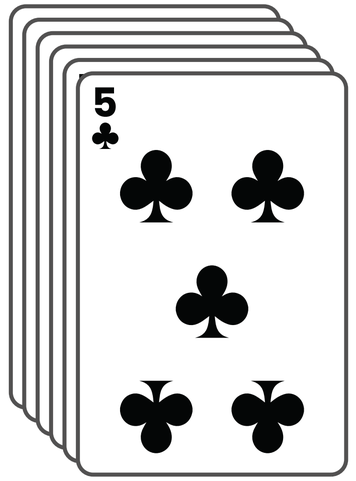
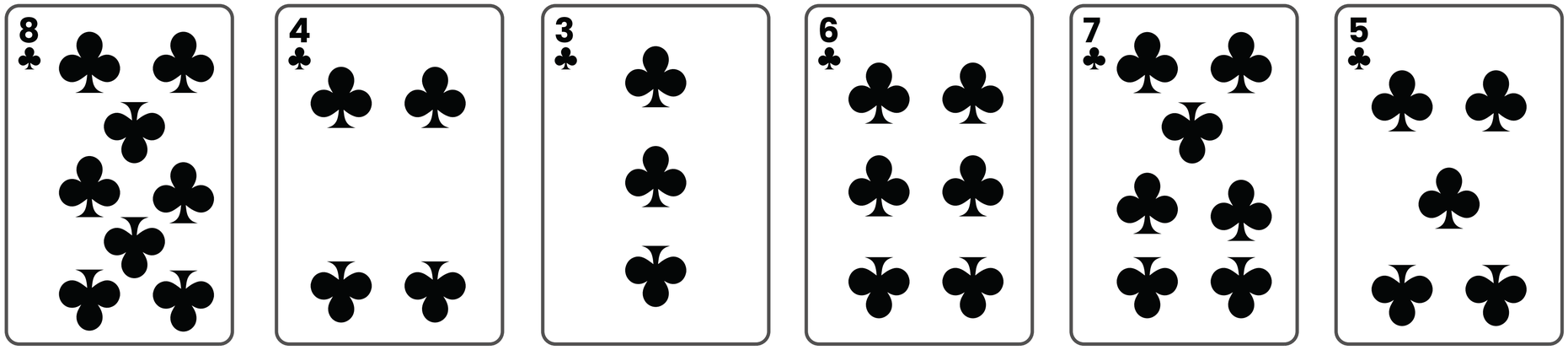
## Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)

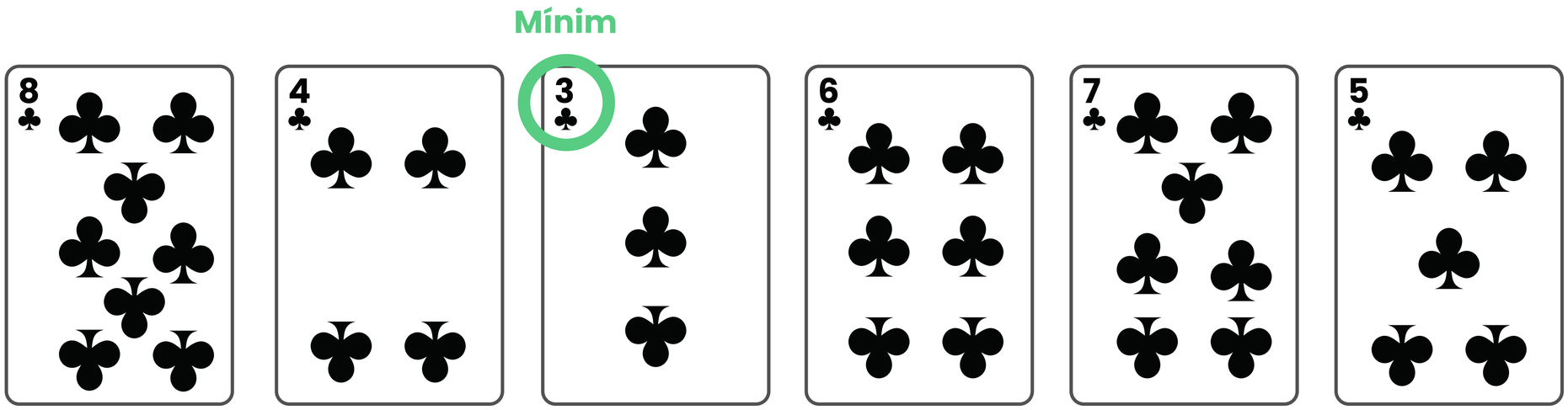
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



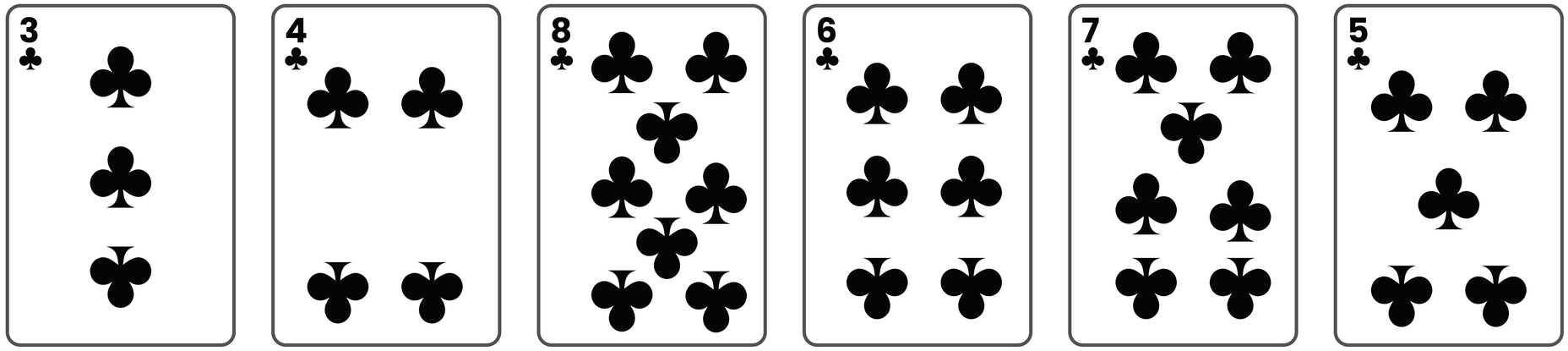
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



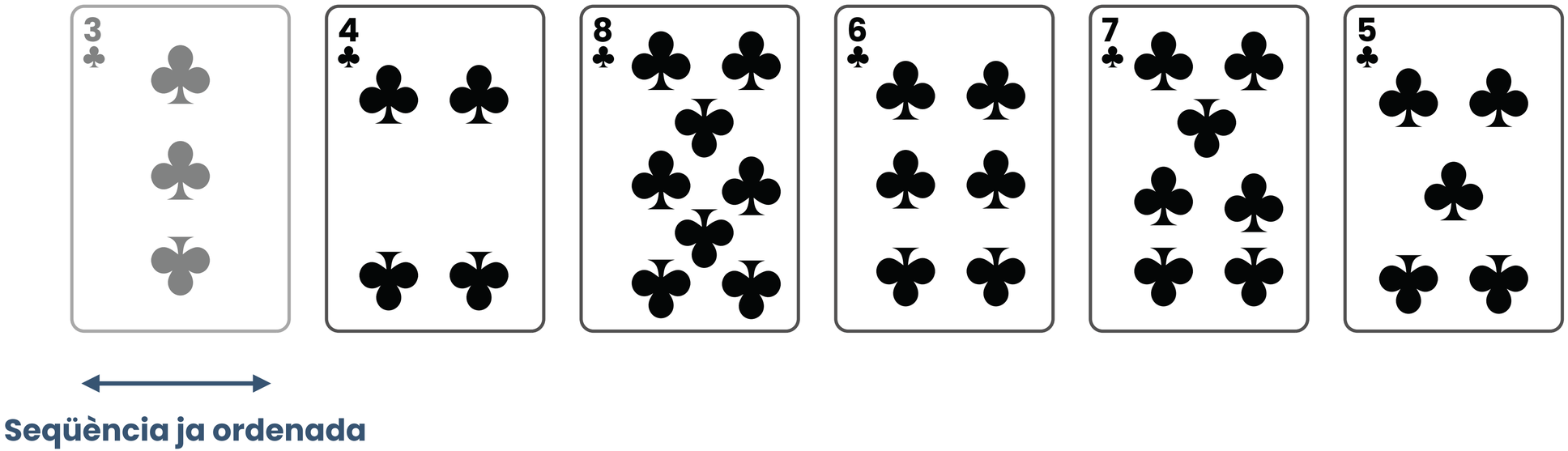
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



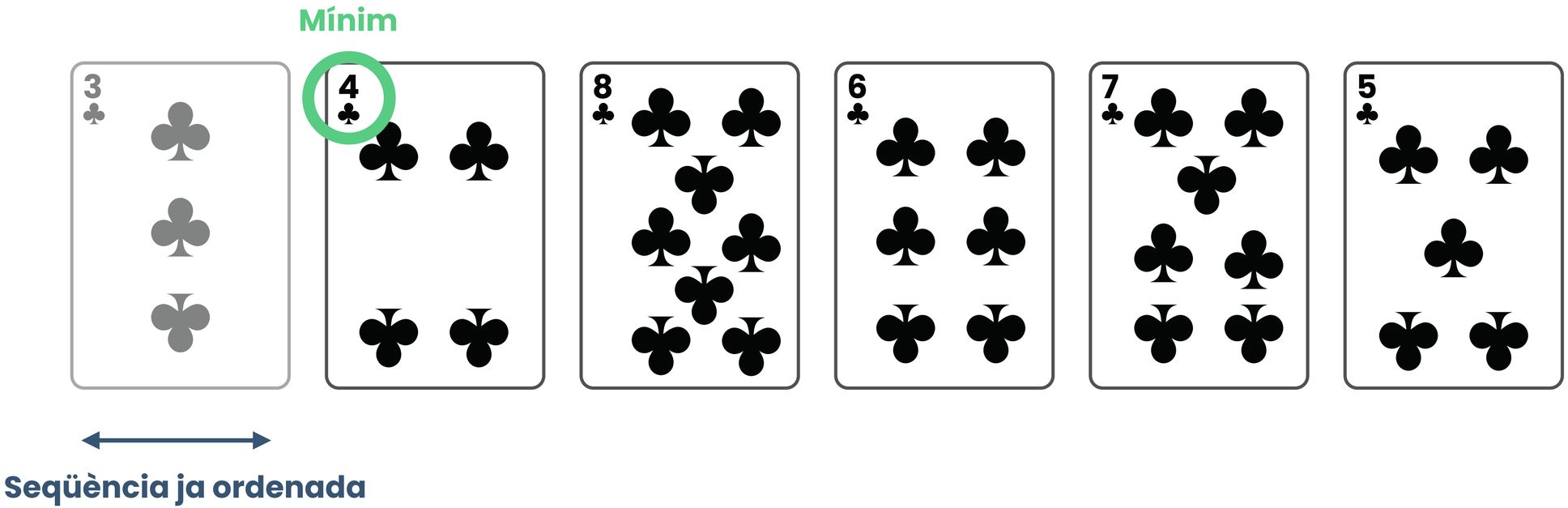
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



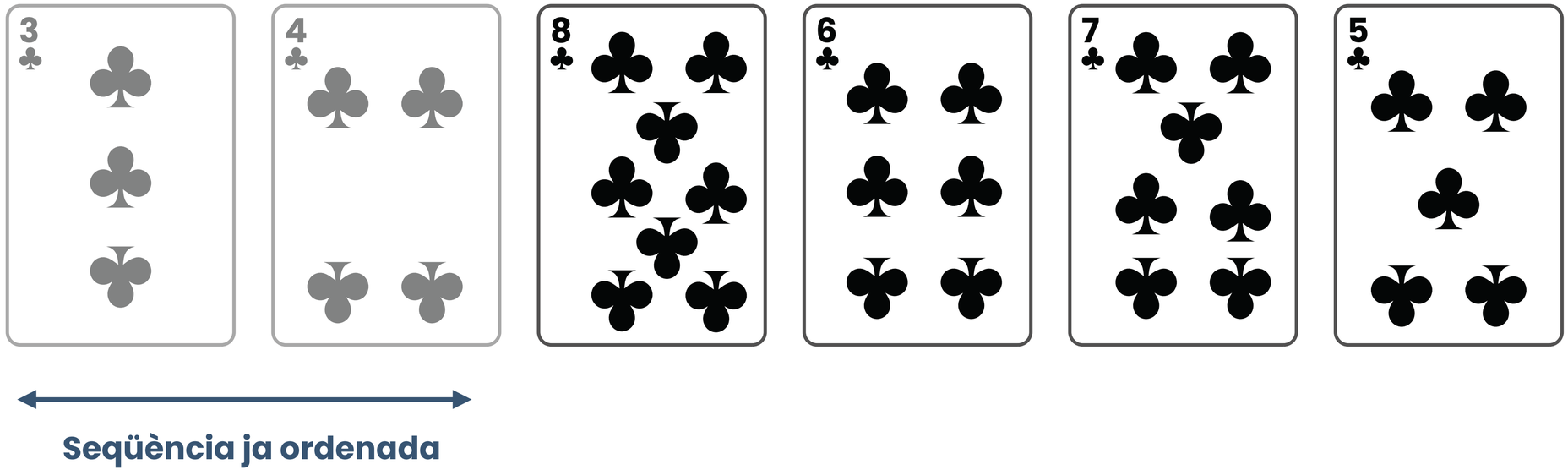
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



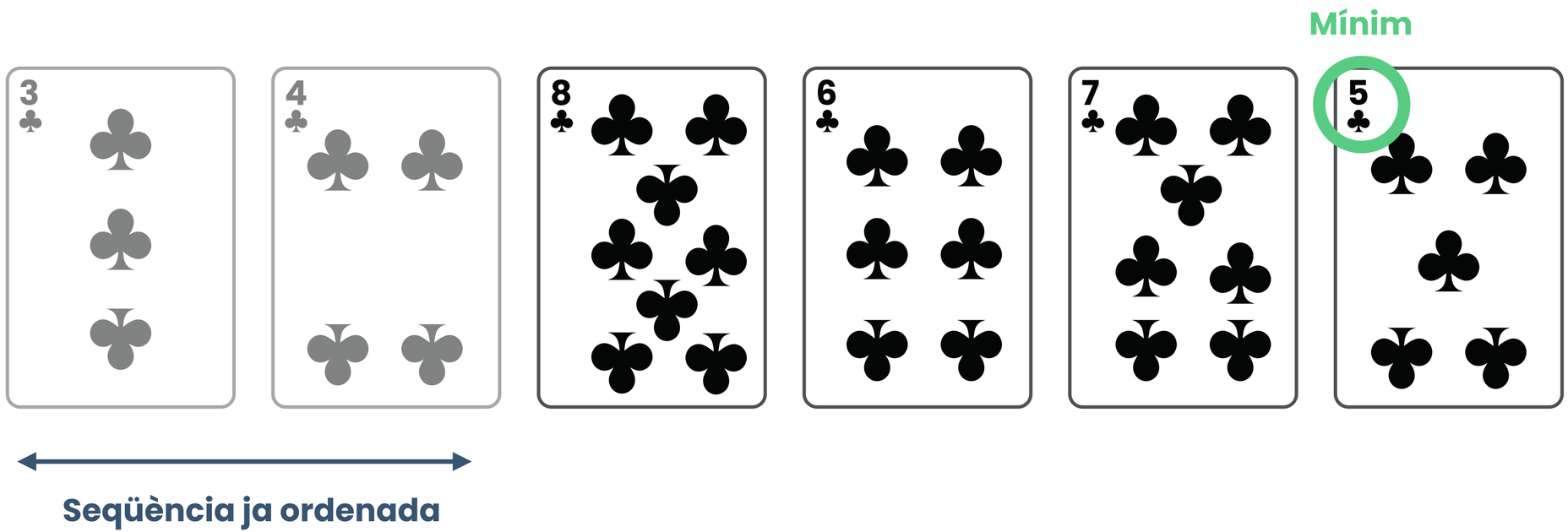
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



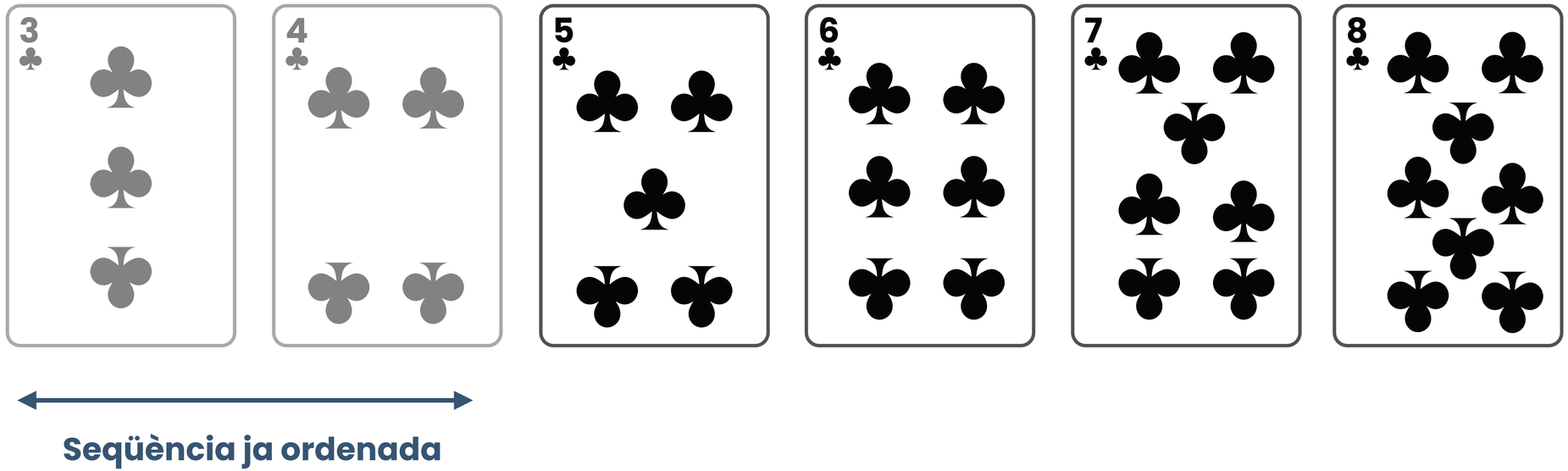
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



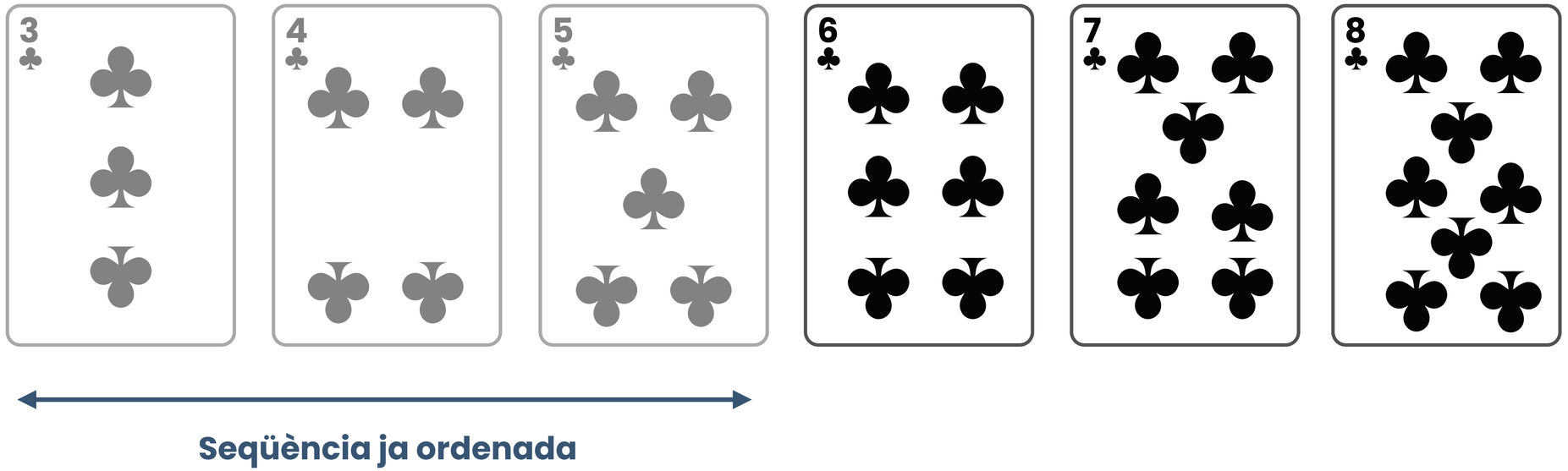
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



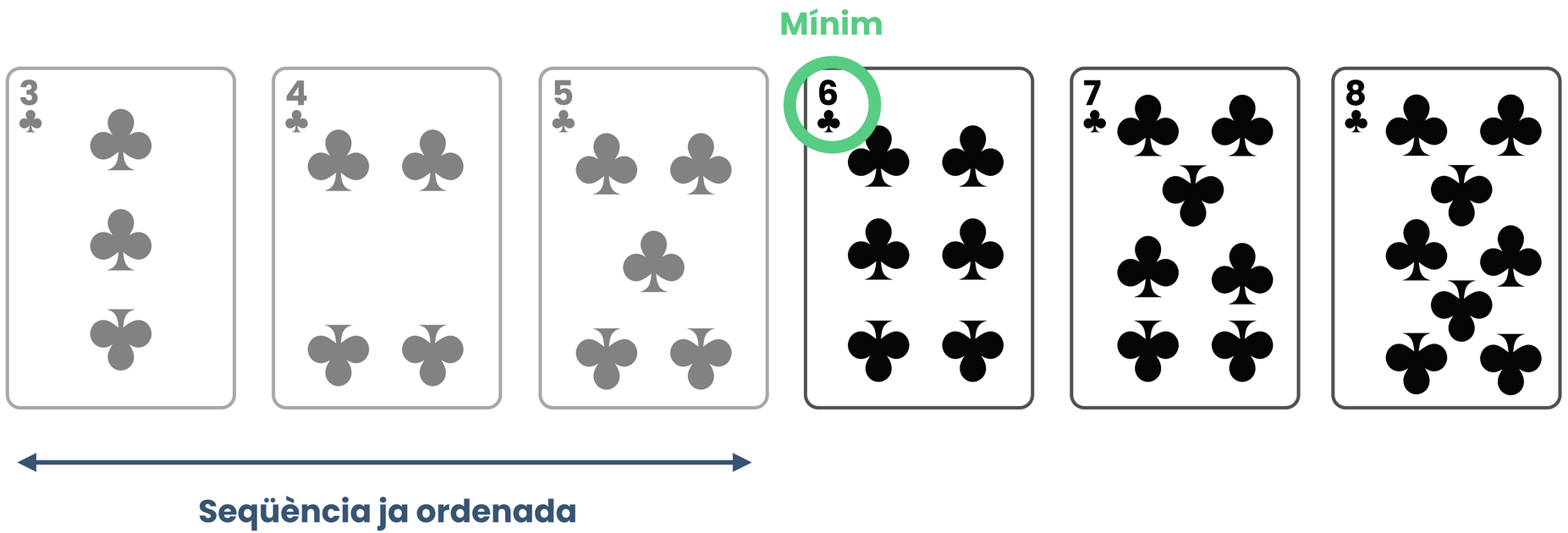
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



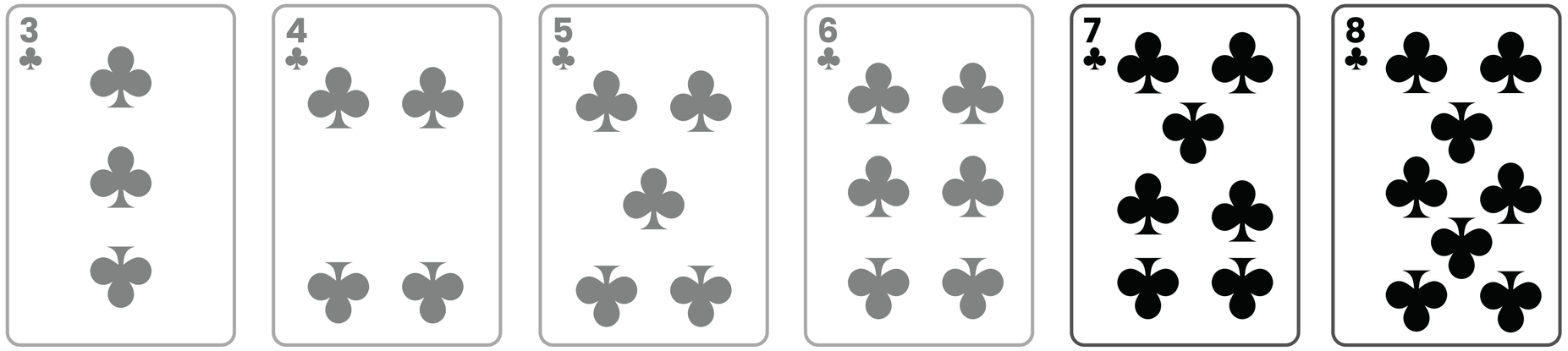
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



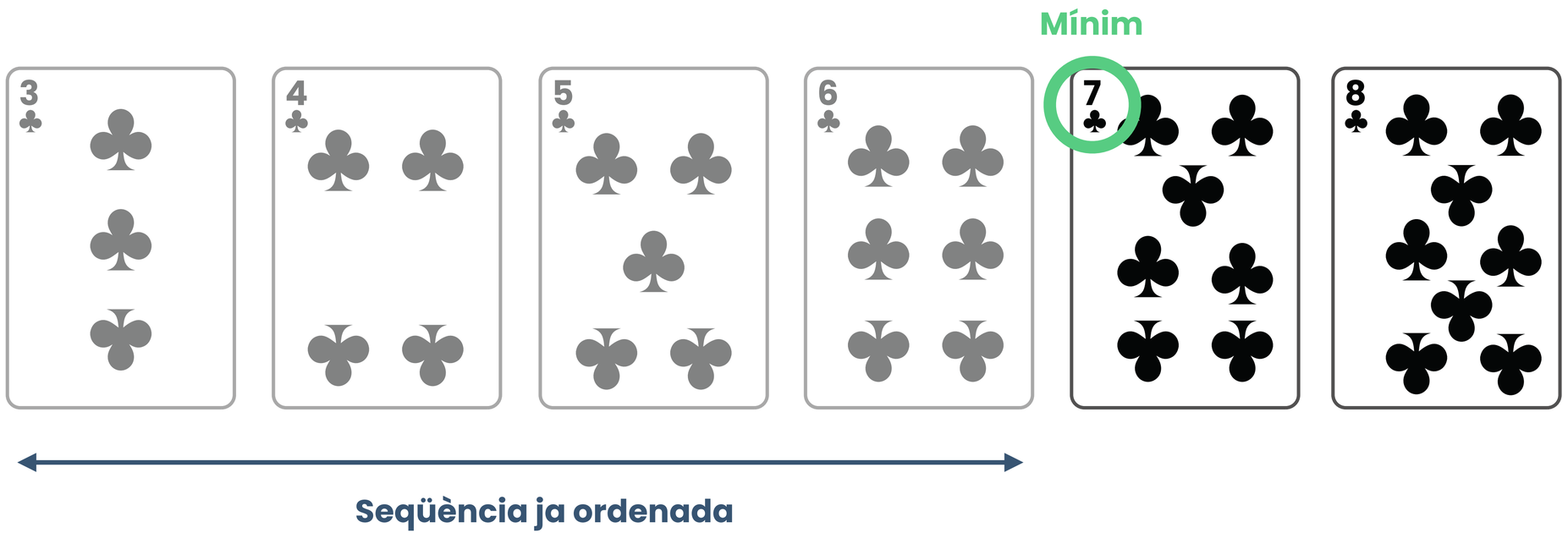
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



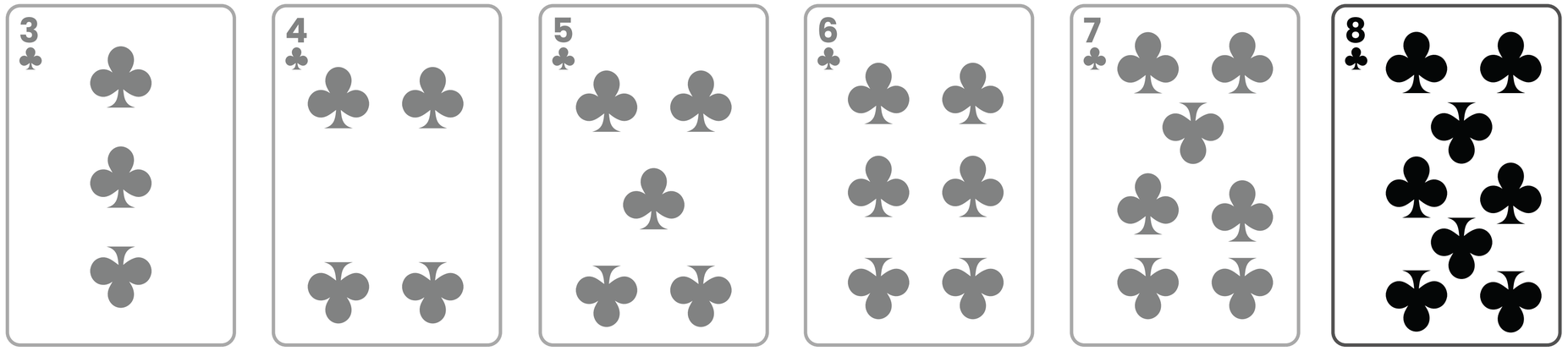
# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)

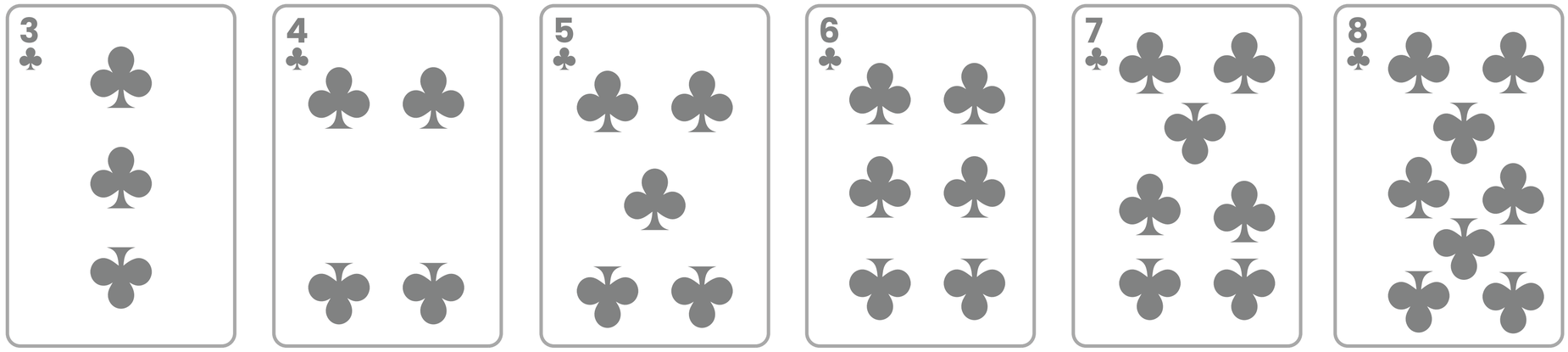


# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)

# Ordenació per selecció

- Aquest algorisme es basa en:
  - Donat un conjunt de registres, seleccionar el registre amb clau mínima
  - Intercanviar-lo amb el registre del conjunt que ocupa la primera posició.
  - Repetir aquests passos sense tenir en compte l'element que acabem de tractar (com si el traguéssim del conjunt)



## Ordenació per selecció

### Algorisme

```
$ Suposem ordenació creixent
acció ordenacio_seleccio (v: taula[] d'enter, n_elems: enter) és
var
 i, j : enter;
 pos_min, min : enter;
fvar
inici
  per (i := 0; i < n_elems-1; i := i+1) fer
     min := v[i];
     pos_min := i;
     $ Cercar el minim (no de tots, només del conjunt "actiu")
     per (j:=i+1; j < n_elems; j:=j+1) fer
       si (v[j] < min)
          min := v[j];
          pos_min := j;
       fsi
     fper
     $ Fem l'intercanvi
     v[pos_min] := v[i];
     v[i] := min;
  fper
facció
```

# Ordenació per selecció

## Algorisme

```
$ Suposem ordenació creixent
acció ordenacio_seleccio (v: taula[] d'enter, n_elems: enter) és
var
 i, j : enter;
 pos_min, min : enter;
fvar
inici
  per (i := 0; i < n_elems-1; i := i+1) fer
     min := v[i];
     pos_min := i;
     $ Cercar el minim (no de tots, només del conjunt "actiu")
     per (j:=i+1; j < n_elems; j:=j+1) fer
       si (v[j] < min)
          min := v[j];
          pos_min := j;
       fsi
     fper
     $ Fem l'intercanvi
     v[pos_min] := v[i];
     v[i] := min;
  fper
facció
```

Aquí podríem fer un intercanvi "sencer" però no cal: la posició i el valor del mínim ja ho tenim guardat a la variable min

# Ordenació per selecció

**Anàlisi de costos:**

- Cada "passada" consisteix en trobar el mínim ($\mathcal{O}(n)$) i intercanviar-lo amb el primer dels no-ordenats.

## Ordenació per selecció

**Anàlisi de costos:**

- Cada "passada" consisteix en trobar el mínim ($\mathcal{O}(n)$) i intercanviar-lo amb el primer dels no-ordenats.

En el **millor cas**, el vector ja està ordenat: { 1, 2, 3, 4, 5 }

No haurà de fer cap intercanvi però haurà de buscar tots els mínims.

**Best case = $O(n^2)$ comparacions, $O(1)$ intercanvis**

**COST = $O(n^2)$**

## Ordenació per selecció

**Anàlisi de costos:**

- Cada "passada" consisteix en trobar el mínim ($\mathcal{O}(n)$) i intercanviar-lo amb el primer dels no-ordenats.

En el **millor cas**, el vector ja està ordenat: { 1, 2, 3, 4, 5 }

No haurà de fer cap intercanvi però haurà de buscar tots els mínims.

**Best case = O(n²) comparacions, O(1) intercanvis** **COST = O(n²)**

En el **pitjor cas**, el vector està inversament ordenat: { 5, 4, 3, 2, 1 }

Farà el mateix que abans però fent els intercanvis.

**Worst case = O(n²) comparacions, O(n) intercanvis** **COST = O(n²)**

## Ordenació per selecció

**Anàlisi de costos:**

- Cada "passada" consisteix en trobar el mínim ($\mathcal{O}(n)$) i intercanviar-lo amb el primer dels no-ordenats.

En el **millor cas**, el vector ja està ordenat: { 1, 2, 3, 4, 5 }

No haurà de fer cap intercanvi però haurà de buscar tots els mínims.

**Best case = O(n²) comparacions, O(1) intercanvis** **COST = O(n²)**

En el **pitjor cas**, el vector està inversament ordenat: { 5, 4, 3, 2, 1 }

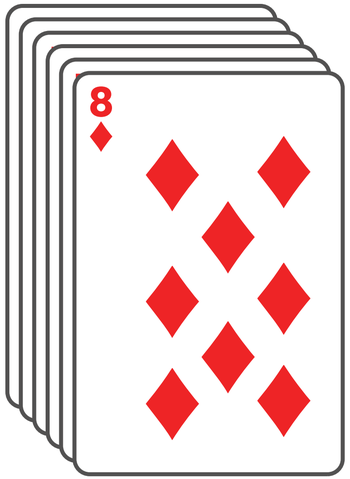
Farà el mateix que abans però fent els intercanvis.

**Worst case = O(n²) comparacions, O(n) intercanvis** **COST = O(n²)**

En el **cas mig:**

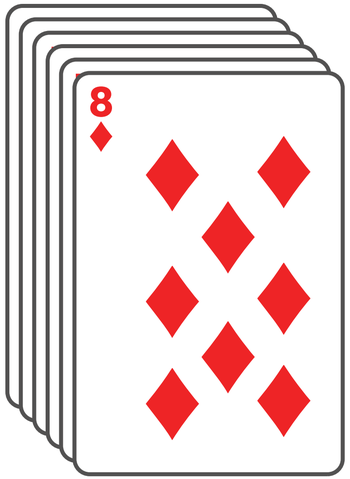
**Average case = O(n²) comparacions, O(n) intercanvis** **COST = O(n²)**

# Ordenació per intercanvi (bombolla)




- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

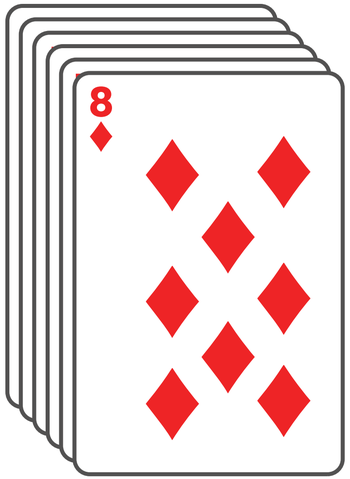
# Ordenació per intercanvi (bombolla)




- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

**Versió 1**

# Ordenació per intercanvi (bombolla)




- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

**Versió 1**

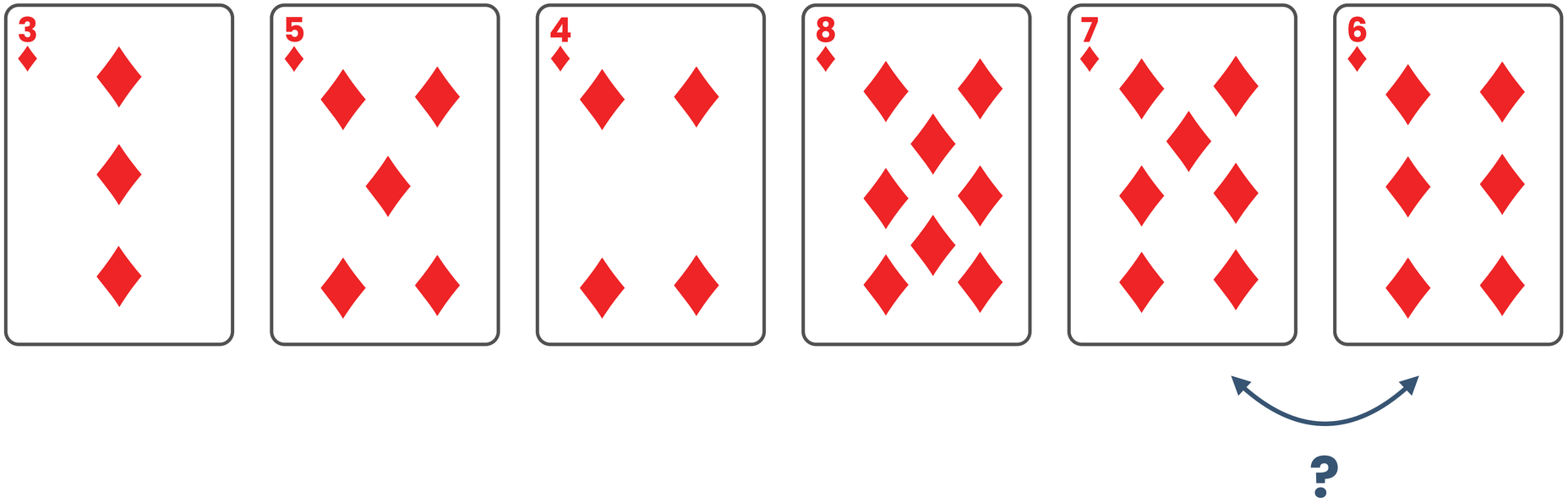
- Aquesta versió és ineficient: la taula pot estar ordenada abans d'acabar totes les passades. És a dir, pot ser que fem més passades de les que cal.
- La versió 2 inclou la modificació de només fer una nova passada si el vector no està ja ordenat.

**Versió 2**

# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

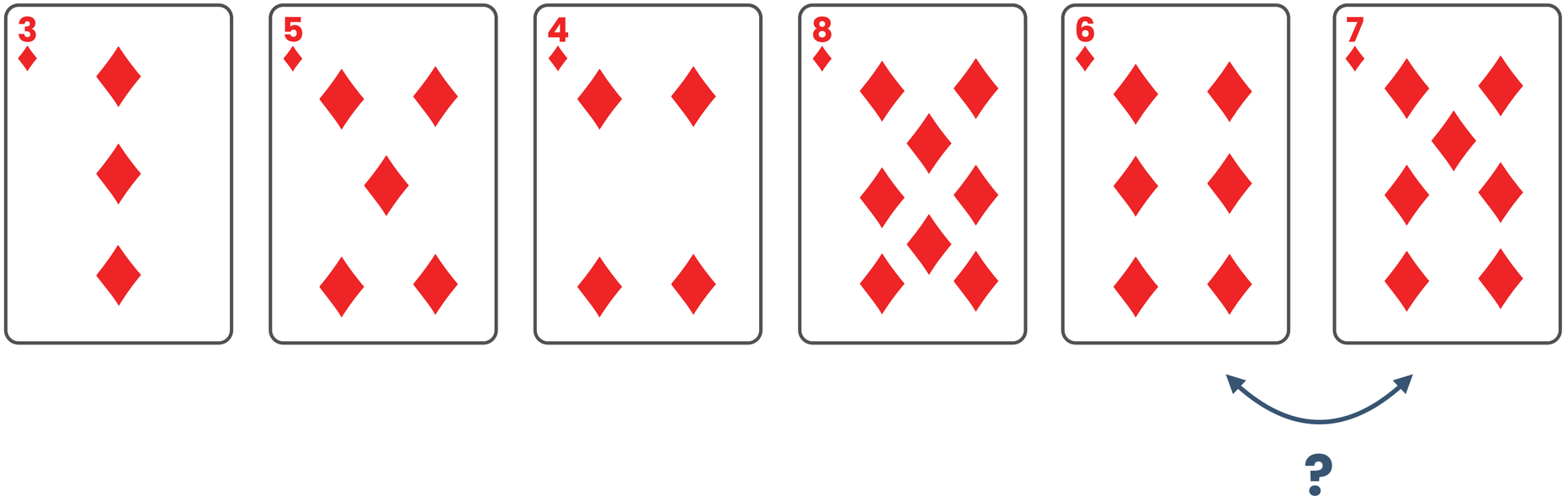
**Passada: 1**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

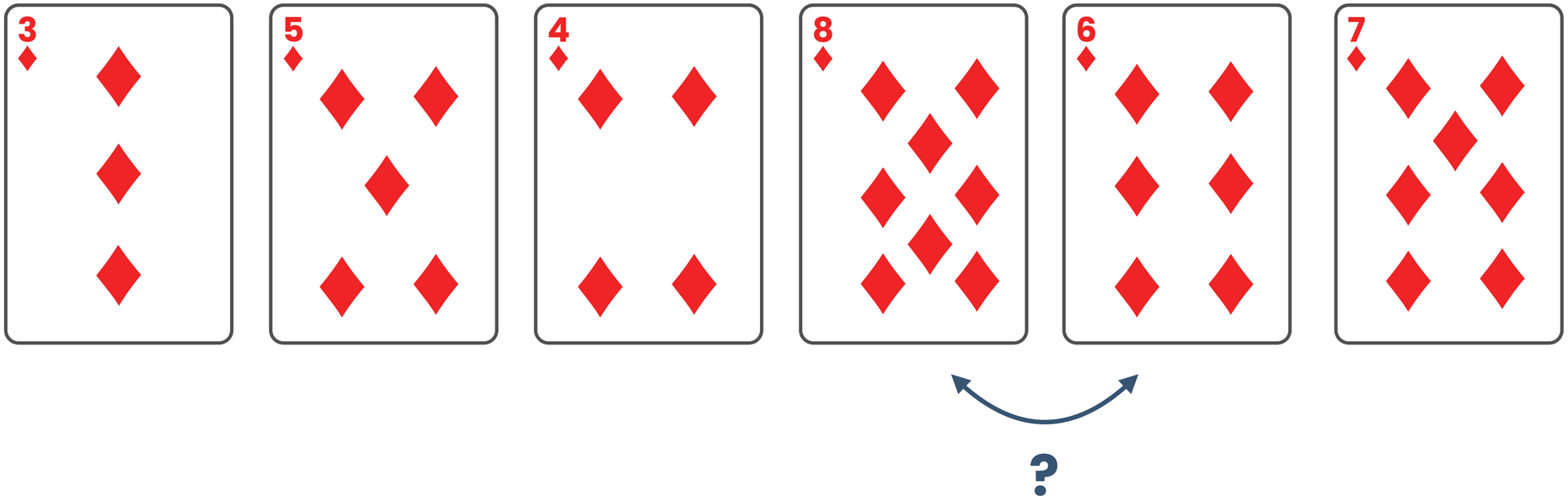
**Passada: 1**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

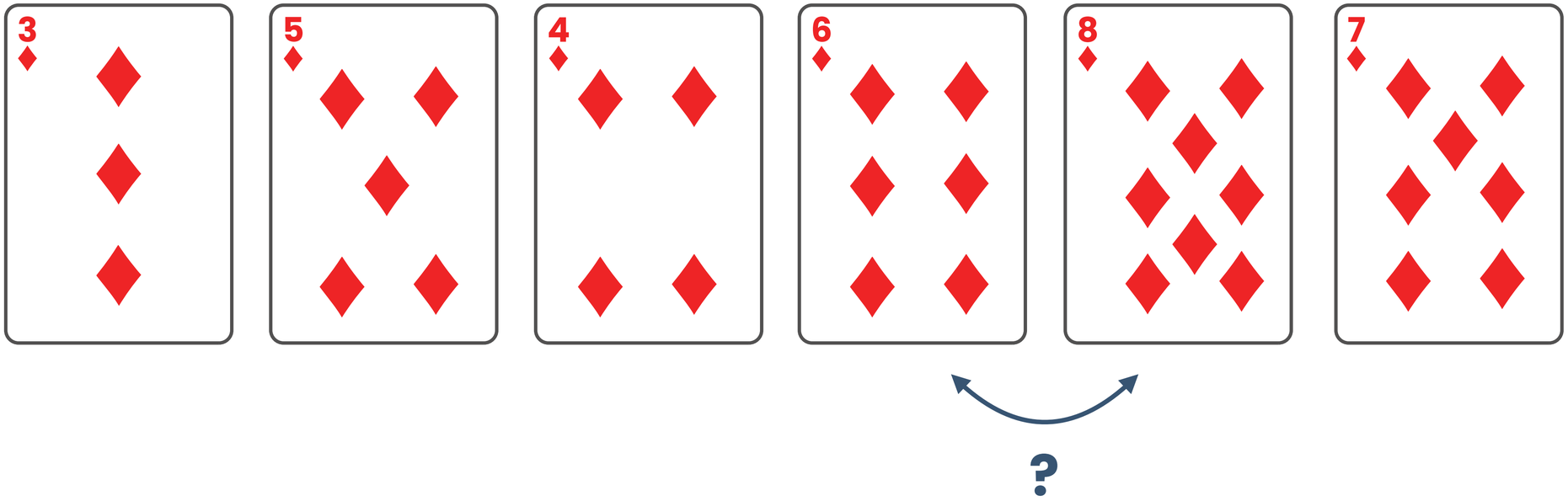
**Passada: 1**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

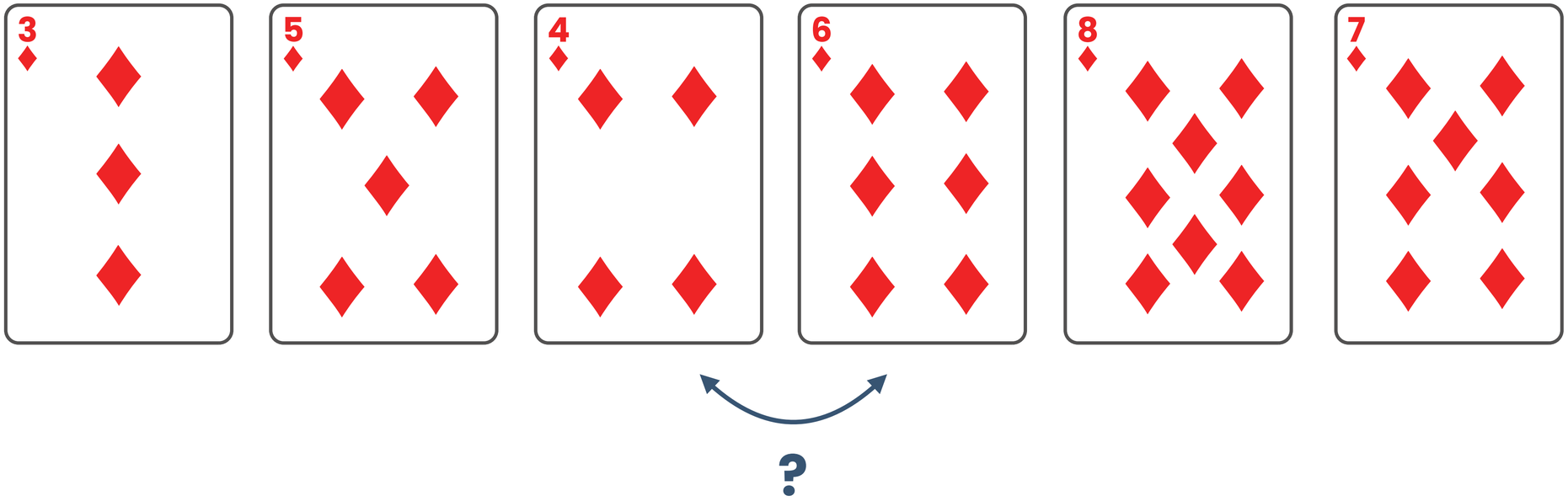
**Passada: 1**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

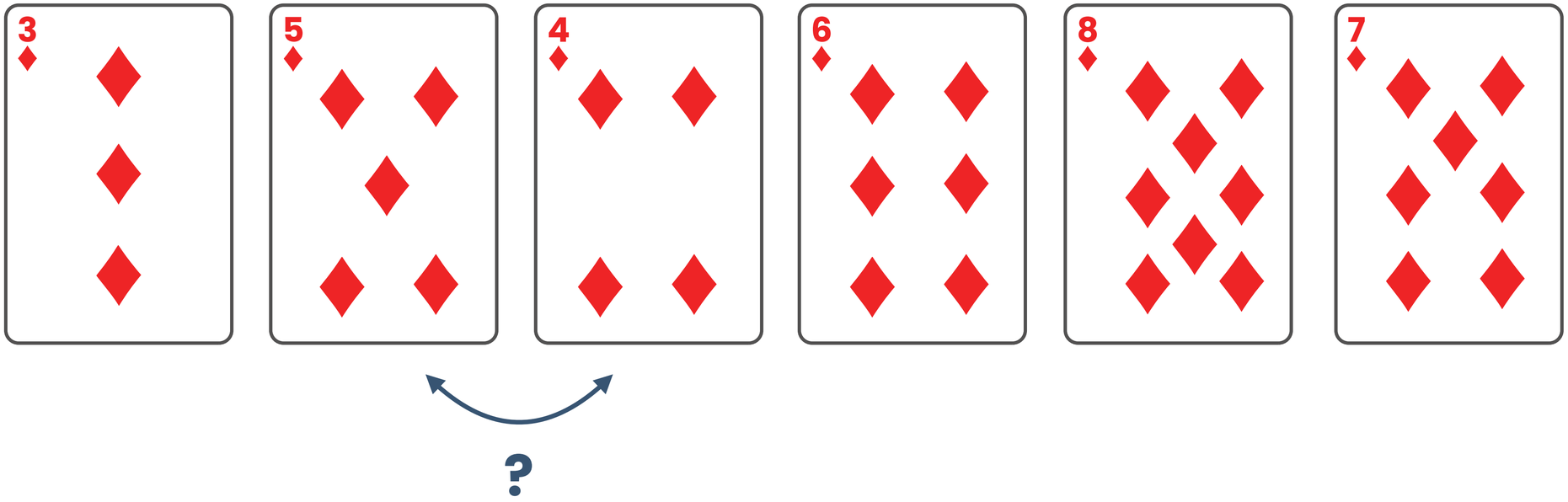
**Passada: 1**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

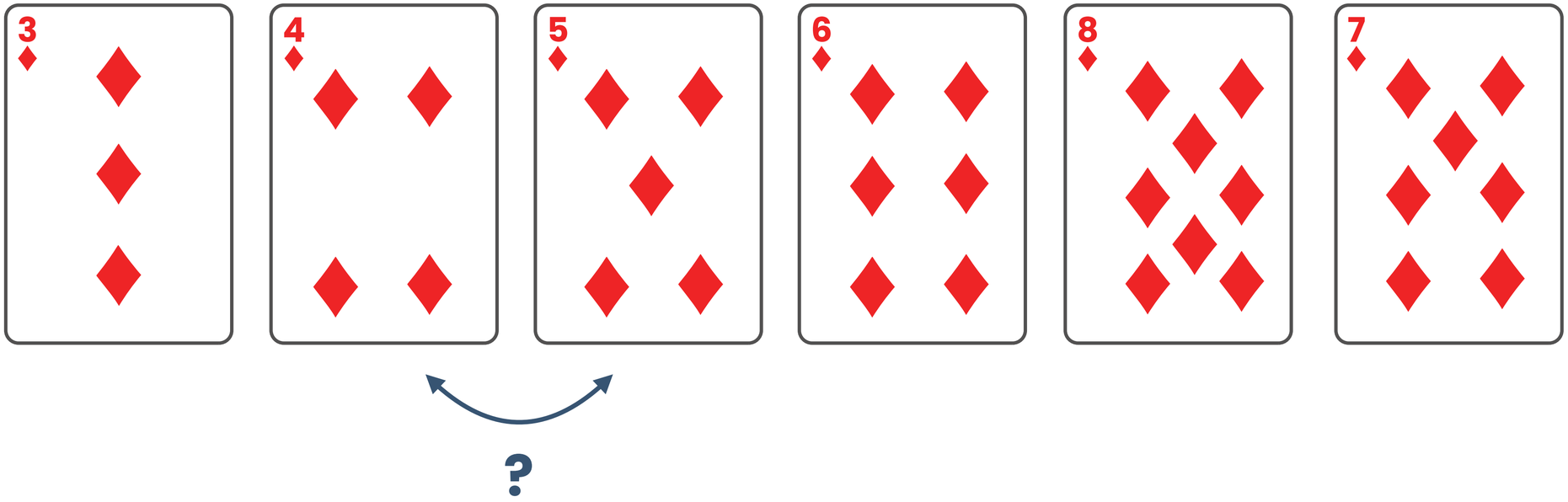
**Passada: 1**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

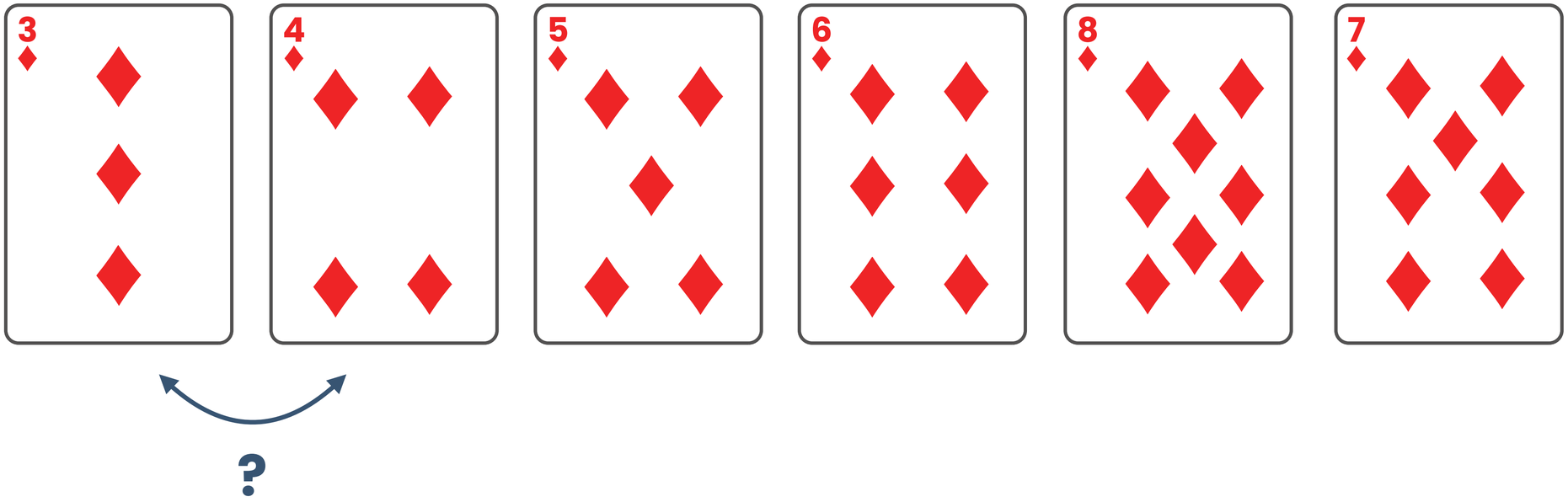
**Passada: 1**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

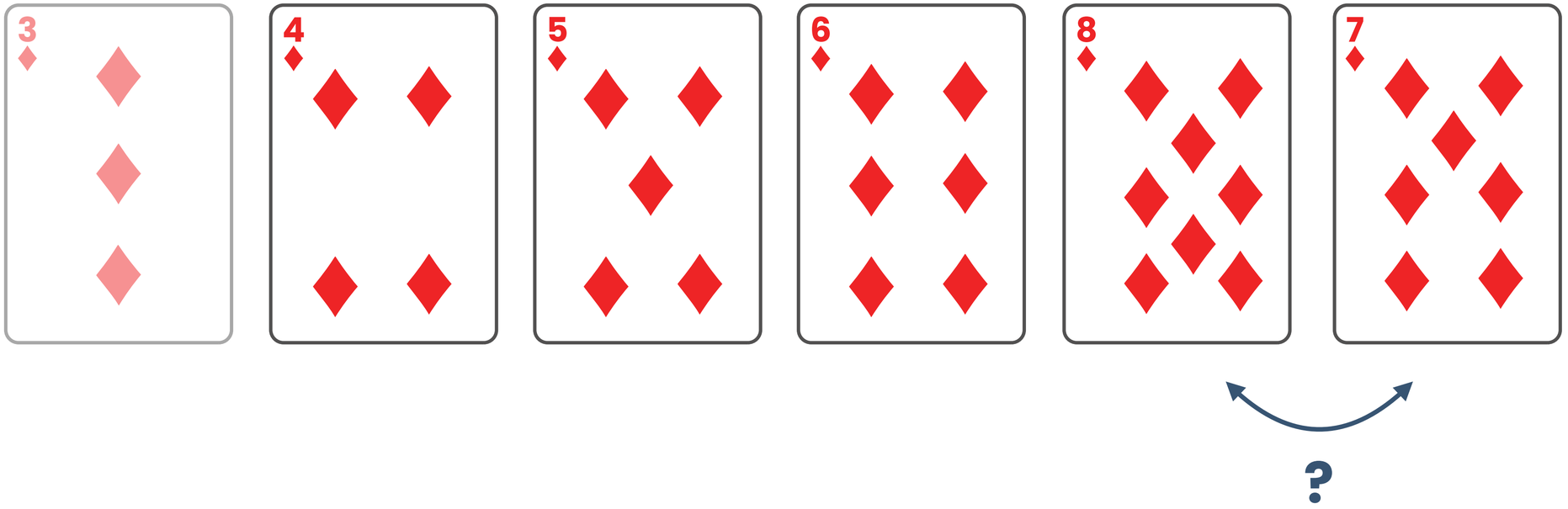
**Passada: 1**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

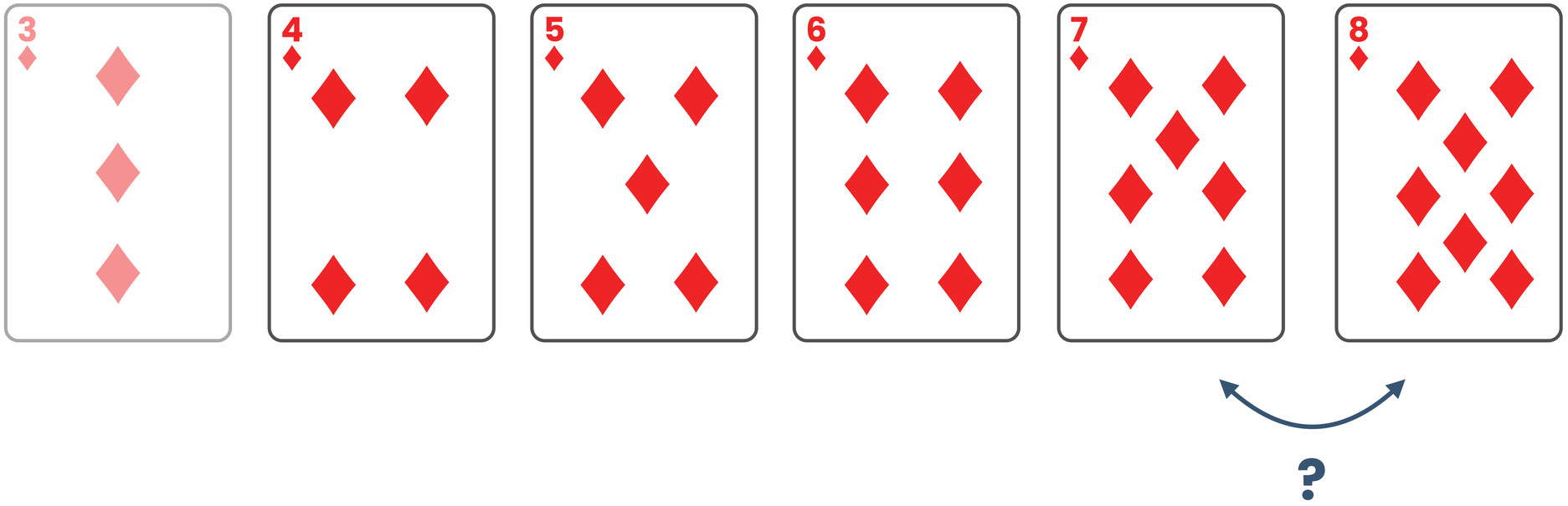
**Passada: 2**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

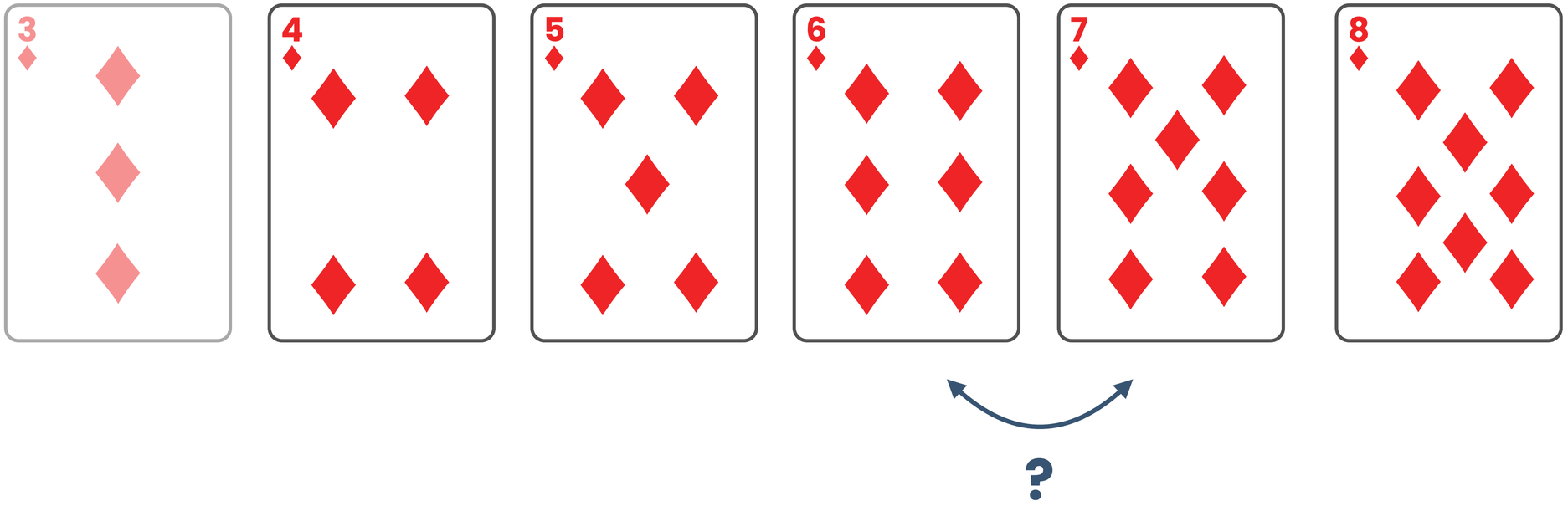
**Passada: 2**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

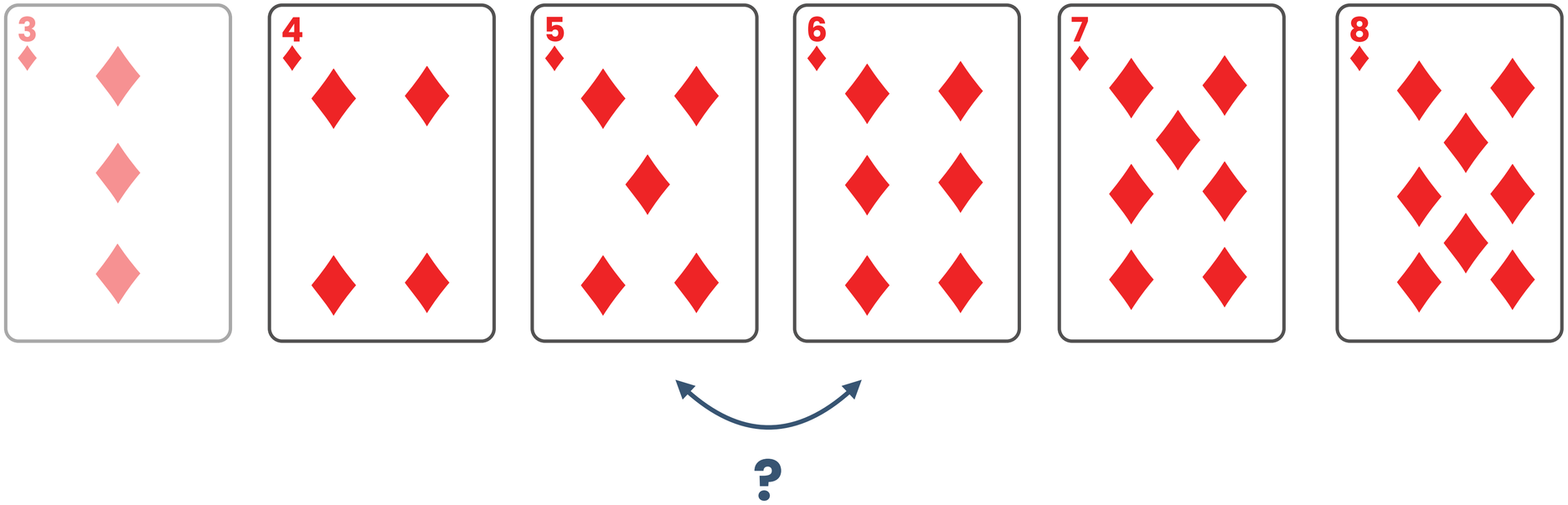
**Passada: 2**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

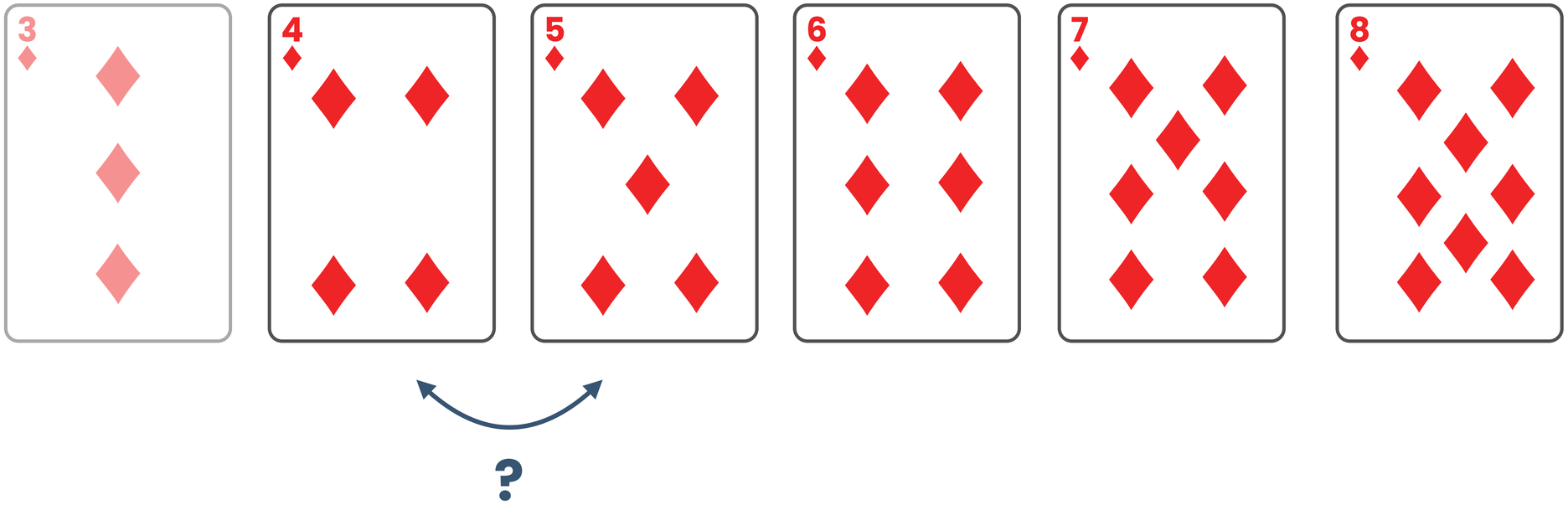
**Passada: 2**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

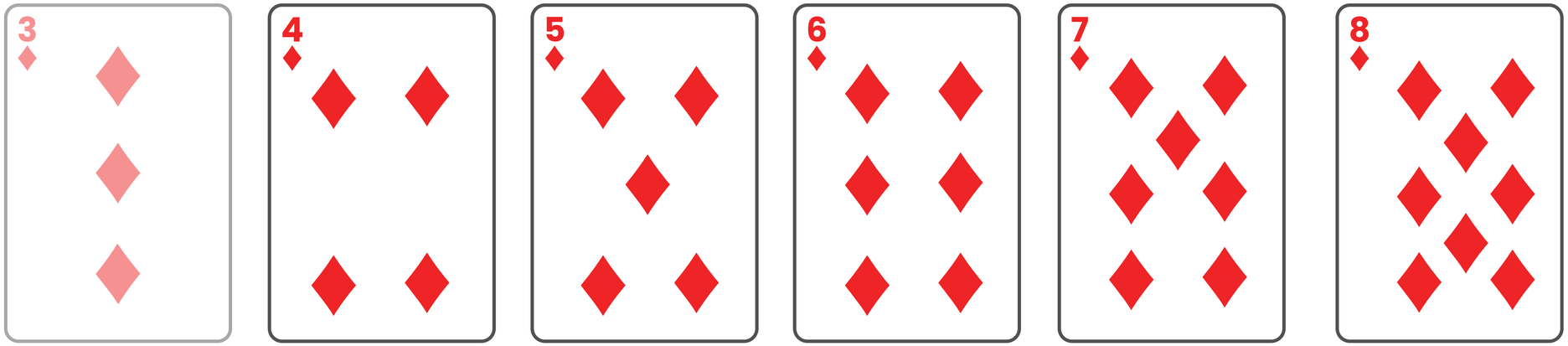
**Passada: 2**



# Ordenació per intercanvi (bombolla)

- Principi: comparar i intercanviar registres adjacents fins que tots estiguin ordenats
- Començant pel final, comparar els registres adjacents per parelles. Si el predecessor és més gran que el successor, intercanviar-los entre ells. Si no, deixar-los com estan.
- Després de la primera passada, l'element més petit ja és a la primera posició. Fem una altra passada excloent el primer element.
- Aquest procés continua fins que s'han realitzat un total de n–1 passades.

**Passada: 2**




?

**Com que ja està ordenat, HEM ACABAT!**

*(A la versió 2. A la 1 encara seguiríem fent passades!)*

# Ordenació per intercanvi (bombolla)

```
$ Algorisme del Wirth (p. 71). Versió 1 (la "dolenta", que
no limita el nombre de passades i sempre en fem n-1)
acció ordenacio_bombolla (v: taula[] d'enter, n: enter) és
var
 i, j: enter;
 aux : enter;
fvar
inici
  per (i := 0; i < n-1; i := i+1) fer $ Bucle "passades"
    per (j := n-1; j > i; j := j-1) fer
      si (v[j] < v[j-1]) llavors
         $ Fem l'intercanvi
         aux := v[j-1];
         v[j-1] := v[j];
         v[j] := aux;
      fsi
    fper
  fper
facció
```

# Ordenació per intercanvi (bombolla)

```
$ Algorisme del Wirth (p. 71). Versió 1 (la "dolenta", que
no limita el nombre de passades i sempre en fem n-1)
acció ordenacio_bombolla (v: taula[] d'enter, n: enter) és
var
 i, j: enter;
 aux : enter;
fvar
inici
  per (i := 0; i < n-1; i := i+1) fer $ Bucle "passades"
    per (j := n-1; j > i; j := j-1) fer
      si (v[j] < v[j-1]) llavors
         $ Fem l'intercanvi
         aux := v[j-1];
         v[j-1] := v[j];
         v[j] := aux;
      fsi
    fper
  fper
facció
```

**Problema greu d'eficiència:**

```
Passada=1 [ 3  5  4  8  6  7 ]
Passada=1 [ 3  5  4  6  8  7 ]
Passada=1 [ 3  5  4  6  8  7 ]
Passada=1 [ 3  4  5  6  8  7 ]
Passada=1 [ 3  4  5  6  8  7 ]
Passada=2 [ 3  4  5  6  7  8 ] *
Passada=2 [ 3  4  5  6  7  8 ]
Passada=2 [ 3  4  5  6  7  8 ]
Passada=2 [ 3  4  5  6  7  8 ]
Passada=3 [ 3  4  5  6  7  8 ]
Passada=3 [ 3  4  5  6  7  8 ]
Passada=3 [ 3  4  5  6  7  8 ]
Passada=4 [ 3  4  5  6  7  8 ]
Passada=4 [ 3  4  5  6  7  8 ]
Passada=5 [ 3  4  5  6  7  8 ]
```

*En aquest cas particular, el vector ja està ordenat amb la segona passada. La 3, 4, i 5 no calen!*

# Ordenació per intercanvi (bombolla)

### Versió 2:

*Una técnica obvia para mejorar este algoritmo es controlar si se ha producido algun cambio en una pasada. Es necesaria, por tanto, una última pasada sin operaciones de intercambio para determinar que el algoritmo puede acabar. - Wirth*

```
$ Versió "bona" de l'algorisme de la bombolla (limita el nombre de passades: si en fem una sense intercanvis, parem).
acció ordenacio_bombolla_v2 (v: taula[] d'enter, n: enter) és
var
 i, j, aux : enter;
 ja_ordenat : booleà;
fvar
inici
 i := 0;
 ja_ordenat := fals;
 mentre (i < n-1 i no(ja_ordenat)) fer
    ja_ordenat := cert;
    per (j := n-1; j > i; j:=j-1) fer
      si (v[j-1] > v[j]) llavors
         $ Fem l'intercanvi
         aux := v[j-1];
         v[j-1] := v[j];
         v[j] := aux;
         ja_ordenat := fals;
      fsi
    fper
    i := i + 1;
  fmentre
facció
```

# Ordenació per intercanvi (bombolla)

### Versió 3:

*La millora anterior (v2) fa que no haguem de fer totes les passades. Ara bé, fins i tot dins d'una mateixa passada, hi ha comparacions que ens podem estalviar: si en la passada anterior no vam haver de fer cap intercanvi a partir d'una certa posició, això vol dir que tots els valors a partir d'aquella posició ja estan al seu lloc, i per tant aquesta zona ja no cal tornar-la a mirar.*

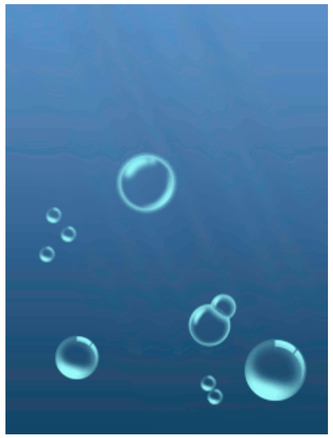
```
$ Versió 3: limitant la longitud de la passada
acció ordenacio_bombolla_v3 (v: taula[] d'enter, n: enter) és
var
 i, j, aux, limit, nou_limit : enter;
 ja_ordenat : booleà;
fvar
inici
 i := 0;
 ja_ordenat := fals;
 limit := 0; nou_limit := 0;
 mentre (i < n-1 i no(ja_ordenat)) fer
    ja_ordenat := cert;
    per (j := n-1; j > limit; j:=j-1) fer
      si (v[j-1] > v[j]) llavors
         $ Fem l'intercanvi
         aux := v[j-1];
         v[j-1] := v[j];
         v[j] := aux;
         ja_ordenat := fals;
        nou_limit := j;
      fsi
    fper
    limit := nou_limit;
    i := i + 1;
  fmentre
facció
```

# Ordenació per intercanvi (bombolla)

**Anàlisi de costos:**

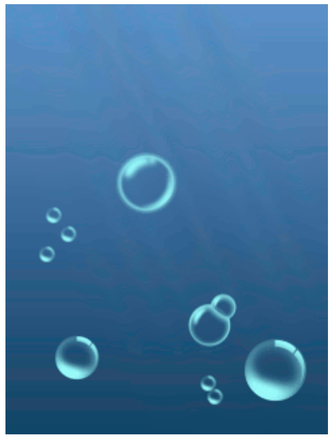
| | **Versió 1**<br>(fa totes les passades) | **Versió 2**<br>(limita el nombre de passades ) | **Versió 3**<br>(limita el nombre de passades i només compara fins al punt de l'últim intercanvi) |
|---|---|---|---|
| **Millor cas**<br>(ja ordenat) | **$O(n^2)$ en el millor cas** (perquè fem totes les passades) | **$O(n)$** (haurà de fer una passada per comprovar que està ordenat, i s'atura (no fa més passades)) | **$O(n)$** (haurà de fer una passada per comprovar que està ordenat, i s'atura (no fa més passades)) |
| **Pitjor cas**<br>(inversament ordenat) | **$O(n^2)$** (Farà n-1 passades, i cada passada comprova n-i-1 parelles) | **$O(n^2)$** (igual que versió 1) | **$O(n^2)$** (com la resta, ha de fer totes les passades i intercanvis) |
| **Cas mig** | **$O(n^2)$** | **$O(n^2)$** | **$O(n+k)$ en el cas mig,** on **k** representa com de desordenat està el vector.<br>• Si la taula està gairebé ordenada (k petit) el cost serà proper a $O(n)$.<br>• Si la taula està molt desordenada, el cost serà proper a $O(n^2)$ (el de les altres versions) |

# Ordenació per intercanvi (bombolla)



- Nombres petits situats al final arriben al seu lloc en una sola passada, mentre que nombres grans situats al principi els hi costa diverses passades arribar al final (es mouen una posició amunt per cada passada).

# Ordenació per intercanvi (bombolla)

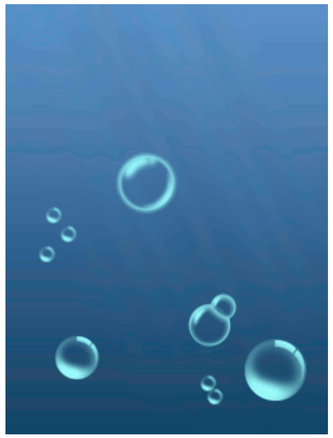


- Nombres petits situats al final arriben al seu lloc en una sola passada, mentre que nombres grans situats al principi els hi costa diverses passades arribar al final (es mouen una posició amunt per cada passada).

- Cas "nombre lleuger a la dreta" (el 6)

| 12 | 18 | 42 | 44 | 55 | 67 | 6 | 94 |
|---|---|---|---|---|---|---|---|

# Ordenació per intercanvi (bombolla)



- Nombres petits situats al final arriben al seu lloc en una sola passada, mentre que nombres grans situats al principi els hi costa diverses passades arribar al final (es mouen una posició amunt per cada passada).

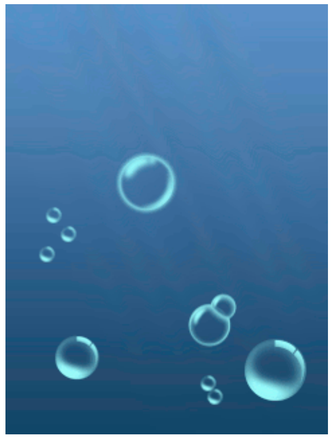
- Cas "nombre lleuger a la dreta" (el 6)

| 12 | 18 | 42 | 44 | 55 | 67 | 6 | 94 |
|---|---|---|---|---|---|---|---|

```
Passada=1 [ 12  18  42  44  55  67  6   94 ]
Passada=1 [ 12  18  42  44  55  6   67  94 ]
Passada=1 [ 12  18  42  44  6   55  67  94 ]
Passada=1 [ 12  18  42  6   44  55  67  94 ]
Passada=1 [ 12  18  6   42  44  55  67  94 ]
Passada=1 [ 12  6   18  42  44  55  67  94 ]
Passada=1 [ 6   12  18  42  44  55  67  94 ] *
```

Arriba a la seva posició en una sola passada

# Ordenació per intercanvi (bombolla)

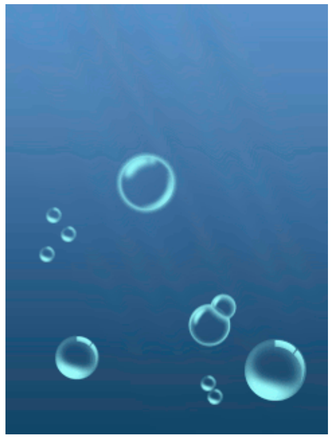


- Nombres petits situats al final arriben al seu lloc en una sola passada, mentre que nombres grans situats al principi els hi costa diverses passades arribar al final (es mouen una posició amunt per cada passada).

- Cas "nombre pesat a l'esquerra" (el 94)

| 94 | 6 | 12 | 18 | 42 | 44 | 55 | 67 |
|---|---|---|---|---|---|---|---|

# Ordenació per intercanvi (bombolla)



- Nombres petits situats al final arriben al seu lloc en una sola passada, mentre que nombres grans situats al principi els hi costa diverses passades arribar al final (es mouen una posició amunt per cada passada).

- Cas "nombre pesat a l'esquerra" (el 94)

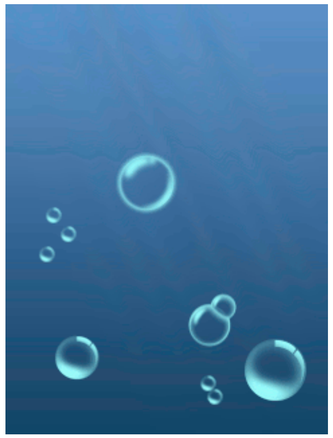
| 94 | 6 | 12 | 18 | 42 | 44 | 55 | 67 |
|---|---|---|---|---|---|---|---|

```
Passada=1 [ 94  6  12  18  42  44  55  67 ]
Passada=1 [ 94  6  12  18  42  44  55  67 ]
Passada=1 [ 94  6  12  18  42  44  55  67 ]
Passada=1 [ 94  6  12  18  42  44  55  67 ]
Passada=1 [ 94  6  12  18  42  44  55  67 ]
Passada=1 [ 94  6  12  18  42  44  55  67 ]
Passada=1 [ 6  94  12  18  42  44  55  67 ]
Passada=2 [ 6  94  12  18  42  44  55  67 ]
Passada=2 [ 6  94  12  18  42  44  55  67 ]
Passada=2 [ 6  94  12  18  42  44  55  67 ]
Passada=2 [ 6  94  12  18  42  44  55  67 ]
Passada=2 [ 6  94  12  18  42  44  55  67 ]
Passada=2 [ 6  94  12  18  42  44  55  67 ]
Passada=2 [ 6  12  94  18  42  44  55  67 ]
```

```
Passada=3 [ 6  12  94  18  42  44  55  67 ]
Passada=3 [ 6  12  94  18  42  44  55  67 ]
Passada=3 [ 6  12  94  18  42  44  55  67 ]
Passada=3 [ 6  12  94  18  42  44  55  67 ]
Passada=3 [ 6  12  18  94  42  44  55  67 ]
Passada=4 [ 6  12  18  94  42  44  55  67 ]
Passada=4 [ 6  12  18  94  42  44  55  67 ]
Passada=4 [ 6  12  18  94  42  44  55  67 ]
Passada=4 [ 6  12  18  42  94  44  55  67 ]
Passada=5 [ 6  12  18  42  94  44  55  67 ]
Passada=5 [ 6  12  18  42  94  44  55  67 ]
Passada=5 [ 6  12  18  42  44  94  55  67 ]
Passada=6 [ 6  12  18  42  44  94  55  67 ]
Passada=6 [ 6  12  18  42  44  55  94  67 ]
Passada=7 [ 6  12  18  42  44  55  67  94 ] *
```

No ariba a la seva posició fins a l'última passada

# Ordenació per intercanvi (bombolla)



- Nombres petits situats al final arriben al seu lloc en una sola passada, mentre que nombres grans situats al principi els hi costa diverses passades arribar al final (es mouen una posició amunt per cada passada).

- Cas "nombre pesat a l'esquerra" (el 94)

| 94 | 6 | 12 | 18 | 42 | 44 | 55 | 67 |
|---|---|---|---|---|---|---|---|

- Variacions:
  - Mètode que alterna la direcció de les comparacions (una passada es fa de dreta a esquerra, la següent d'esquerra a dreta... etc)

# Comparativa empírica dels algorismes clàssics

| Algorisme | Mida | Cas | Temps(s)* |
|---|---|---|---|
| **Inserció** | 10000 | Ordenat | 0.00002210 |
| **Selecció** | 10000 | Ordenat | 0.05887290 |
| **Bombolla** | 10000 | Ordenat | 0.05893820 |
| **Inserció** | 10000 | Invers | 0.06276520 |
| **Selecció** | 10000 | Invers | 0.06636920 |
| **Bombolla** | 10000 | Invers | 0.14371120 |
| **Inserció** | 10000 | Aleatori | 0.03131840 |
| **Selecció** | 10000 | Aleatori | 0.05953160 |
| **Bombolla** | 10000 | Aleatori | 0.16666380 |

| |
|---|
| 0.00001440 |

*Bombolla versió 2 (la que s'atura si una passada no fem cap intercanvi)*

- Quines conclusions en pots treure, d'aquesta comparativa?

** El temps d'execució, mesurat en segons, no té valor com a magnitud absoluta, ja que depèn de les característiques específiques de la màquina on s'ha executat. No obstant això, és útil per comparar el rendiment relatiu entre diferents algorismes en un mateix entorn d'execució.*

# Comparativa empírica dels algorismes clàssics

?

### Com pot ser que es trigui més a ordenar un vector aleatori que un inversament ordenat?

- És una molt bona pregunta, ja que és molt contraintuitiu: un vector inversament ordenat ja és el "pitjor cas" que ens podríem trobar. Per tant, ordenar un vector inversament ordenat o un d'aleatori hauria de trigar el mateix.
- Però per què no passa en el cas de la bombolla? Per un concepte que es diu "*Predicció de branques*".

#### Branch prediction

- La predicció de branques és una tècnica que utilitzen els processadors moderns per millorar la velocitat d'execució dels programes. Es basa en intentar predir quin serà el resultat d'una instrucció condicional (`if, while, for...`) en base als resultats anteriors. El processador fa una "aposta" sobre quin serà el resultat; si l'encerta, pot continuar sense interrupcions i s'estalvia temps. Si s'equivoca, ha de rectificar, i perd temps.
- A l'algorisme de la bombolla hi ha la comparació `(v[j-1] > v[j])` que és la que determina si fem l'intercanvi. En el cas del vector inversament ordenat, aquesta condició és sempre certa (sempre hem de fer l'intercanvi), de manera que la predicció és molt fàcil: el processador fa l'aposta de que serà cert i l'encerta sempre.
- En canvi, quan el vector és aleatori, aquesta predicció fallarà, de mitjana, el 50% de les vegades, i les vegades que s'equivoca el temps perdut és elevat.
- Per això, tot i que l'algorisme fa el mateix, a la pràctica ordenar un vector aleatori pot trigar més que ordenar-ne un d'inversament ordenat, perquè el seu comportament és menys previsible per al processador.