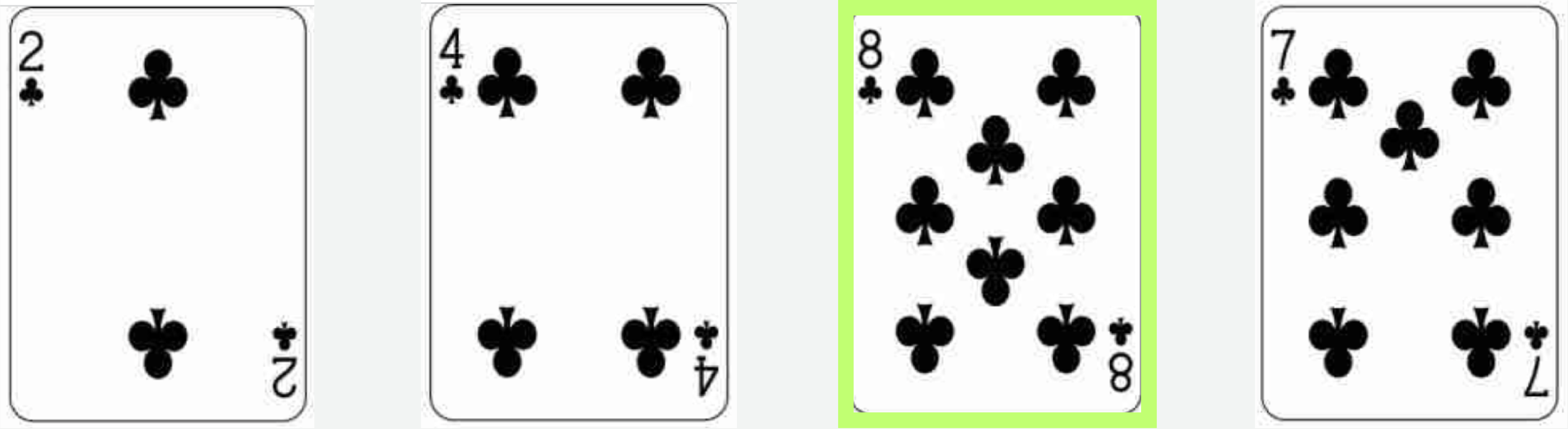


Posición actual

Posición siguiente

1º ITERACIÓN

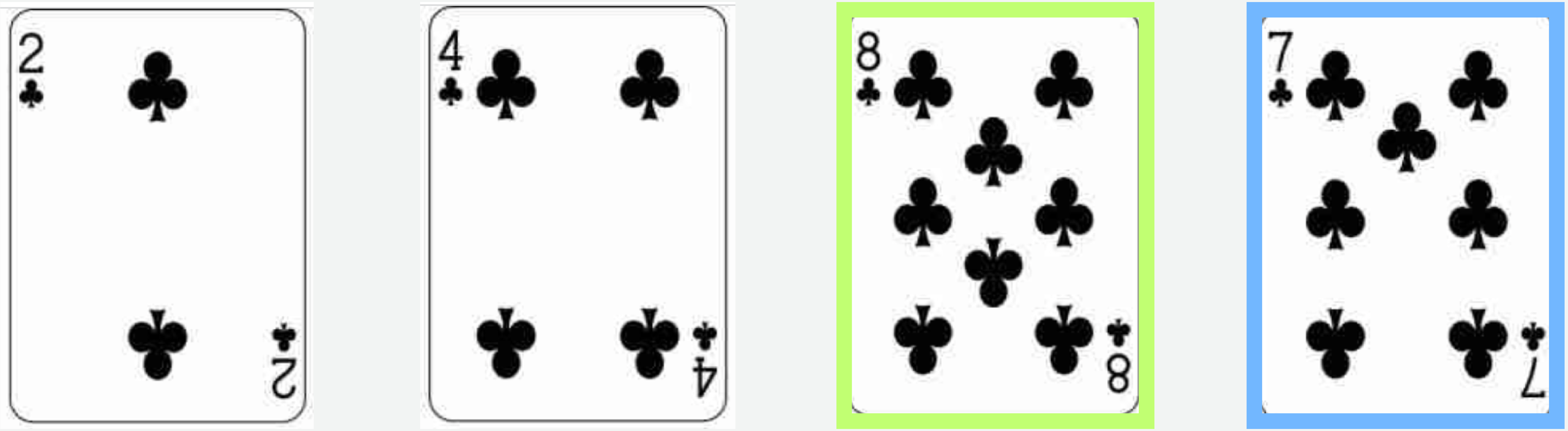
Intercambios = 1



Posición actual

1º ITERACIÓN

Intercambios = 1

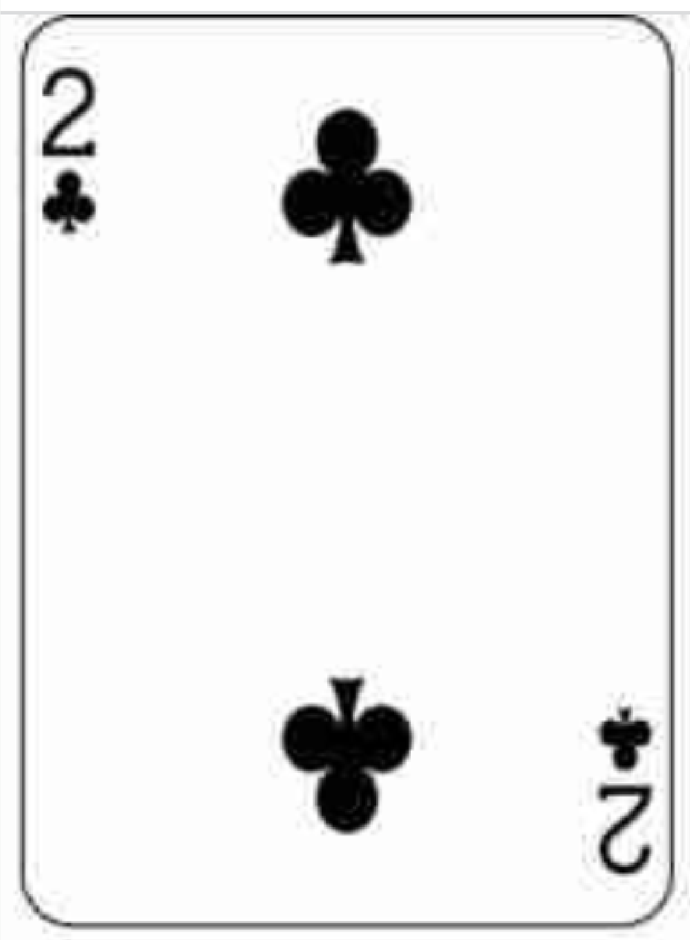


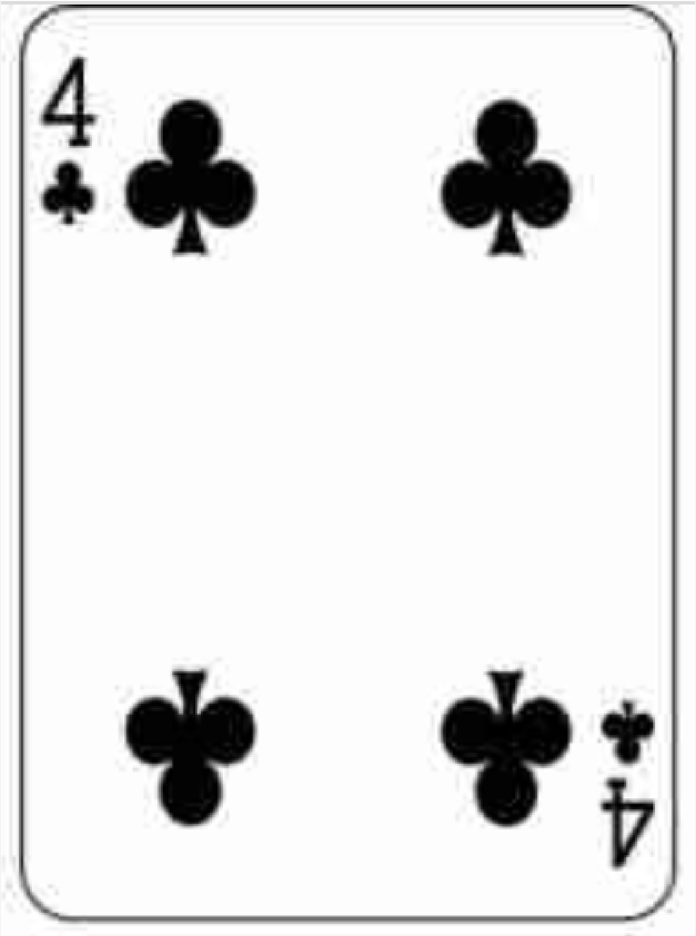
Posición actual

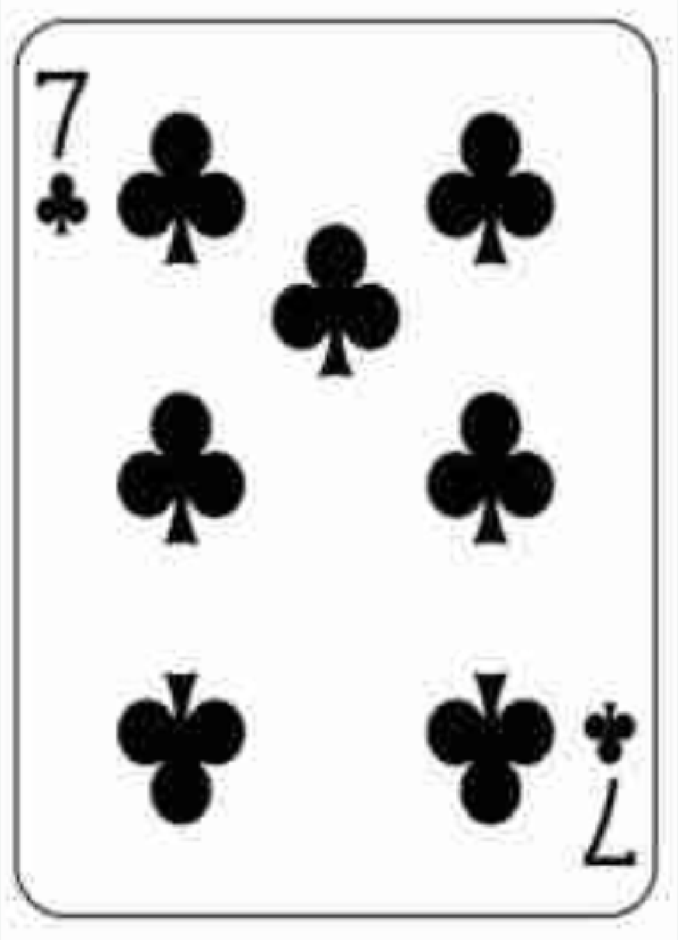
Posición siguiente

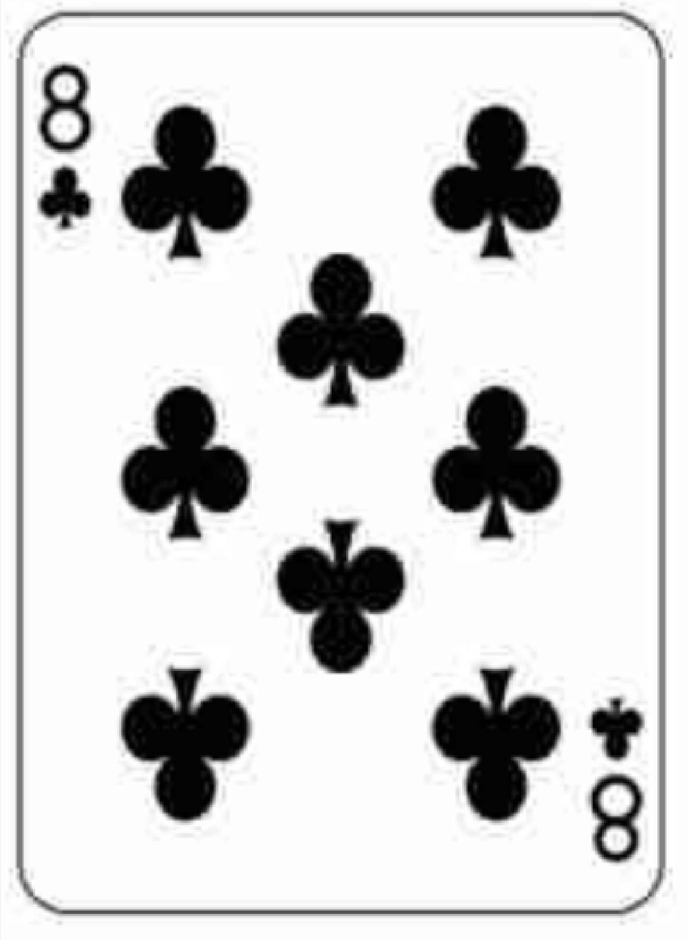
1º ITERACIÓN

Intercambios = 2



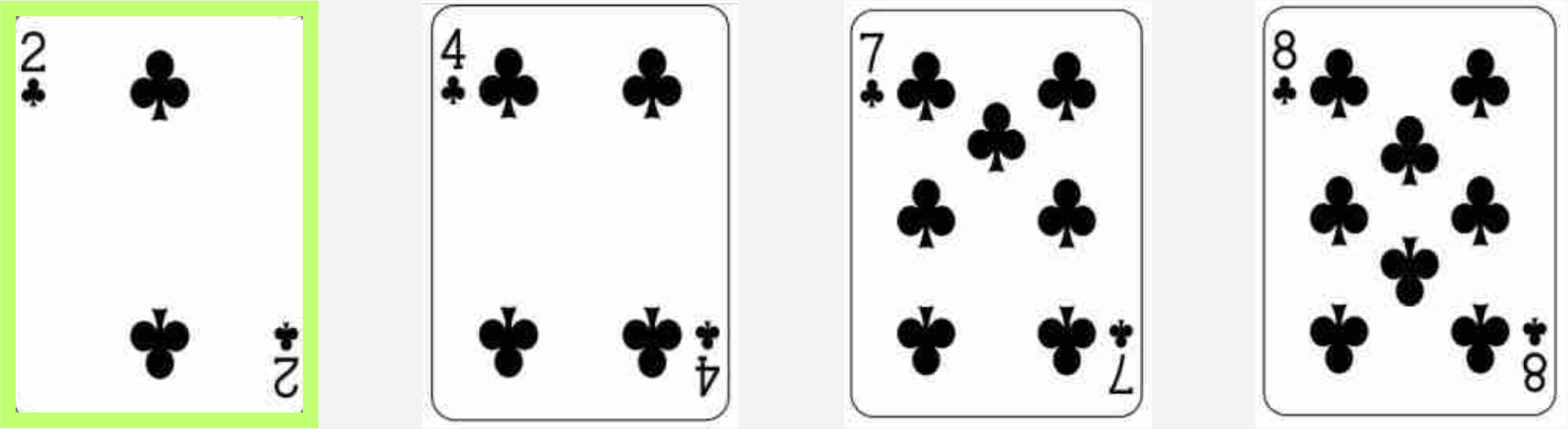






2º ITERACIÓN

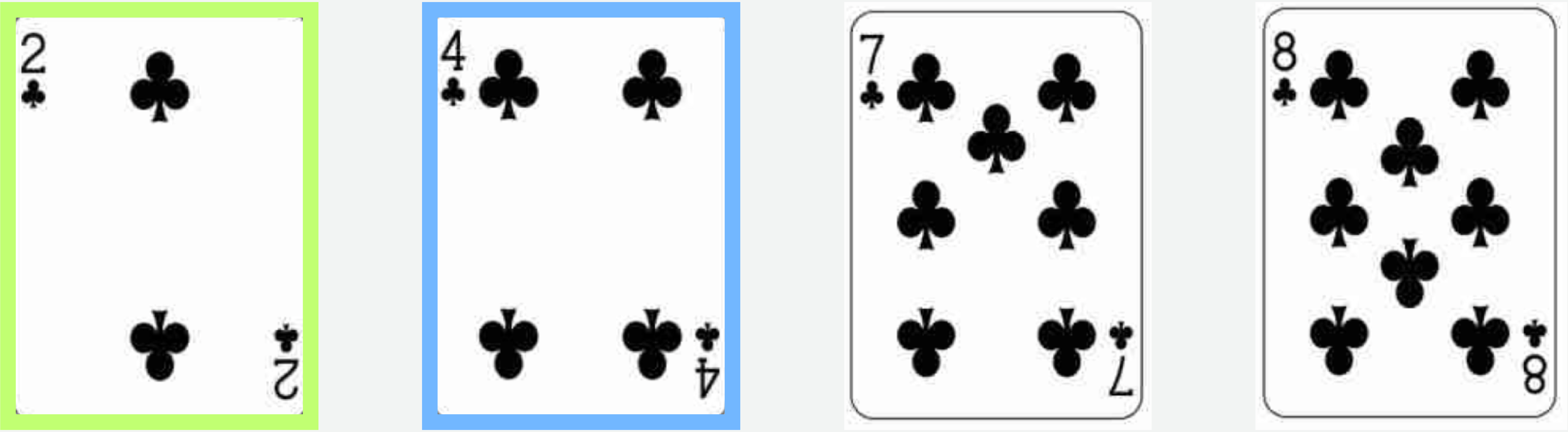
Intercambios = 0



Posición actual

2º ITERACIÓN

Intercambios = 0

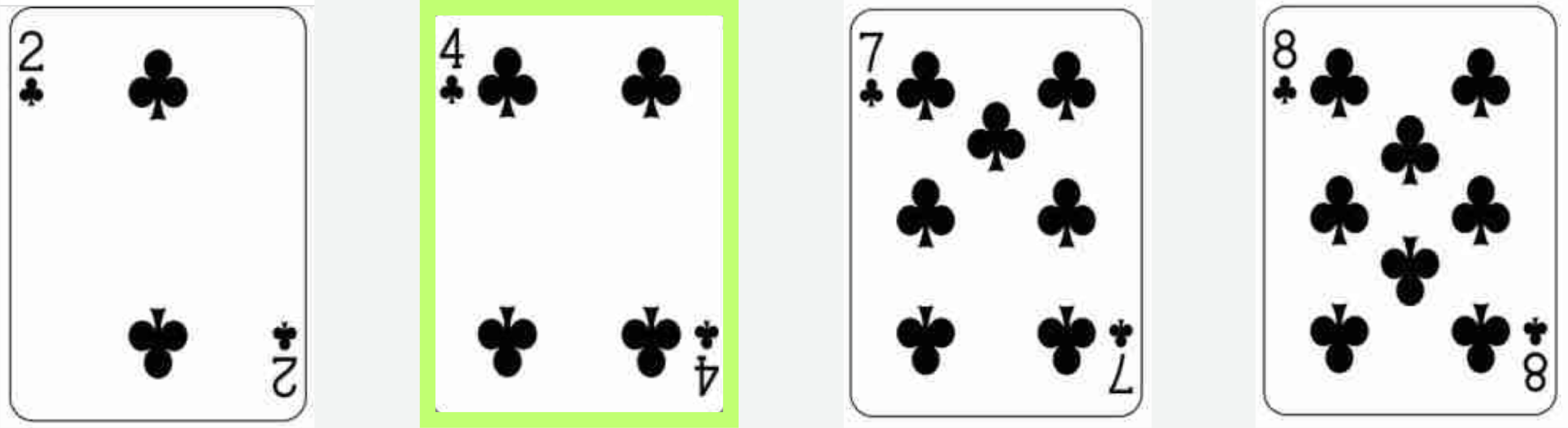


Posición actual

Posición siguiente

2º ITERACIÓN

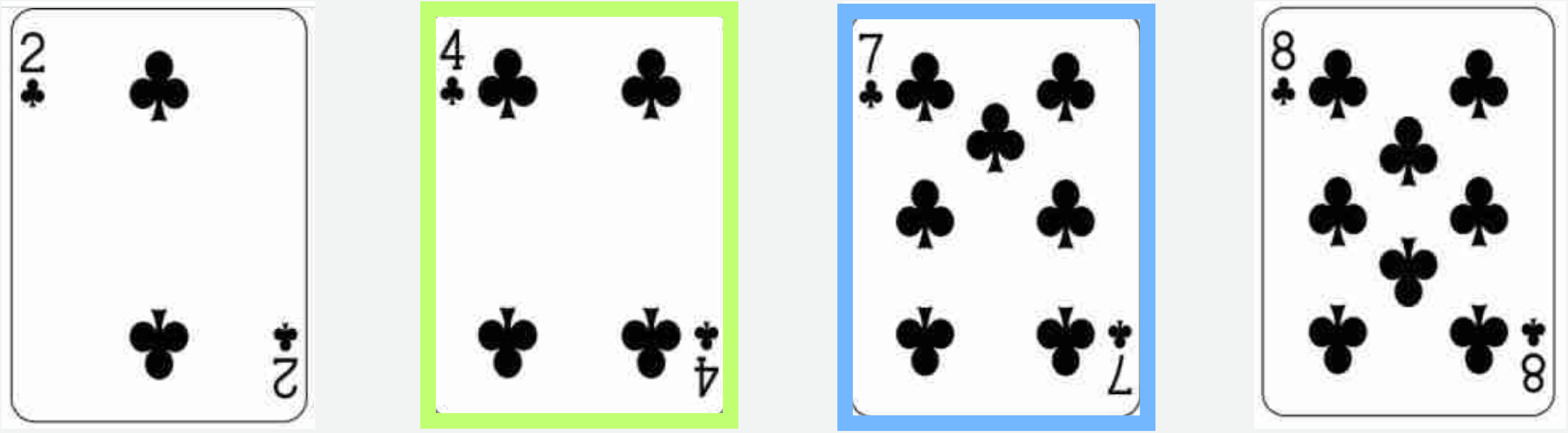
Intercambios = 0



Posición actual

2º ITERACIÓN

Intercambios = 0

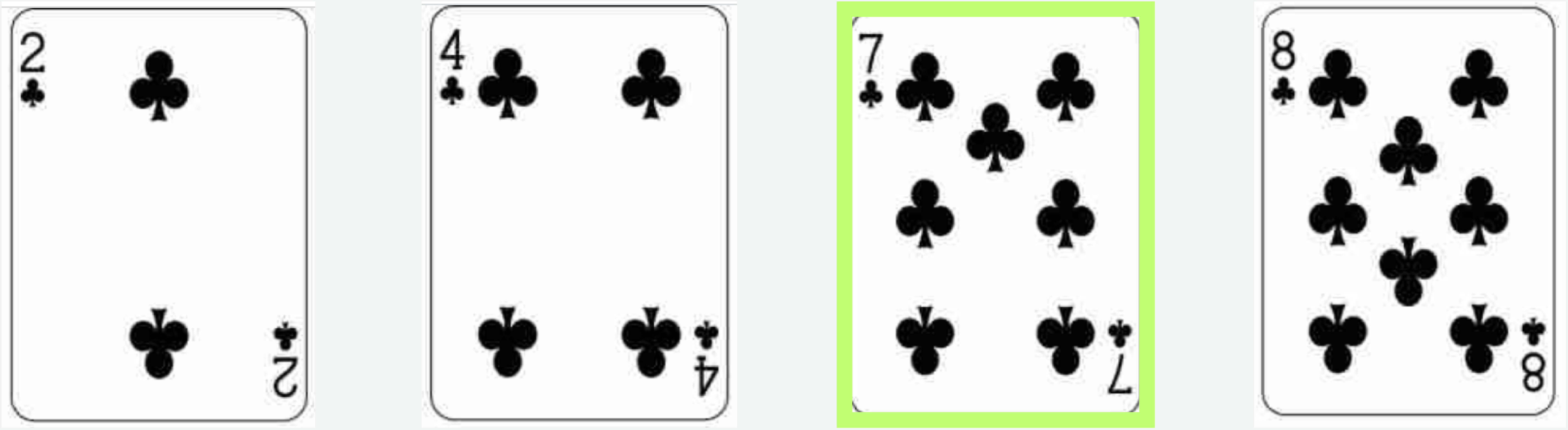


Posición actual

Posición siguiente

2º ITERACIÓN

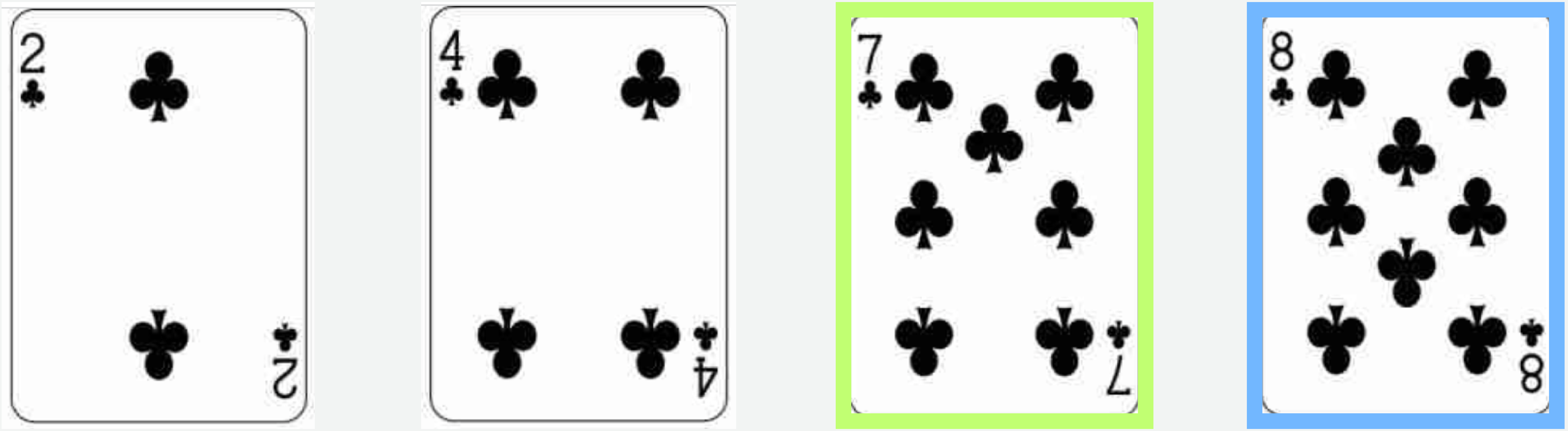
Intercambios = 0



Posición actual

2º ITERACIÓN

Intercambios = 0

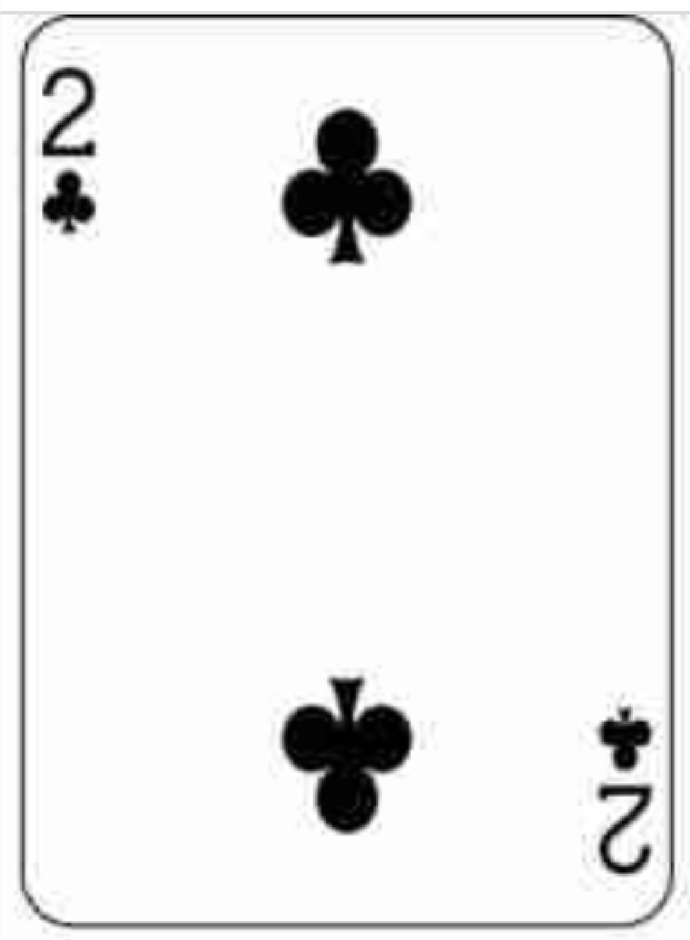
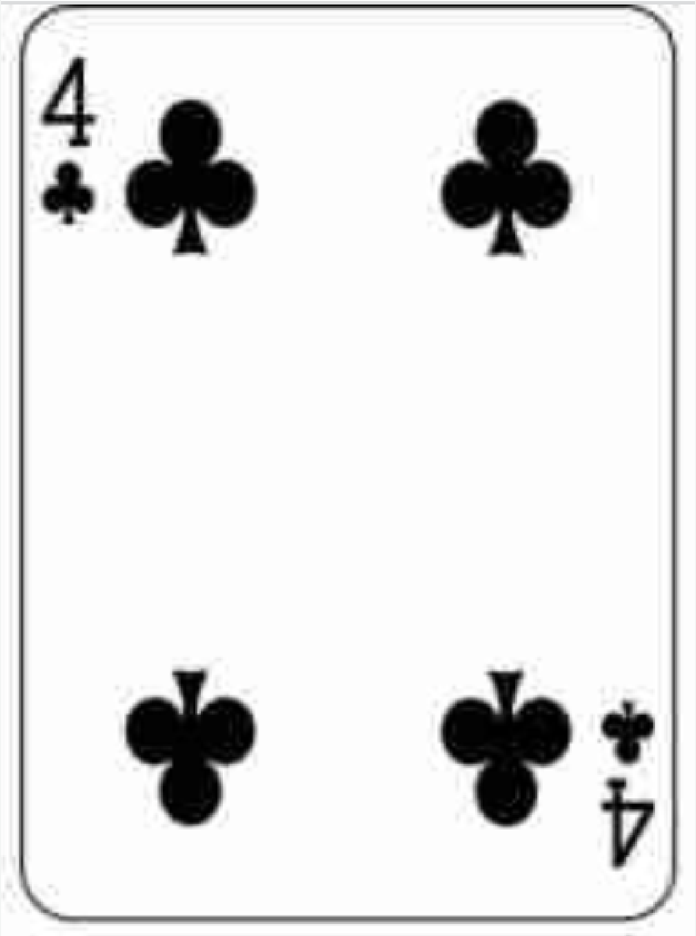
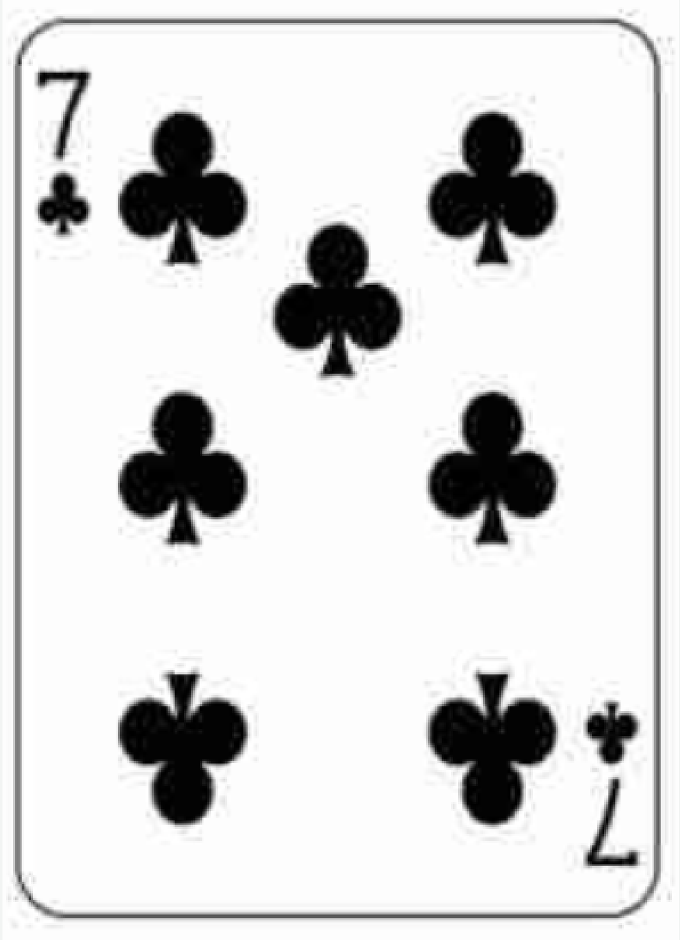
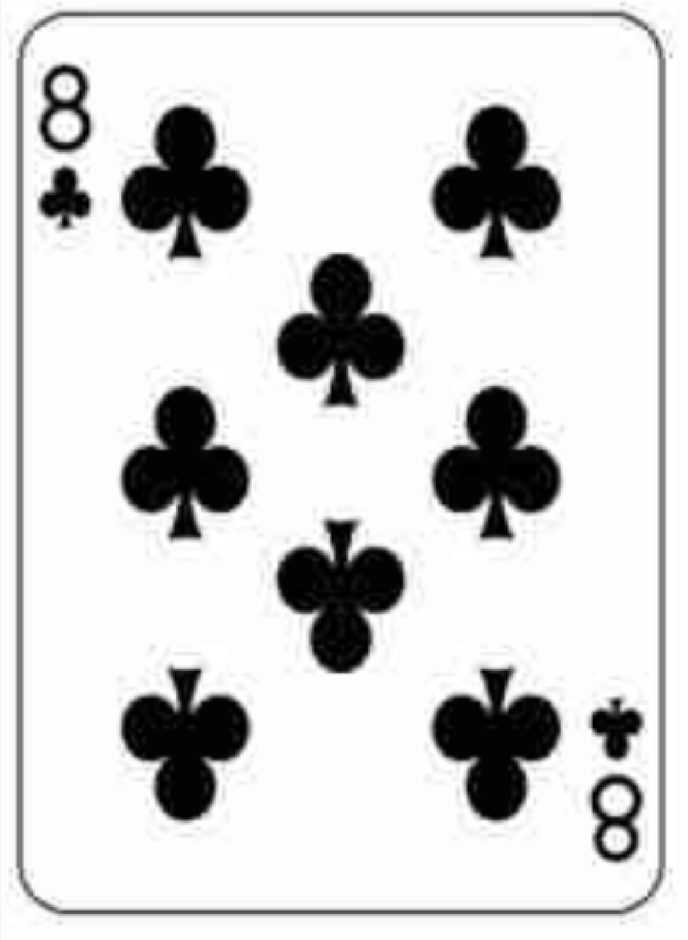


Posición actual

Posición siguiente

2º ITERACIÓN

Intercambios = 0

EJEMPLO: BUBBLE SORT

Inducción:

1. Base inductiva: arreglo de largo 1:
   a. Como tiene un solo elemento, ya está ordenado
2. Hipótesis inductiva: Bubble Sort ordena todo arreglo de largo n
3. Tesis inductiva: Bubble Sort ordena todo arreglo de largo n+1

EJEMPLO: BUBBLE SORT

Tesis inductiva: Bubble Sort ordena todo arreglo de largo n+1

- Observamos que al finalizar la 1° iteración, el máximo quedará al final del arreglo y se mantendrá en esa posición dado que nunca se va a cumplir la condición de intercambio con un elemento anterior

EJEMPLO: BUBBLE SORT

Tesis inductiva: Bubble Sort ordena todo arreglo de largo n+1

- Tomando A un arreglo de largo n+1 con max(A) el máximo del arreglo A, y A' el mismo arreglo pero sacando max(A):
  - BS(A) = BS(A') + [max(A)]
- Como A' es de largo n, por HI, BS(A') entregará un arreglo ordenado
- Al agregarle max(A) al final, se mantiene el orden
- BS(A) entrega un arreglo ordenado

EJEMPLO: BUBBLE SORT

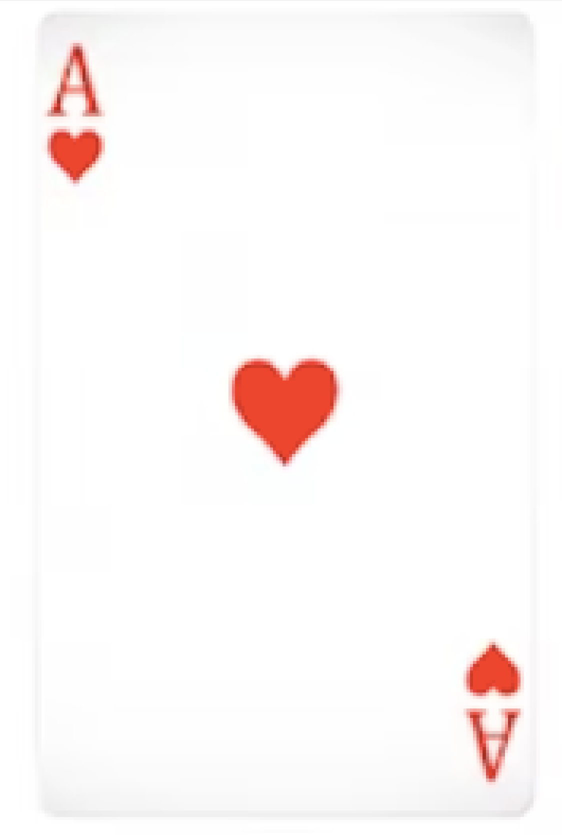
- Bubble Sort termina si no se producen intercambios en una iteración. Eso ocurre cuando el arreglo está ordenado
- Como demostramos que Bubble Sort ordena cualquier arreglo dado, siempre termina

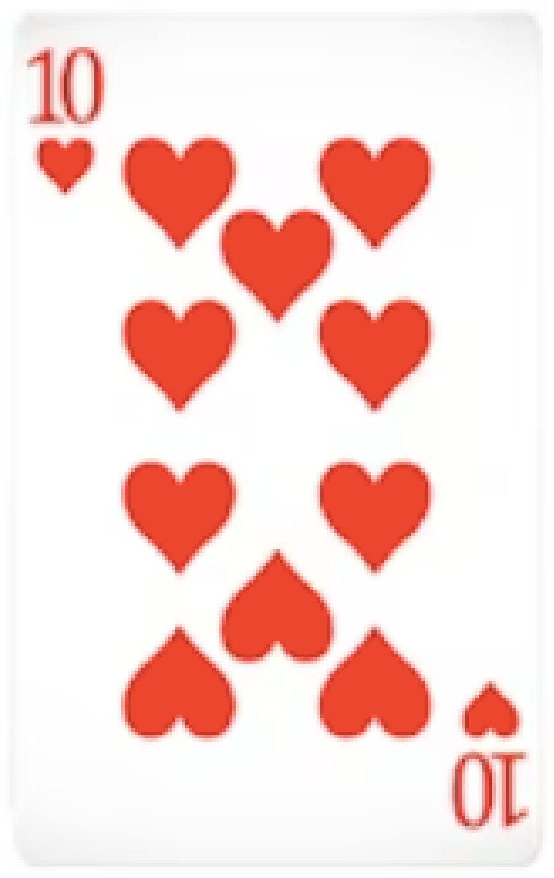
Intro a Sorting

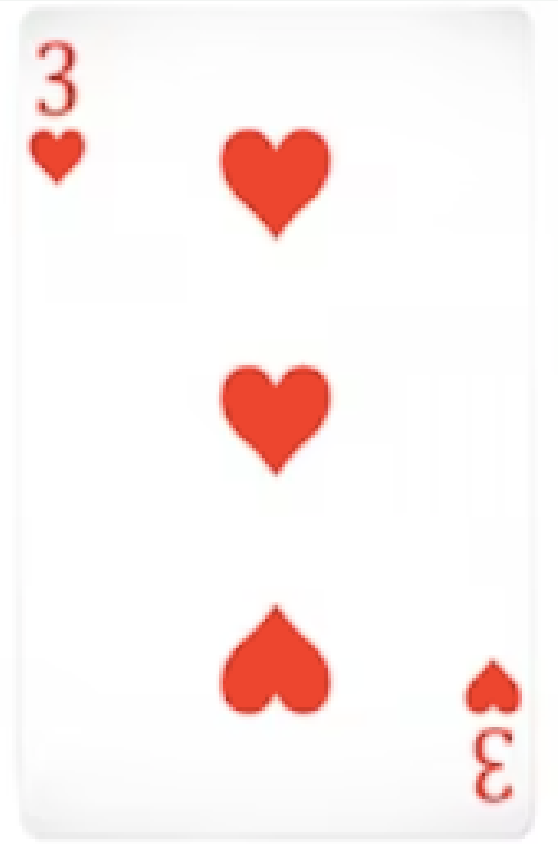
SELECTION SORT

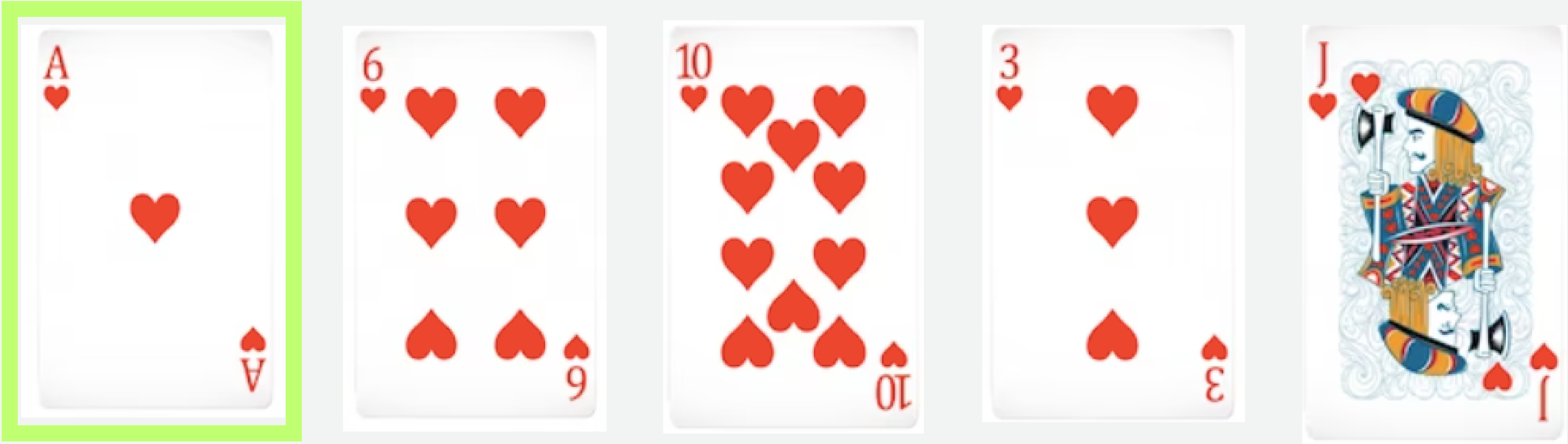
1. Tenemos una secuencia desordenada
2. Iniciar en posición 0
3. Buscar el menor dato 'x' en la secuencia
4. Intercambiar ese elemento 'x' con el elemento actual de la secuencia
5. Avanzar uno en la secuencia
6. Si aún queda secuencia, volver al paso 2

EJEMPLO



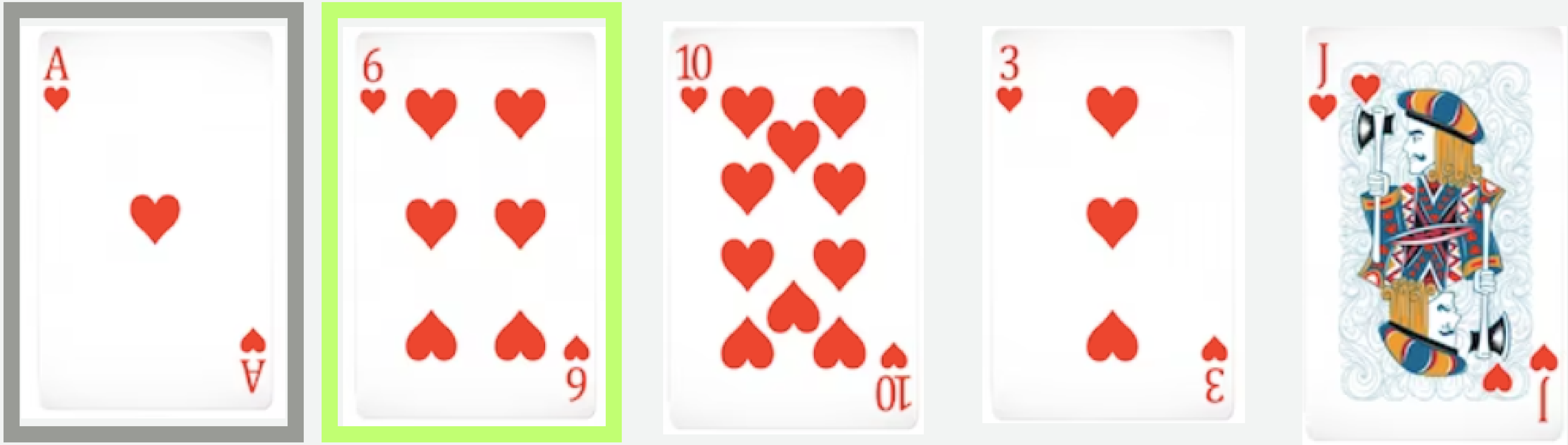






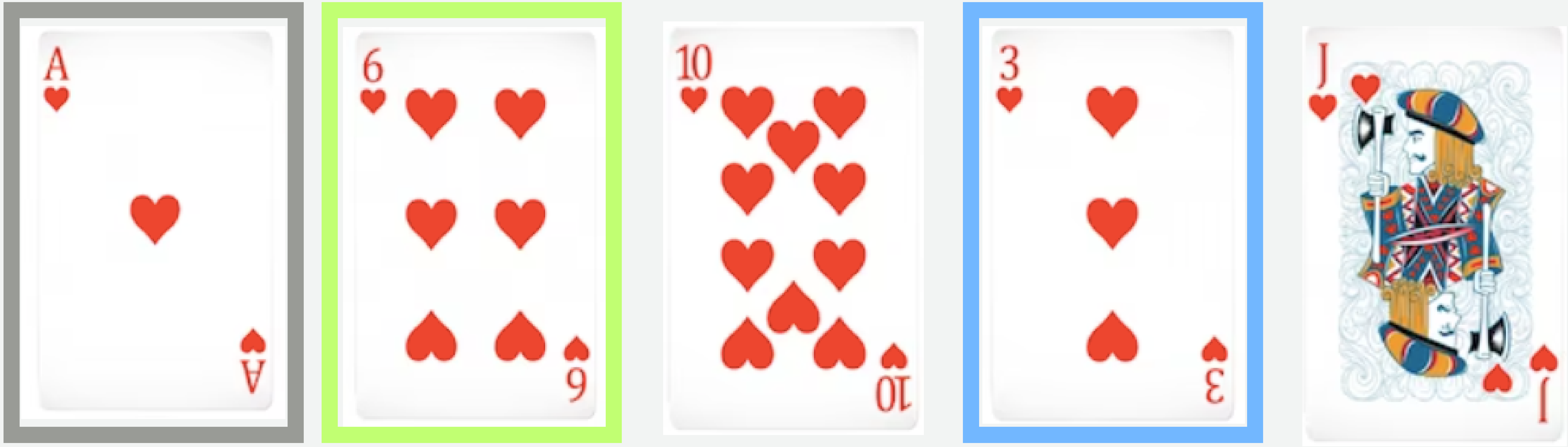
Posición actual y
menor elemento

Como es el menor elemento no necesita ser ordenado



Elemento ordenado

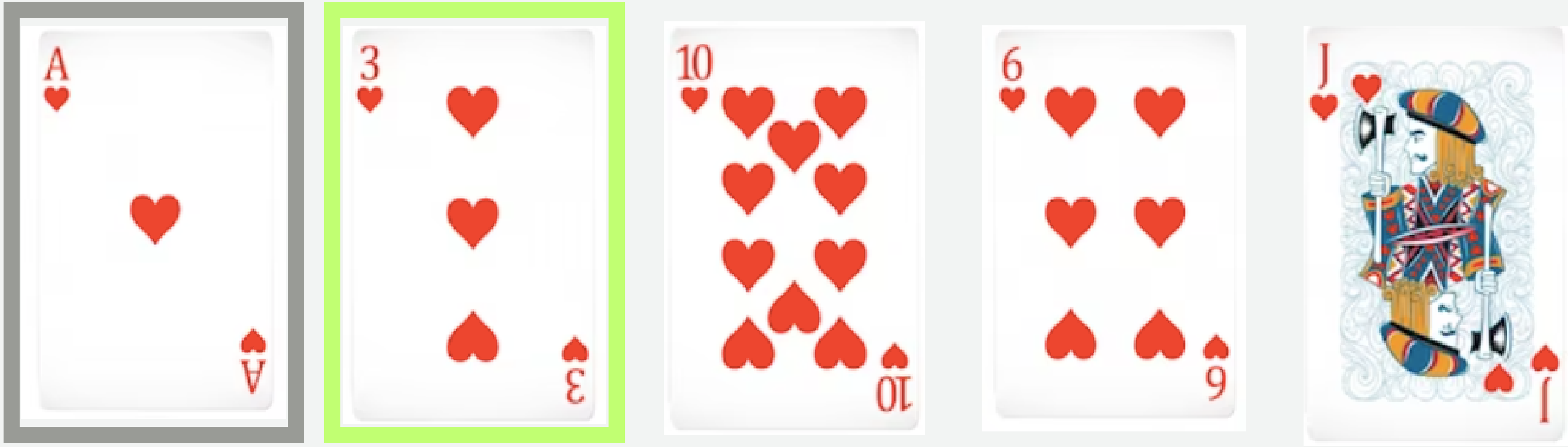
Posición actual



Elemento ordenado

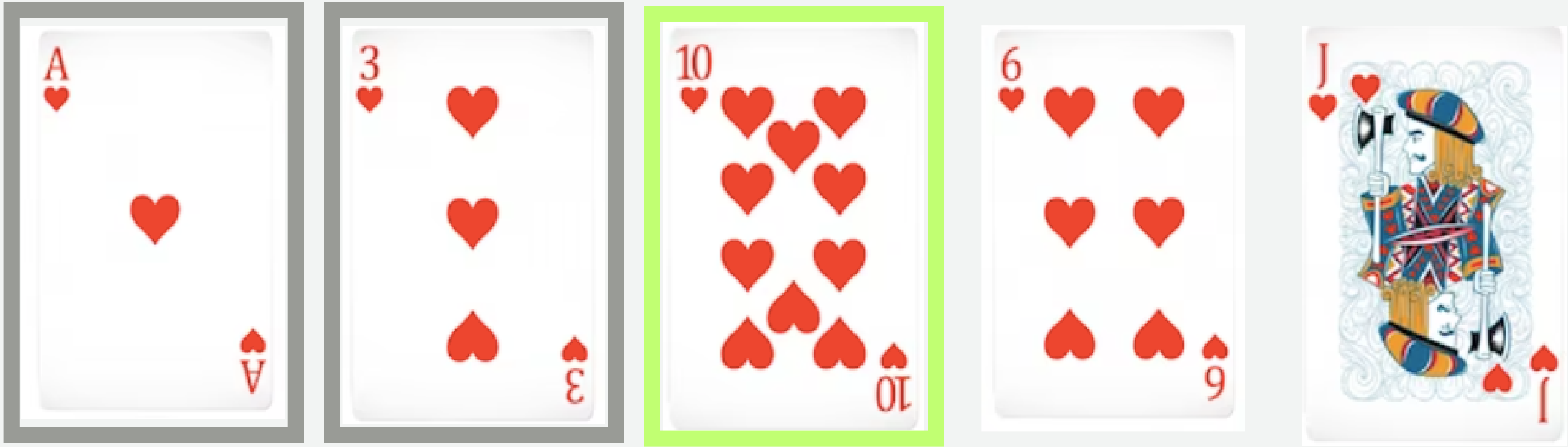
Menor elemento

Posición actual



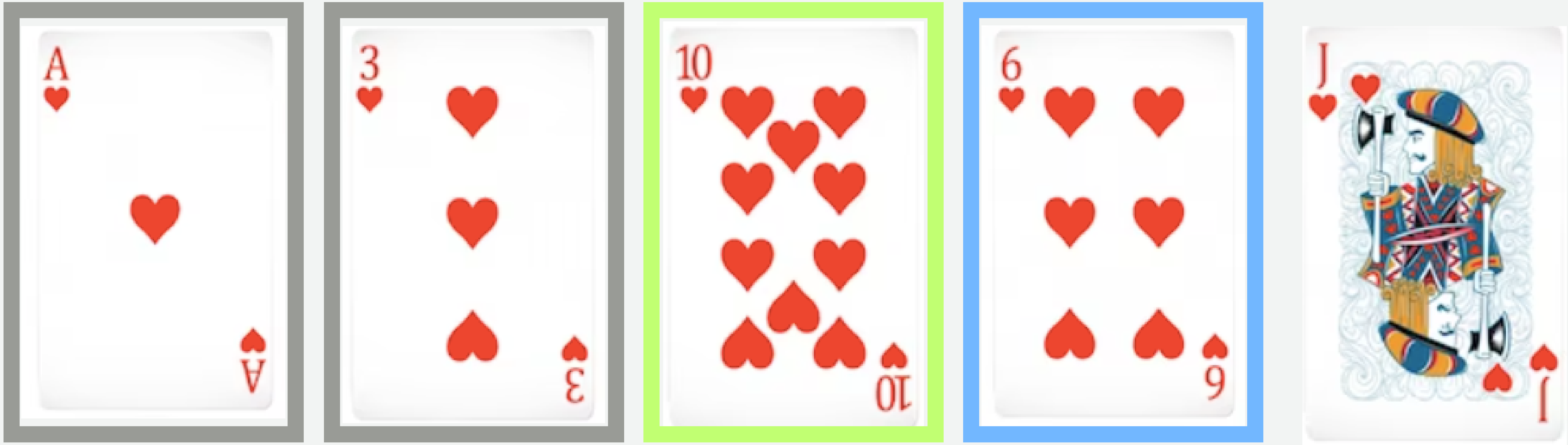
Elemento ordenado

Posición actual



Elemento ordenado

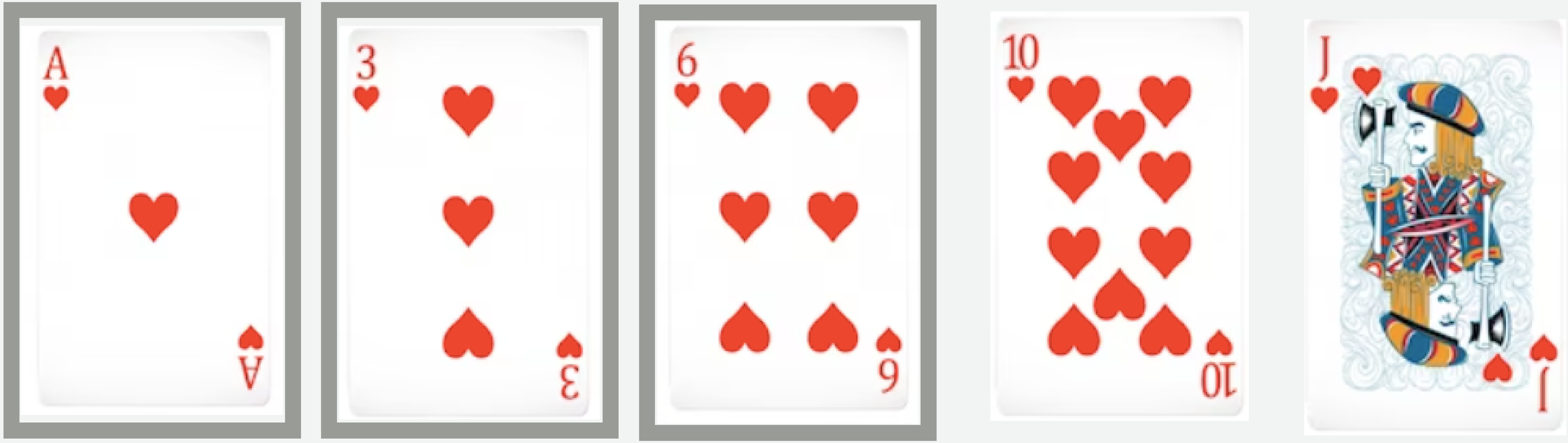
Posición actual



Elemento ordenado

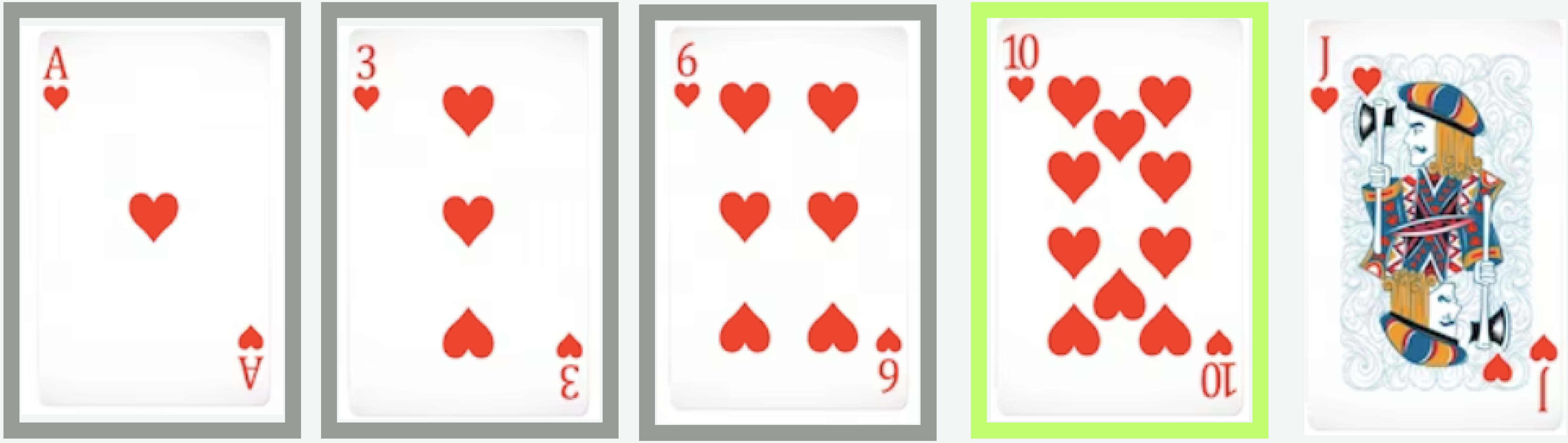
Menor elemento

Posición actual



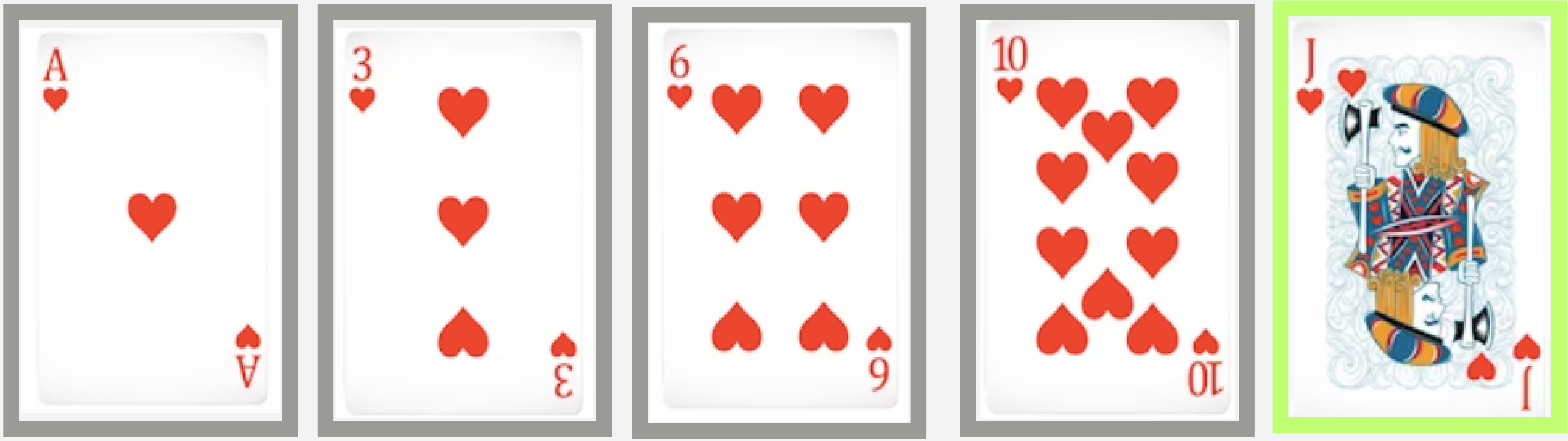
Elemento ordenado

Posición actual



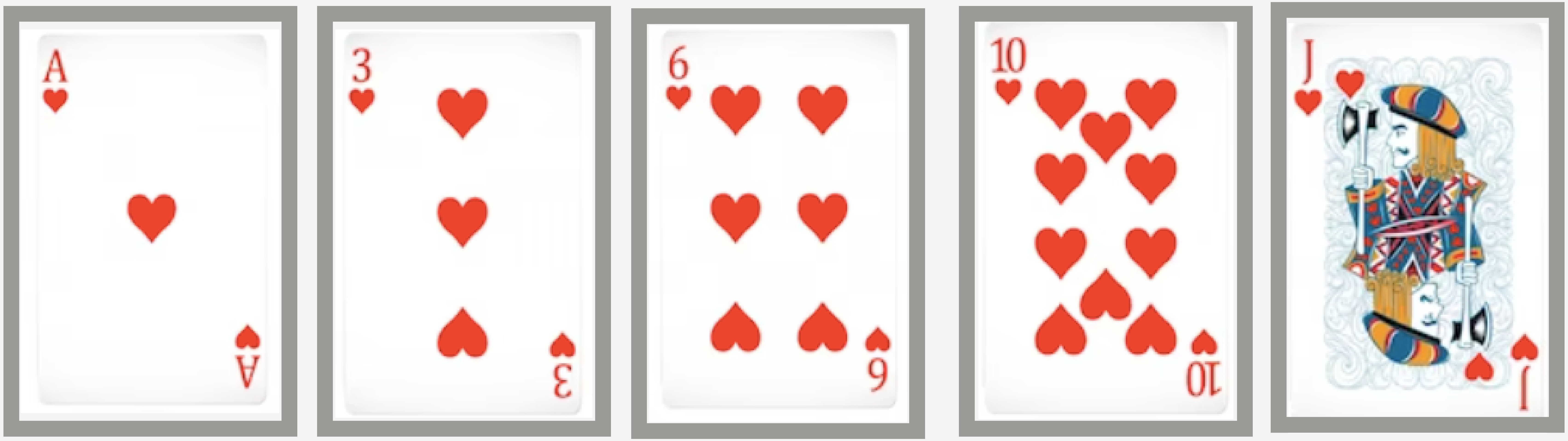
Elemento ordenado

Posición actual



Elemento ordenado

Posición actual

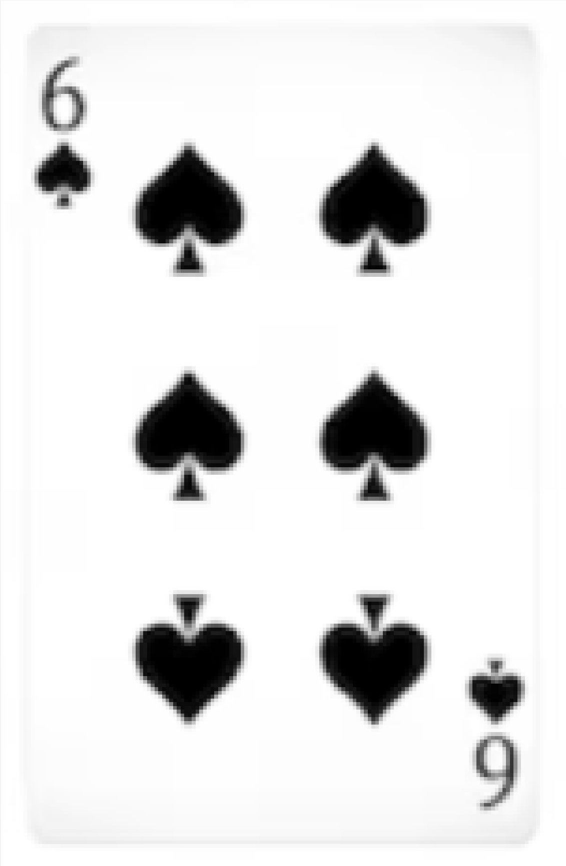


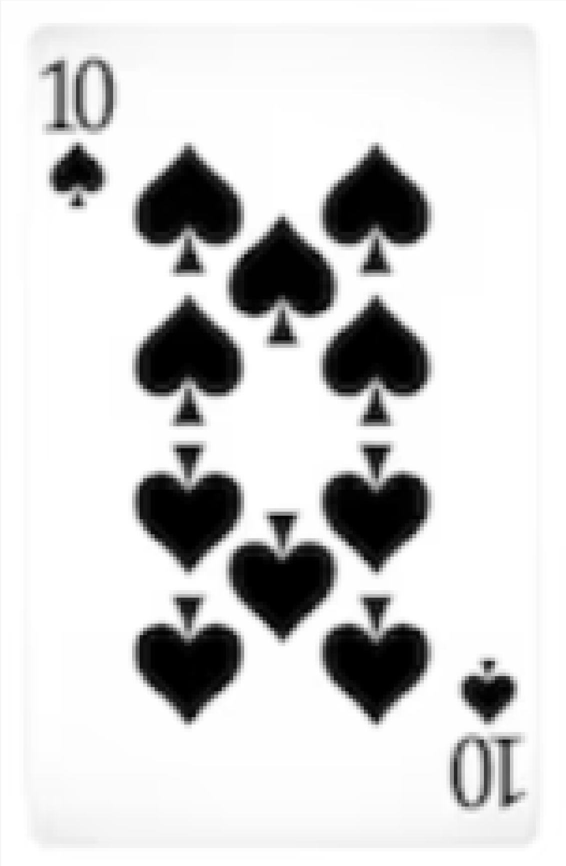
Elemento ordenado

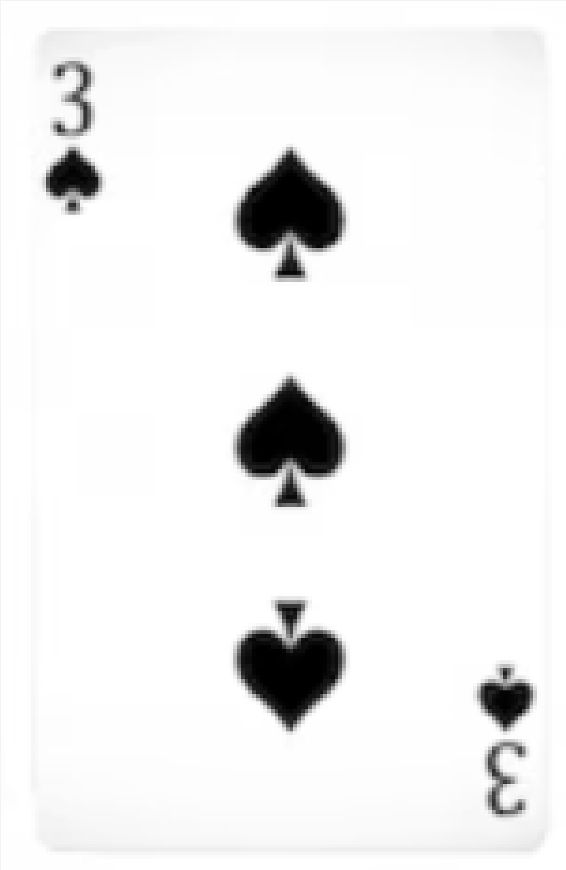
INSERTION SORT

1. Tenemos una secuencia desordenada
2. Tomar el primer dato 'x' de la secuencia
3. Insertar 'x' en los elementos anteriores de manera que quede ordenado
4. Avanzar en la secuencia
5. Si aún queda secuencia, volver al paso 2

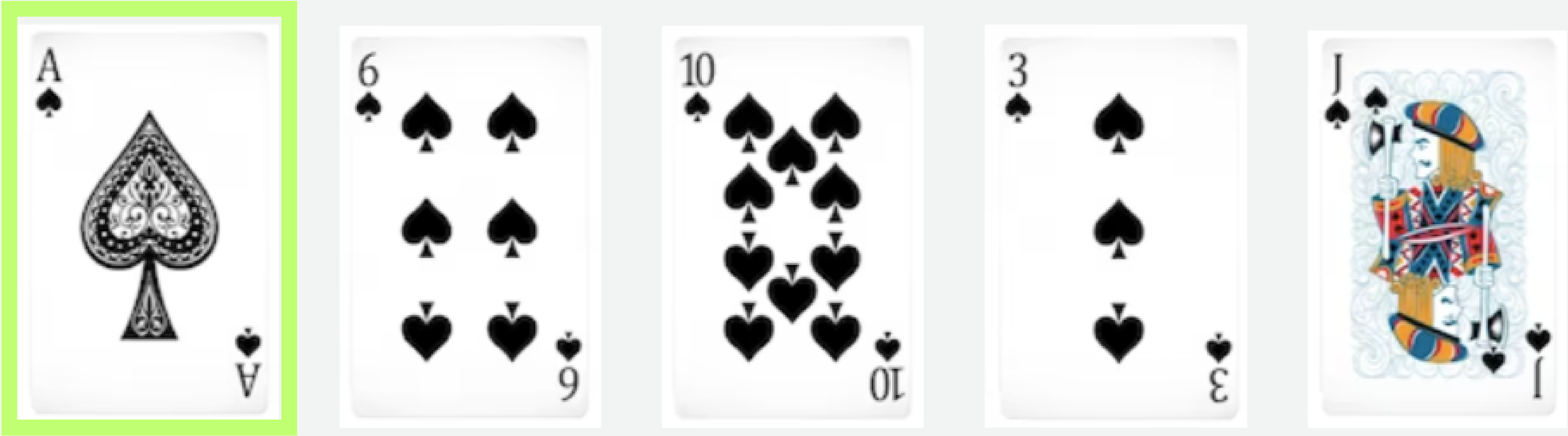
EJEMPLO





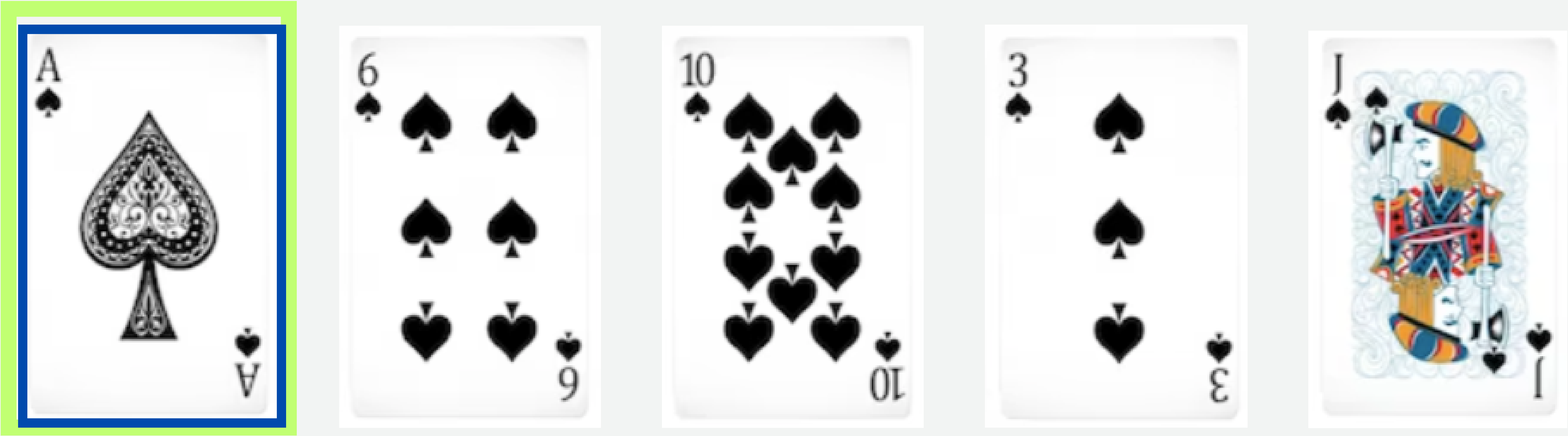


Nos ubicamos en el comienzo del arreglo



Posición actual

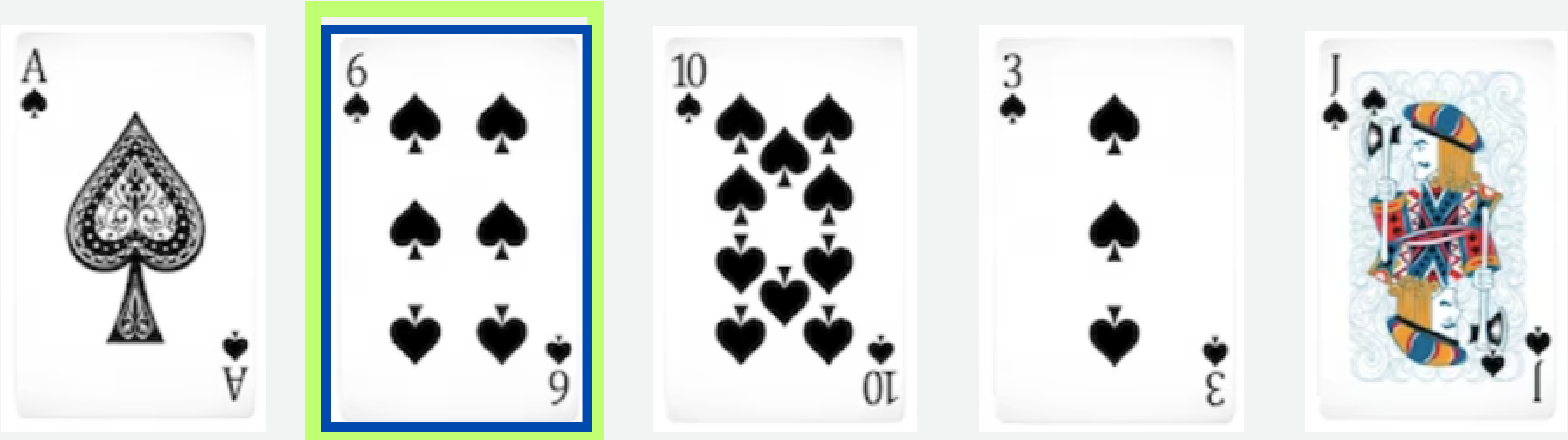
Nos ubicamos en el comienzo del arreglo



Posición actual

Elemento actual

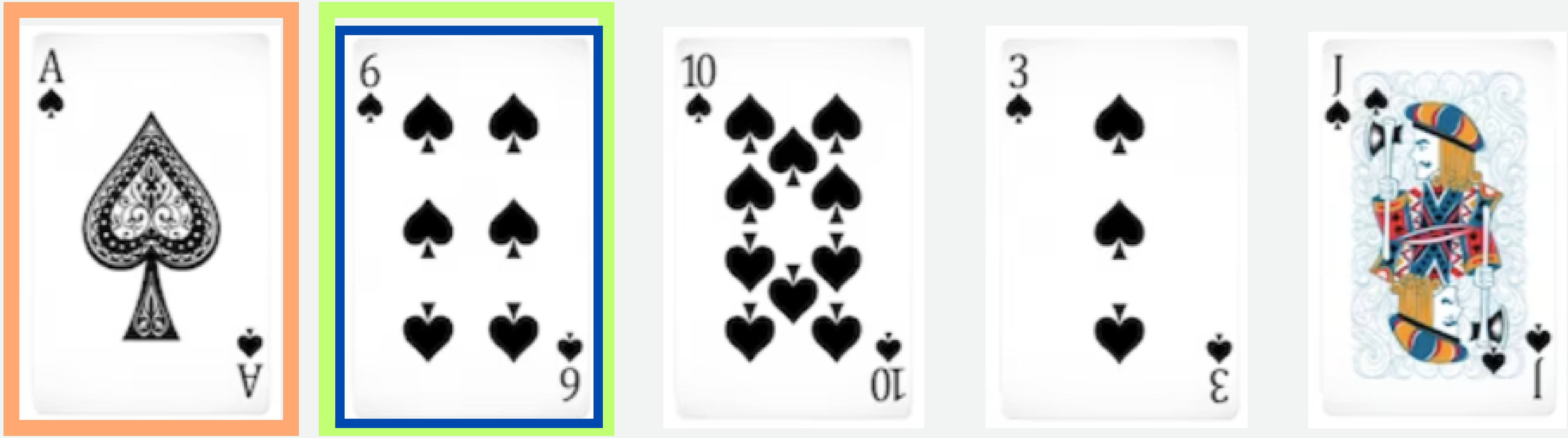
Como no hay elemento anteriores, avanzamos



Posición actual

Elemento actual

Observamos elementos anteriores

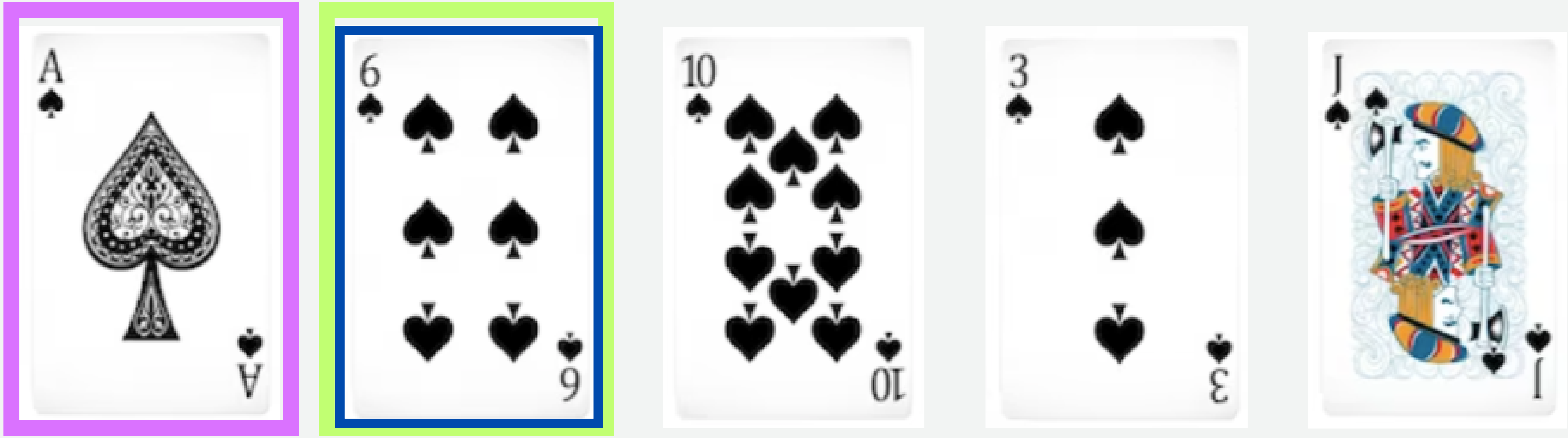


Posición actual

Elemento actual

Elementos anteriores

Comparamos con los anteriores e insertamos ordenadamente

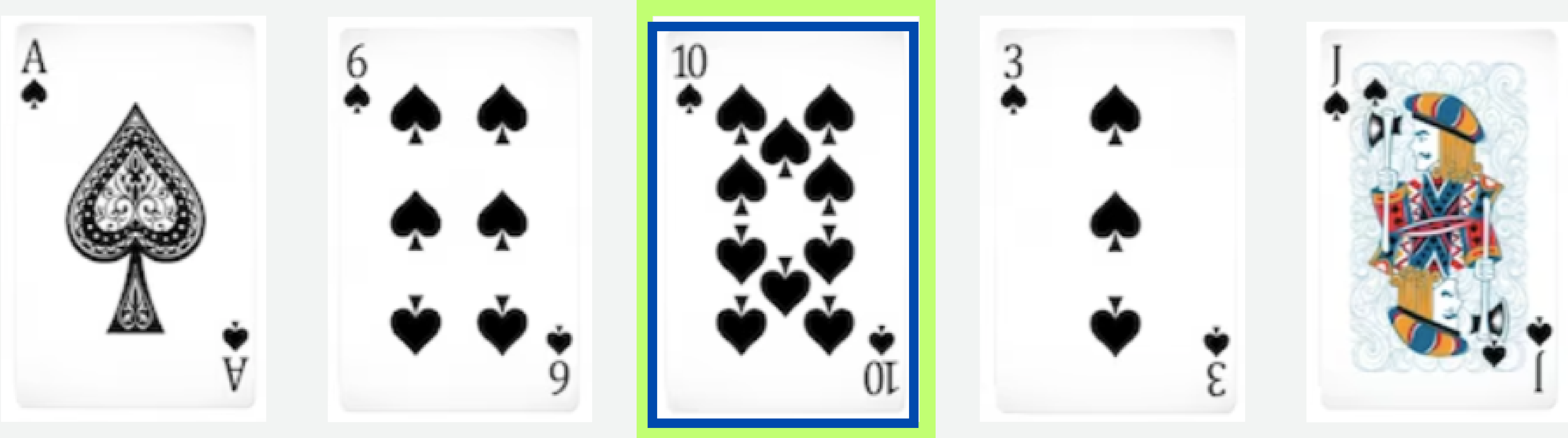


Posición actual

Elemento actual

Elemento a comparar

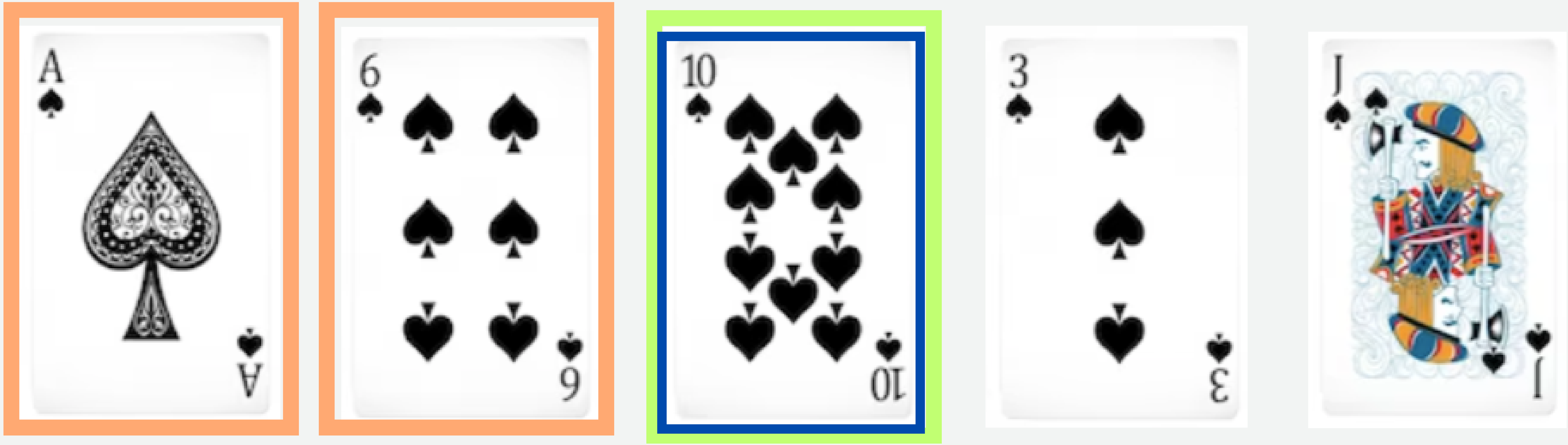
Como ya se encuentra ordenado, avanzamos



Posición actual

Elemento actual

Observamos elementos anteriores

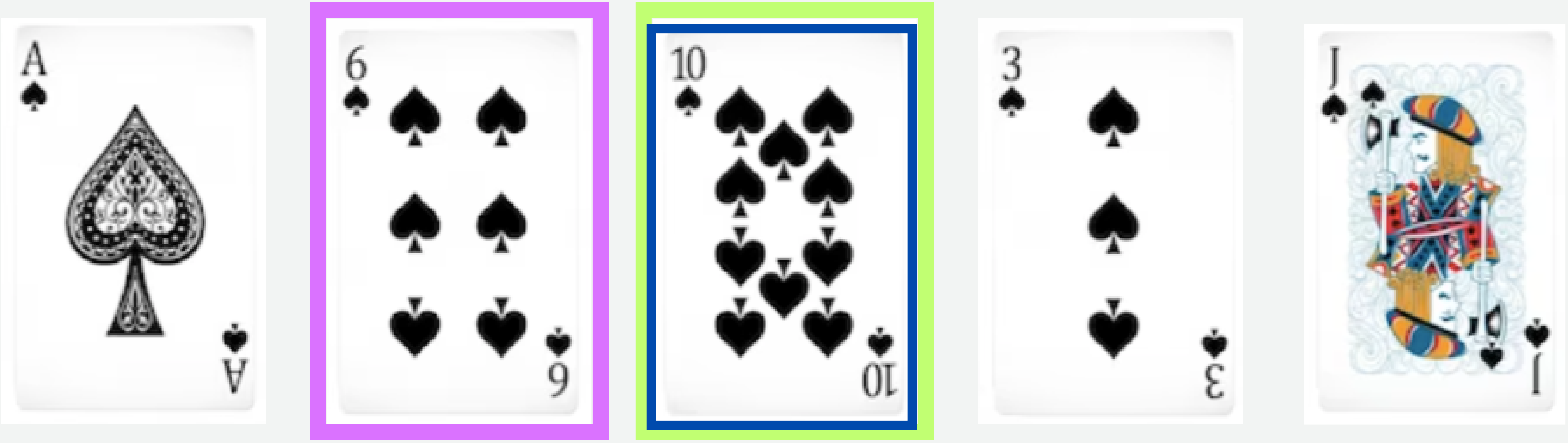


Posición actual

Elemento actual

Elementos anteriores

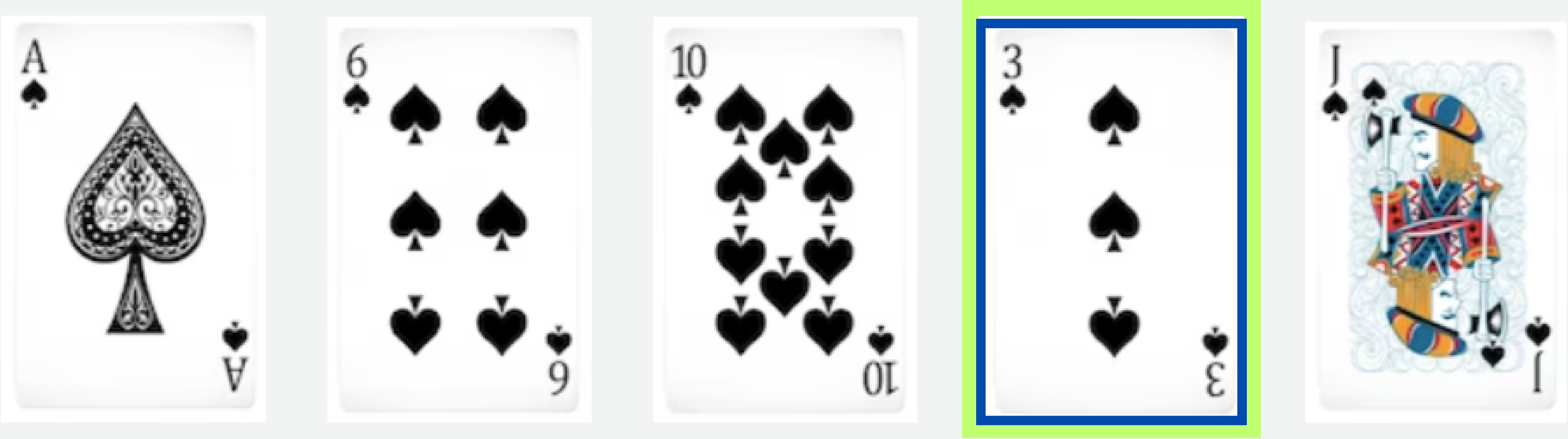
Comparamos con el elemento anterior, si se encuentra ordenado, dejamos de comparar y avanzamos



Posición actual

Elemento a comparar

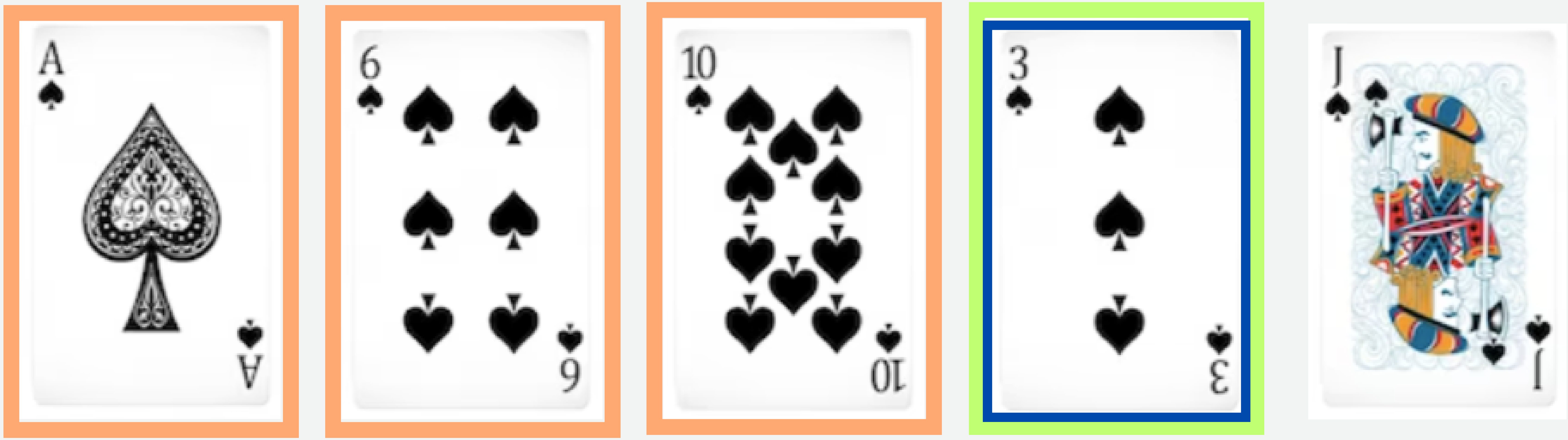
Avanzamos de posición



Posición actual

Elemento actual

Observamos elementos anteriores

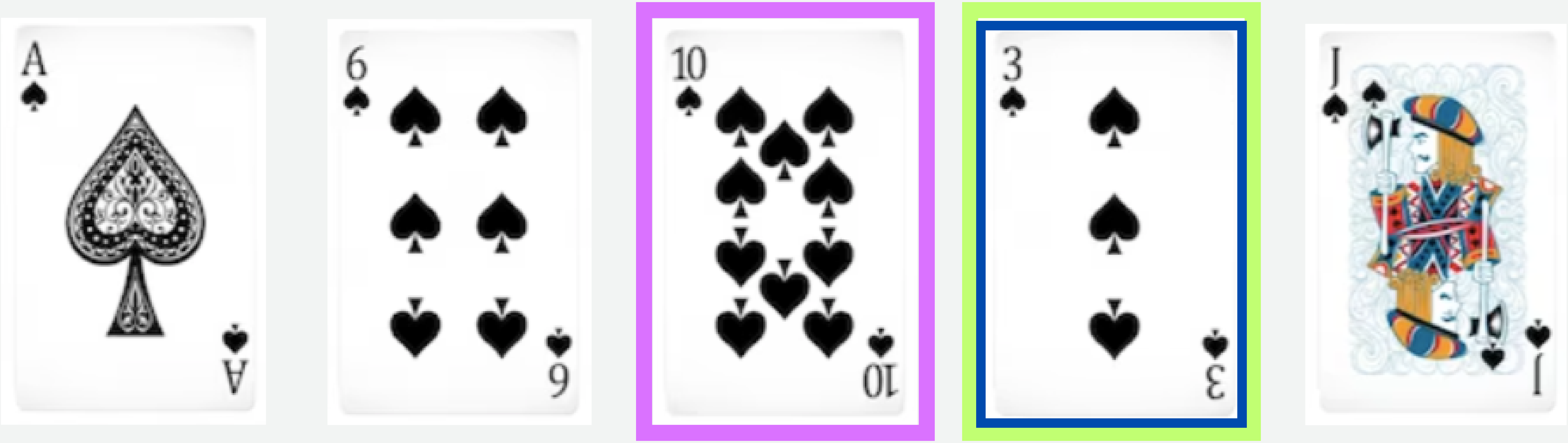


Posición actual

Elemento actual

Elementos anteriores

Comparamos siempre el elemento actual, que se puede insertar, la posición actual se mantiene hasta terminar de comparar

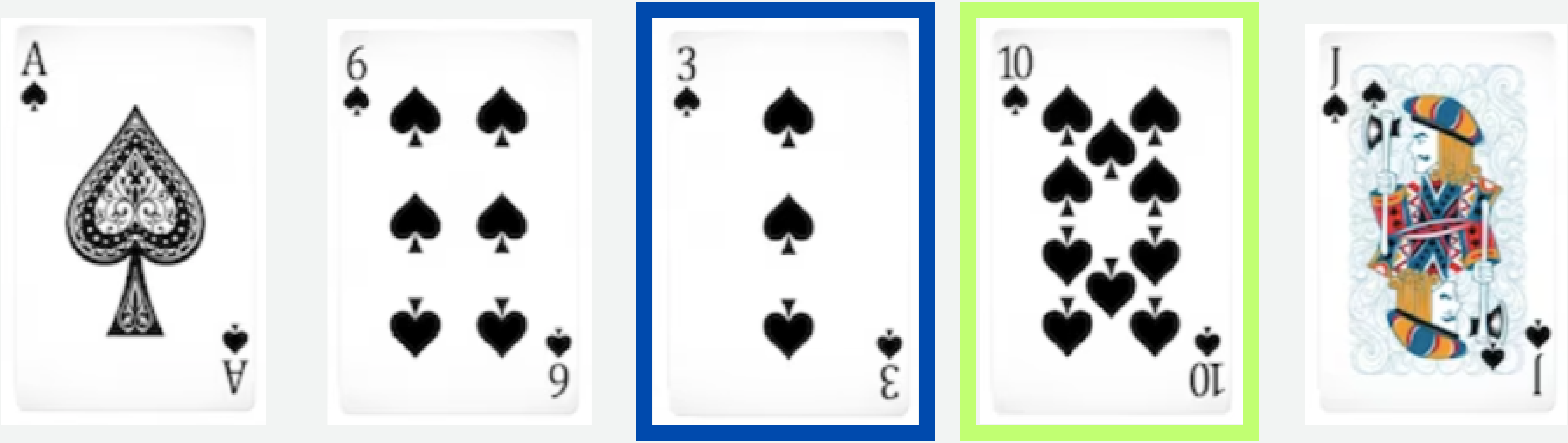


Posición actual

Elemento actual

Elemento a comparar

Comparamos uno por uno, si es necesario, insertamos
Cuando se ordena el elemento actual es cuando puede avanzar la posición

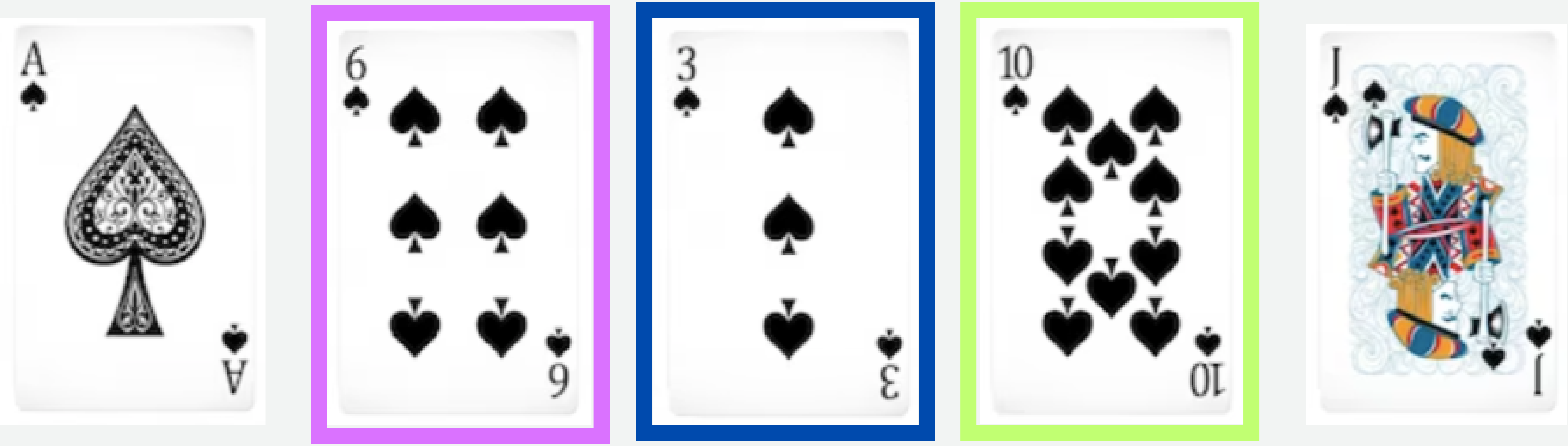


Posición actual

Elemento actual

Elemento a comparar

Comparamos uno por uno, si es necesario, insertamos

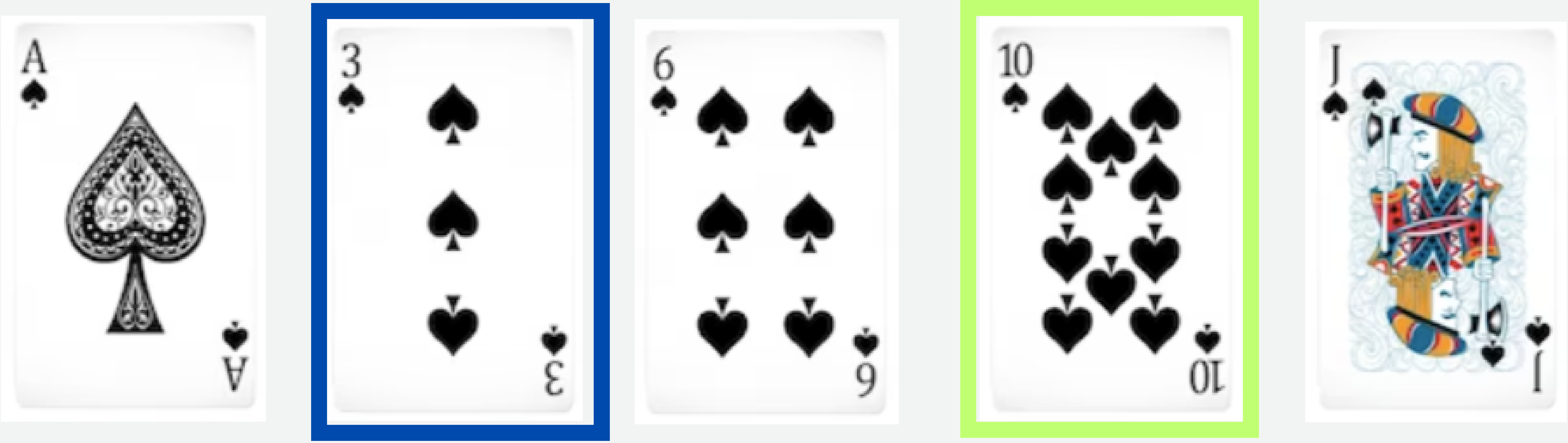


Posición actual

Elemento actual

Elemento a comparar

Comparamos uno por uno, si es necesario, insertamos



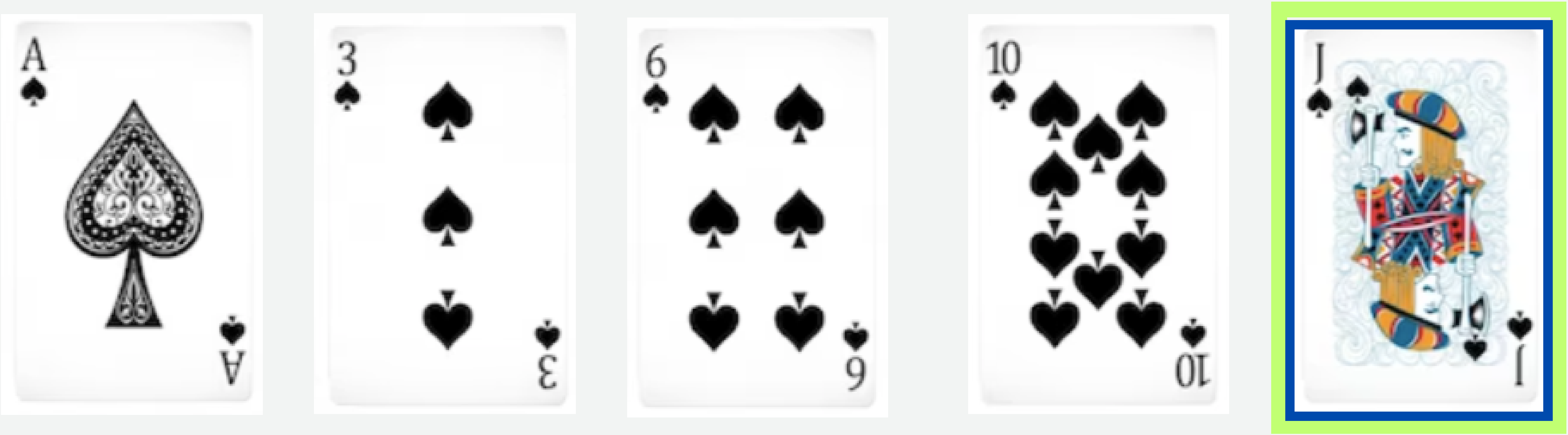
Posición actual

Elemento actual

Elemento a comparar

Comparamos uno por uno, si es necesario, insertamos

Como terminamos de comparar, avanzamos

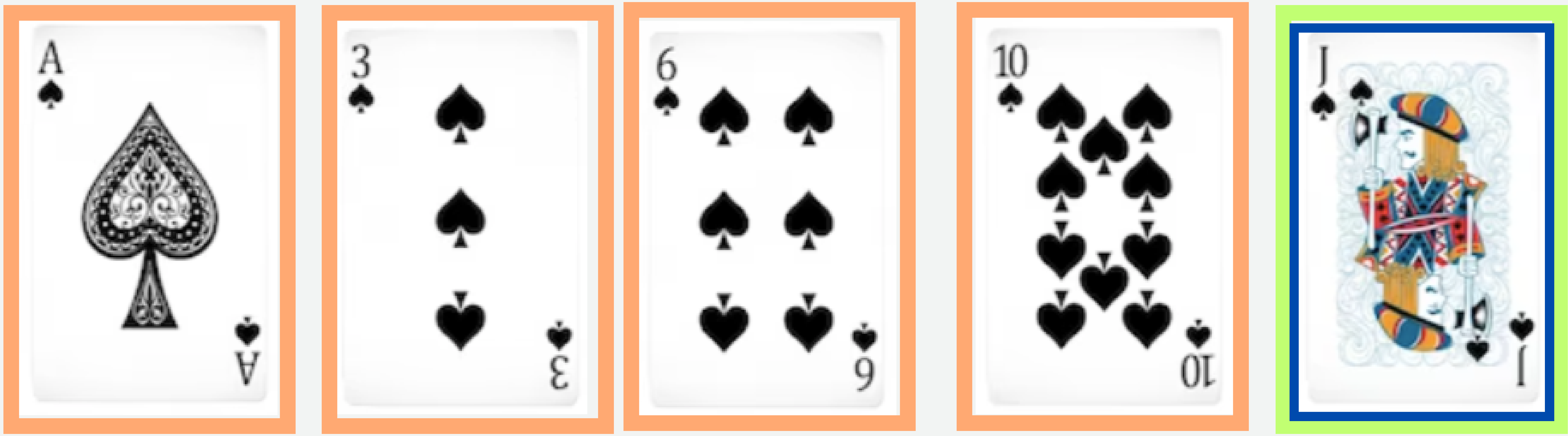


Posición actual

Elemento actual

Elemento a comparar

Como terminamos de comparar, avanzamos

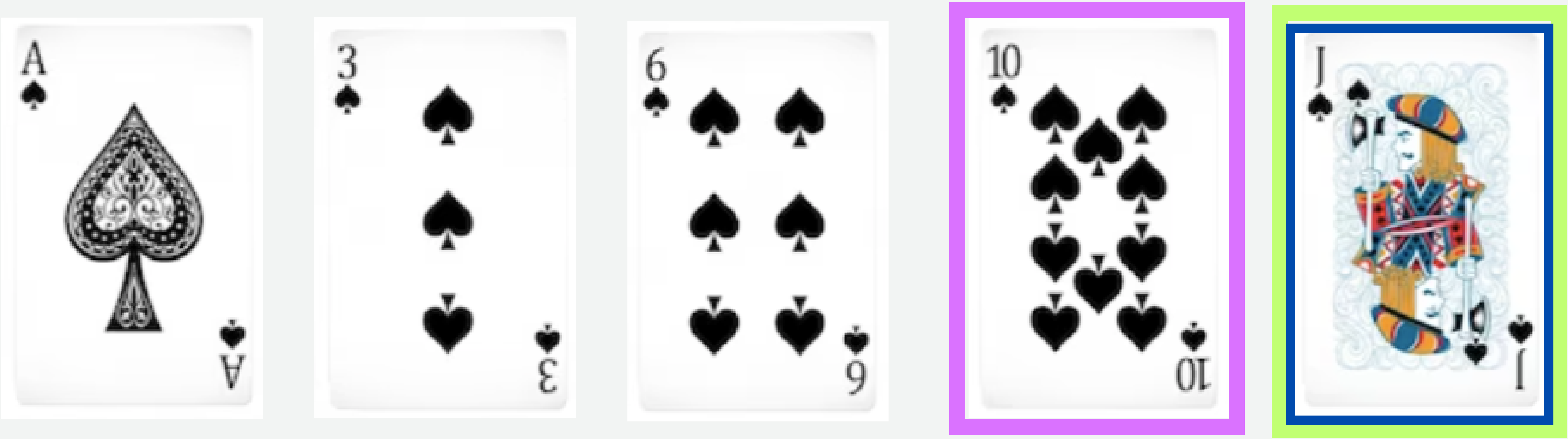


Posición actual

Elemento actual

Elementos anteriores

Si está ordenado, dejamos de comparar



Posición actual

Elemento actual

Elemento a comparar

Fianlmente, el arreglo queda ordenado