

# TEMA 7.

## CERCA I ORDENACIÓ.

TEORIA.

FONAMENTS DE PROGRAMACIÓ II

CURS: 2024-25

UNIVERSITAT ROVIRA I VIRGILI

*The key to finding a needle in a haystack  
is to first sort the hay.*

*-- Anonymous*

# **ALGORISMES DE CERCA**

# Cerca

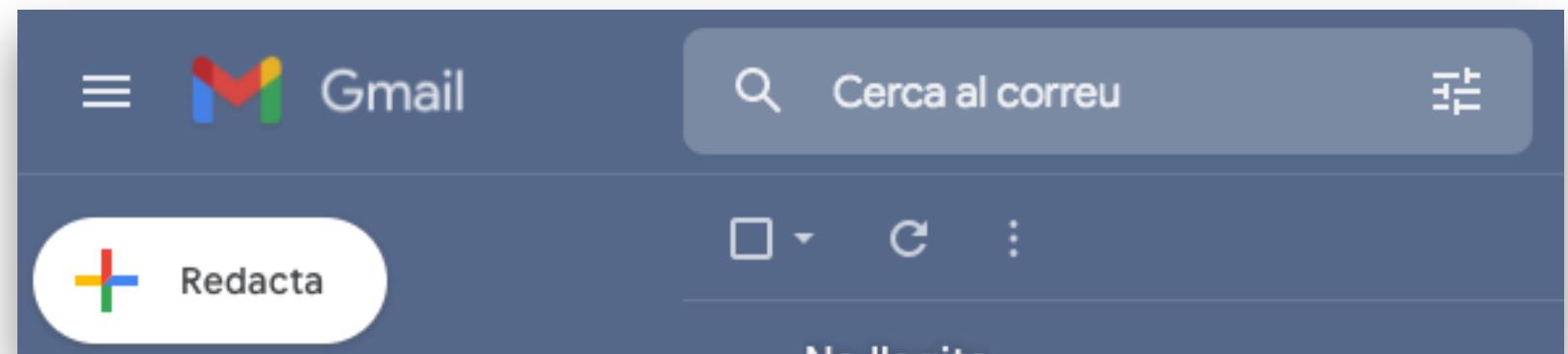
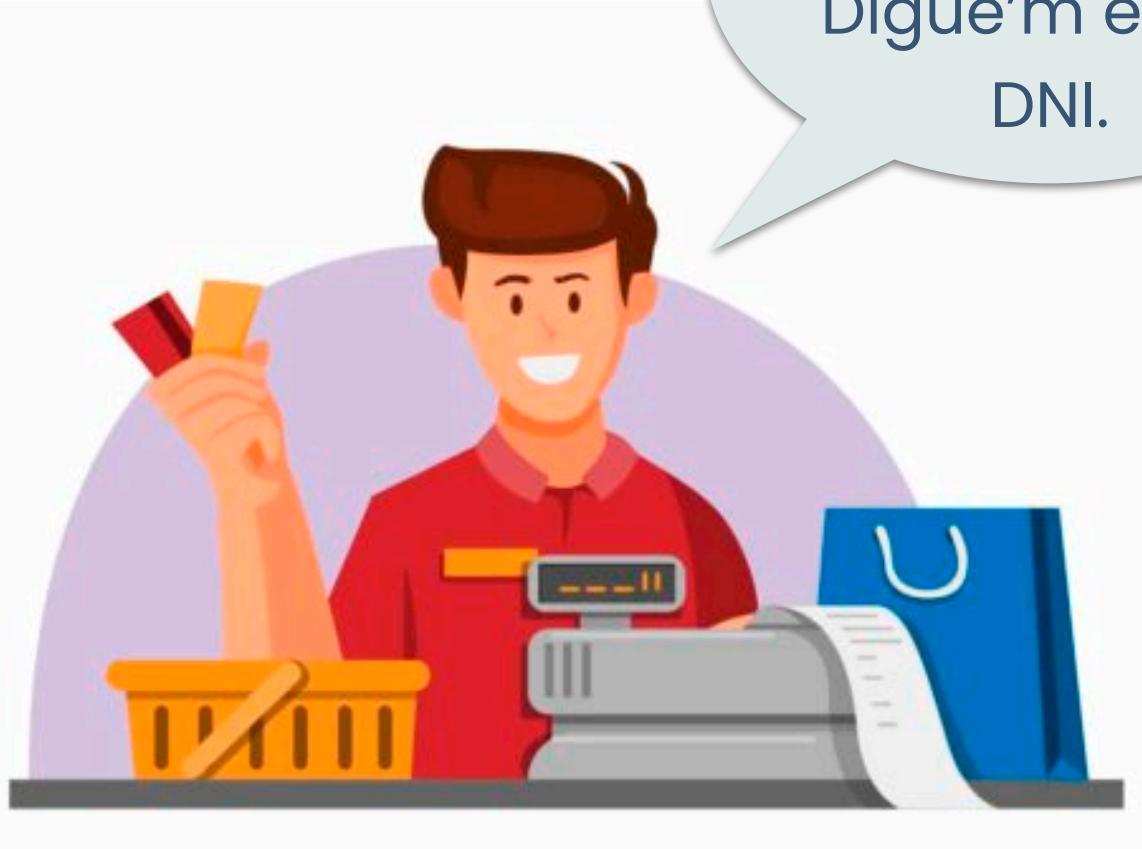
## Necessitat

- Moltíssimes de les tasques automàtiques quotidianes involucren fer accions de cerca.

# Cerca

## Necessitat

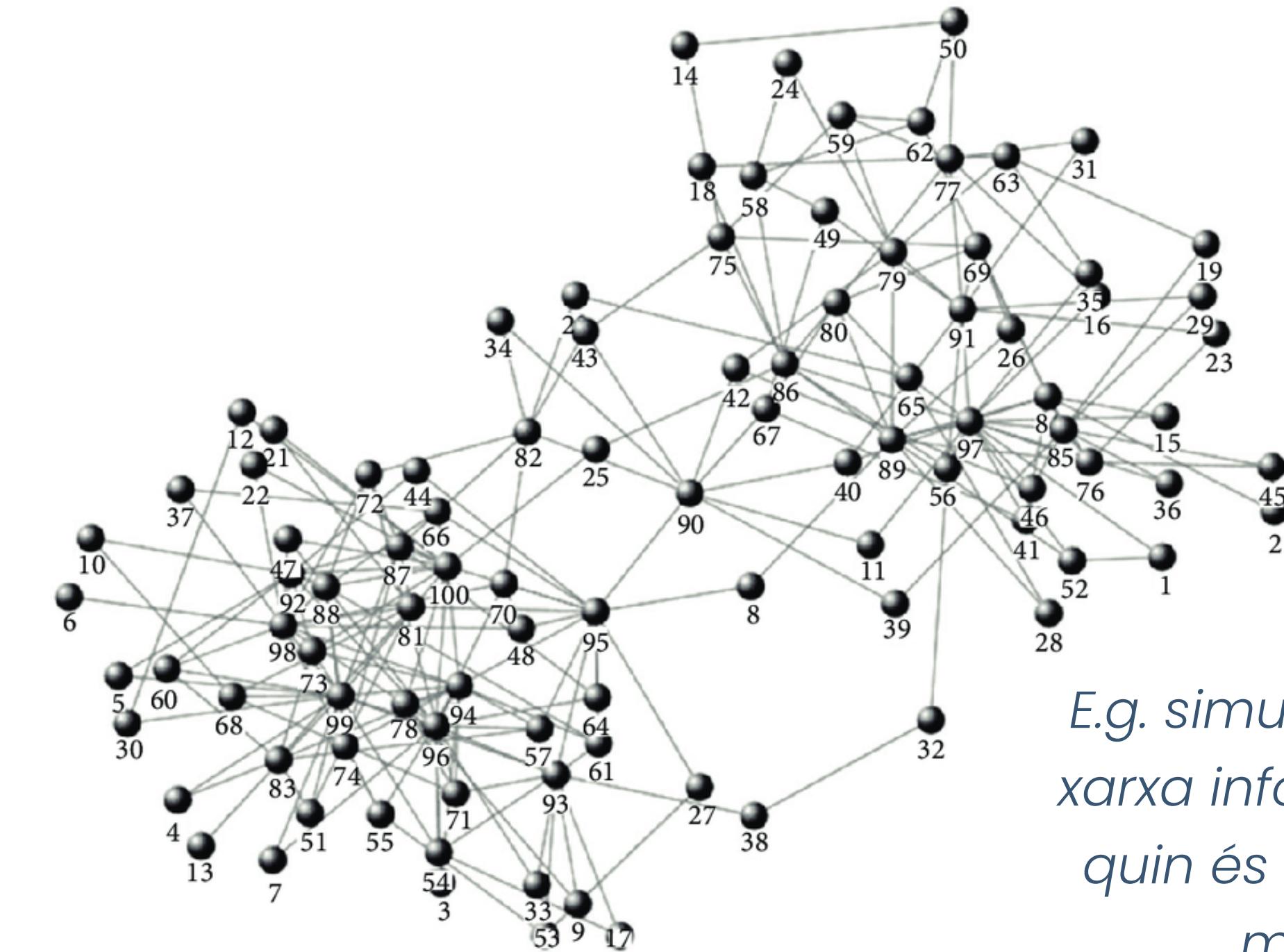
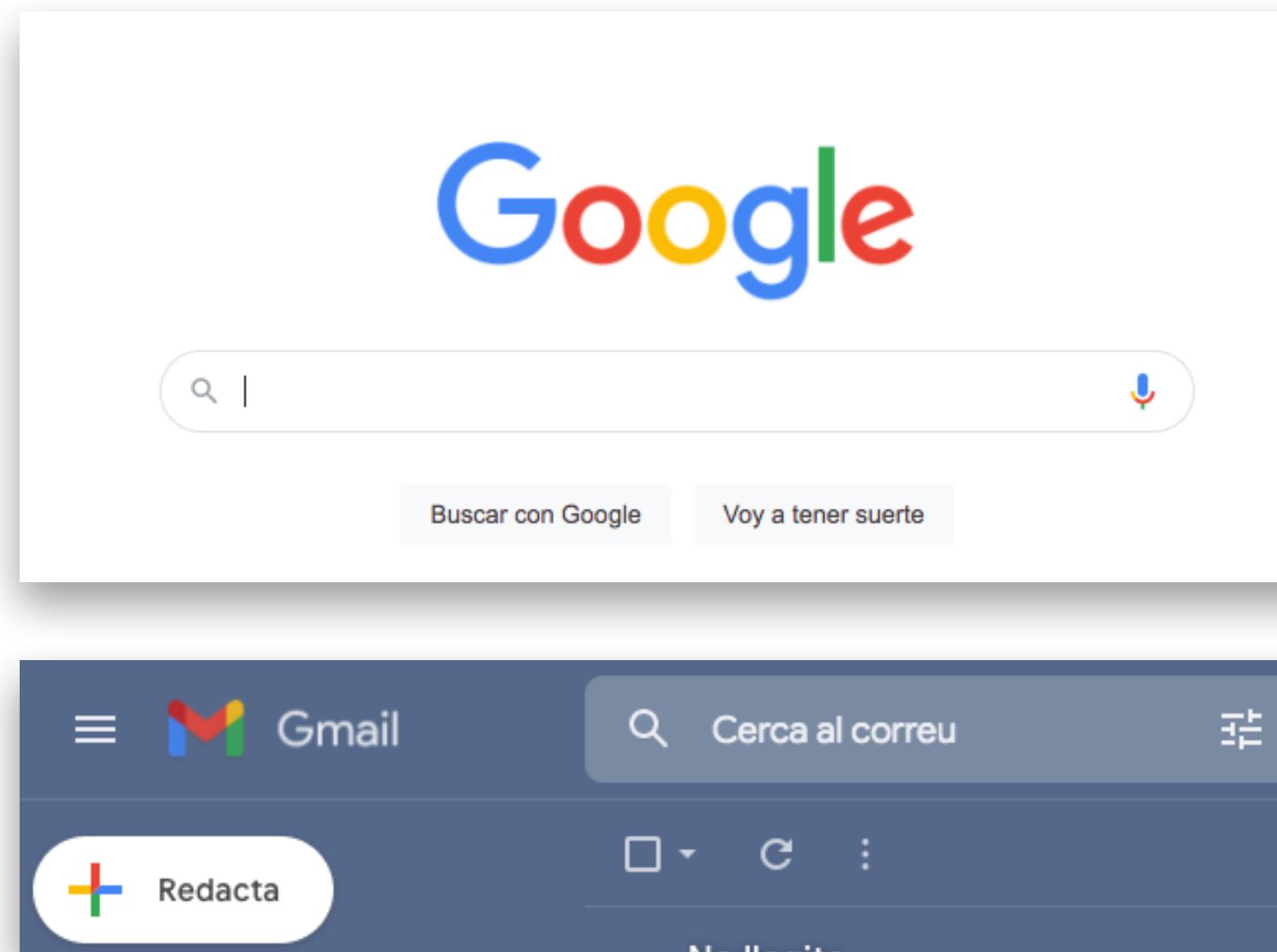
- Moltíssimes de les tasques automàtiques quotidianes involucren fer accions de cerca.
- Exemples: Cercar a Google, cercar dins del nostre correu, fer una consulta a una base de dades segons el nostre DNI...



# Cerca

## Necessitat

- Moltíssimes de les tasques automàtiques quotidianes involucren fer accions de cerca.
- Exemples: Cercar a Google, cercar dins del nostre correu, fer una consulta a una base de dades segons el nostre DNI...



E.g. simular un atac dirigit en una xarxa informàtica requereix trobar quin és el node de la xarxa amb major connectivitat.

Tens la targeta client?  
Digue'm el teu  
DNI.

# Cerca

## Algorismes de cerca

- Cerca: Indicar si un element **e** es troba dins d'un conjunt **D** (taula, fitxer, registre...etc)



# Cerca

## Algorismes de cerca

- Cerca: Indicar si un element **e** es troba dins d'un conjunt **D** (taula, fitxer, registre...etc)
- Suposarem que tenim un conjunt d'elements de tipus **registre**. I per cadascun d'ells tenim una **clau  $K_i$**  (key) que ens permet identificar-lo. El nostre objectiu és **trobar la clau** que busquem ( **$K_i=K$** ). La clau acostuma a ser el camp més identificatiu del registre.



# Cerca

## Algorismes de cerca

- Cerca: Indicar si un element **e** es troba dins d'un conjunt **D** (taula, fitxer, registre...etc)
- Suposarem que tenim un conjunt d'elements de tipus **registre**. I per cadascun d'ells tenim una **clau K<sub>i</sub>** (key) que ens permet identificar-lo. El nostre objectiu és **trobar la clau** que busquem (**K<sub>i</sub>=K**). La clau acostuma a ser el camp més identificatiu del registre.

**Clau: DNI**

Nom: Paquita  
**DNI: 12334562**  
Adreça: Tarragona

Nom: Jaume  
**DNI: 65983233**  
Adreça: Reus

Nom: Josefina  
**DNI: 52665432**  
Adreça: Igualada

Nom: Pepet  
**DNI: 39332255**  
Adreça: Lleida

**Registres**



# Cerca

## Algorismes de cerca

- Cerca: Indicar si un element **e** es troba dins d'un conjunt **D** (taula, fitxer, registre...etc)
- Suposarem que tenim un conjunt d'elements de tipus **registre**. I per cadascun d'ells tenim una **clau K<sub>i</sub>** (key) que ens permet identificar-lo. El nostre objectiu és **trobar la clau** que busquem (**K<sub>i</sub>=K**). La clau acostuma a ser el camp més identificatiu del registre.

**Clau: DNI**

Nom: Paquita
<b>DNI: 12334562</b>
Adreça: Tarragona

Nom: Jaume
<b>DNI: 65983233</b>
Adreça: Reus

Nom: Josefina
<b>DNI: 52665432</b>
Adreça: Igualada

Nom: Pepet
<b>DNI: 39332255</b>
Adreça: Lleida



## Registres

- Neecessitem que el procés de cerca sigui **eficient**, per assegurar-nos que trobem l'element en poc temps. Ara tenim molta capacitat computacional, però per altra banda, les bases de dades són molt més grans.

## Cerca en conjunts no ordenats: **Cerca seqüencial**

- Primera aproximació: cerca seqüencial.
- És l'única opció si el conjunt de dades no està ordenat.

## Cerca en conjunts no ordenats: Cerca seqüencial

- Primera aproximació: cerca seqüencial.
- És l'única opció si el conjunt de dades no està ordenat.
- Consisteix en comparar la clau que busquem (**K**) amb cadascuna de les claus del conjunt de registres (**K<sub>i</sub>**) fins que el trobem (*cas successful*) o fins que s'arriba al final (*cas unsuccessful*).
- L'existència de **K** es pot assegurar tant aviat es troba però la no-existència no es pot assegurar fins haver recorregut tot el conjunt.

# Cerca en conjunts no ordenats: Cerca seqüencial

- Primera aproximació: cerca seqüencial.
- És l'única opció si el conjunt de dades no està ordenat.
- Consisteix en comparar la clau que busquem (**K**) amb cadascuna de les claus del conjunt de registres (**K<sub>i</sub>**) fins que el trobem (*cas successful*) o fins que s'arriba al final (*cas unsuccessful*).
- L'existència de **K** es pot assegurar tant aviat es troba però la no-existència no es pot assegurar fins haver recorregut tot el conjunt.

```
funció cerca_simple (vector: taula[] d'enter,  
mida: enter, clau: enter) retorna booleà és  
var  
    i: enter;  
    trobat: booleà;  
fvar  
inici  
    i := 0;  
    trobat := fals;  
    mentre (i<mida i no(trobat)) fer  
        si (vector[i] = clau)  
            trobat := cert;  
        fsi  
        i:=i+1;  
    fmentre  
    retorna (trobat);  
ffunció
```

# Cerca en conjunts no ordenats: Cerca seqüencial

- Primera aproximació: cerca seqüencial.
- És l'única opció si el conjunt de dades no està ordenat.
- Consisteix en comparar la clau que busquem (**K**) amb cadascuna de les claus del conjunt de registres (**K<sub>i</sub>**) fins que el trobem (cas successful) o fins que s'arriba al final (cas unsuccessful).
- L'existència de **K** es pot assegurar tant aviat es troba però la no-existència no es pot assegurar fins haver recorregut tot el conjunt.
- També ens pot interessar retornar la posició en la qual hem trobat l'element, per després poder accedir a la resta del registre:

```
funció cerca_simple (vector: taula[] d'enter,
mida: enter, clau: enter) retorna enter és
var
    i: enter;
    posicio: enter;
fvar
inici
    i := 0;
    posicio := -1;
mentre (i < mida i no(trobat)) fer
    si (vector[i] = clau)
        posicio := i;
    fsi
    i:=i+1;
fmentre
retorna (posicio);
ffunció
```

# Cerca en conjunts no ordenats: Cerca seqüencial

## Anàlisi de costos

```
funció cerca_simple (vector: taula[] d'enter,
mida: enter, clau: enter) retorna enter és
var
    i: enter;
    posicio: enter;
fvar
inici
    i := 0;
    posicio := -1;
mentre (i<mida i no(trobat)) fer
    si (vector[i] = clau)
        posicio := i;
    fsi
    i:=i+1;
fmentre
retorna (posicio);
ffunció
```

# Cerca en conjunts no ordenats: Cerca seqüencial

## Anàlisi de costos

- En el **millor cas**, l'element està a la primera posició i només fem una iteració.

Busquem 1:

{ 1, 5, 4, 2, 3 }

A la primera iteració ja fem trobat=true

```
funció cerca_simple (vector: taula[] d'enter,
mida: enter, clau: enter) retorna enter és
var
    i: enter;
    posicio: enter;
fvar
inici
    i := 0;
    posicio := -1;
mentre (i < mida i no(trobat)) fer
    si (vector[i] = clau)
        posicio := i;
    fsi
    i:=i+1;
fmentre
retorna (posicio);
ffunció
```

# Cerca en conjunts no ordenats: Cerca seqüencial

## Anàlisi de costos

- En el **millor cas**, l'element està a la primera posició i només fem una iteració.

Busquem 1:

{ 1, 5, 4 ,2 ,3 }

A la primera iteració ja fem trobat=true

**Best case = O(1)**

```
funció cerca_simple (vector: taula[] d'enter,
mida: enter, clau: enter) retorna enter és
var
    i: enter;
    posicio: enter;
fvar
inici
    i := 0;
    posicio := -1;
mentre (i<mida i no(trobat)) fer
    si (vector[i] = clau)
        posicio := i;
    fsi
    i:=i+1;
fmentre
retorna (posicio);
ffunció
```

# Cerca en conjunts no ordenats: Cerca seqüencial

## Anàlisi de costos

- En el **millor cas**, l'element està a la primera posició i només fem una iteració.

Busquem **1**:

{ 1, 5, 4, 2, 3 }

A la primera iteració ja fem trobat=true

- En el **pitjor cas**, l'element no hi és i hem de recórrer tot el conjunt.

Busquem **6**:

{ 1, 5, 4, 2, 3 }

Hem de recórrer tot el vector

**Best case = O(1)**

```
funció cerca_simple (vector: taula[] d'enter,
mida: enter, clau: enter) retorna enter és
var
    i: enter;
    posicio: enter;
fvar
inici
    i := 0;
    posicio := -1;
mentre (i < mida i no(trobat)) fer
    si (vector[i] = clau)
        posicio := i;
    fsi
    i:=i+1;
fmentre
retorna (posicio);
ffunció
```

# Cerca en conjunts no ordenats: Cerca seqüencial

## Anàlisi de costos

- En el **millor cas**, l'element està a la primera posició i només fem una iteració.

Busquem **1**:

{ 1, 5, 4, 2, 3 }

A la primera iteració ja fem trobat=true

**Best case = O(1)**

- En el **pitjor cas**, l'element no hi és i hem de recórrer tot el conjunt.

Busquem **6**:

{ 1, 5, 4, 2, 3 }

Hem de recórrer tot el vector

**Worst case = O(n)**

```
funció cerca_simple (vector: taula[] d'enter,
mida: enter, clau: enter) retorna enter és
var
    i: enter;
    posicio: enter;
fvar
inici
    i := 0;
    posicio := -1;
mentre (i < mida i no(trobat)) fer
    si (vector[i] = clau)
        posicio := i;
    fsi
    i:=i+1;
fmentre
retorna (posicio);
ffunció
```

# Cerca en conjunts no ordenats: Cerca seqüencial

## Anàlisi de costos

- En el **millor cas**, l'element està a la primera posició i només fem una iteració.

Busquem **1**:

{ 1, 5, 4, 2, 3 }

A la primera iteració ja fem trobat=true

- En el **pitjor cas**, l'element no hi és i hem de recórrer tot el conjunt.

Busquem **6**:

{ 1, 5, 4, 2, 3 }

Hem de recórrer tot el vector

- En el cas **mig**, l'element és al mig, hem de fer  $N/2$  iteracions.

**Best case = O(1)**

**Worst case = O(n)**

```
funció cerca_simple (vector: taula[] d'enter,
mida: enter, clau: enter) retorna enter és
var
    i: enter;
    posicio: enter;
fvar
inici
    i := 0;
    posicio := -1;
mentre (i < mida i no(trobat)) fer
    si (vector[i] = clau)
        posicio := i;
    fsi
    i:=i+1;
fmentre
retorna (posicio);
ffunció
```

# Cerca en conjunts no ordenats: Cerca seqüencial

## Anàlisi de costos

- En el **millor cas**, l'element està a la primera posició i només fem una iteració.

Busquem **1**:

{ 1, 5, 4, 2, 3 }

A la primera iteració ja fem trobat=true

**Best case = O(1)**

- En el **pitjor cas**, l'element no hi és i hem de recórrer tot el conjunt.

Busquem **6**:

{ 1, 5, 4, 2, 3 }

Hem de recórrer tot el vector

**Worst case = O(n)**

- En el cas **mig**, l'element és al mig, hem de fer N/2 iteracions.

**Average case = O(n)**

```
funció cerca_simple (vector: taula[] d'enter,
mida: enter, clau: enter) retorna enter és
var
    i: enter;
    posicio: enter;
fvar
inici
    i := 0;
    posicio := -1;
mentre (i < mida i no(trobat)) fer
    si (vector[i] = clau)
        posicio := i;
    fsi
    i:=i+1;
fmentre
retorna (posicio);
ffunció
```

# Cerca en conjunts ordenats

Què faríeu si una persona us entrega una guia telefònica i us demana **a quina persona pertany el telèfon 977558282**? En aquest cas no tenim més remei que fer servir la **cerca seqüencial** anterior.

En una guia telefònica, la “**clau**” és el **cognom** de la persona, i aquest conjunt de claus està **ordenat** alfabèticament.

Per aquest motiu és molt més fàcil trobar qui és el número de telèfon d'una persona que trobar a qui pertany cert número de telèfon.

# Cerca en conjunts ordenats

Què faríeu si una persona us entrega una guia telefònica i us demana **a quina persona pertany el telèfon 977558282**? En aquest cas no tenim més remei que fer servir la **cerca seqüencial** anterior.

En una guia telefònica, la “**clau**” és el **cognom** de la persona, i aquest conjunt de claus està **ordenat** alfabèticament.

Per aquest motiu és molt més fàcil trobar qui és el número de telèfon d'una persona que trobar a qui pertany cert número de telèfon.

## Cerca en conjunts ordenats

- Quan el conjunt de claus està ordenat, els mètodes de cerca poden ser més eficients.

# Cerca en conjunts ordenats

Què faríeu si una persona us entrega una guia telefònica i us demana **a quina persona pertany el telèfon 977558282**? En aquest cas no tenim més remei que fer servir la **cerca seqüencial** anterior.

En una guia telefònica, la “**clau**” és el **cognom** de la persona, i aquest conjunt de claus està **ordenat** alfabèticament.

Per aquest motiu és molt més fàcil trobar qui és el número de telèfon d'una persona que trobar a qui pertany cert número de telèfon.

## Cerca en conjunts ordenats

- Quan el conjunt de claus està ordenat, els mètodes de cerca poden ser més eficients.
- En el cas de la cerca seqüencial, només podem contemplar els casos:
  - $K_i = K$  (trobat, cerca acabada),
  - i  $K_i \neq K$  (element encara no trobat)

# Cerca en conjunts ordenats

Què faríeu si una persona us entrega una guia telefònica i us demana **a quina persona pertany el telèfon 977558282**? En aquest cas no tenim més remei que fer servir la **cerca seqüencial** anterior.

En una guia telefònica, la “**clau**” és el **cognom** de la persona, i aquest conjunt de claus està **ordenat** alfabèticament.

Per aquest motiu és molt més fàcil trobar quin és el número de telèfon d'una persona que trobar a qui pertany cert número de telèfon.

## Cerca en conjunts ordenats

- Quan el conjunt de claus està ordenat, els mètodes de cerca poden ser més eficients.
- En el cas de la cerca seqüencial, només podem contemplar els casos:
  - $K_i = K$  (trobat, cerca acabada),
  - i  $K_i \neq K$  (element encara no trobat)
- En canvi, si les dades tenen un ordre implícit, contemplarem els casos:
  - $K_i < K$  (descartem els registres anteriors a  $K_i$ )
  - o bé  $K_i = K$  (trobat, cerca acabada)
  - o bé  $K_i > K$  (descartem els registres posteriors a  $K_i$ )

# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

**Busquem: el 5**

1	3	5	10	22	25	30	42	50
---	---	---	----	----	----	----	----	----

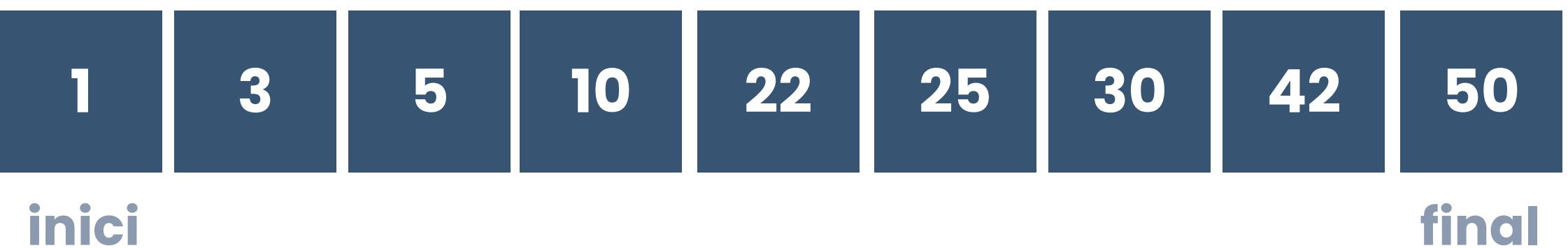
# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

**Busquem: el 5**



# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

**Busquem: el 5**



# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

**Busquem: el 5**



**22 > 5.** Descartem la part superior

# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

**Busquem: el 5**



# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

**Busquem: el 5**



# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

**Busquem: el 5**



**3 < 5.** Descartem la part inferior

# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

**Busquem: el 5**



# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

**Busquem: el 5**



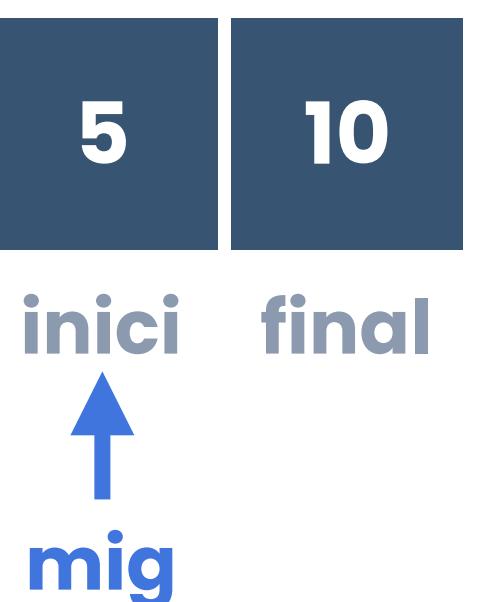
# Cerca en conjunts ordenats: Cerca binària

\* O cerca dicotòmica / bisecció

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

**Busquem: el 5**



**5 = 5.** Trobat!

# Cerca en conjunts ordenats: Cerca binària

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

```
$ Suposem ordenació creixent  
funció cerca_binària (vector: taula[ ] d'enter,  
mida: enter, clau: enter) retorna booleà és
```

**ffunció**

# Cerca en conjunts ordenats: Cerca binària

## Funcionament:

- Requereix que el conjunt de registres estigui ordenat.
- Començar a cercar pel valor a la meitat de la taula.  
Això ens determina quina meitat de la taula explorarem a continuació i quina descartarem. Amb la meitat que ens quedem, repetim aquest procés.

```
$ Suposem ordenació creixent
funció cerca_binària (vector: taula[] d'enter,
mida: enter, clau: enter) retorna booleà és
var
    ini, fi, mig: enter;
    trobat: booleà;
fvar
inici
    ini := 0;
    fi := mida-1;
    trobat := fals;
mentre (ini <= fi i no(trobat)) fer
    mig := (ini+fi) div 2;
    si (vector[mig] < clau)
        $ Descartem la meitat inferior
        ini := mig + 1;
    sino si (vector[mig] > clau)
        $ Descartem la meitat superior
        fi := mig - 1;
    sino $ Trobat a la posició "mig"
        trobat := cert;
fsi
fmentre
retorna (trobat);
ffunció
```

# Cerca en conjunts ordenats: Cerca binària

## Variacions:

- Si hi ha valors repetits, aquest algorisme no assegura quin dels valors repetits retornarà
  - Variació per retornar sempre l'element esquerre
  - Variació per retornar sempre l'element dret
  - ....

```
$ Suposem ordenació creixent
funció cerca_binària (vector: taula[] d'enter,
mida: enter, clau: enter) retorna booleà és
var
    ini, fi, mig: enter;
    trobat: booleà;
fvar
inici
    ini := 0;
    fi := mida-1;
    trobat := fals;
mentre (ini <= fi i no(trobat)) fer
    mig := (ini+fi) div 2;
    si (vector[mig] < clau)
        $ Descartem la meitat inferior
        ini := mig + 1;
    sino si (vector[mig] > clau)
        $ Descartem la meitat superior
        fi := mig - 1;
    sino $ Trobat a la posició "mig"
        trobat := cert;
fsi
fmentre
retorna (trobat);
ffunció
```

# Cerca en conjunts ordenats: Cerca binària

## Variacions:

- Si hi ha valors repetits, aquest algorisme no assegura quin dels valors repetits retornarà
  - Variació per retornar sempre l'element esquerre
  - Variació per retornar sempre l'element dret
  - ....

## Un possible problema:

- En vectors molt grans, la suma `ini+fi`, que necessitem per calcular  $(\text{ini}+\text{fi})/2$  pot donar un overflow del tipus que utilitzem. Per evitar-ho, es pot calcular com: `ini + (fi-ini)/2`

```
$ Suposem ordenació creixent
funció cerca_binària (vector: taula[] d'enter,
mida: enter, clau: enter) retorna booleà és
var
    ini, fi, mig: enter;
    trobat: booleà;
fvar
inici
    ini := 0;
    fi := mida-1;
    trobat := fals;
mentre (ini <= fi i no(trobat)) fer
    mig := (ini+fi) div 2;
    si (vector[mig] < clau)
        $ Descartem la meitat inferior
        ini := mig + 1;
    sino si (vector[mig] > clau)
        $ Descartem la meitat superior
        fi := mig - 1;
    sino $ Trobat a la posició "mig"
        trobat := cert;
fsi
fmentre
retorna (trobat);
ffunció
```

# Cerca en conjunts ordenats: Cerca binària

## Preguntes per pensar-hi:

- Què passaria si en comptes de que la condició fos  
`ini <= fi`, aquesta fos `ini < fi`?

```
$ Suposem ordenació creixent
funció cerca_binària (vector: taula[] d'enter,
mida: enter, clau: enter) retorna booleà és
var
    ini, fi, mig: enter;
    trobat: booleà;
fvar
inici
    ini := 0;
    fi := mida-1;
    trobat := fals;
mentre (ini <= fi i no(trobat)) fer
    mig := (ini+fi) div 2;
    si (vector[mig] < clau)
        $ Descartem la meitat inferior
        ini := mig + 1;
    sino si (vector[mig] > clau)
        $ Descartem la meitat superior
        fi := mig - 1;
    sino $ Trobat a la posició "mig"
        trobat := cert;
fsi
fmentre
retorna (trobat);
ffunció
```

# Cerca en conjunts ordenats: Cerca binària

## Preguntes per pensar-hi:

- Què passaria si en comptes de que la condició fos `ini <= fi`, aquesta fos `ini < fi`?
- Què passaria si al reduir el conjunt de dades no excloc `mig`? És a dir, faig `ini := mig` en comptes de `ini := mig+1` (i el mateix per a `fi`)?

```
$ Suposem ordenació creixent
funció cerca_binària (vector: taula[] d'enter,
mida: enter, clau: enter) retorna booleà és
var
    ini, fi, mig: enter;
    trobat: booleà;
fvar
inici
    ini := 0;
    fi := mida-1;
    trobat := fals;
mentre (ini <= fi i no(trobat)) fer
    mig := (ini+fi) div 2;
    si (vector[mig] < clau)
        $ Descartem la meitat inferior
        ini := mig + 1;
    sino si (vector[mig] > clau)
        $ Descartem la meitat superior
        fi := mig - 1;
    sino $ Trobat a la posició "mig"
        trobat := cert;
fsi
fmentre
retorna (trobat);
ffunció
```

# Cerca en conjunts ordenats: Cerca binària

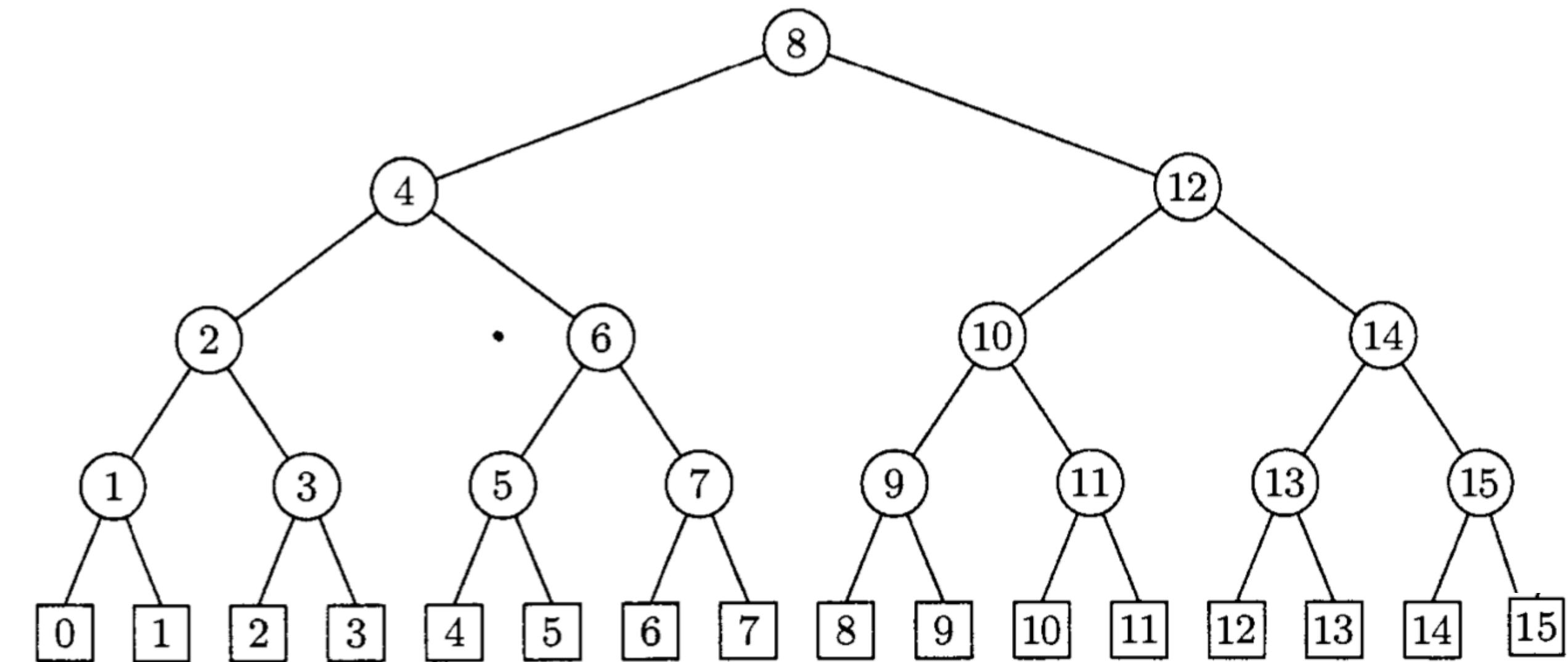
## Preguntes per pensar-hi:

- Què passaria si en comptes de que la condició fos `ini <= fi`, aquesta fos `ini < fi`?
- Què passaria si al reduir el conjunt de dades no excloc `mig`? És a dir, faig `ini := mig` en comptes de `ini := mig+1` (i el mateix per a `fi`)?
- Què hauria de canviar d'aquest codi si el vector estés ordenat decreixentment?

```
$ Suposem ordenació creixent
funció cerca_binària (vector: taula[] d'enter,
mida: enter, clau: enter) retorna booleà és
var
    ini, fi, mig: enter;
    trobat: booleà;
fvar
inici
    ini := 0;
    fi := mida-1;
    trobat := fals;
mentre (ini <= fi i no(trobat)) fer
    mig := (ini+fi) div 2;
    si (vector[mig] < clau)
        $ Descartem la meitat inferior
        ini := mig + 1;
    sino si (vector[mig] > clau)
        $ Descartem la meitat superior
        fi := mig - 1;
    sino $ Trobat a la posició "mig"
        trobat := cert;
fsi
fmentre
retorna (trobat);
ffunció
```

# Cerca en conjunts ordenats: Cerca binària

## Anàlisi de costos



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

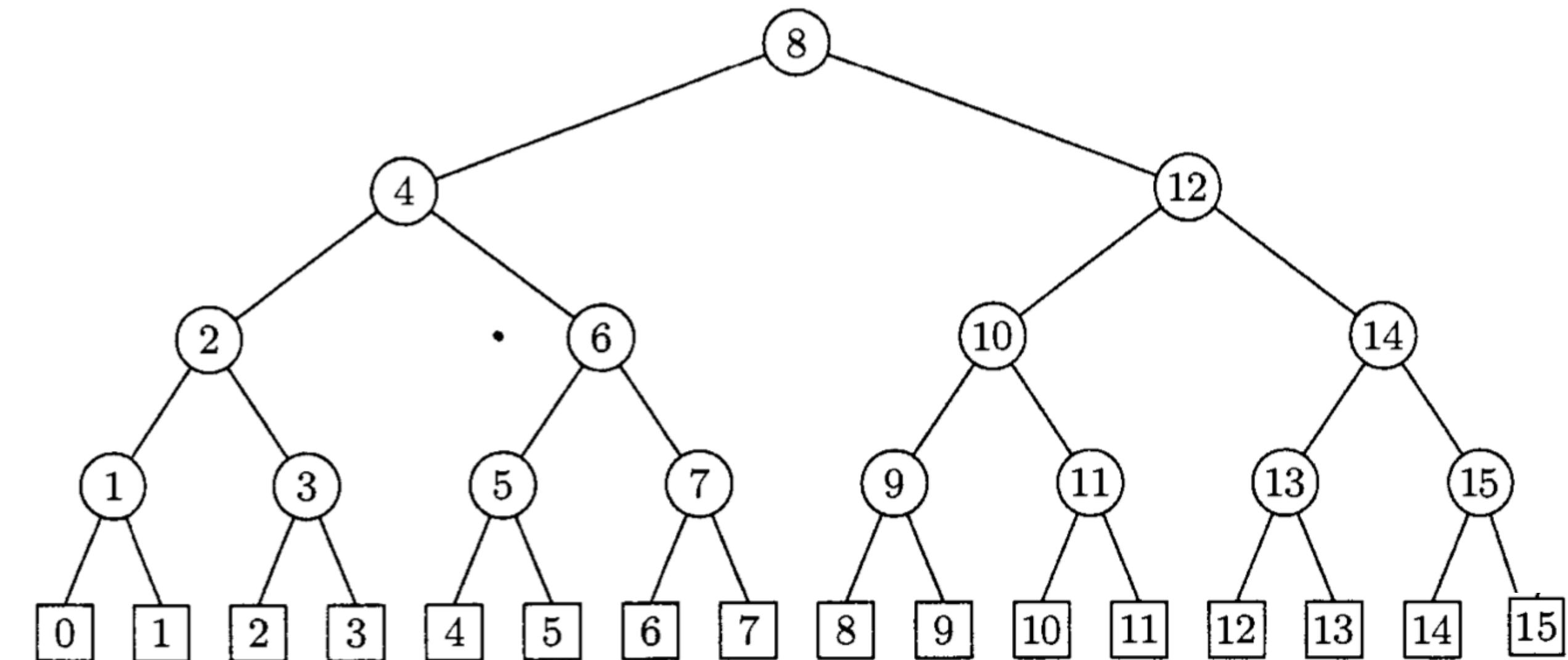
# Cerca en conjunts ordenats: Cerca binària

## Anàlisi de costos

- En el **millor cas**, l'element està a la posició del mig.

Només cal una iteració.

E.g. busquem el '8'



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

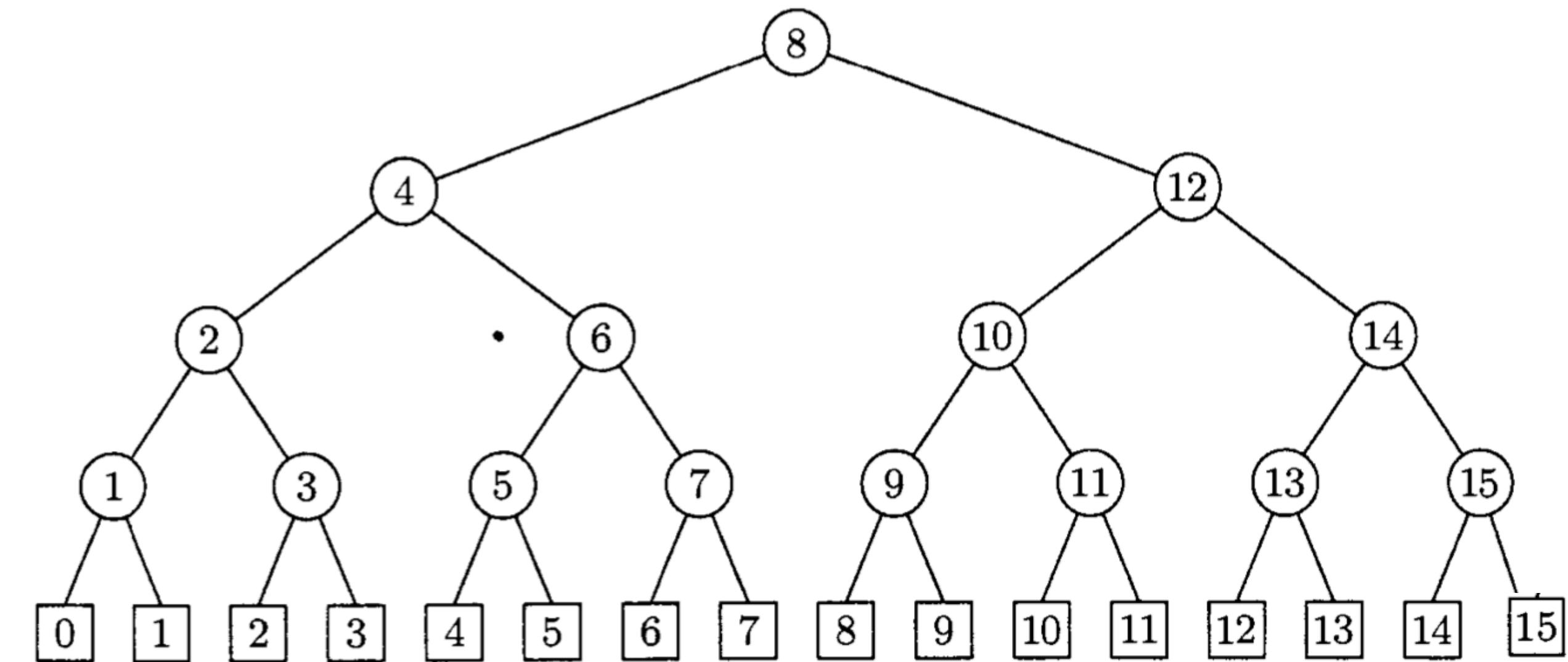
# Cerca en conjunts ordenats: Cerca binària

## Anàlisi de costos

- En el **millor cas**, l'element està a la posició del mig.  
Només cal una iteració.

E.g. busquem el '8'

**Best case =  $\mathcal{O}(1)$**



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

# Cerca en conjunts ordenats: Cerca binària

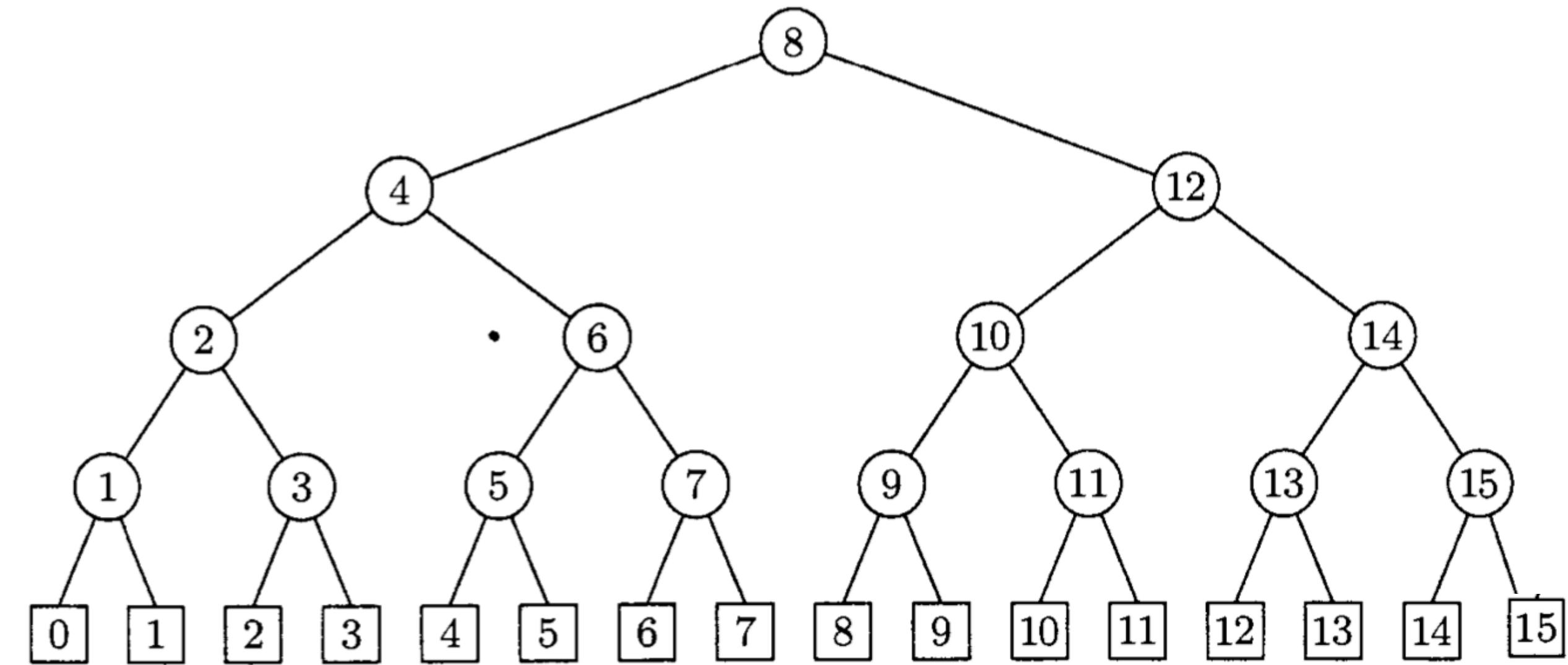
## Anàlisi de costos

- En el **millor cas**, l'element està a la posició del mig.  
Només cal una iteració.

E.g. busquem el '8'

**Best case =  $\mathcal{O}(1)$**

- En el **pitjor cas**, l'element està al nivell més profund  
de l'arbre. Quantes iteracions es fan?



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

# Cerca en conjunts ordenats: Cerca binària

## Anàlisi de costos

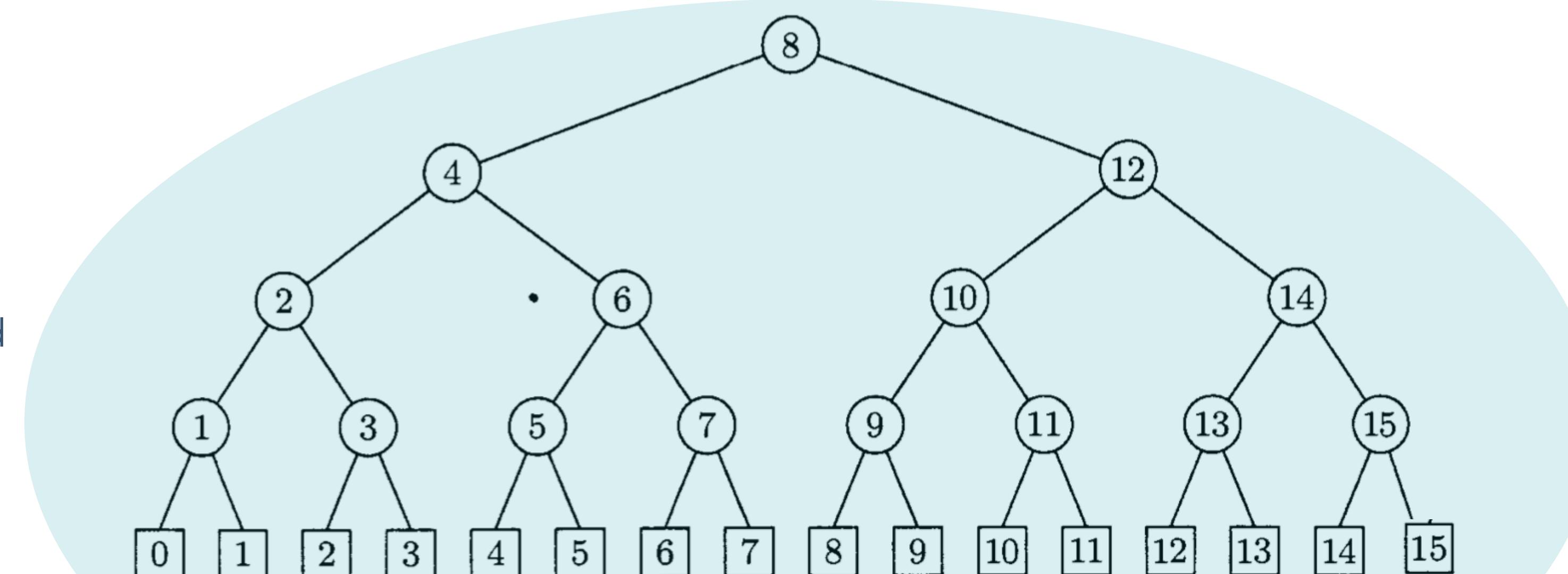
- En el **millor cas**, l'element està a la posició del mig.  
Només cal una iteració.

E.g. busquem el '8'

**Best case =  $\mathcal{O}(1)$**

- En el **pitjor cas**, l'element està al nivell més profund  
de l'arbre. Quantes iteracions es fan?

**Busquem: el 5**



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

# Cerca en conjunts ordenats: Cerca binària

## Anàlisi de costos

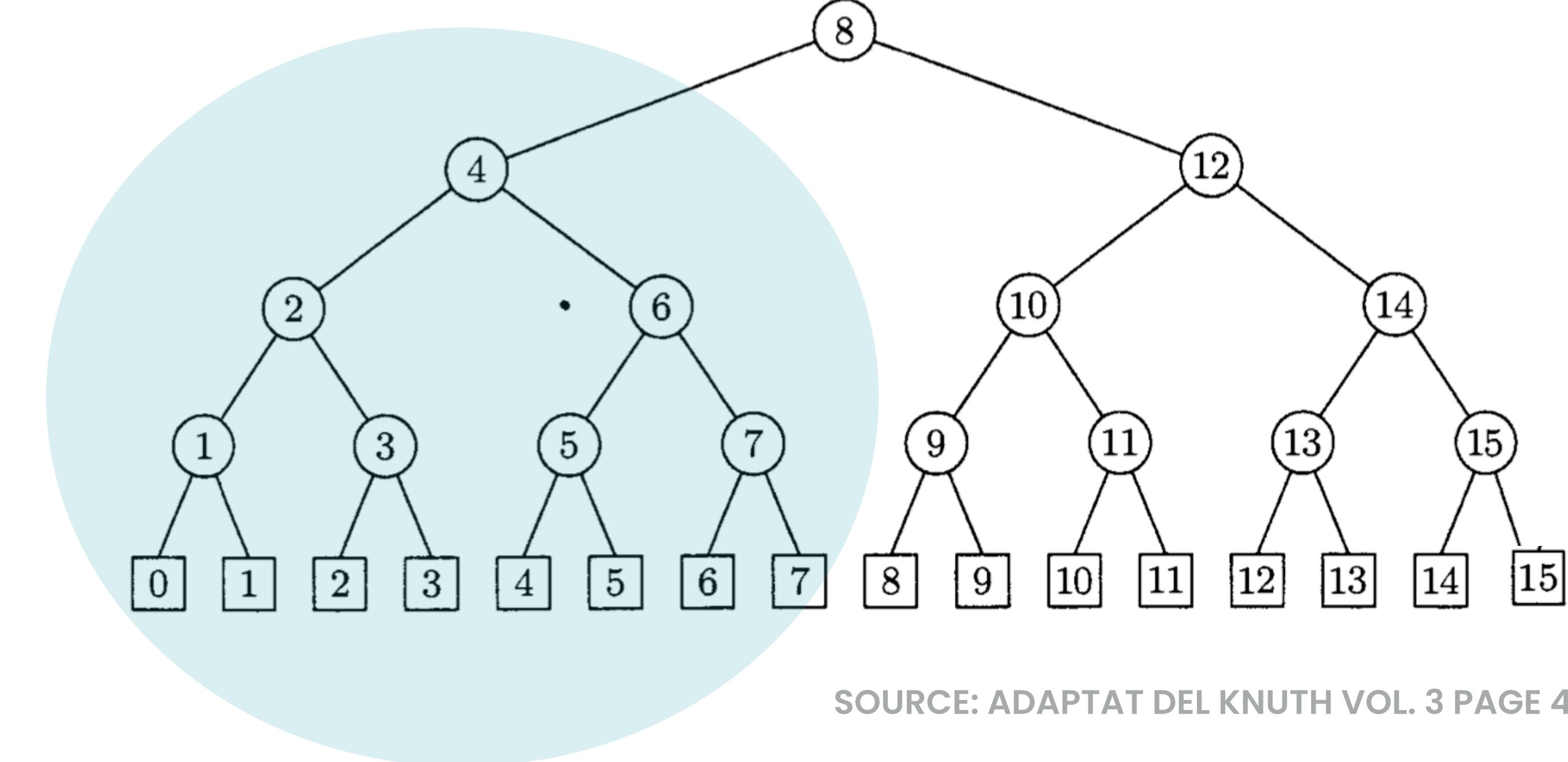
- En el **millor cas**, l'element està a la posició del mig.  
Només cal una iteració.

E.g. busquem el '8'

**Best case =  $\mathcal{O}(1)$**

- En el **pitjor cas**, l'element està al nivell més profund  
de l'arbre. Quantes iteracions es fan?

**Busquem: el 5**



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

# Cerca en conjunts ordenats: Cerca binària

## Anàlisi de costos

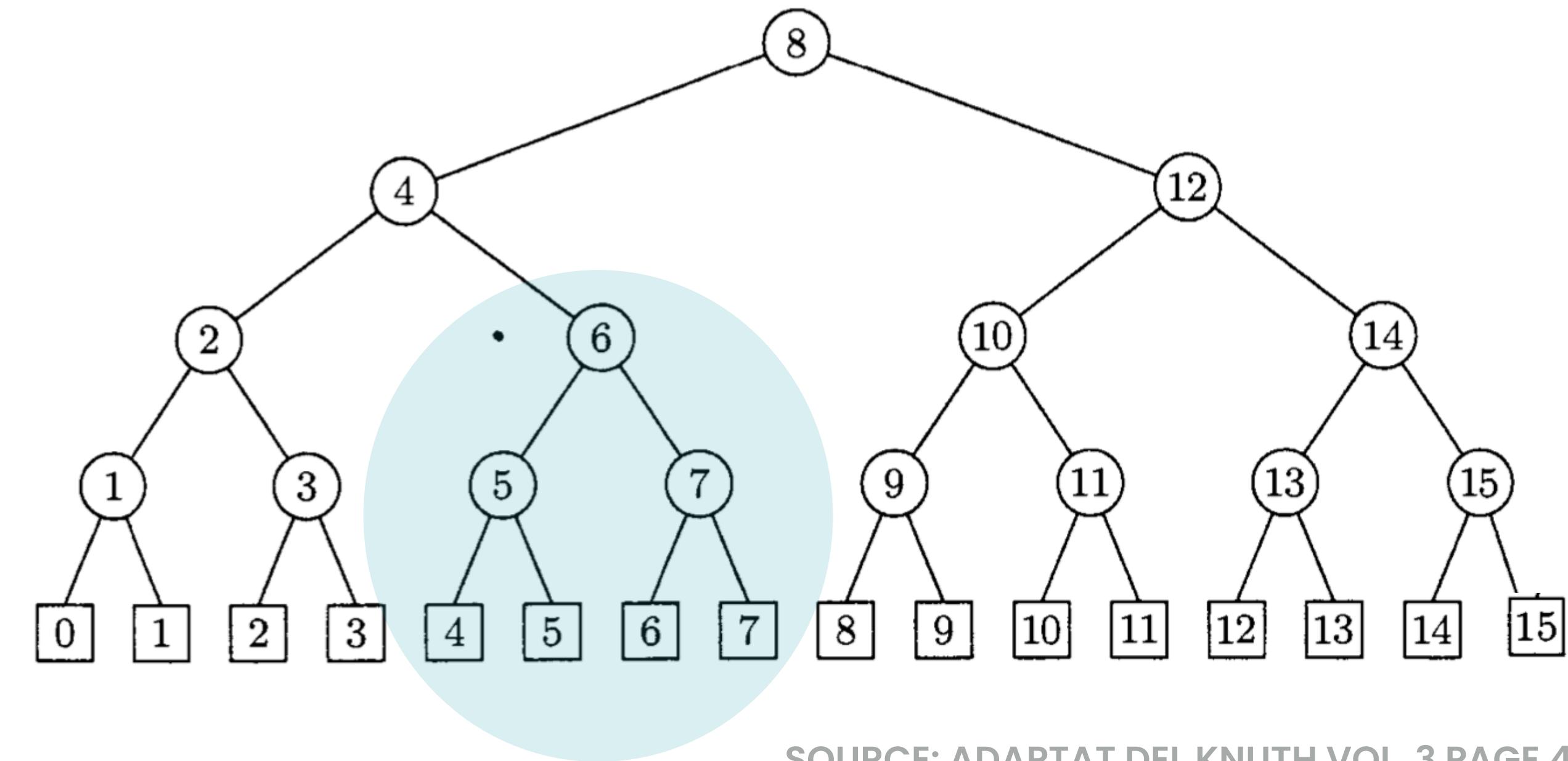
- En el **millor cas**, l'element està a la posició del mig.  
Només cal una iteració.

E.g. busquem el '8'

**Best case =  $\mathcal{O}(1)$**

- En el **pitjor cas**, l'element està al nivell més profund  
de l'arbre. Quantes iteracions es fan?

**Busquem: el 5**



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

# Cerca en conjunts ordenats: Cerca binària

## Anàlisi de costos

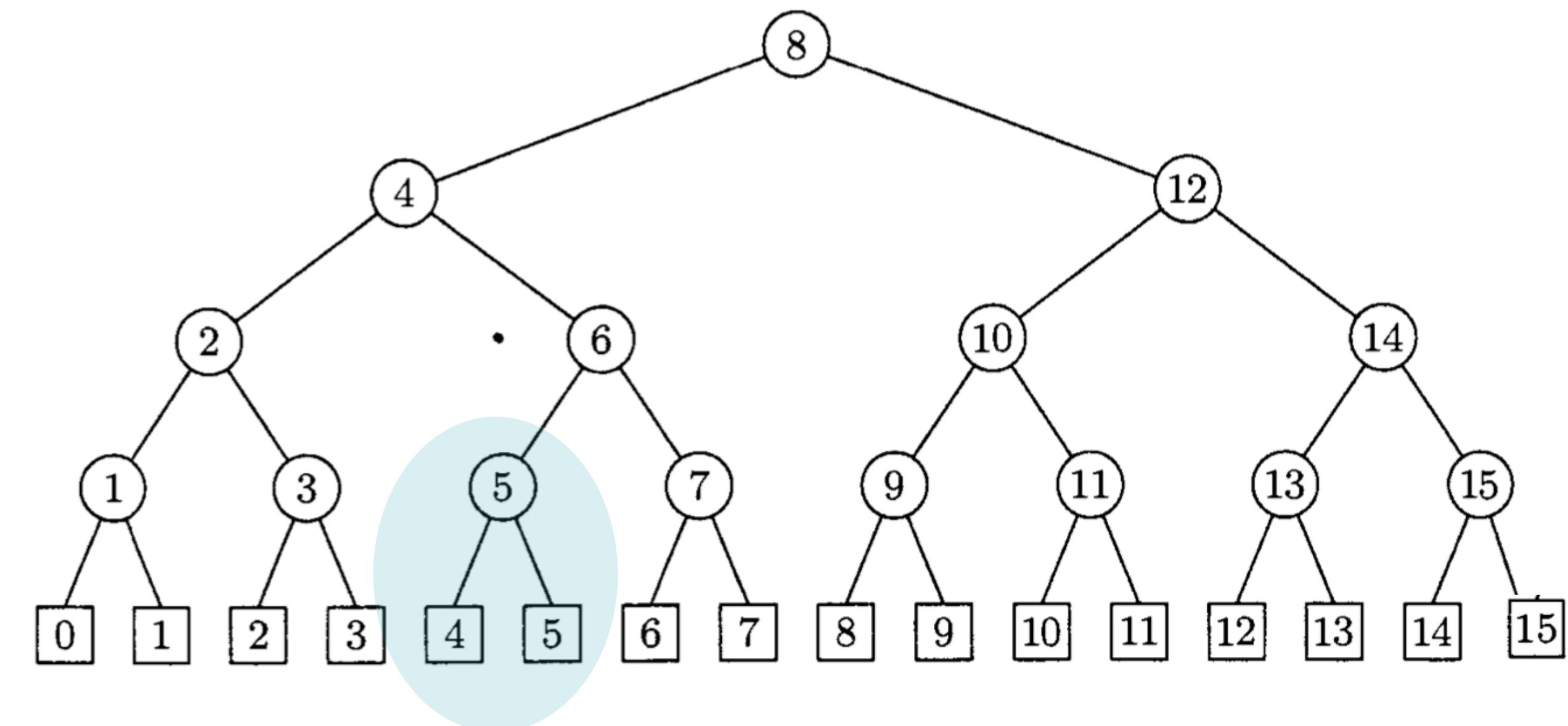
- En el **millor cas**, l'element està a la posició del mig.  
Només cal una iteració.

E.g. busquem el '8'

**Best case =  $\mathcal{O}(1)$**

- En el **pitjor cas**, l'element està al nivell més profund  
de l'arbre. Quantes iteracions es fan?

**Busquem: el 5**



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

# Cerca en conjunts ordenats: Cerca binària

## Anàlisi de costos

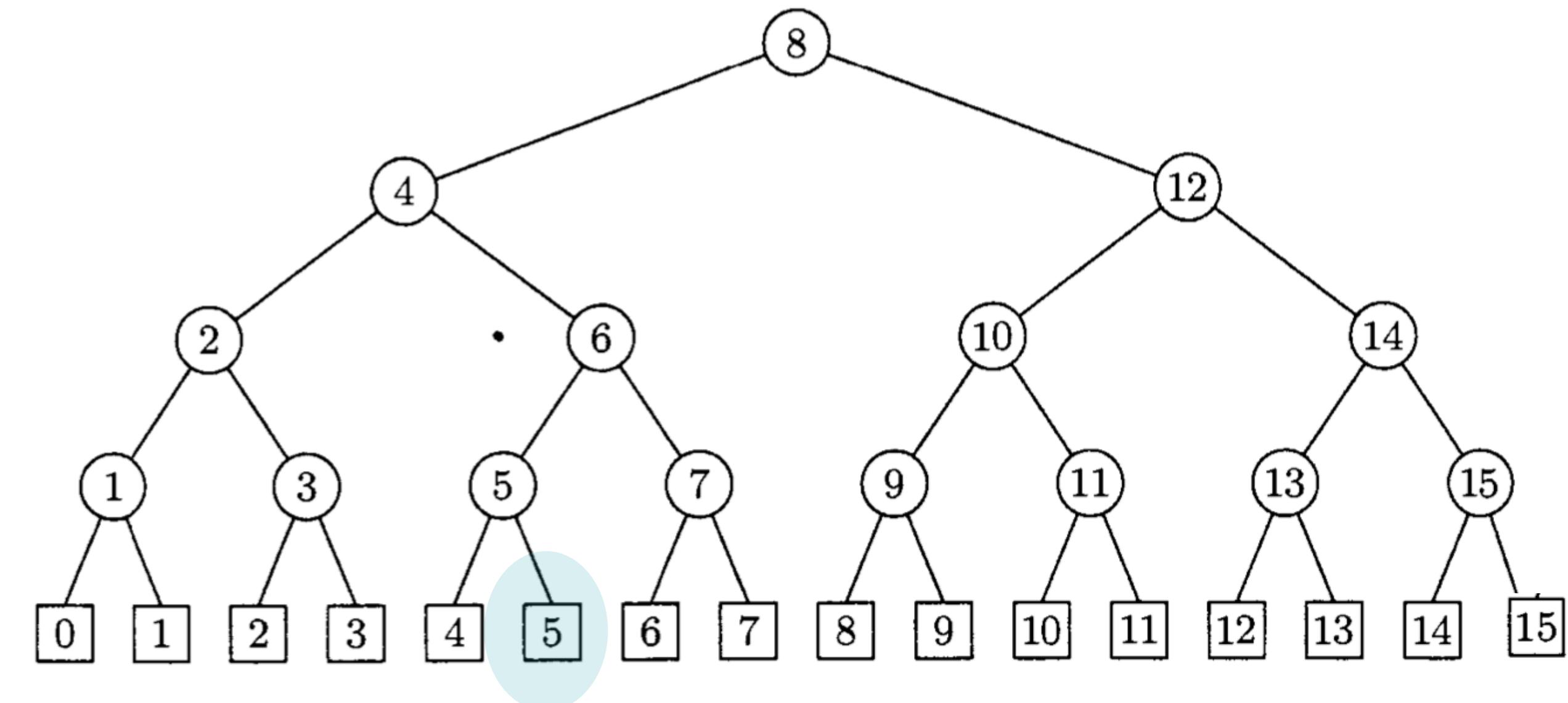
- En el **millor cas**, l'element està a la posició del mig.  
Només cal una iteració.

E.g. busquem el '8'

**Best case =  $\mathcal{O}(1)$**

- En el **pitjor cas**, l'element està al nivell més profund  
de l'arbre. Quantes iteracions es fan?

**Busquem: el 5**



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

# Cerca en conjunts ordenats: Cerca binària

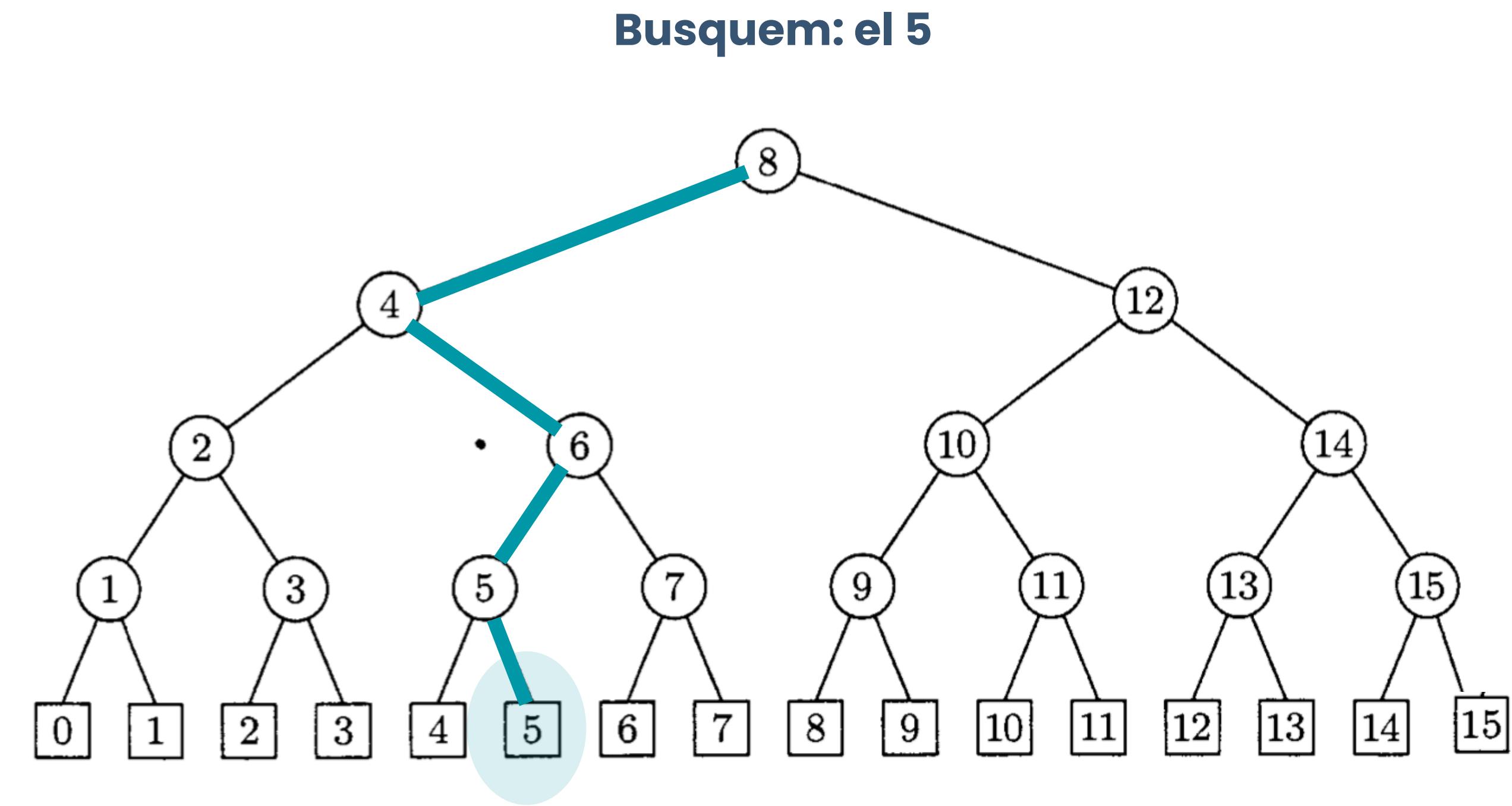
## Anàlisi de costos

- En el **millor cas**, l'element està a la posició del mig.  
Només cal una iteració.

E.g. busquem el '8'

**Best case =  $\mathcal{O}(1)$**

- En el **pitjor cas**, l'element està al nivell més profund  
de l'arbre. Quantes iteracions es fan?



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

# Cerca en conjunts ordenats: Cerca binària

## Anàlisi de costos

- En el **millor cas**, l'element està a la posició del mig.  
Només cal una iteració.

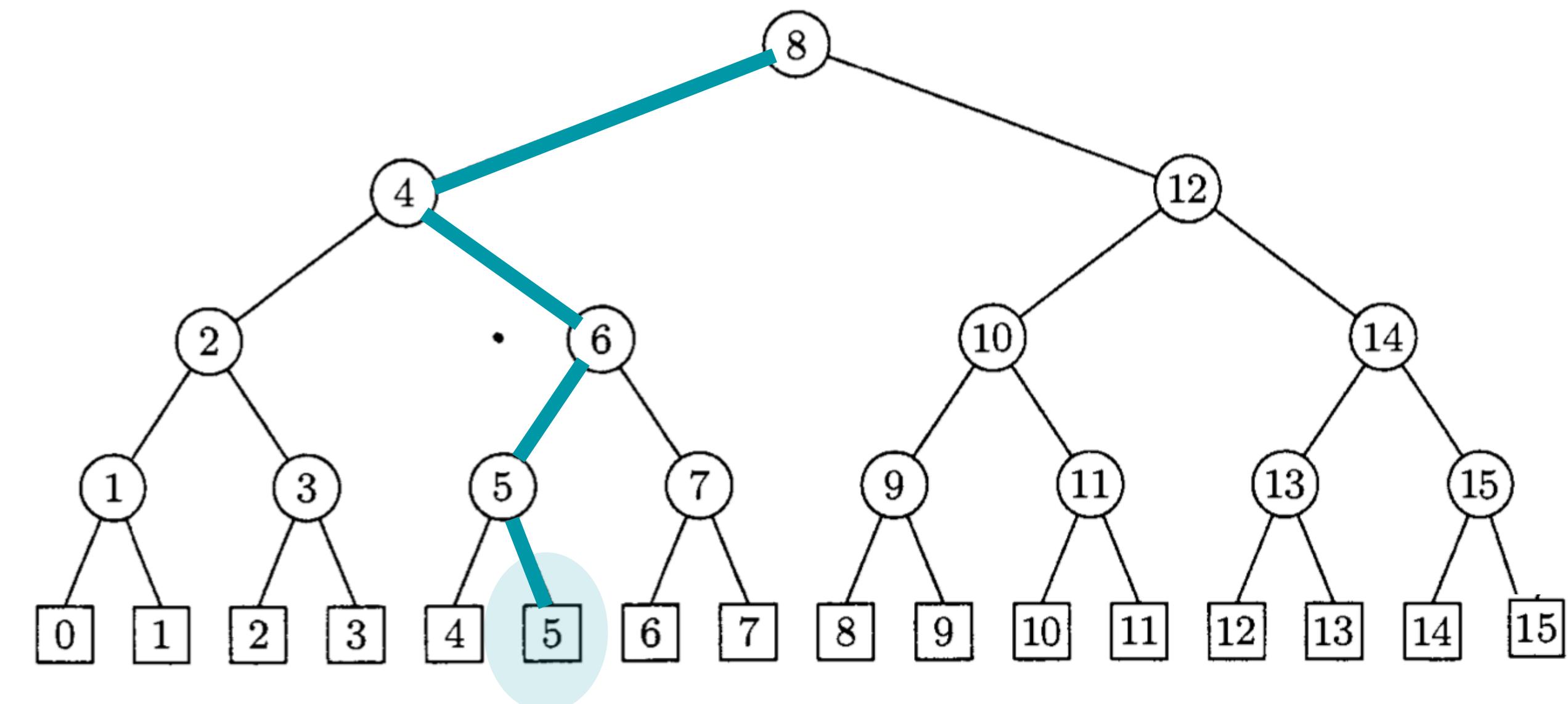
E.g. busquem el '8'

**Best case =  $\mathcal{O}(1)$**

- En el **pitjor cas**, l'element està al nivell més profund  
de l'arbre. Quantes iteracions es fan?

**Worst case =  $\mathcal{O}(\log n)$**

**Busquem: el 5**



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

# Cerca en conjunts ordenats: Cerca binària

## Anàlisi de costos

- En el **millor cas**, l'element està a la posició del mig.  
Només cal una iteració.

E.g. busquem el '8'

**Best case =  $\mathcal{O}(1)$**

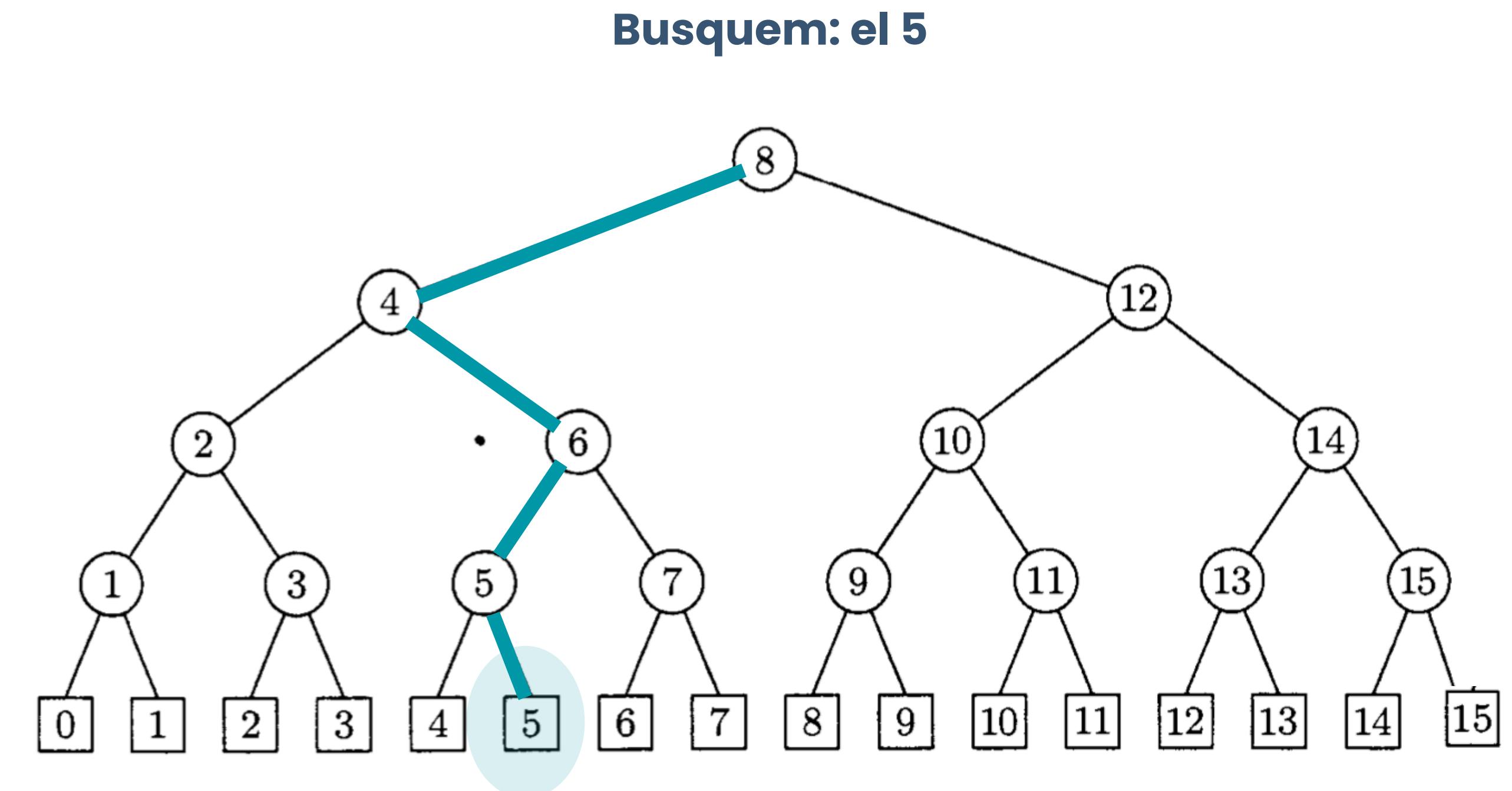
- En el **pitjor cas**, l'element està al nivell més profund  
de l'arbre. Quantes iteracions es fan?

**Worst case =  $\mathcal{O}(\log n)$**

- En el **cas mig**, s'han de tenir en compte els casos  
exitosos vs. no exitosos. Càcul aquí:

[https://en.wikipedia.org/wiki/Binary\\_search\\_algorithm#Derivation\\_of\\_average\\_case](https://en.wikipedia.org/wiki/Binary_search_algorithm#Derivation_of_average_case)

**Average case =  $\mathcal{O}(\log n)$**



SOURCE: ADAPTAT DEL KNUTH VOL. 3 PAGE 412

# Cerca en conjunts ordenats: Cerca binària

## Versió recursiva

```
int cerca_binaria_rec(int *vector, int clau, int ini, int end){
    // Calcular el punt mig.
    int mid = ini + (end - ini)/2;           // Divisió entera!

    if (ini > end)                          // Cas directe: no trobat
        return -1;
    else if (vector[mid] == clau)           // Trobat, retorna índex
        return mid;
    else if (vector[mid] > clau)            //Buscar a la part inferior
        return cerca_binaria_rec(vector, clau, ini, mid-1);
    else                                    //Buscar a la part superior
        return cerca_binaria_rec(vector, clau, mid+1, end);
}
```

# Cerca en conjunts ordenats: Cerca binària

## Versió recursiva

```
int cerca_binaria(int *vector, int clau, int num_elems){  
    // Inici = 0  
    // Final = num_elems - 1  
    return cerca_binaria_rec(vector, clau, 0, num_elems-1);  
}
```

FUNCió WRAPPER

```
int cerca_binaria_rec(int *vector, int clau, int ini, int end){  
    // Calcular el punt mig.  
    int mid = ini + (end - ini)/2;           // Divisió entera!  
  
    if (ini > end)                          // Cas directe: no trobat  
        return -1;  
    else if (vector[mid] == clau)            // Trobat, retorna índex  
        return mid;  
    else if (vector[mid] > clau)             //Buscar a la part inferior  
        return cerca_binaria_rec(vector, clau, ini, mid-1);  
    else                                    //Buscar a la part superior  
        return cerca_binaria_rec(vector, clau, mid+1, end);  
}
```

VERSió RECURSIVA DE L'ALGORISME DE CERCA BINÀRIA

# Cerca en conjunts ordenats: Cerca binària

## Versió recursiva

### Comparativa dels costos

#### Cost temporal:

- Tant la versió iterativa com la recursiva fan com a màxim  $\lfloor \log(n) \rfloor + 1$  comparacions en el pitjor cas

#### Cost espacial:

- A la pila, la versió iterativa té un cost constant  $\mathcal{O}(1)$
- En canvi, la versió recursiva apilarà tants stack frames com crides:  $\mathcal{O}(\log n)$

```
int cerca_binaria(int *vector, int clau, int num_elems){  
    // Inici = 0  
    // Final = num_elems - 1  
    return cerca_binaria_rec(vector, clau, 0, num_elems-1);  
}
```

FUNCÍÓ WRAPPER

```
int cerca_binaria_rec(int *vector, int clau, int ini, int end){  
    // Calcular el punt mig.  
    int mid = ini + (end - ini)/2;           // Divisió entera!  
  
    if (ini > end)                          // Cas directe: no trobat  
        return -1;  
    else if (vector[mid] == clau)            // Trobat, retorna índex  
        return mid;  
    else if (vector[mid] > clau)             // Buscar a la part inferior  
        return cerca_binaria_rec(vector, clau, ini, mid-1);  
    else                                     // Buscar a la part superior  
        return cerca_binaria_rec(vector, clau, mid+1, end);  
}
```

VERSIÓ RECURSIVA DE L'ALGORISME DE CERCA BINÀRIA

# **ALGORISMES**

## **D'ORDENACIÓ**

# Ordenació

## Necessitat i definicions

- Mantenir un conjunt de dades ordenat ens serveix per poder fer cerques més eficients (tot i que no surten a compte si només s'han de fer una vegada)
- Alguns processos matemàtics requereixen de l'ordenació d'elements. E.g. trobar la mediana, els quartils...

Cada element és un registre  $r$  amb clau  $k$

Definim que tenim els següents registres:

$$r_1, r_2, \dots, r_n$$

Ordenar consisteix en permutar aquests registres

$$a_{k_1}, a_{k_2}, \dots, a_{k_n}$$

De manera que donada una funció d'ordenació  $f$  es verifiqui:

$$f(a_{k_1}) \leq f(a_{k_2}) \leq \dots \leq f(a_{k_n})$$



SOURCE: WIRTH PAGE 62

# Ordenació

## Necessitat i definicions

- Mantenir un conjunt de dades ordenat ens serveix per poder fer cerques més eficients (tot i que no surten a compte si només s'han de fer una vegada)
- Alguns processos matemàtics requereixen de l'ordenació d'elements. E.g. trobar la mediana, els quartils...

Cada element és un registre  $r$  amb clau  $k$

Definim que tenim els següents registres:

$$r_1, r_2, \dots, r_n$$

Ordenar consisteix en permutar aquests registres

$$a_{k_1}, a_{k_2}, \dots, a_{k_n}$$

De manera que donada una funció d'ordenació  $f$  es verifiqui:

$$f(a_{k_1}) \leq f(a_{k_2}) \leq \dots \leq f(a_{k_n})$$



SOURCE: WIRTH PAGE 62

- Un mètode d'ordenació es denomina **estable** si l'ordre relatiu dels registres amb la mateixa clau no s'altera pel procés d'ordenació
- L'ordenació de taules es realitza **in situ**, és a dir, no es declara una altra taula que s'omple amb els valors ordenats, sinó que només es pot fer servir la memòria que ocupa la nostra taula inicial.

# Ordenació

## Mètodes directes d'ordenació

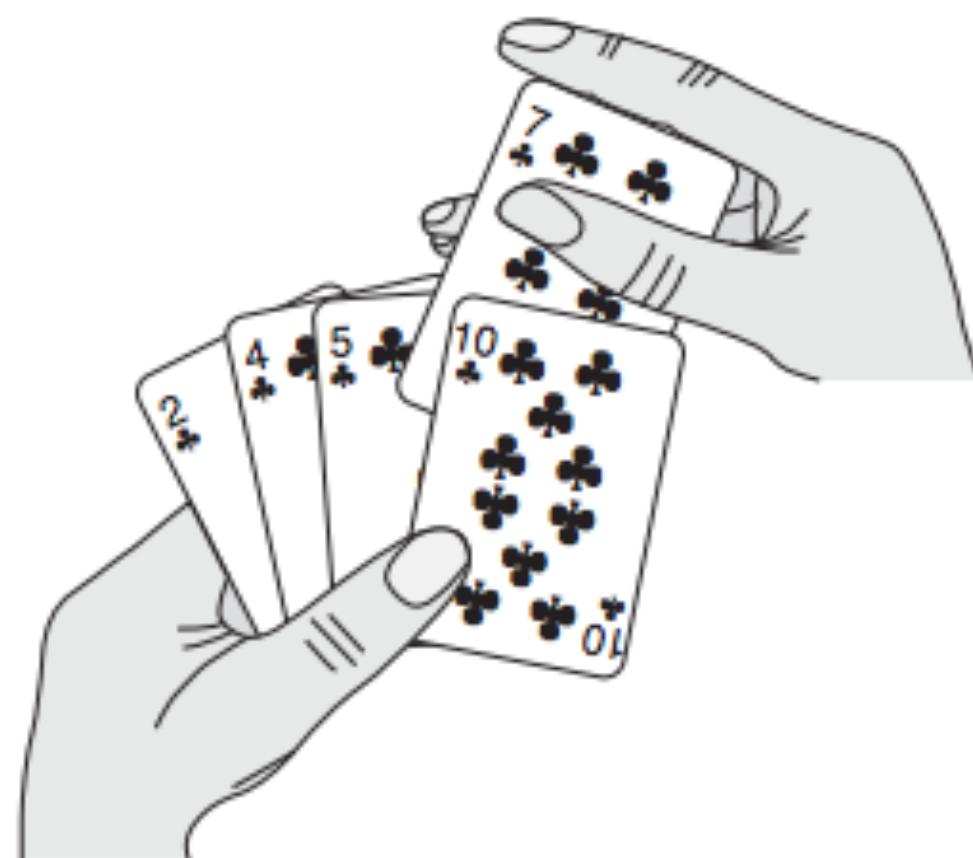
- Són els mètodes més bàsics, tot i no ser els més eficients
- Són importants perquè estableixen les bases de molts altres mètodes

# Ordenació

## Mètodes directes d'ordenació

- Són els mètodes més bàsics, tot i no ser els més eficients
- Són importants perquè estableixen les bases de molts altres mètodes

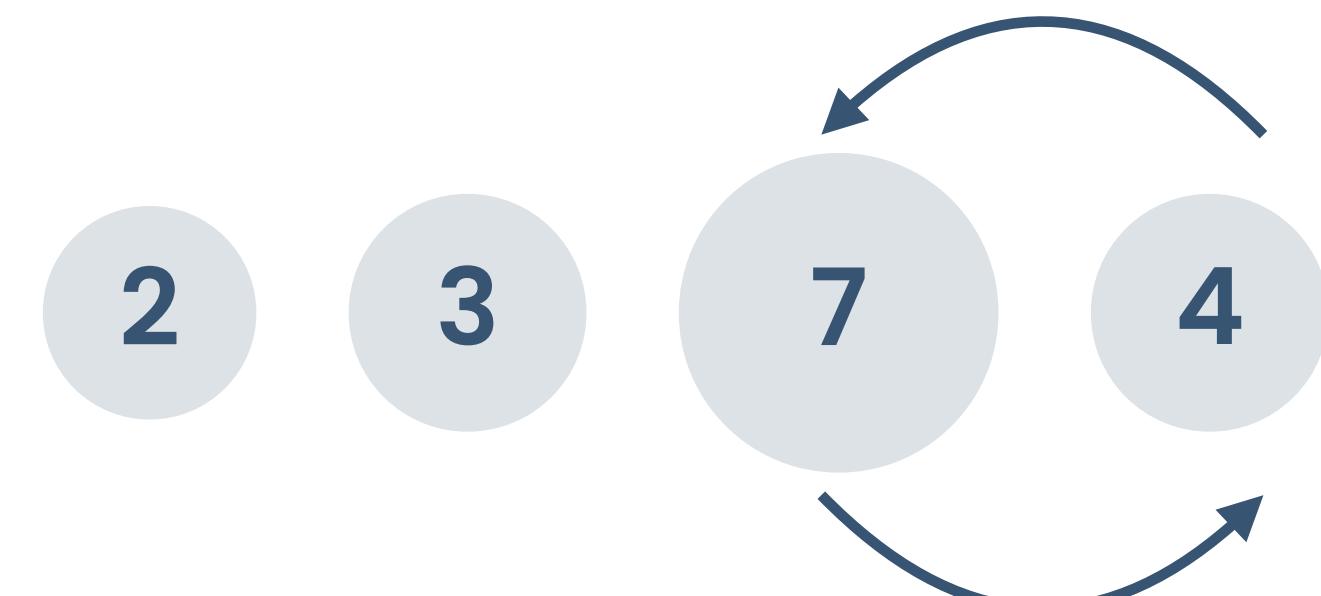
### Ordenació per inserció



### Ordenació per selecció



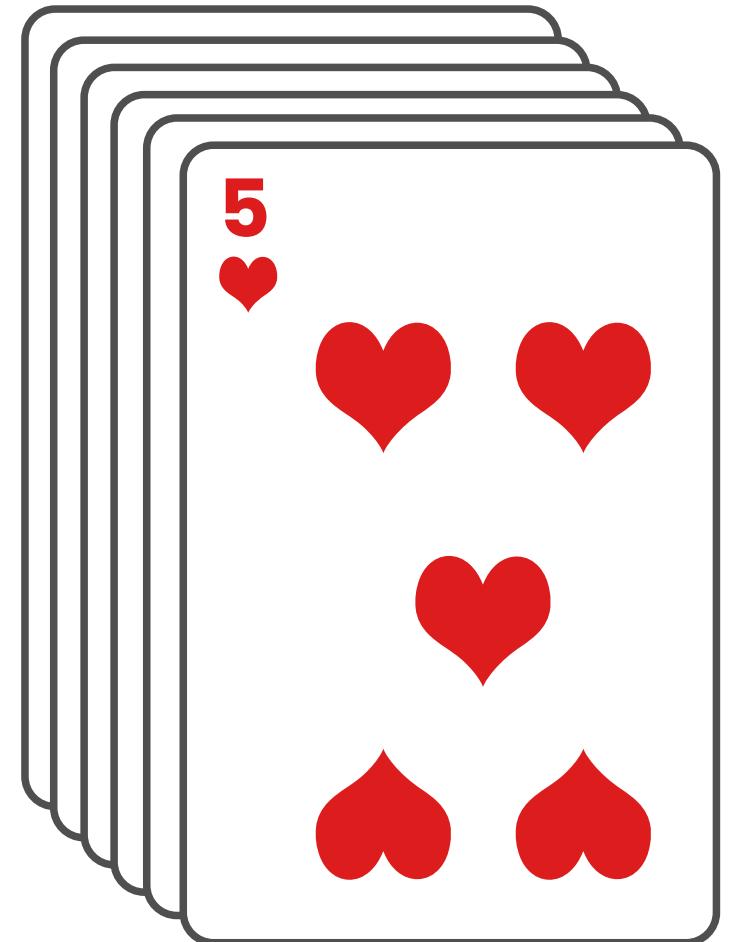
### Ordenació per intercanvi (bombolla)



## Ordenació per inserció

- **El mètode dels jugadors de cartes:**

Es basa en la mateixa estratègia que fan servir les persones quan ordenen cartes a la mà durant una partida: van col·locant cada nova carta en la posició que li correspon dins de les que ja tenen ordenades.



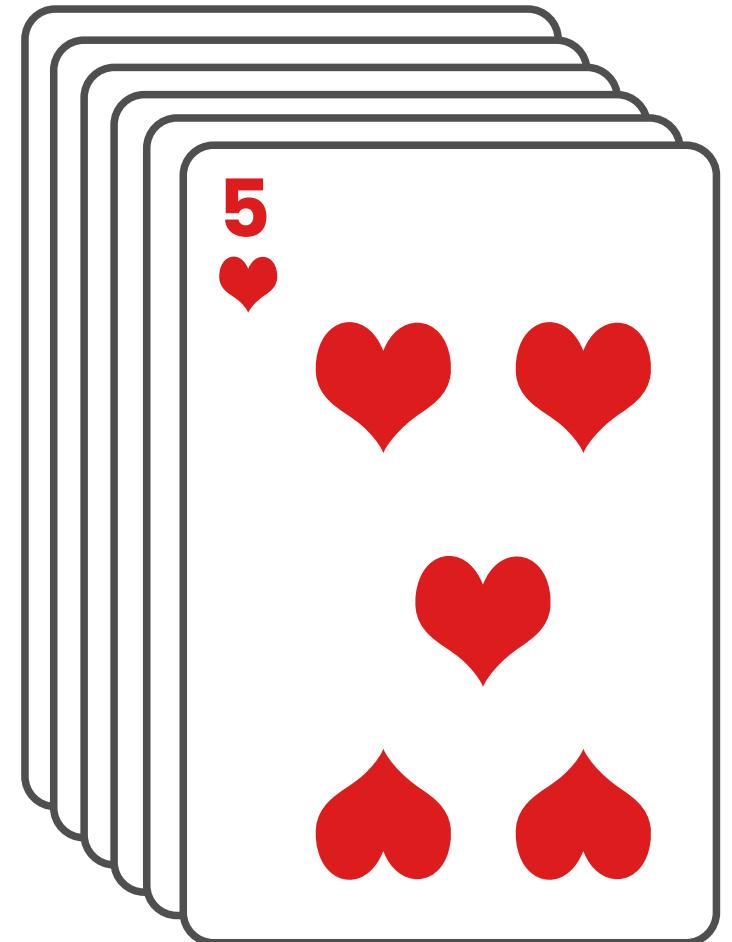
## Ordenació per inserció

- **El mètode dels jugadors de cartes:**

Es basa en la mateixa estratègia que fan servir les persones quan ordenen cartes a la mà durant una partida: van col·locant cada nova carta en la posició que li correspon dins de les que ja tenen ordenades.

- **Idea clau:**

Partim d'una seqüència de claus inicial  $K_1, K_2, \dots, K_n$  i volem arribar a una seqüència de claus ordenada. L'algorisme suposa que, a cada pas  $i$ , la subaula  $K_0, \dots, K_{i-1}$  ja està ordenada, i inserim la clau  $K_i$  en el lloc que li correspon dins d'aquesta subseqüència.



## Ordenació per inserció

- **El mètode dels jugadors de cartes:**

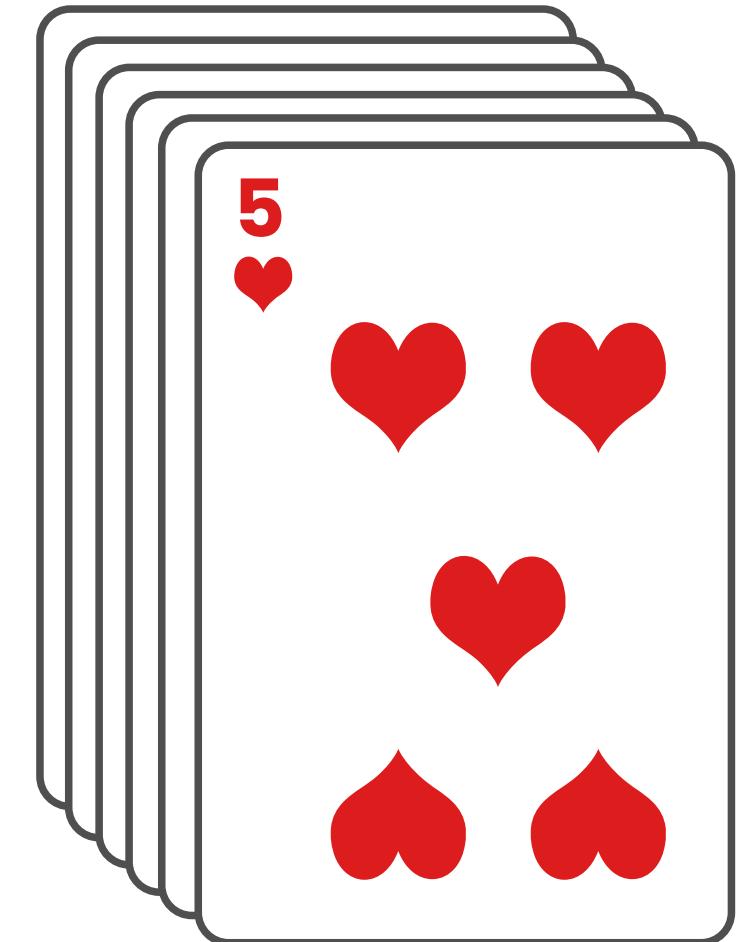
Es basa en la mateixa estratègia que fan servir les persones quan ordenen cartes a la mà durant una partida: van col·locant cada nova carta en la posició que li correspon dins de les que ja tenen ordenades.

- **Idea clau:**

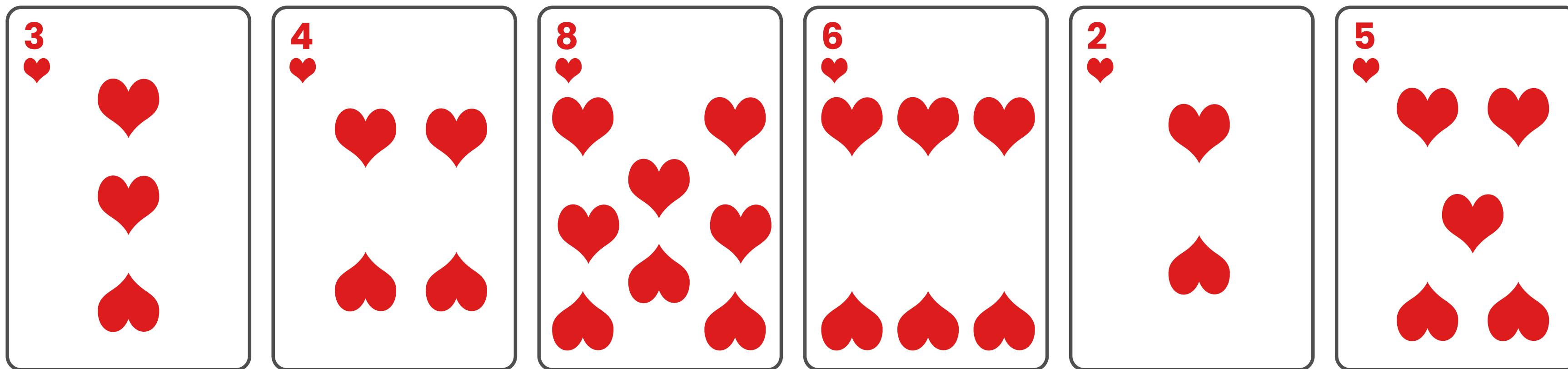
Partim d'una seqüència de claus inicial  $K_1, K_2, \dots, K_n$  i volem arribar a una seqüència de claus ordenada. L'algorisme suposa que, a cada pas  $i$ , la subaula  $K_0, \dots, K_{i-1}$  ja està ordenada, i inserim la clau  $K_i$  en el lloc que li correspon dins d'aquesta subseqüència.

- **Funcionament:**

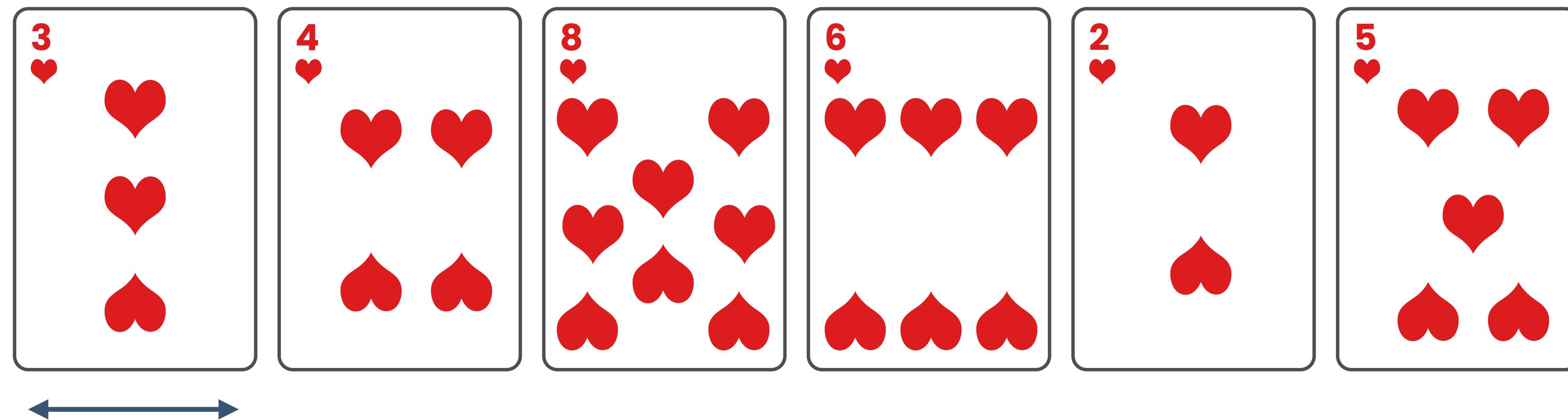
- Comencem pel segon element perquè el primer no té amb qui comparar-se.
- Cada clau  $K_i$  es compara amb els anteriors (busca un lloc on inserir-se).
- Mentre  $K_i$  sigui menor que els elements anteriors, aquests es desplacen una posició cap a la dreta per fer espai.
- Quan es troba la posició adequada, s'hi insereix la clau  $K_i$
- Això es repeteix per a totes les claus restants.



## Ordenació per inserció

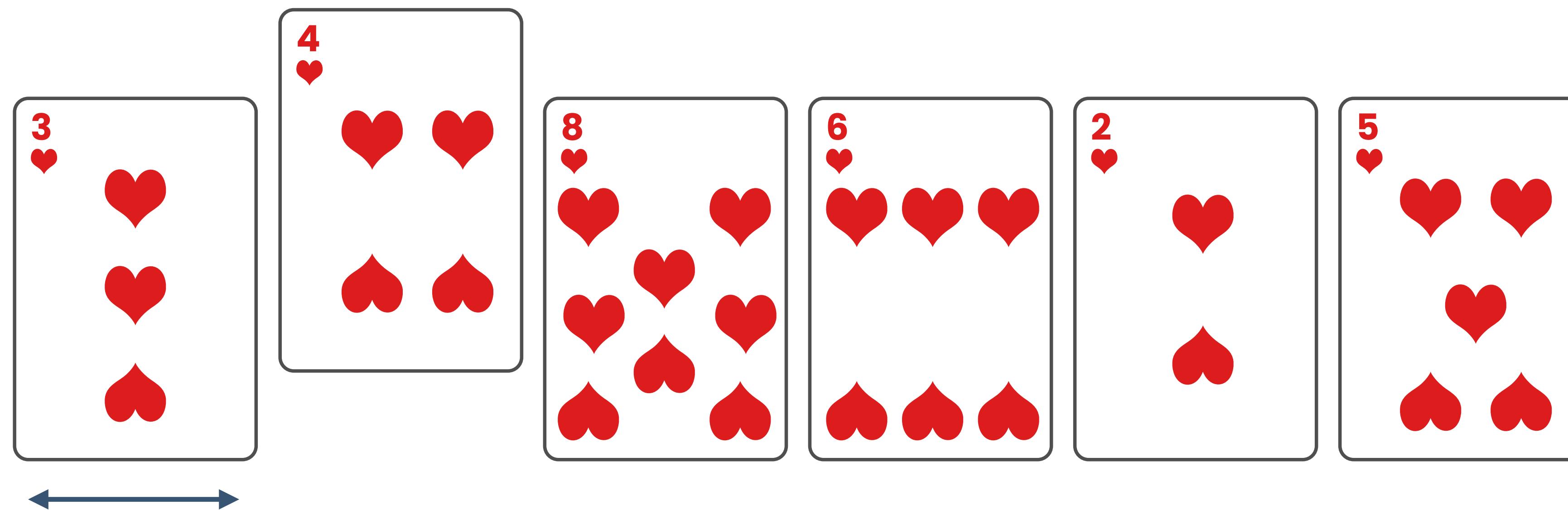


## Ordenació per inserció



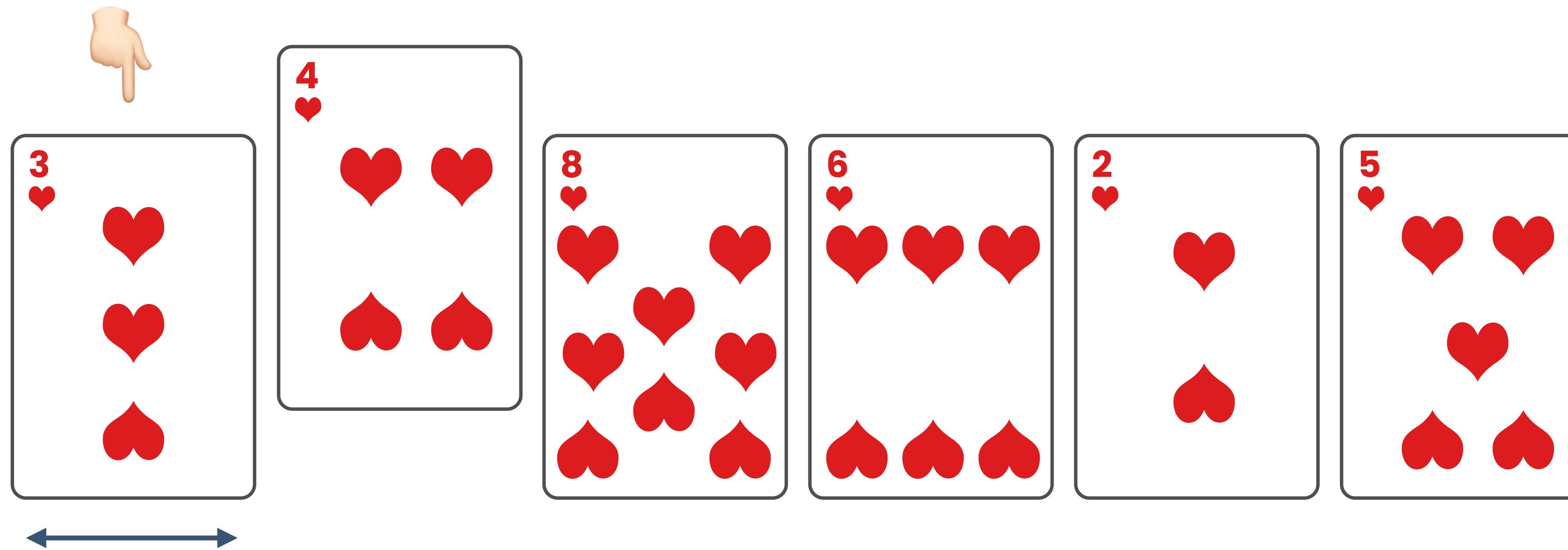
Subseqüència parcialment ordenada

## Ordenació per inserció

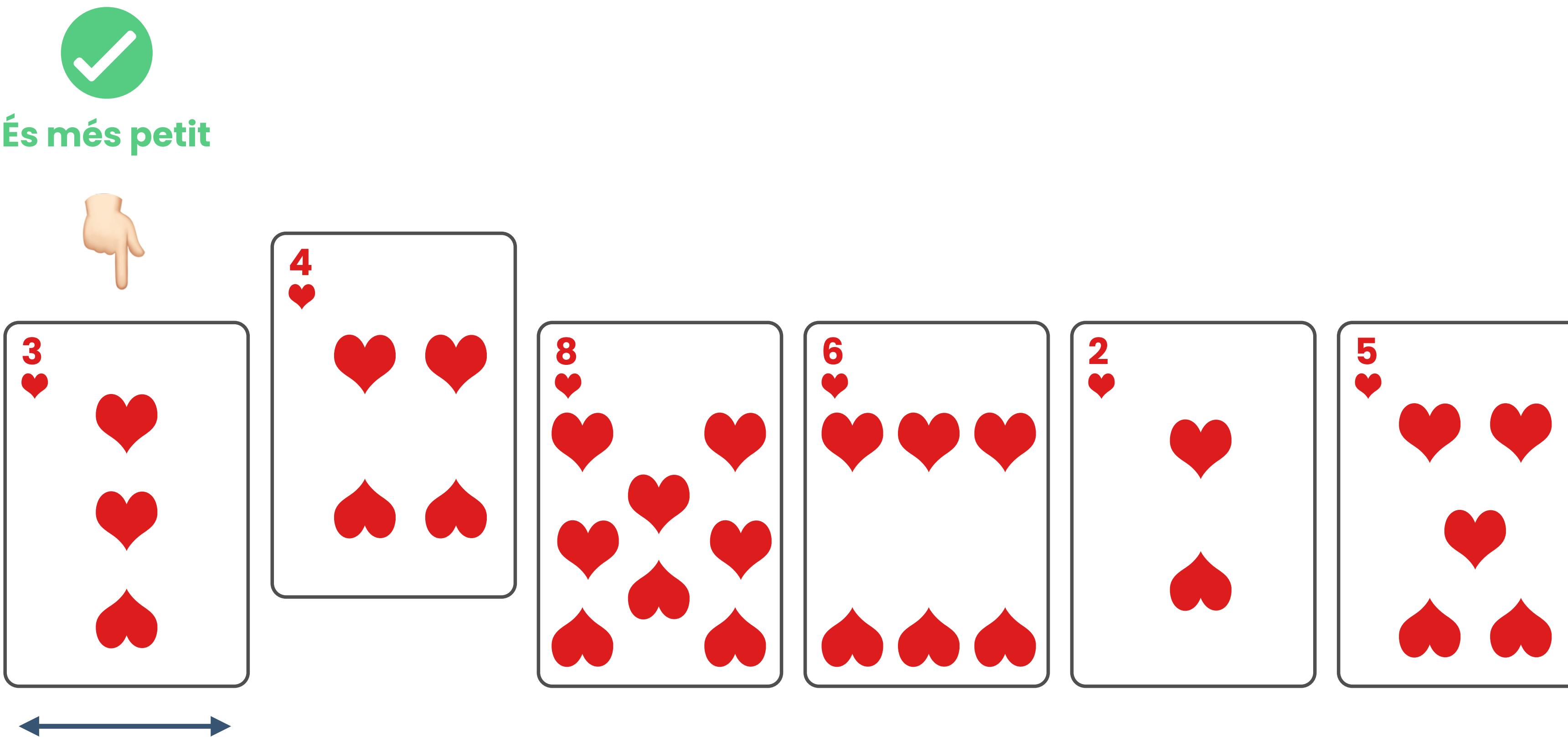


Subseqüència parcialment ordenada

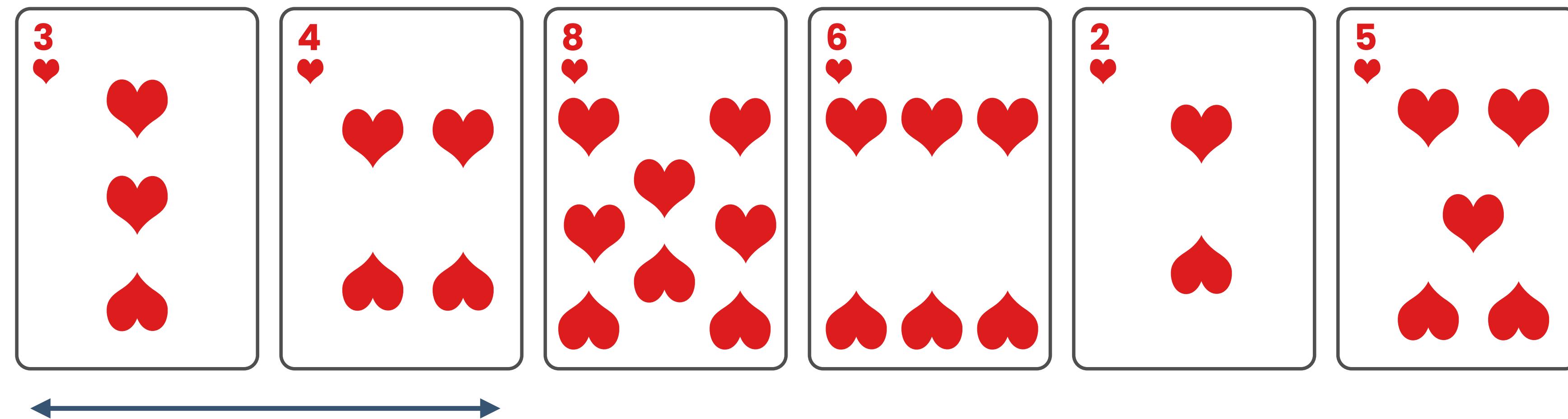
## Ordenació per inserció



## Ordenació per inserció

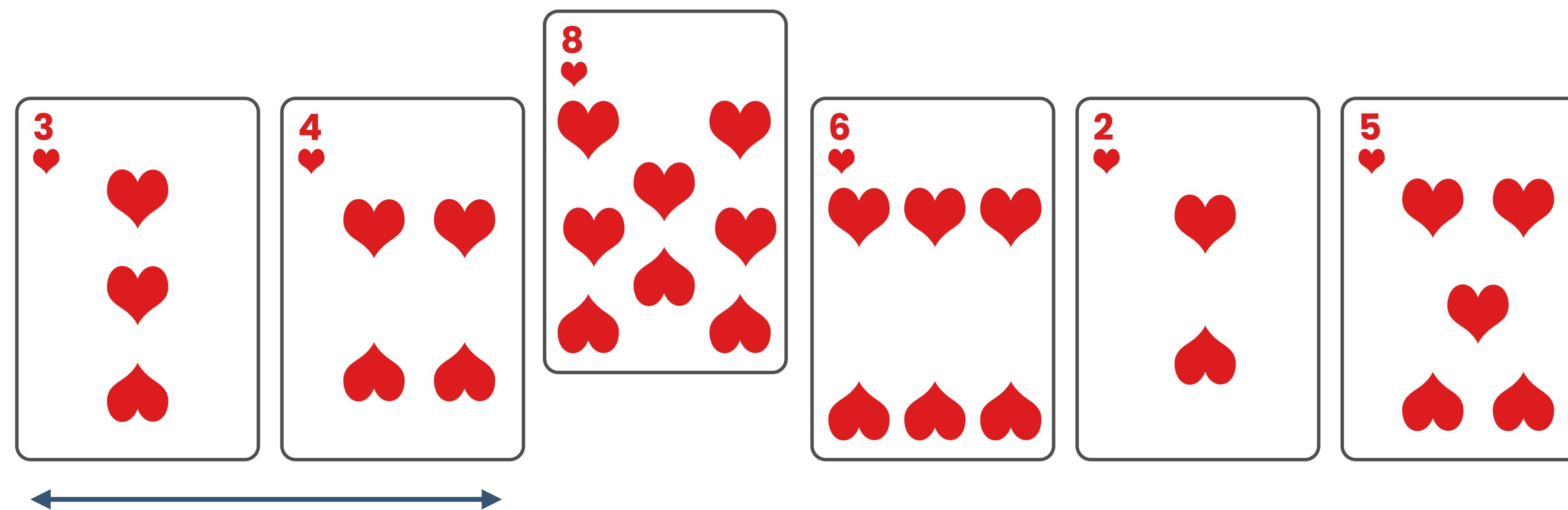


## Ordenació per inserció

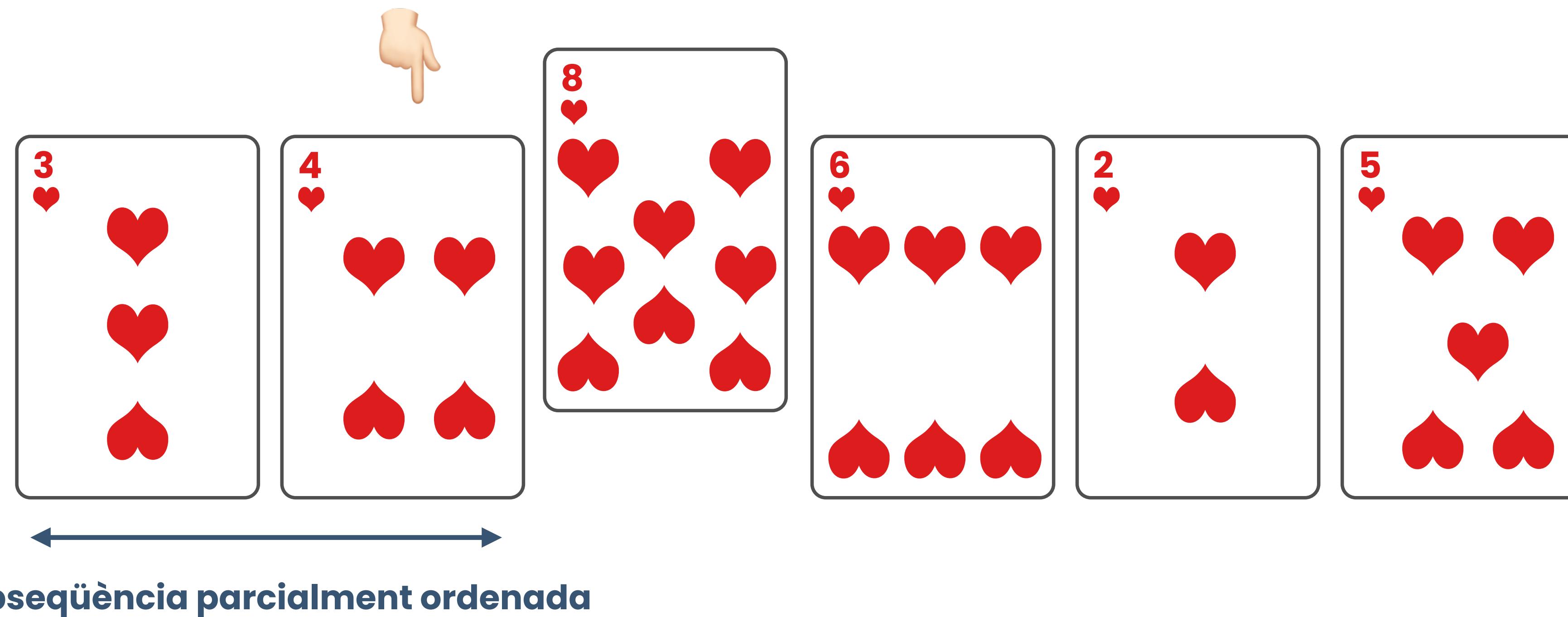


Subseqüència parcialment ordenada

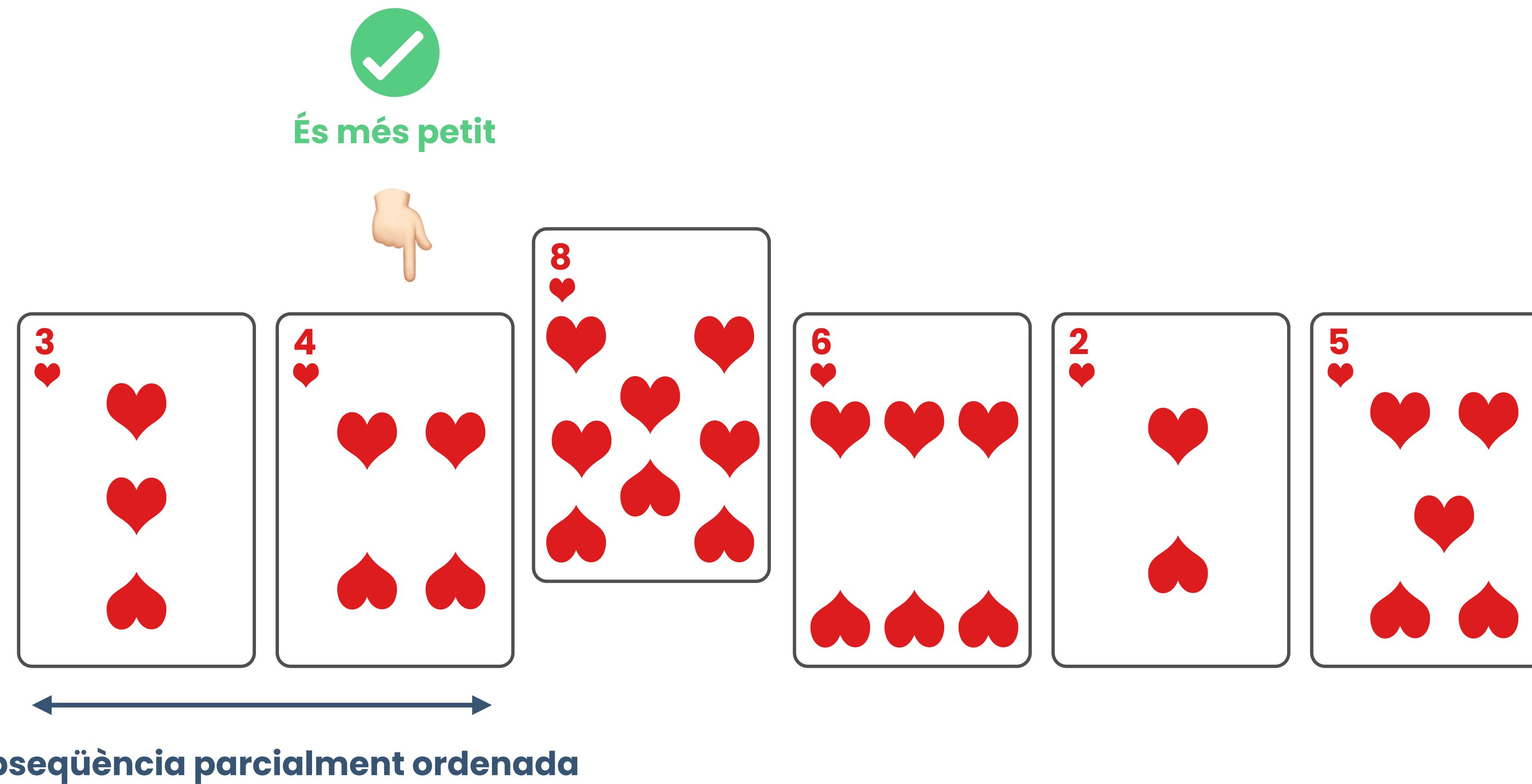
## Ordenació per inserció



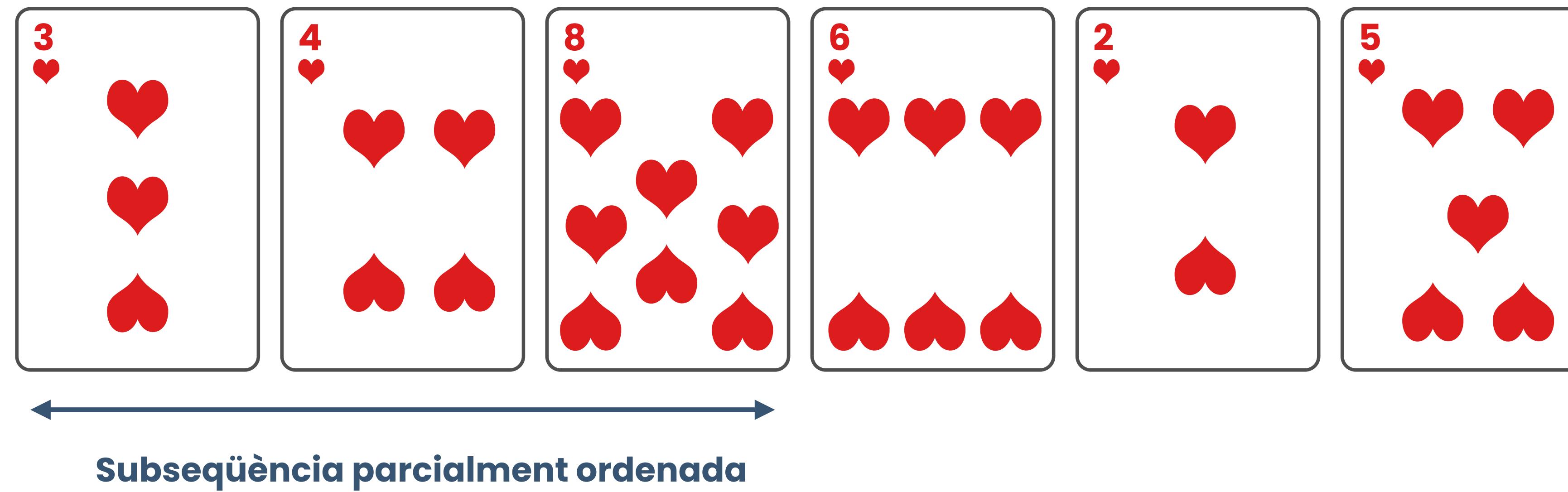
## Ordenació per inserció



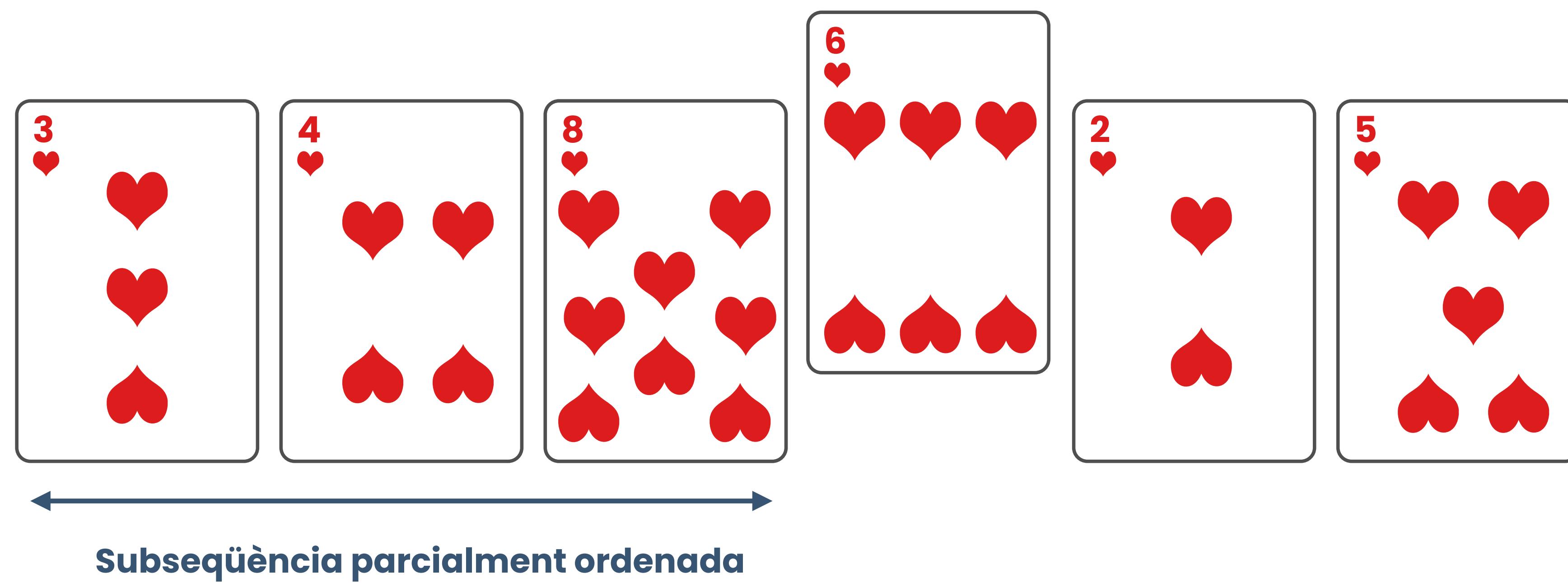
## Ordenació per inserció



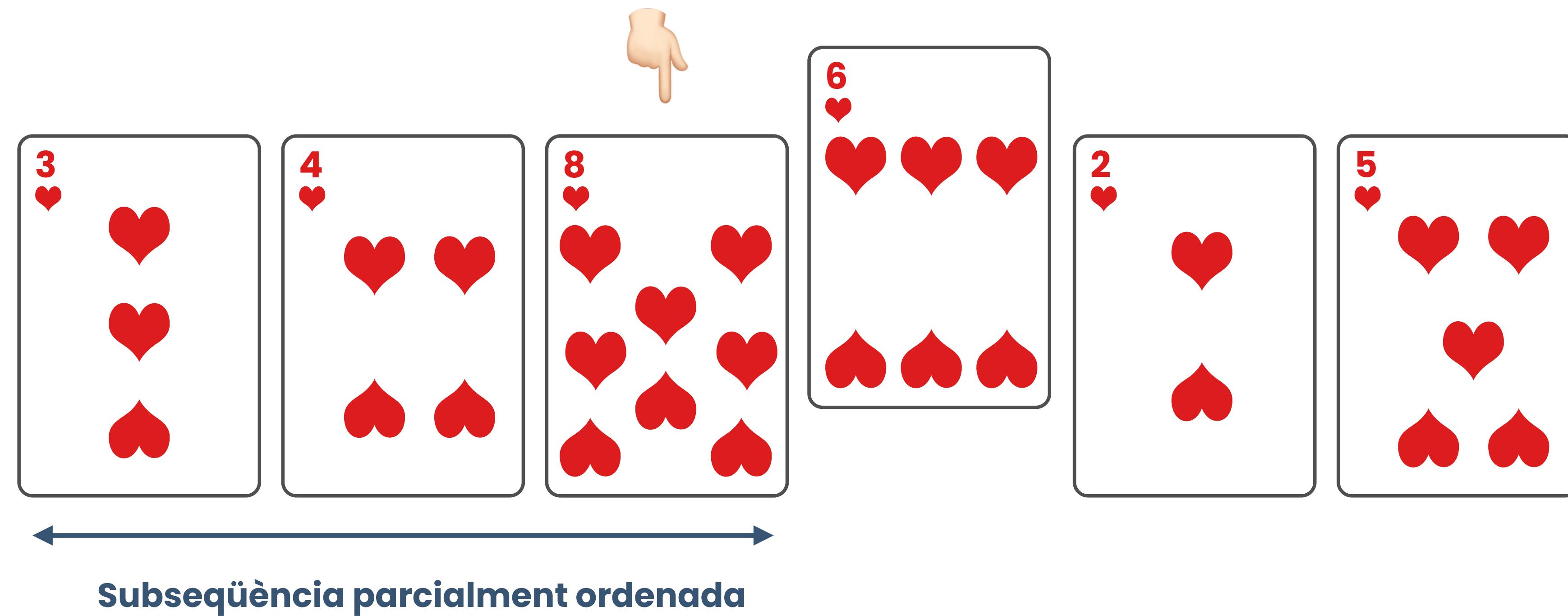
## Ordenació per inserció



## Ordenació per inserció



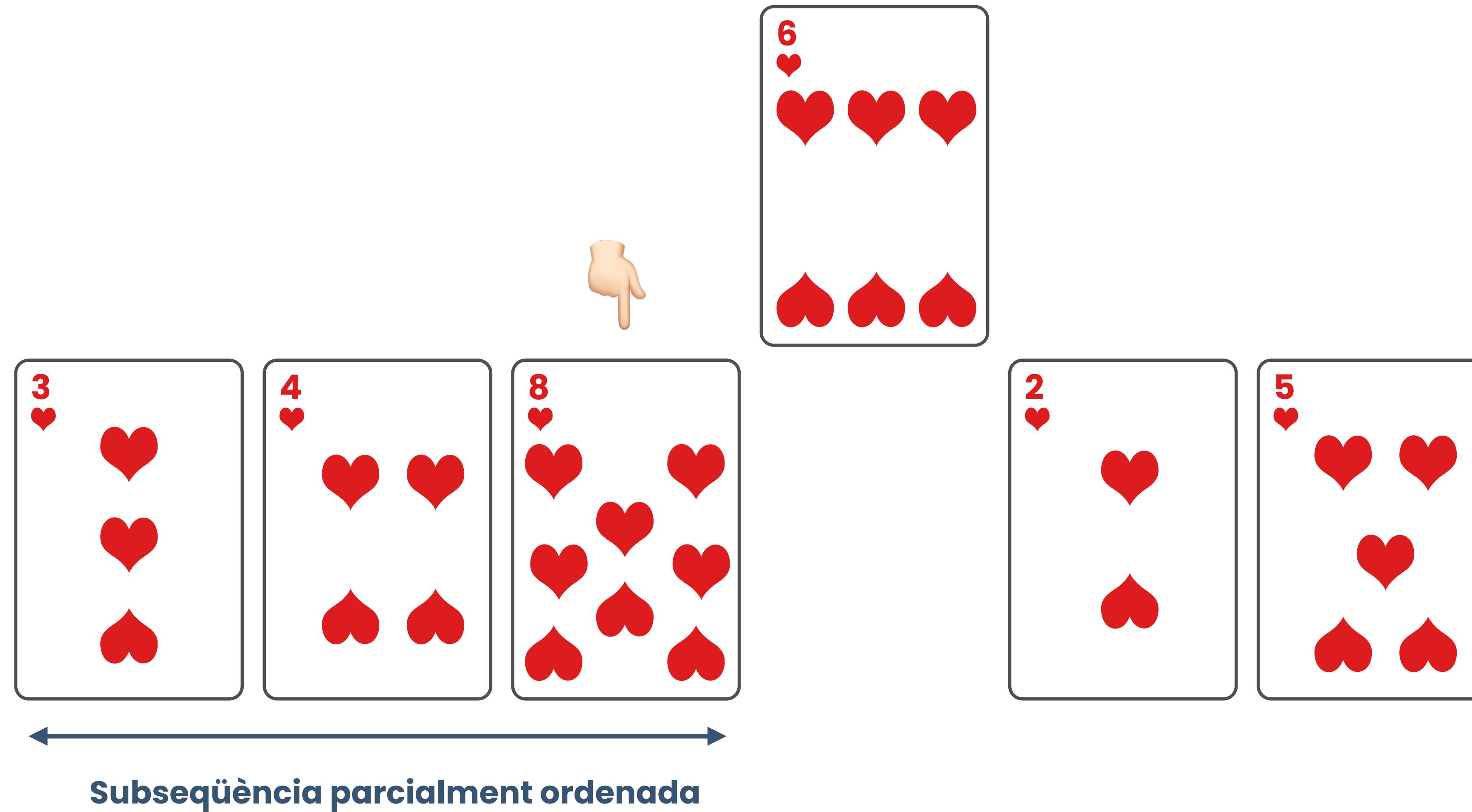
## Ordenació per inserció



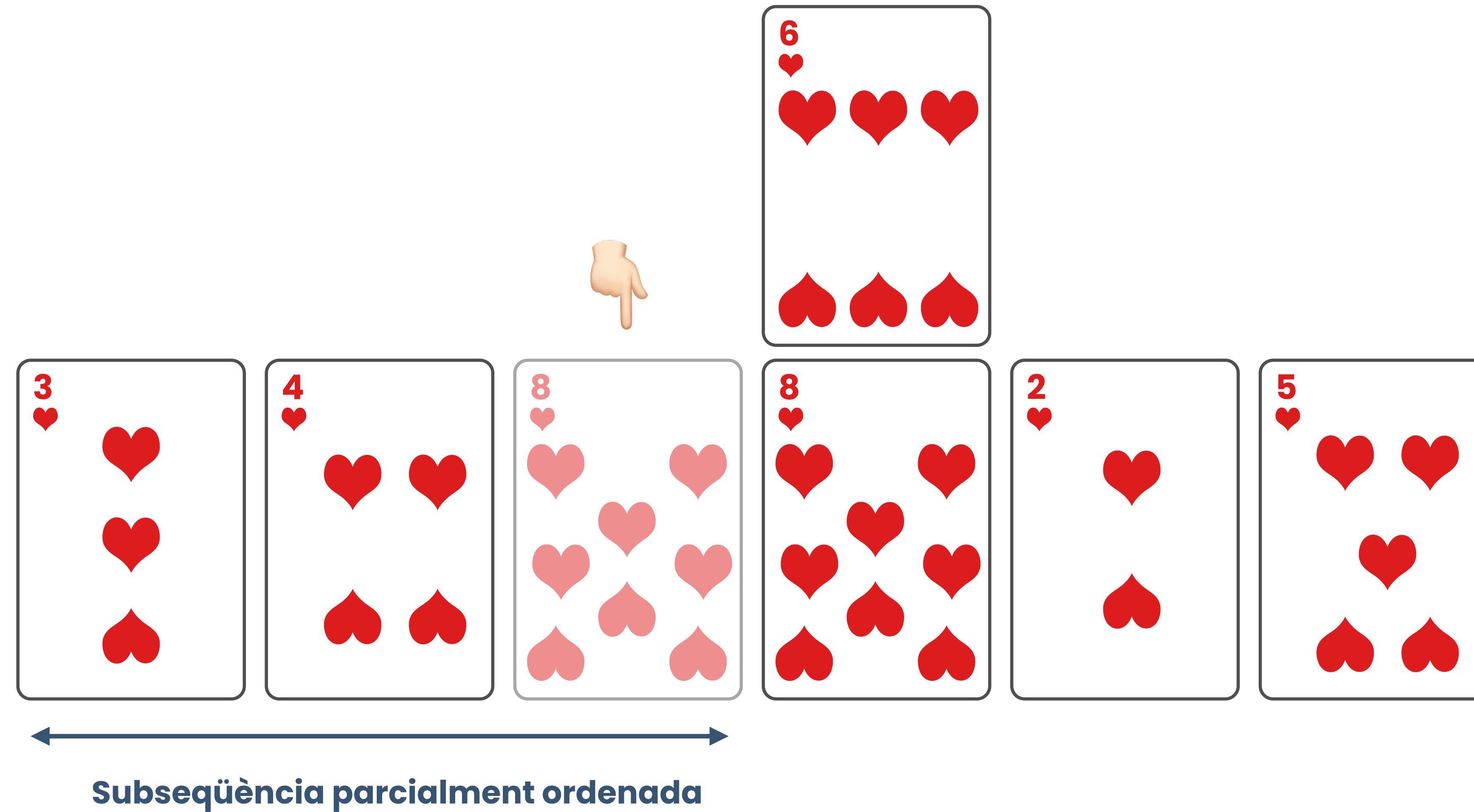
## Ordenació per inserció



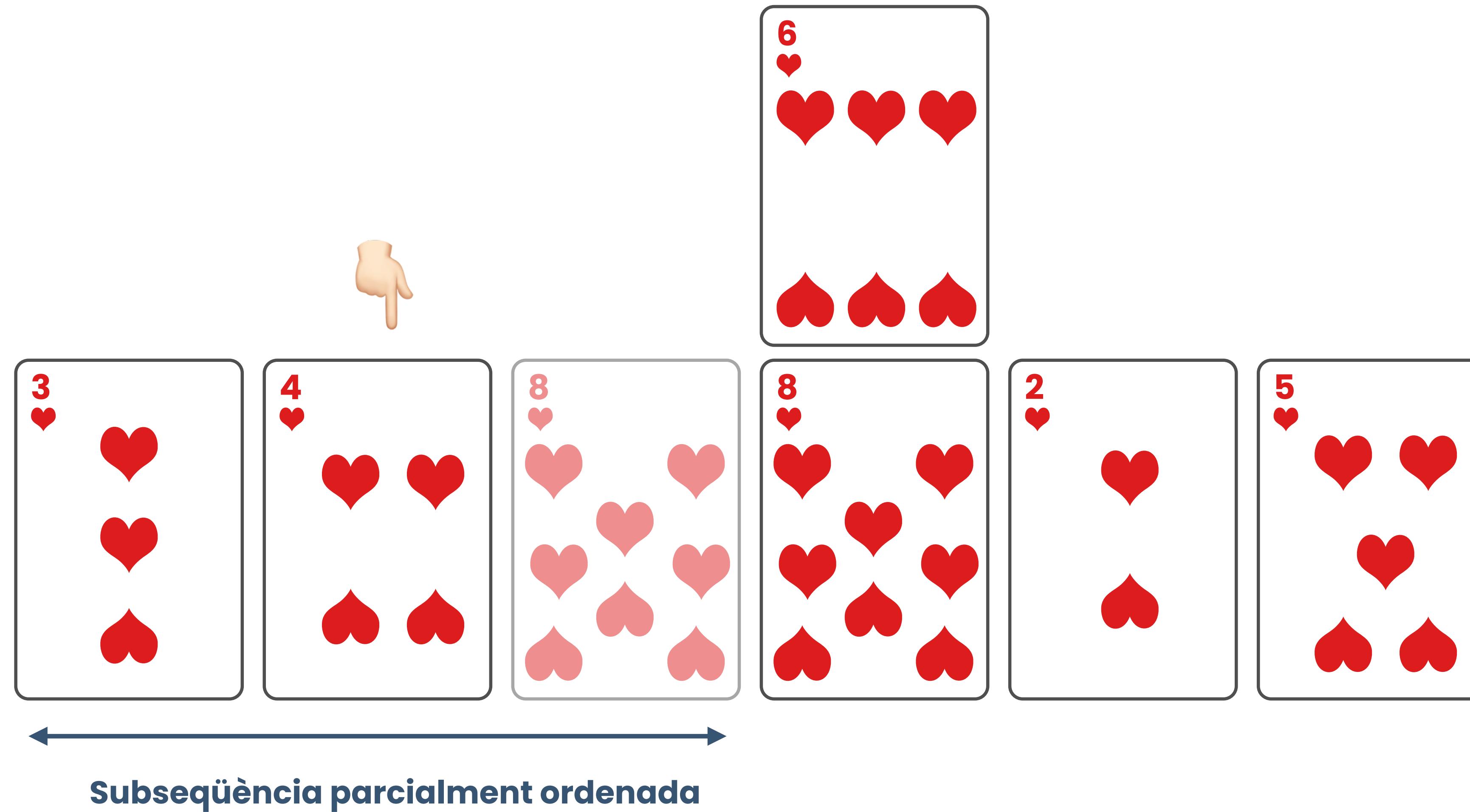
## Ordenació per inserció



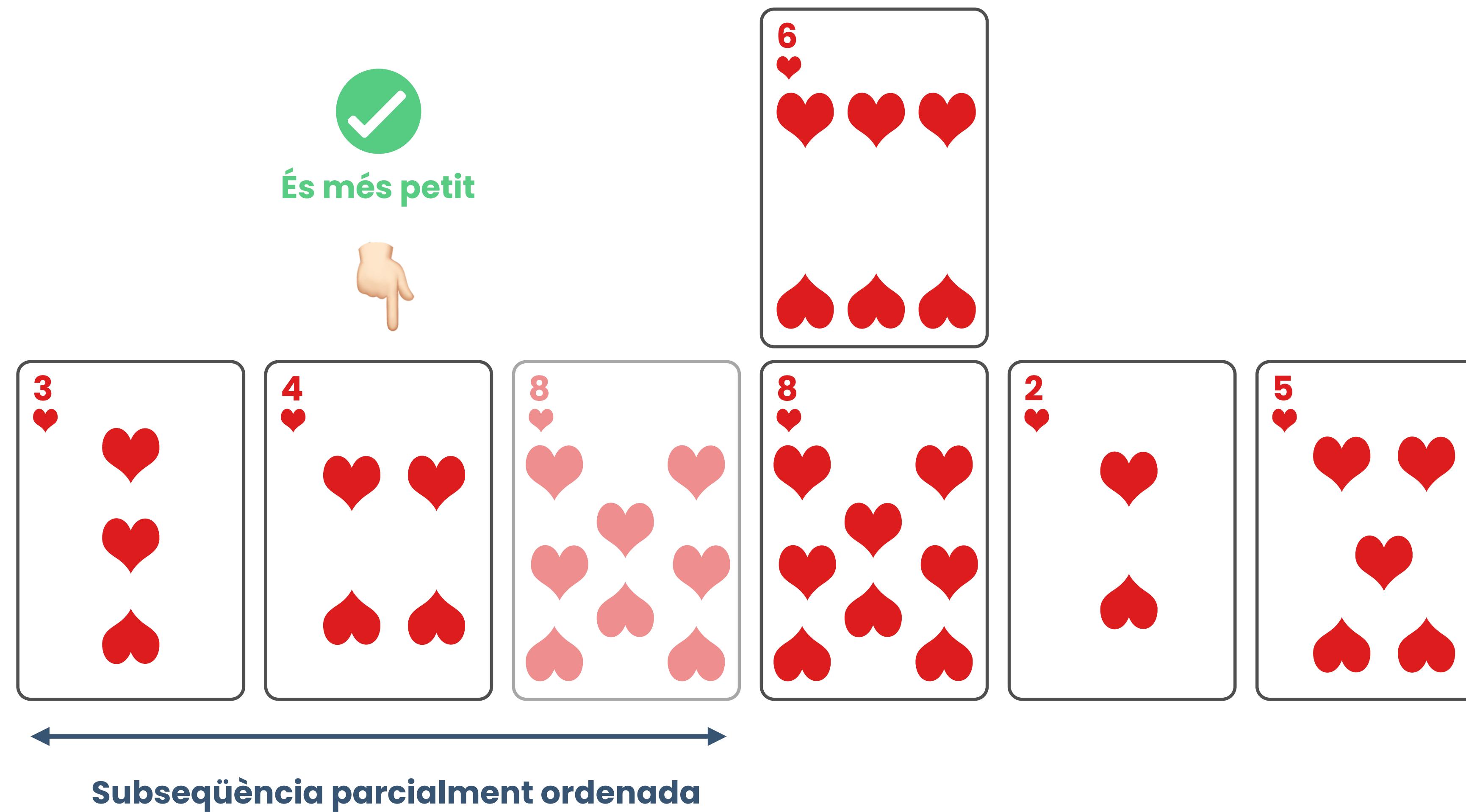
## Ordenació per inserció



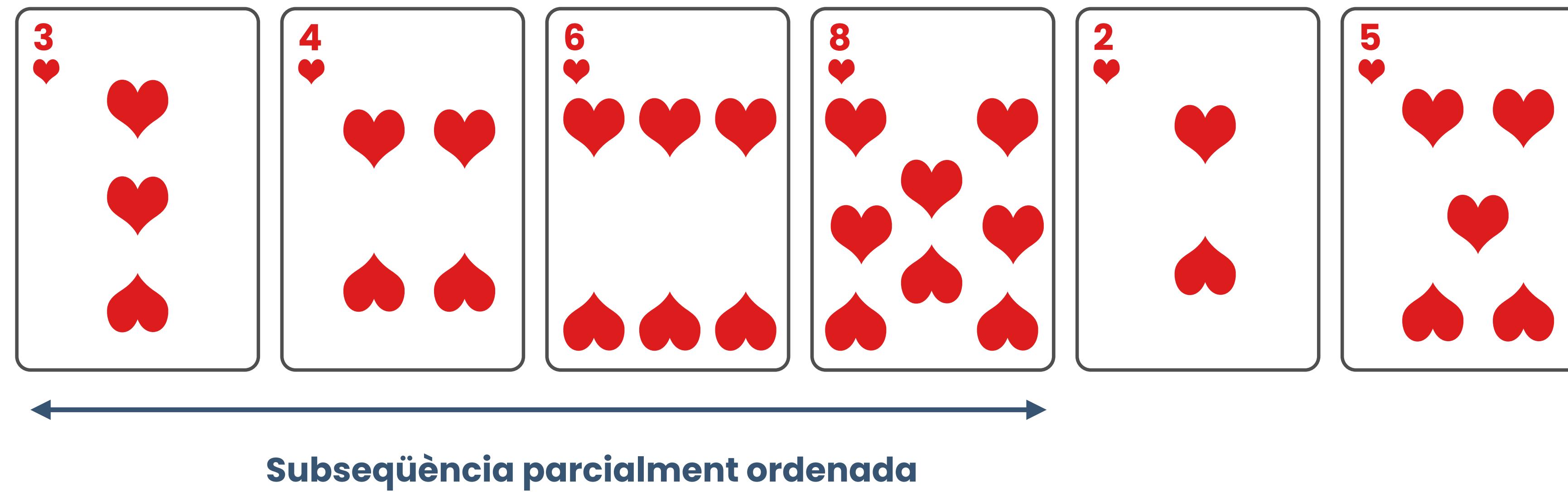
## Ordenació per inserció



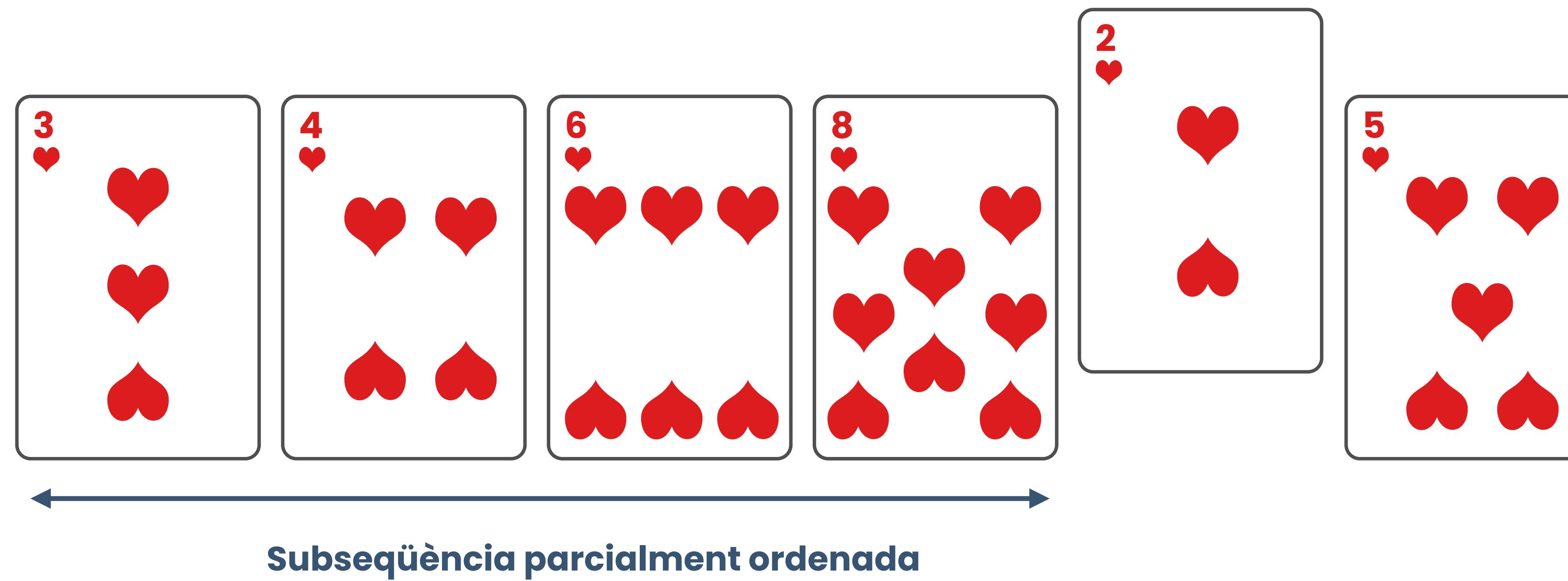
## Ordenació per inserció



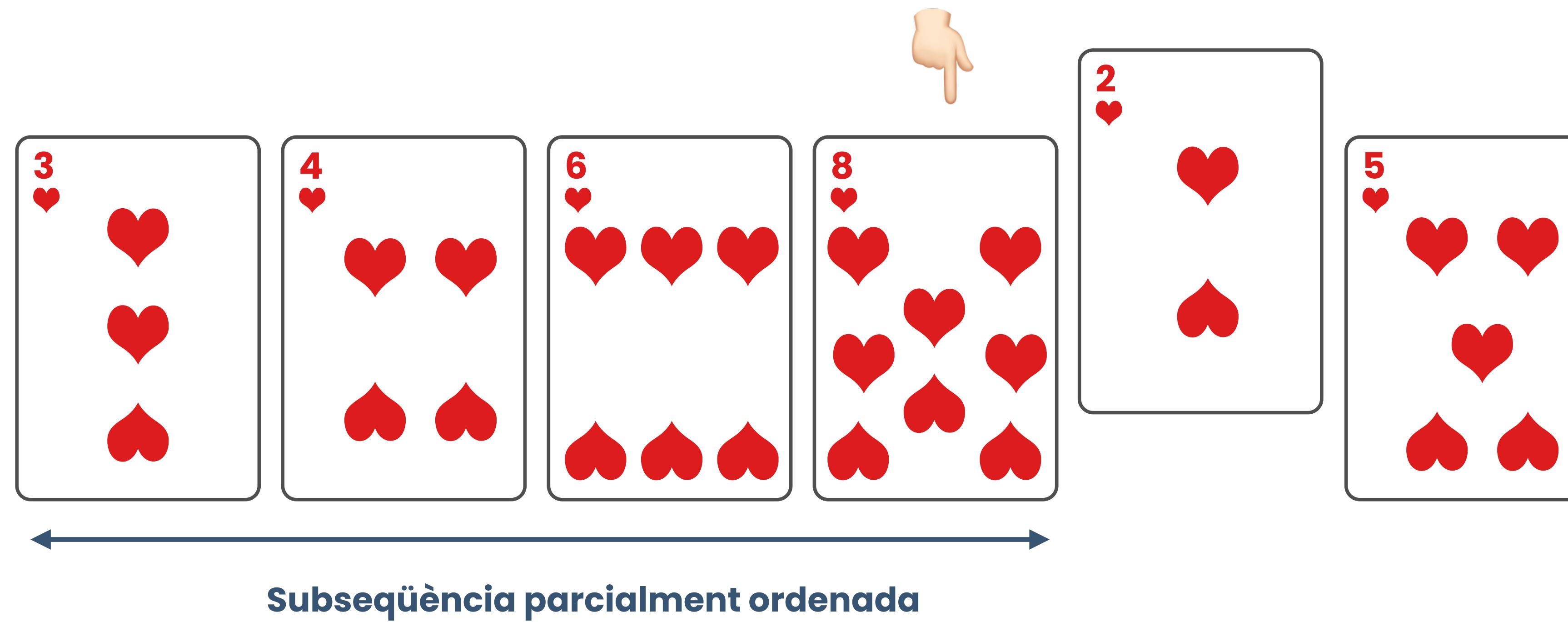
## Ordenació per inserció



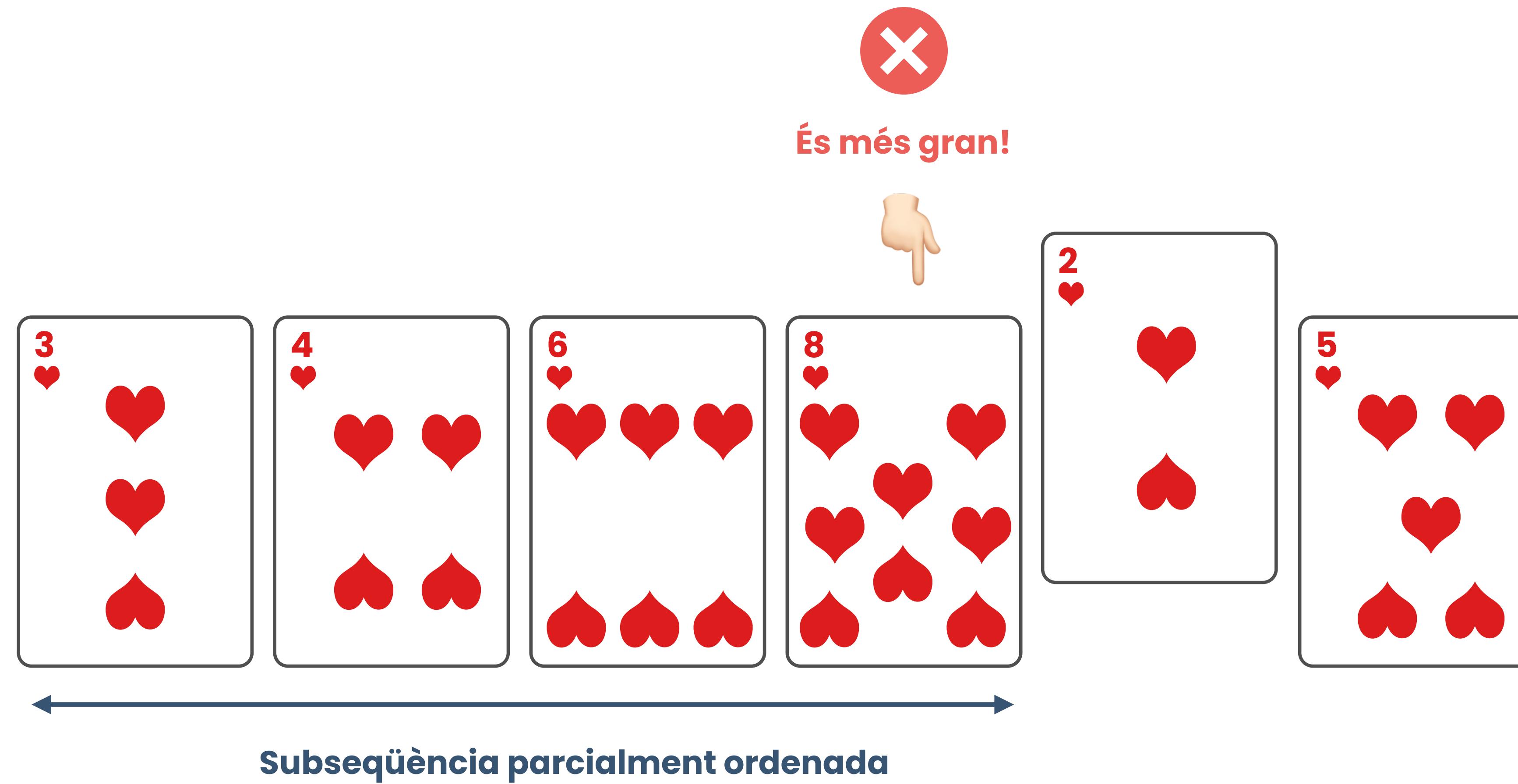
## Ordenació per inserció



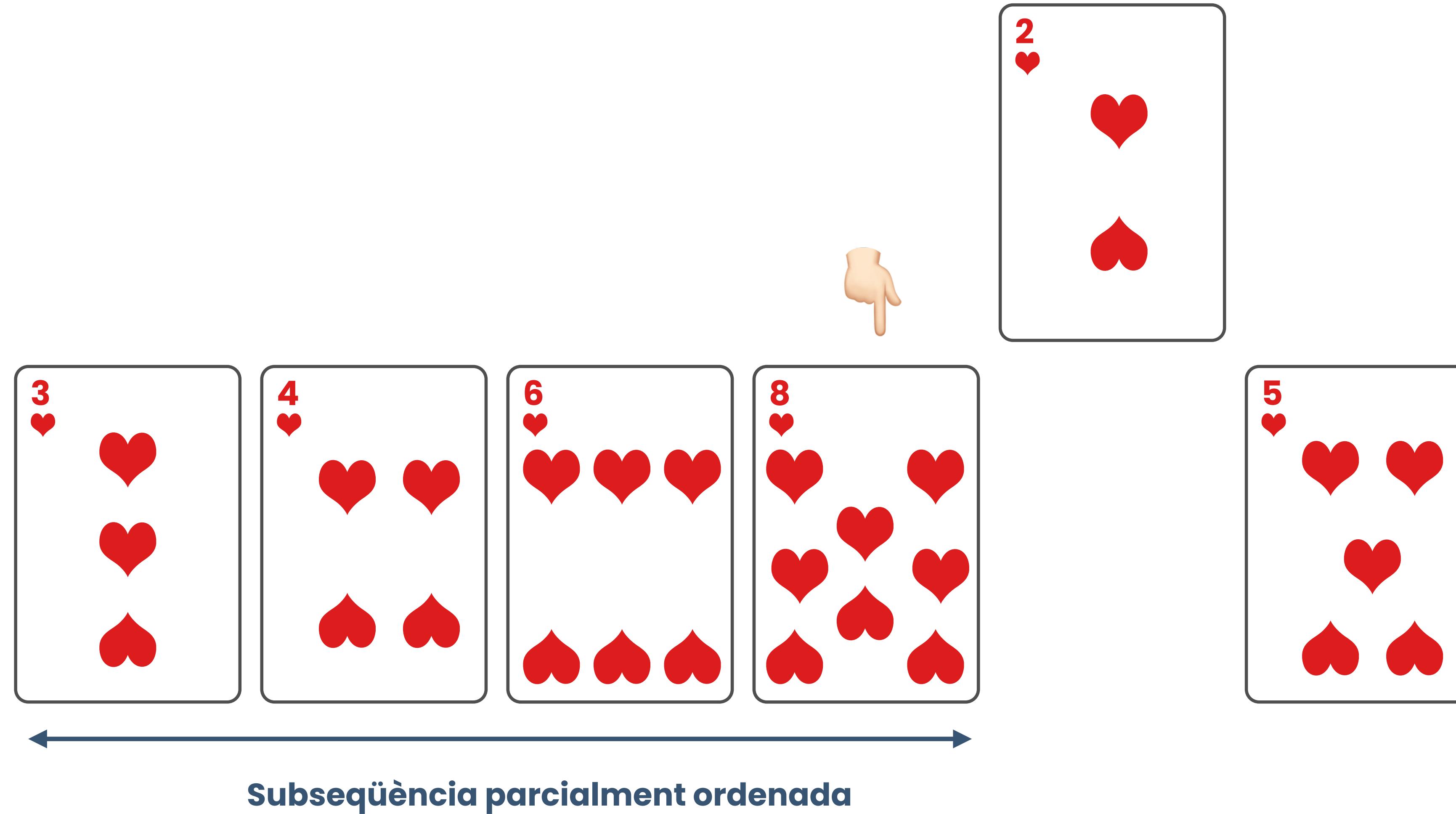
## Ordenació per inserció



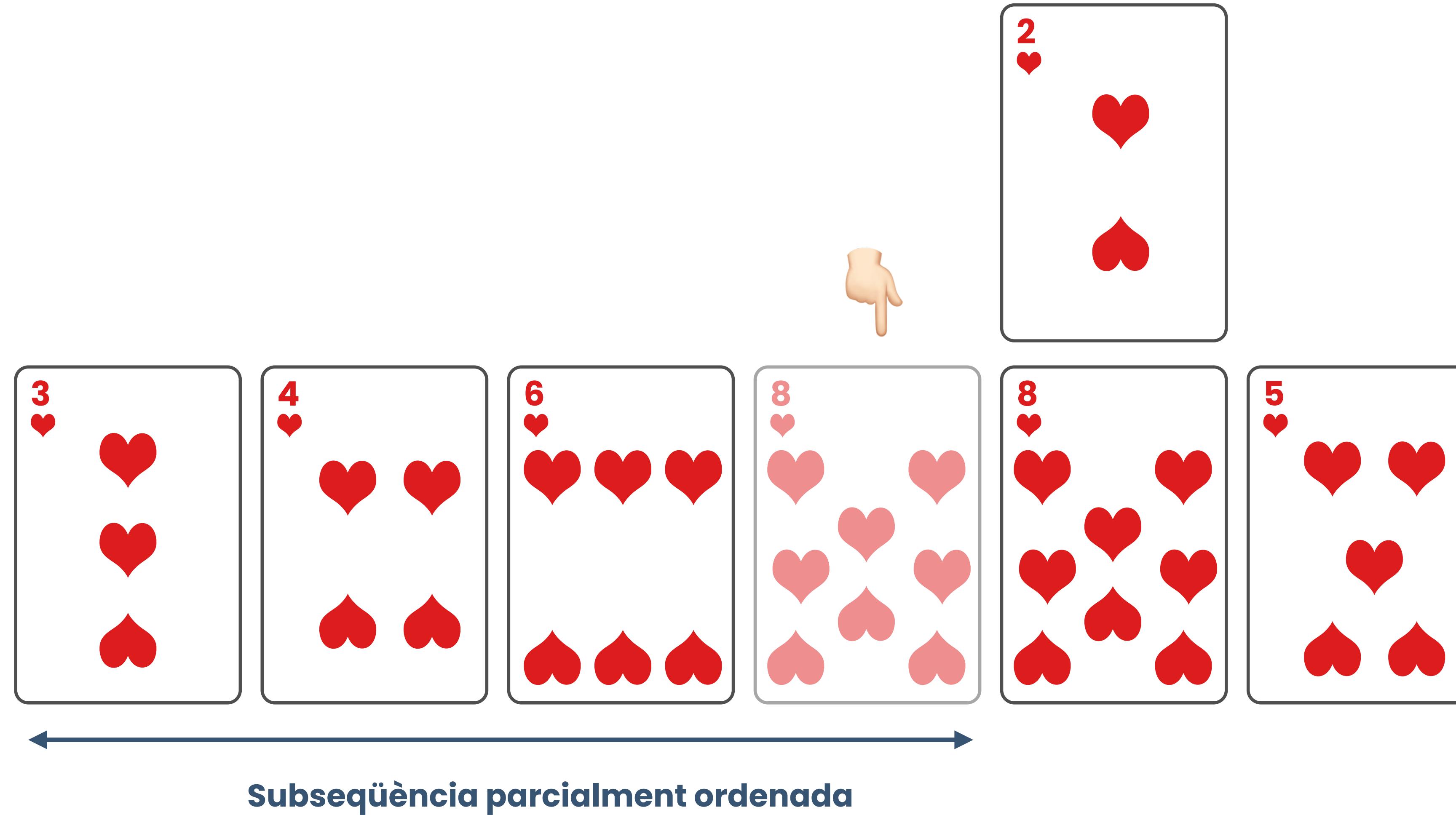
## Ordenació per inserció



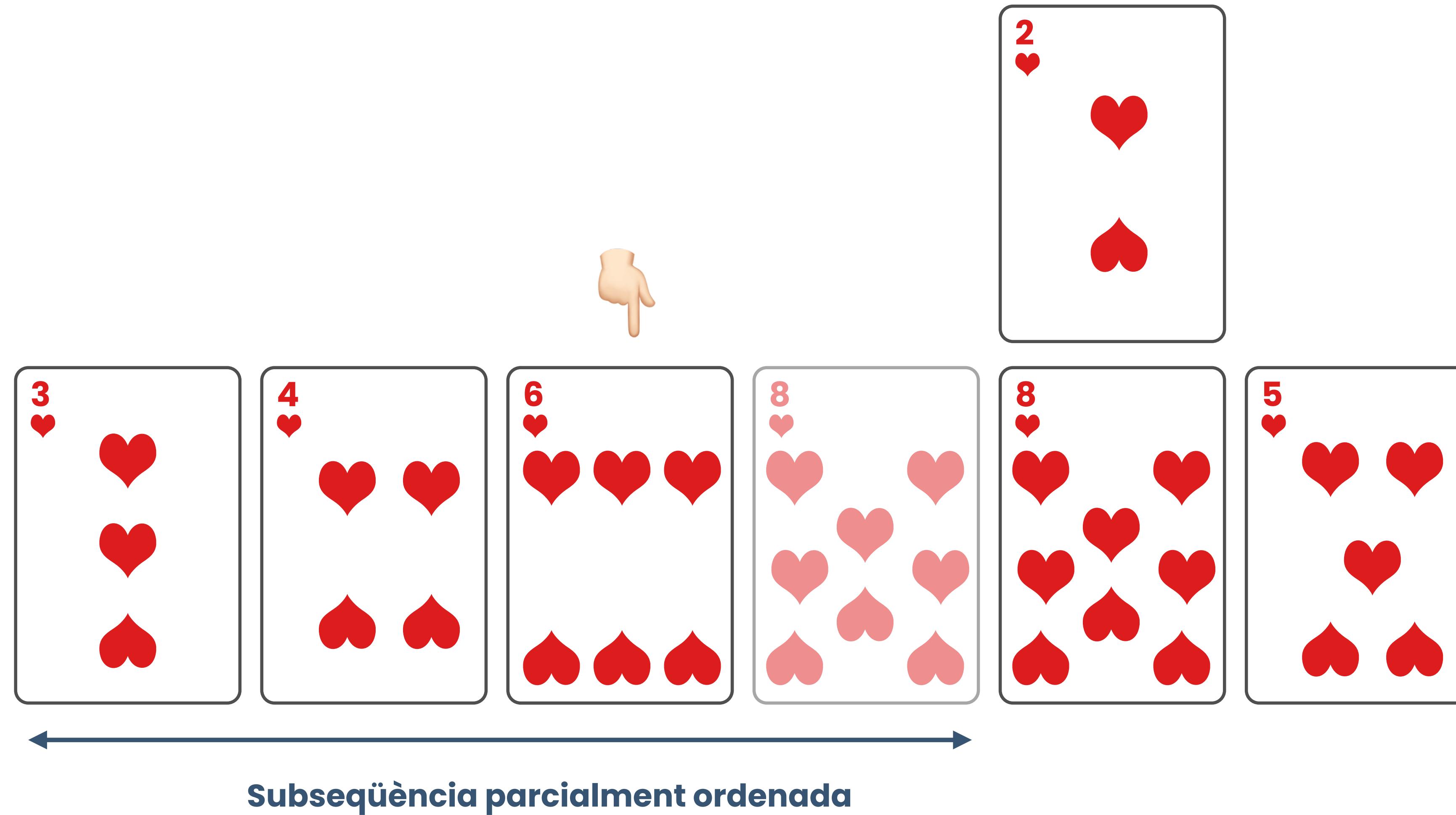
## Ordenació per inserció



## Ordenació per inserció



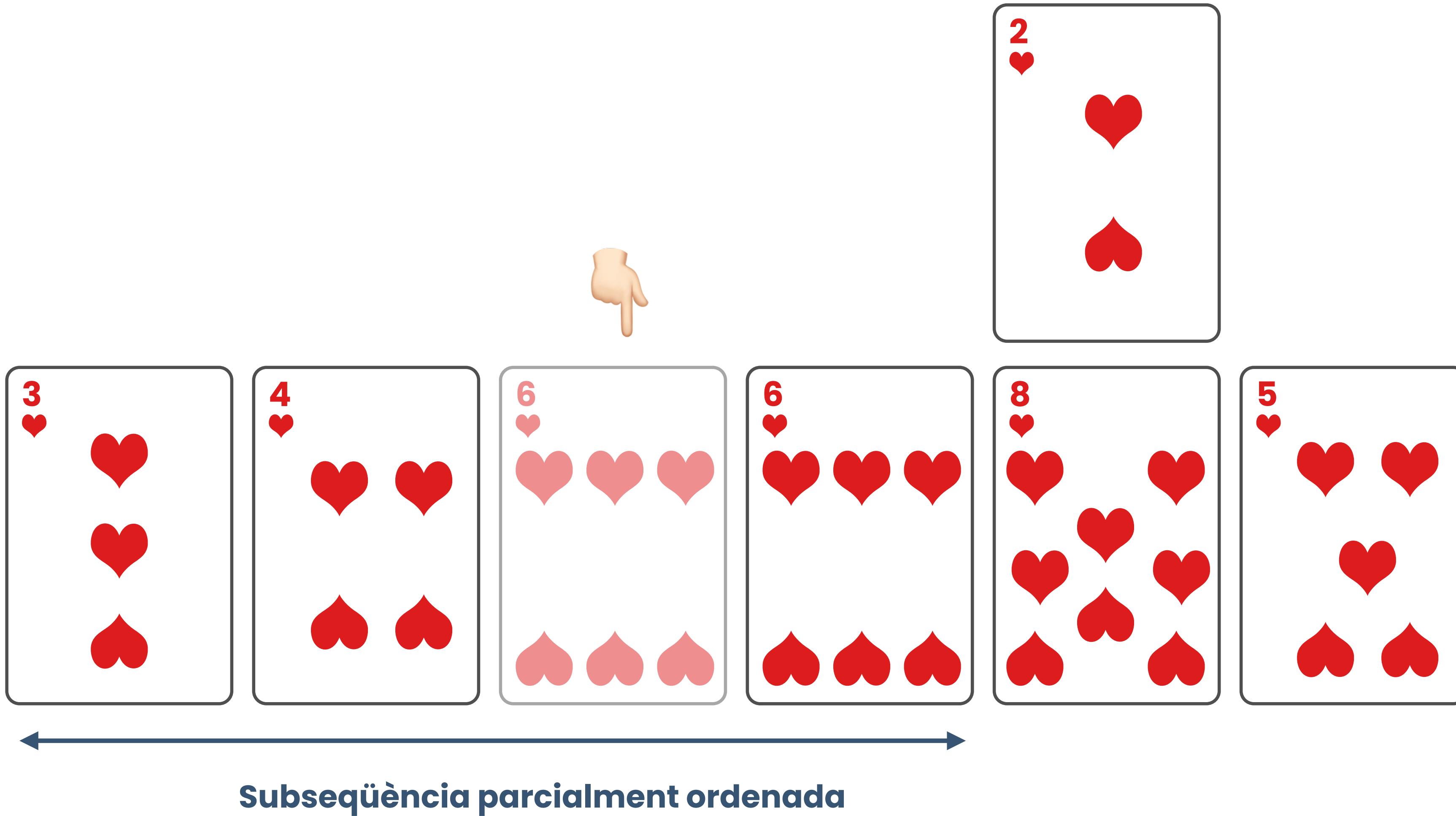
## Ordenació per inserció



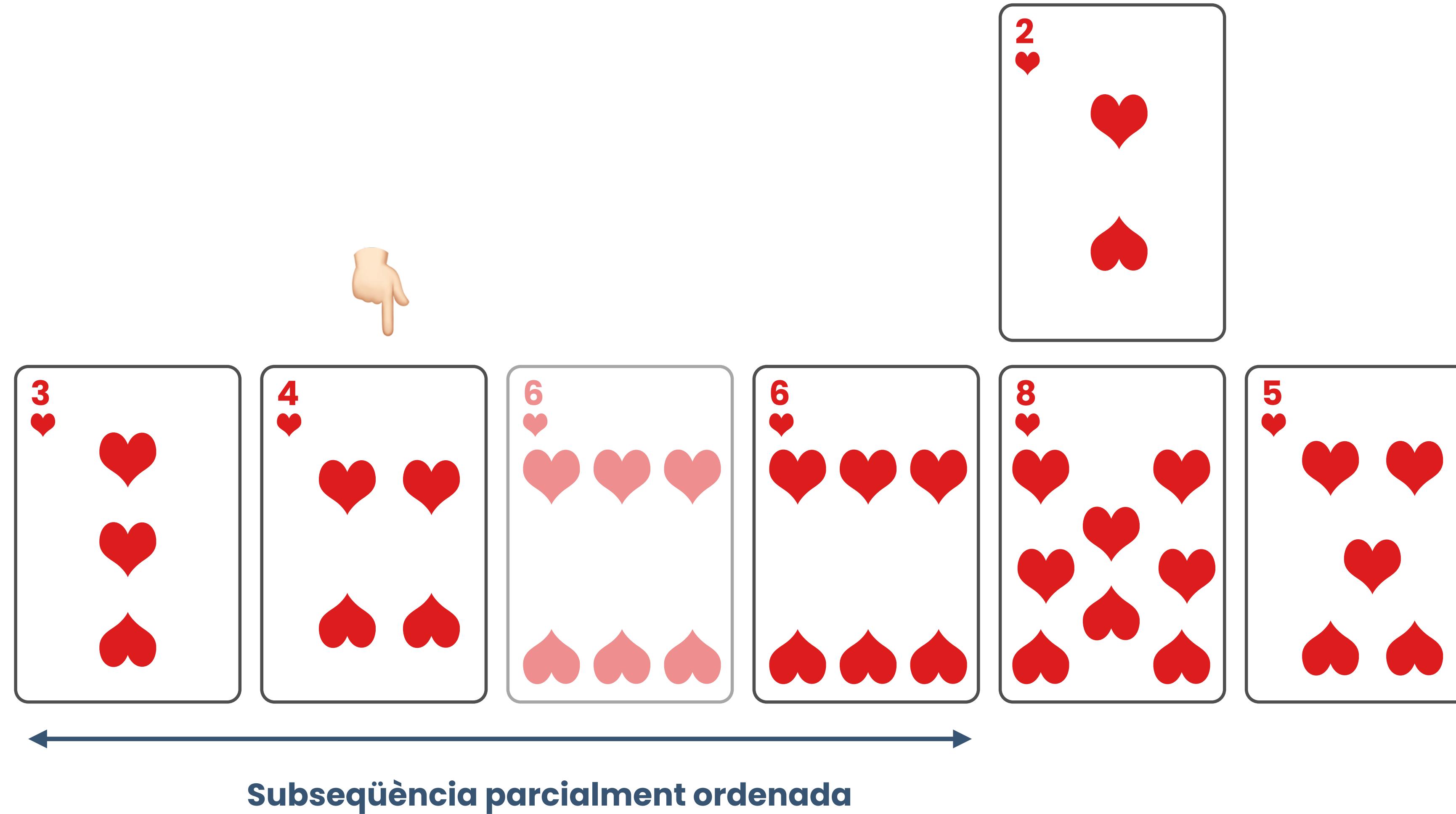
## Ordenació per inserció



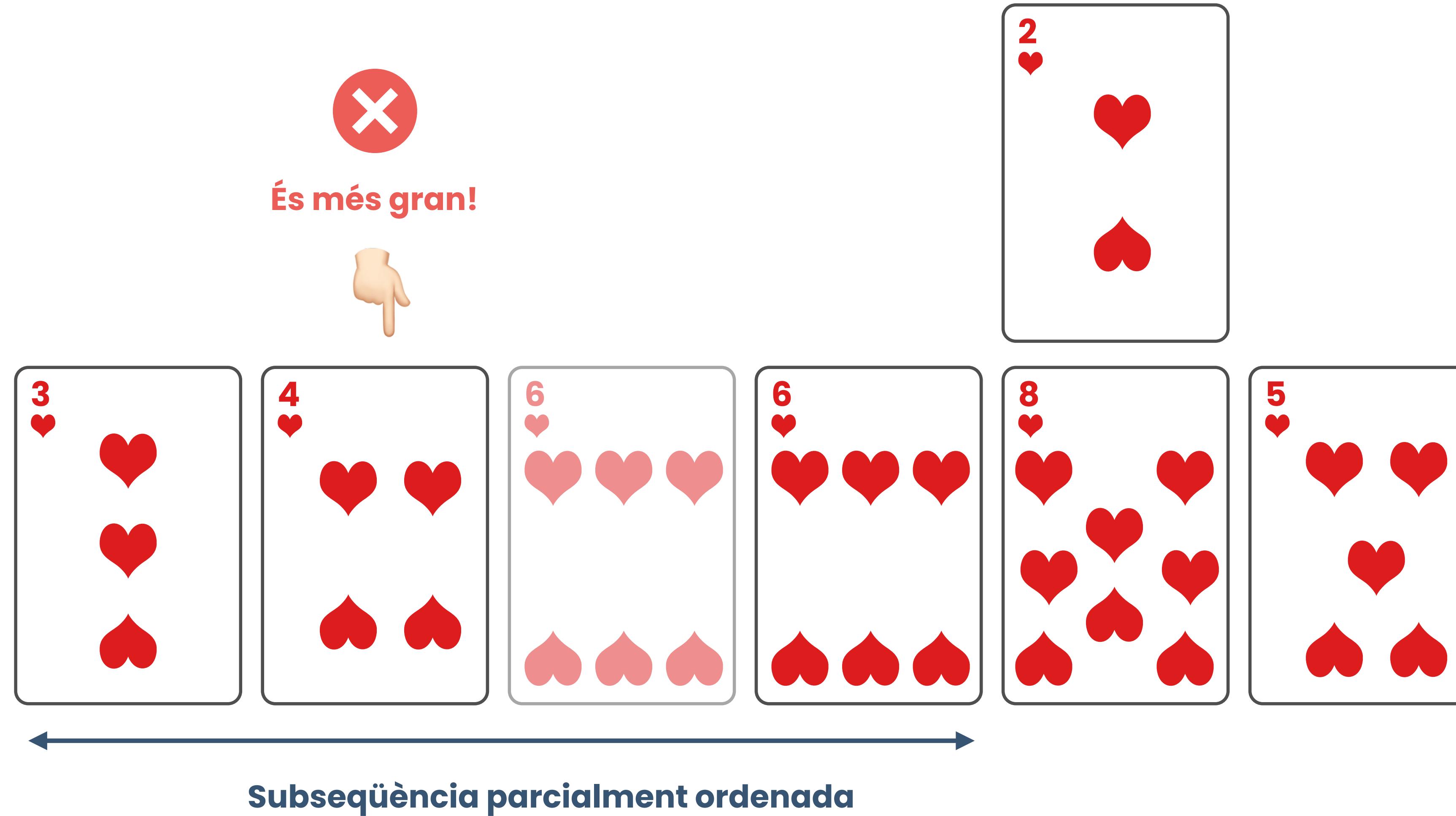
## Ordenació per inserció



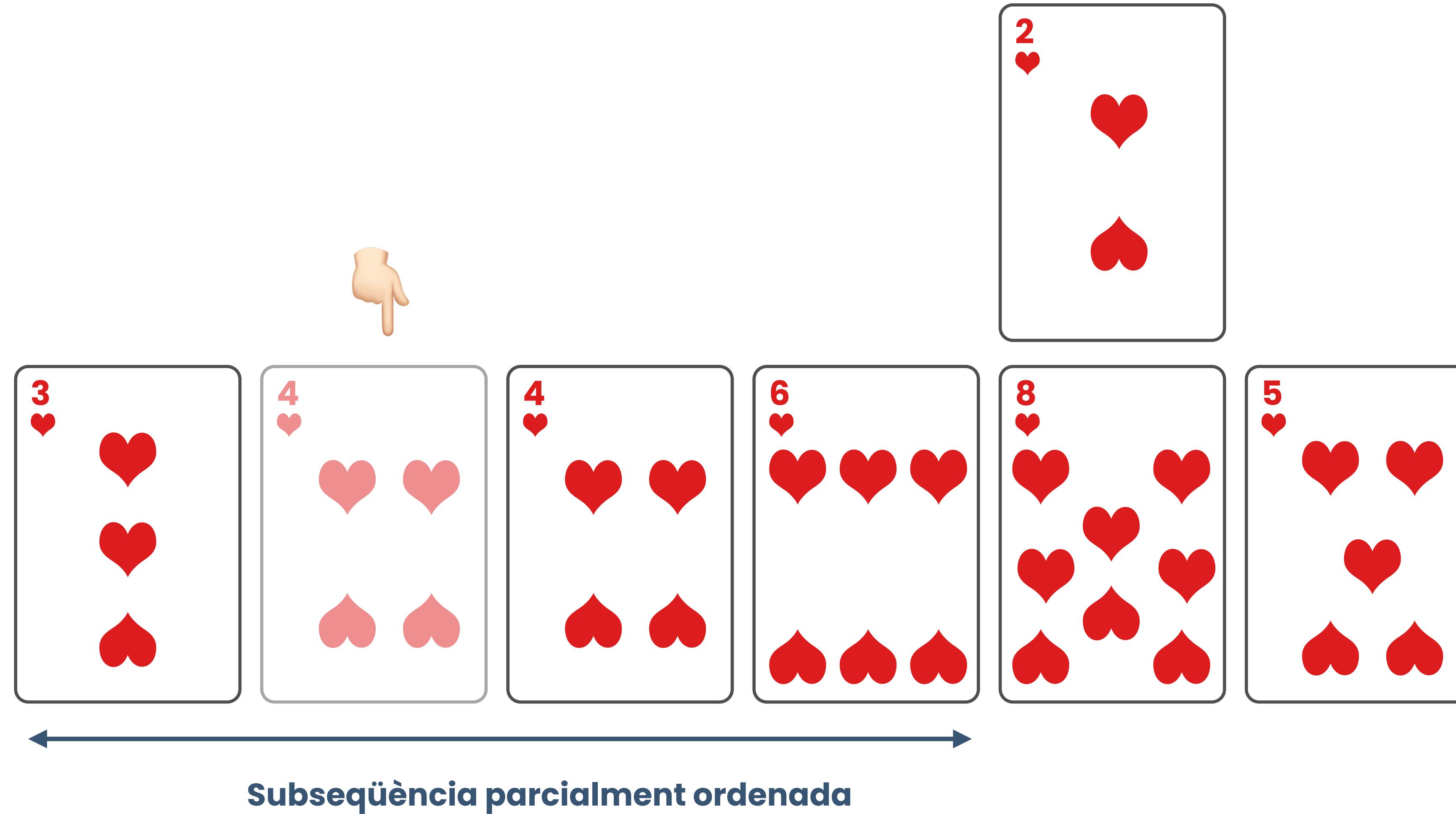
## Ordenació per inserció



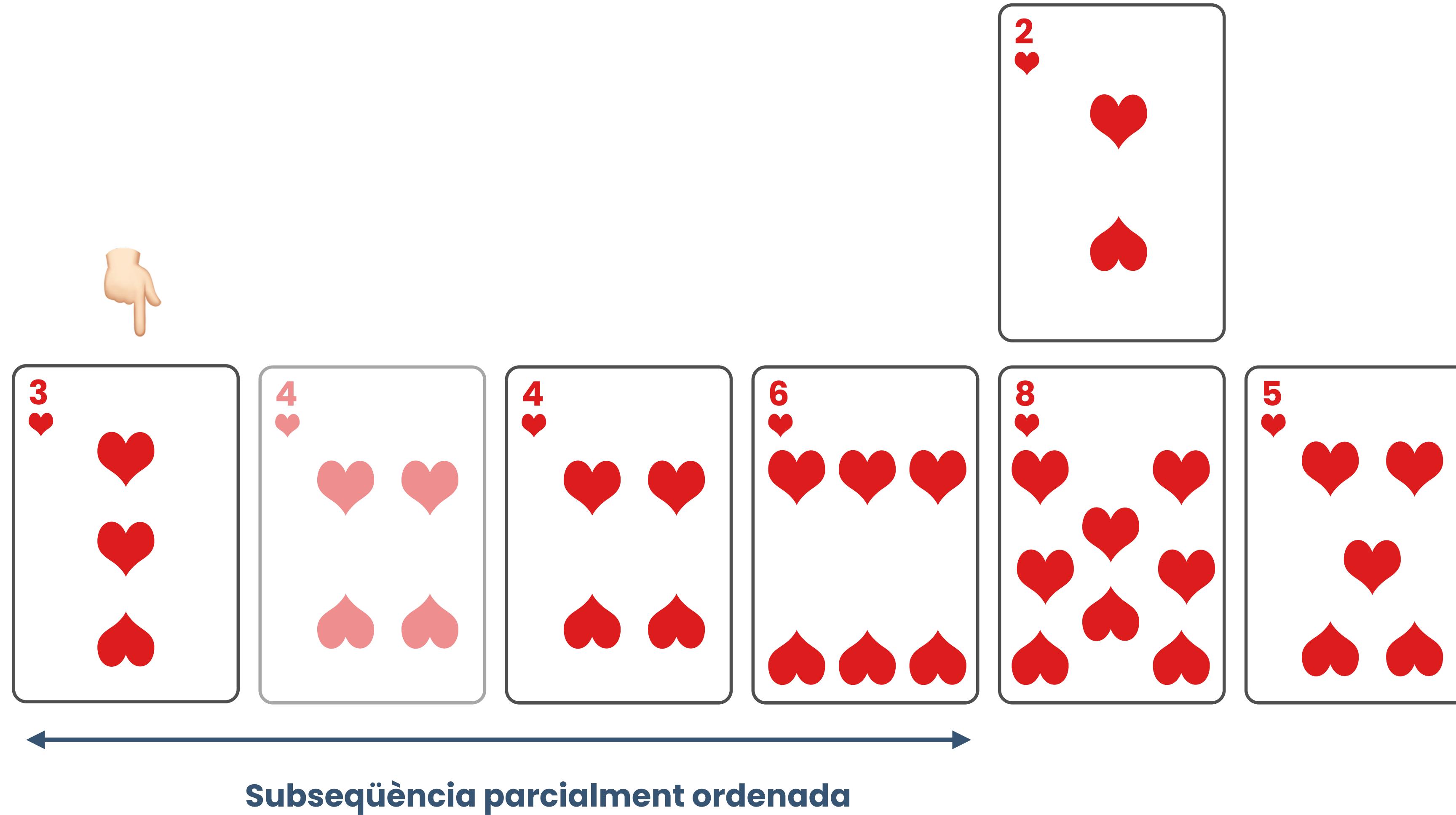
## Ordenació per inserció



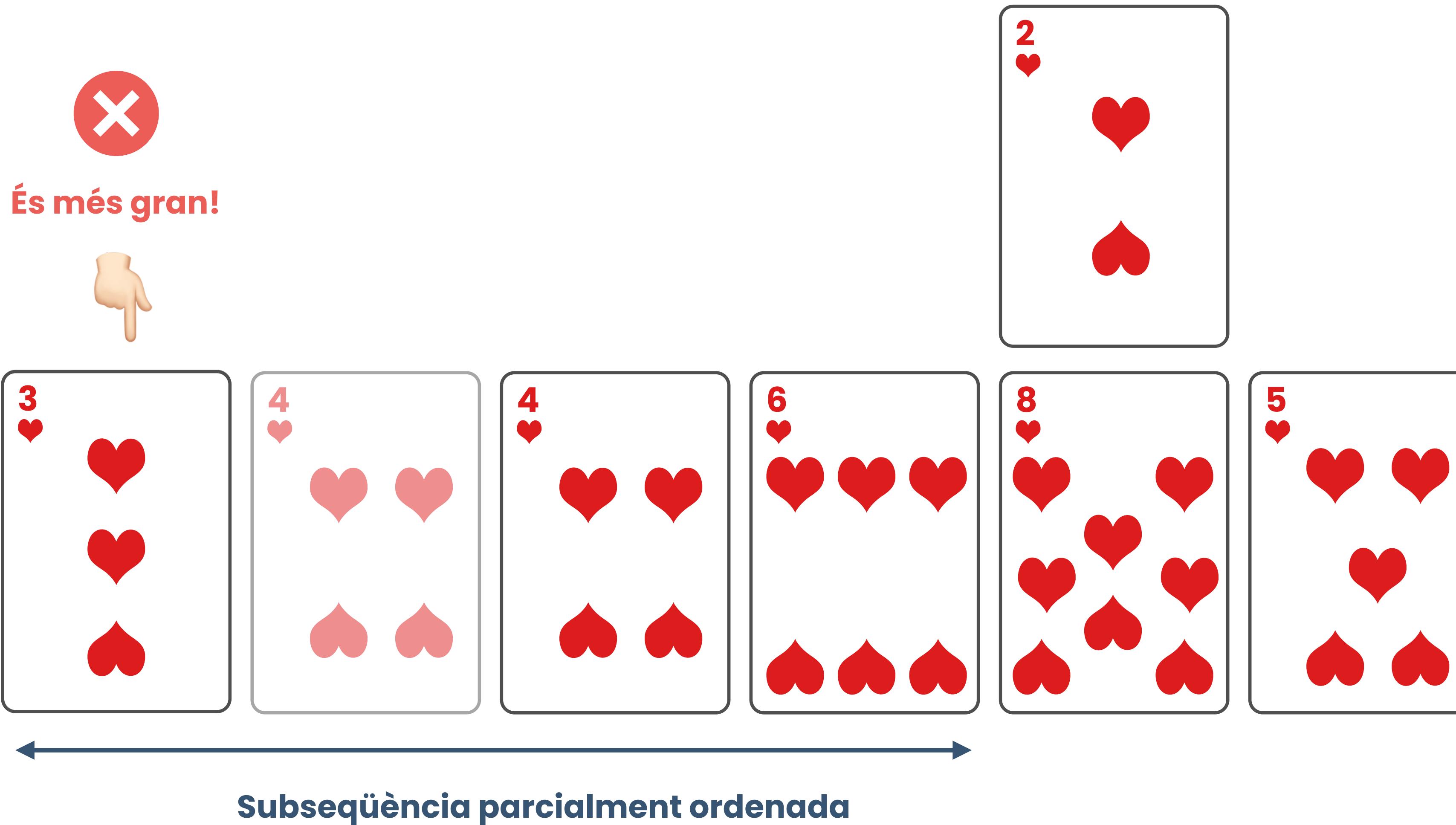
## Ordenació per inserció



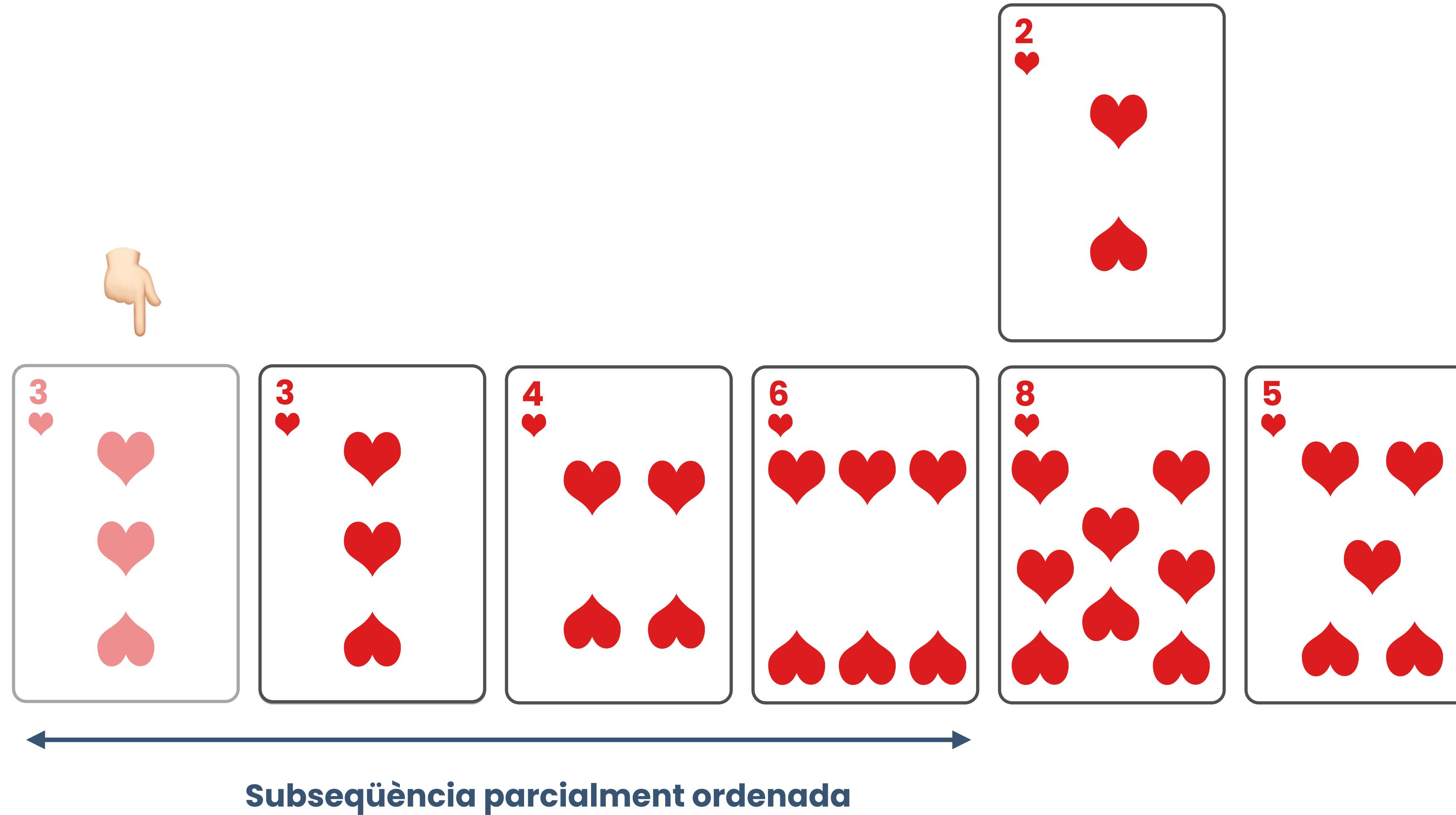
## Ordenació per inserció



## Ordenació per inserció



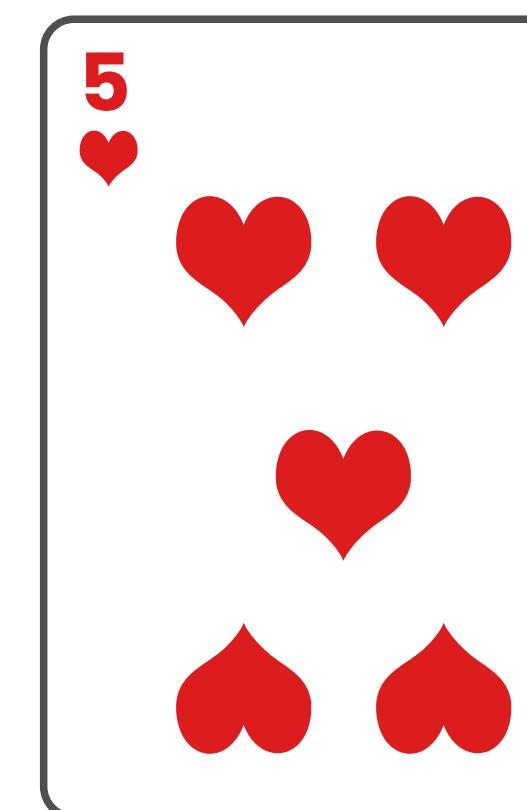
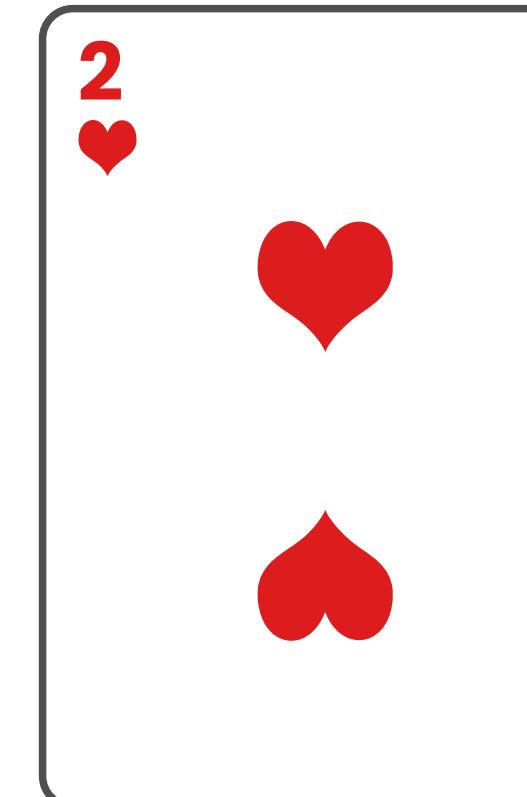
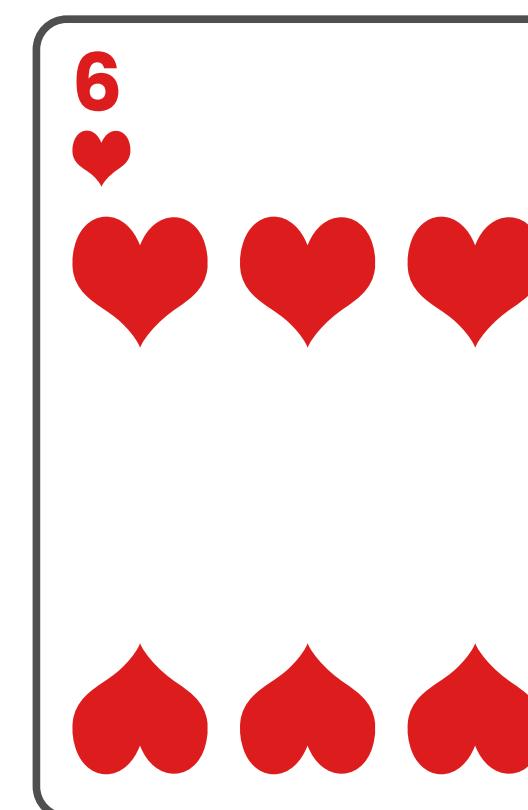
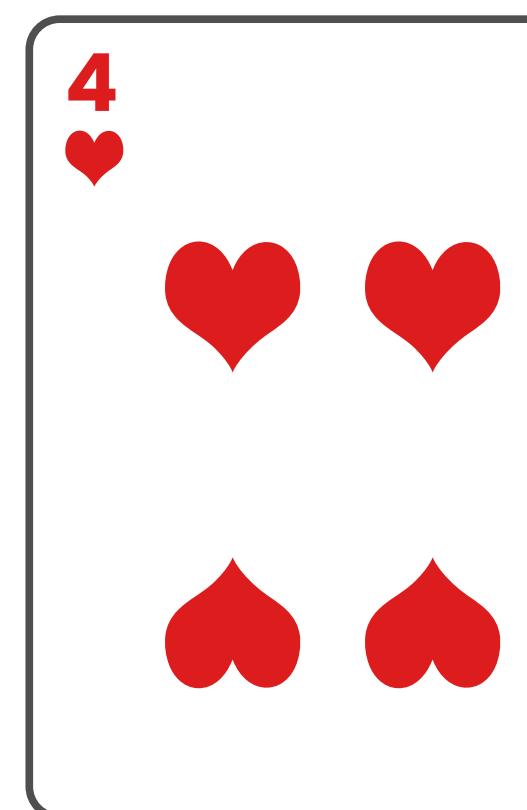
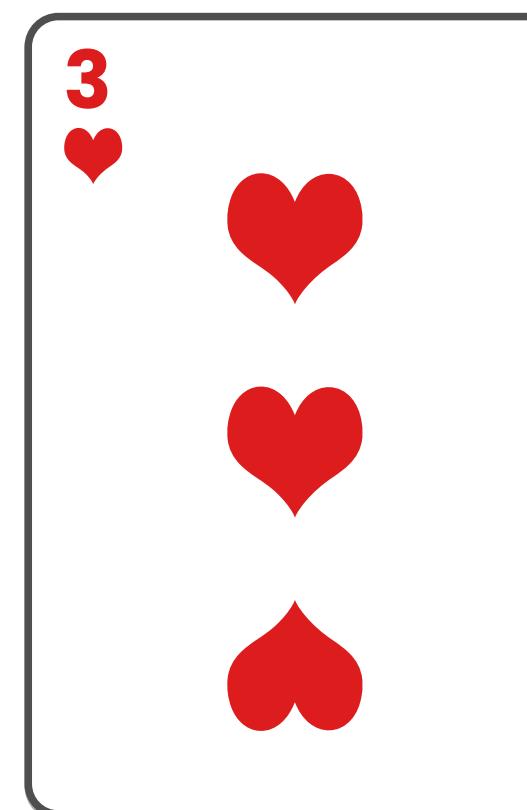
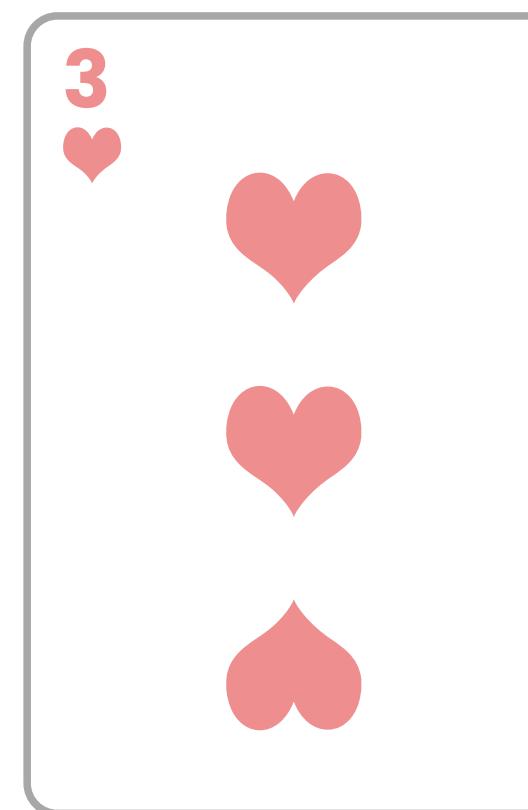
## Ordenació per inserció



## Ordenació per inserció

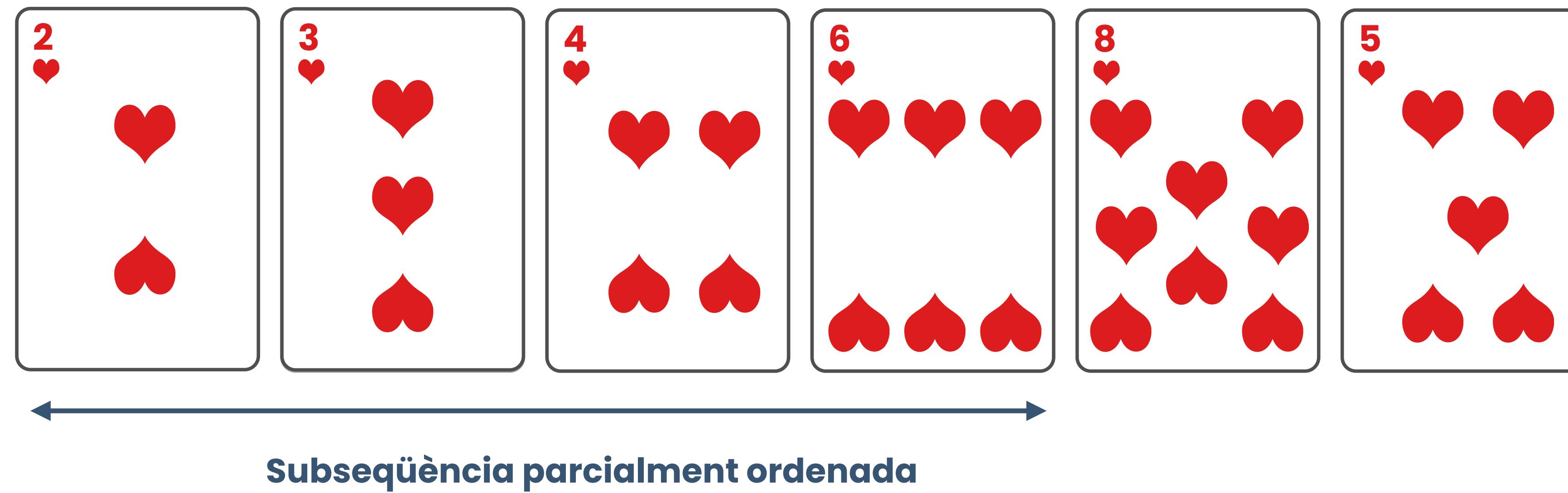


No puc anar més a l'esquerra



Subseqüència parcialment ordenada

## Ordenació per inserció



## Ordenació per inserció



## Ordenació per inserció



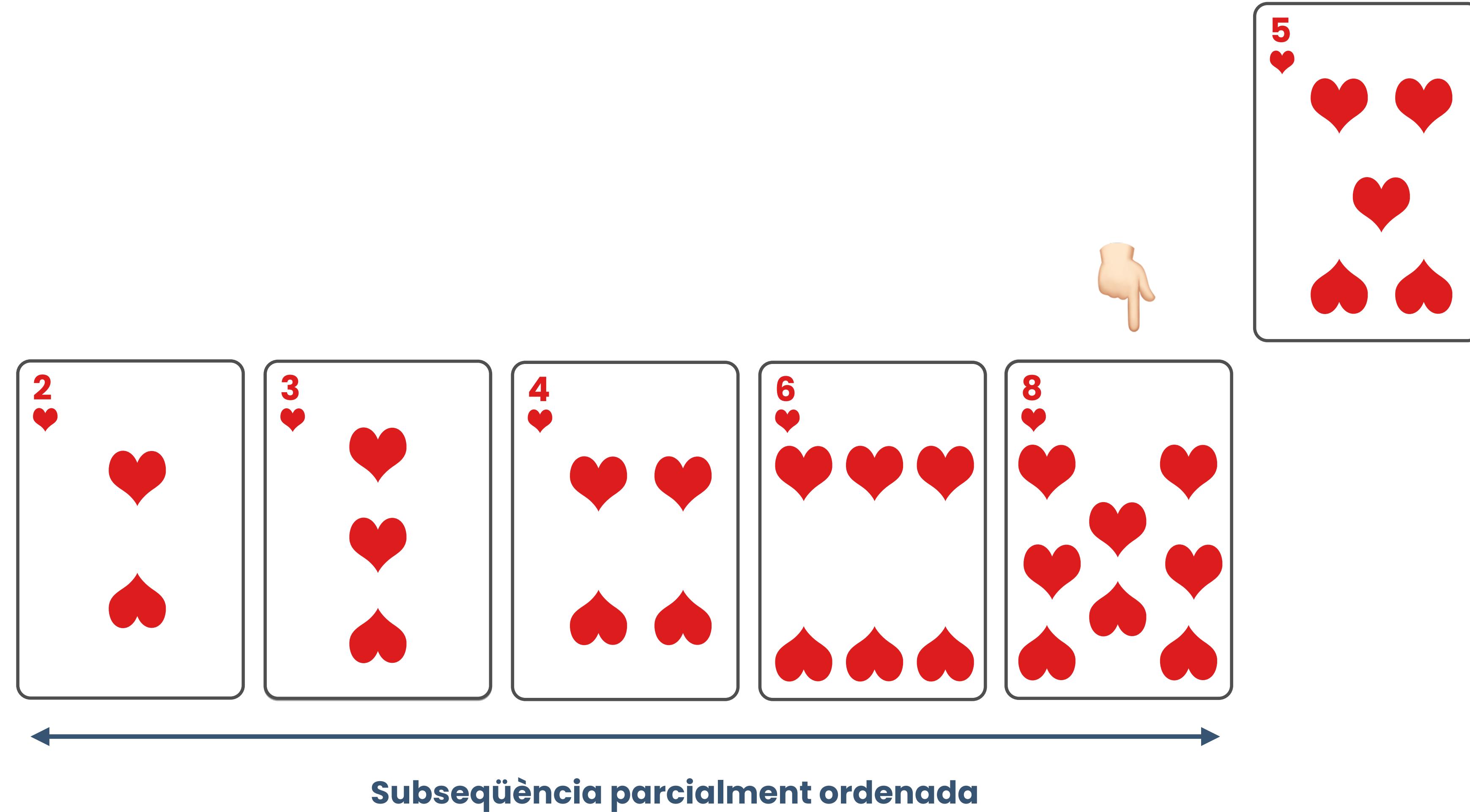
## Ordenació per inserció



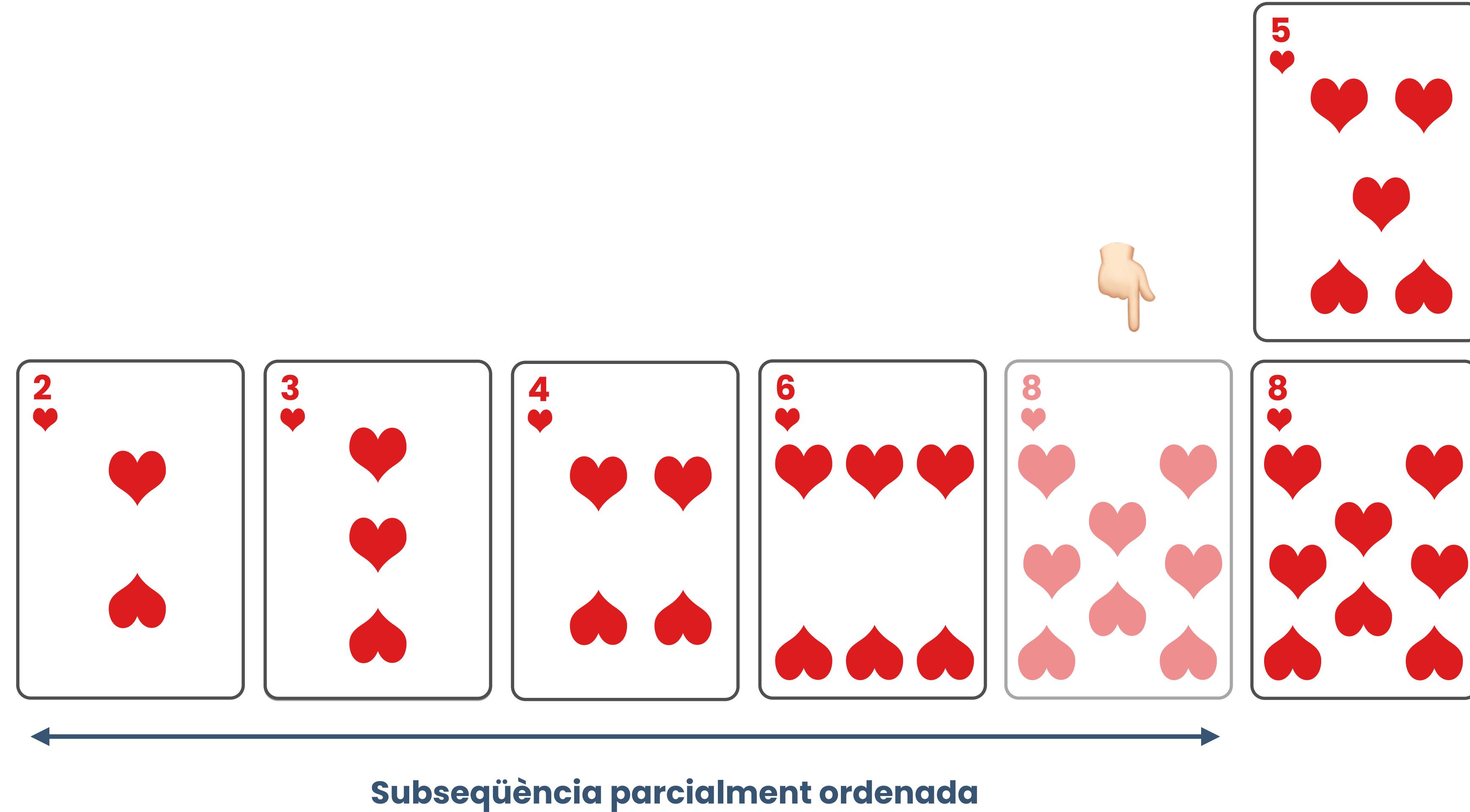
## Ordenació per inserció



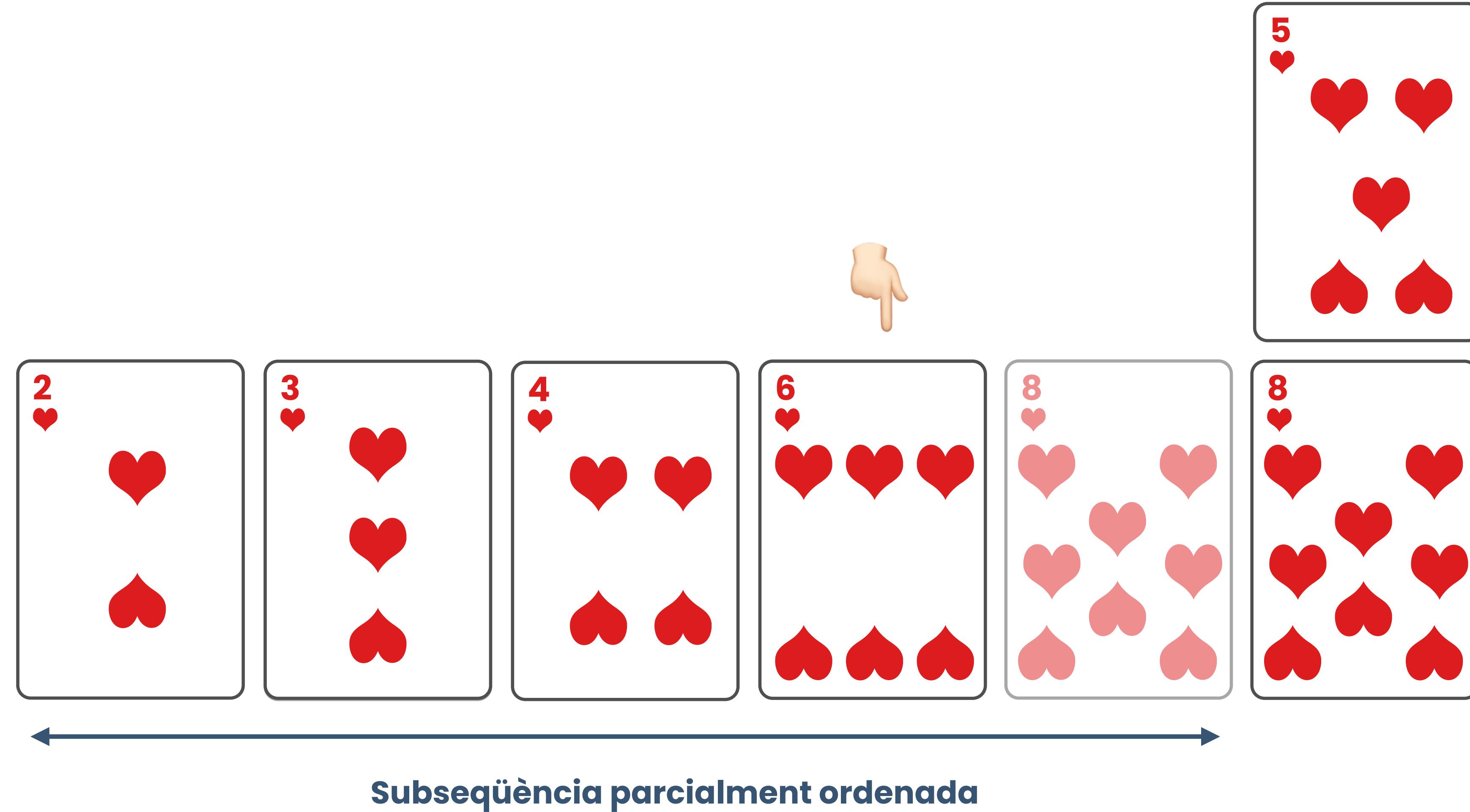
## Ordenació per inserció



## Ordenació per inserció



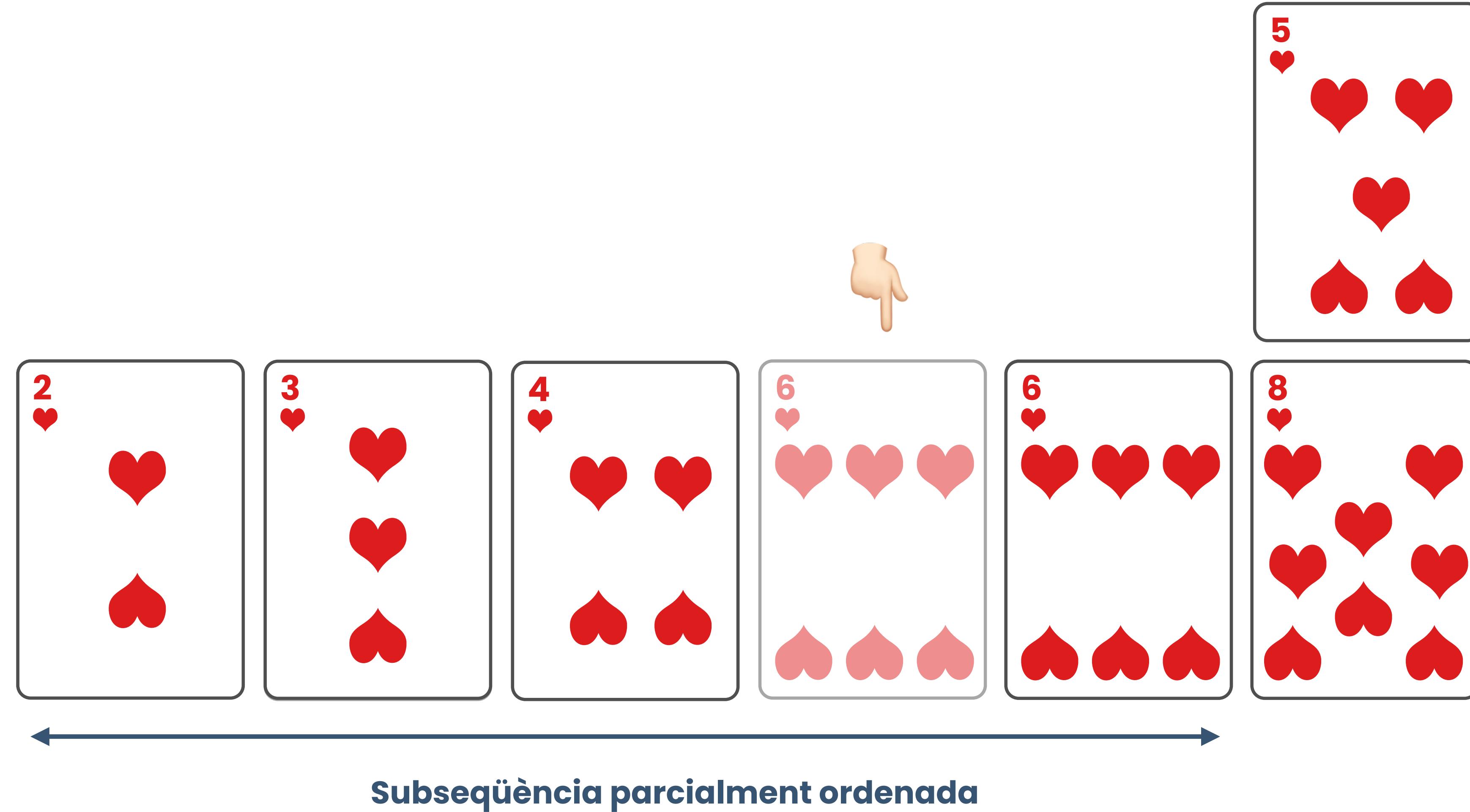
## Ordenació per inserció



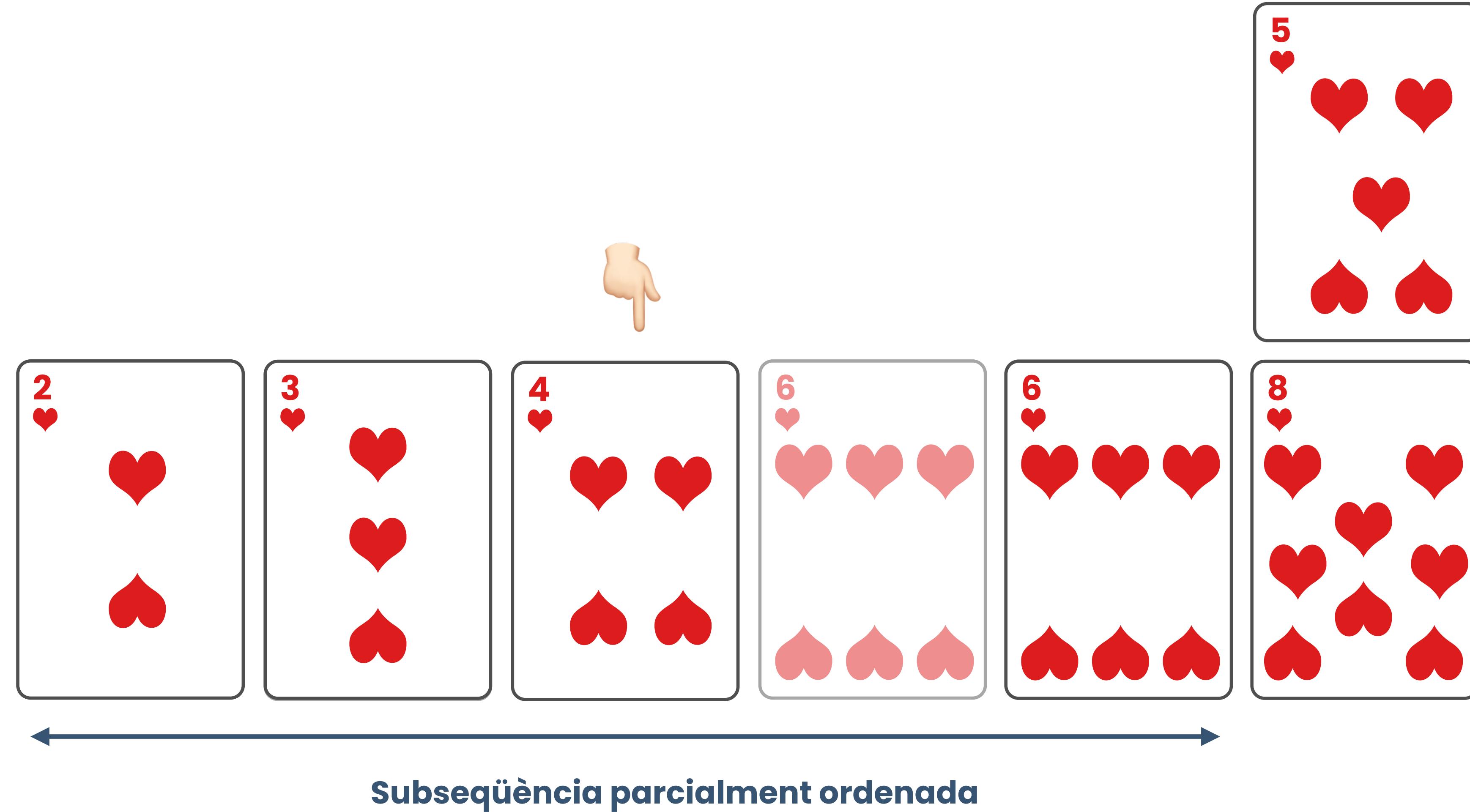
## Ordenació per inserció



## Ordenació per inserció



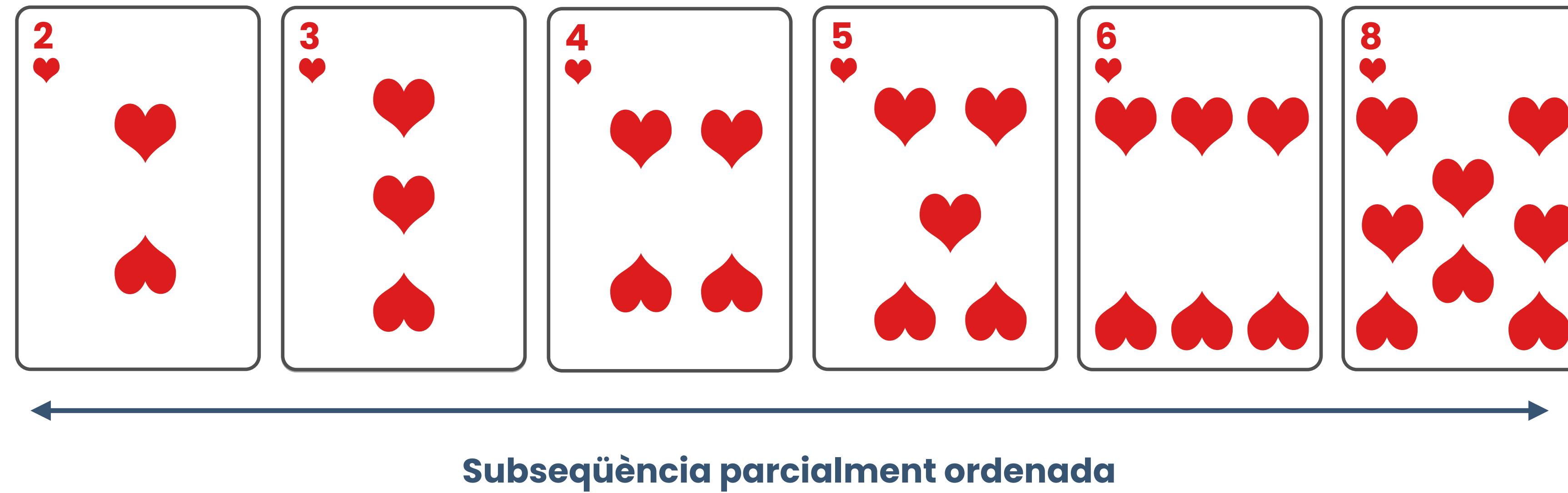
## Ordenació per inserció



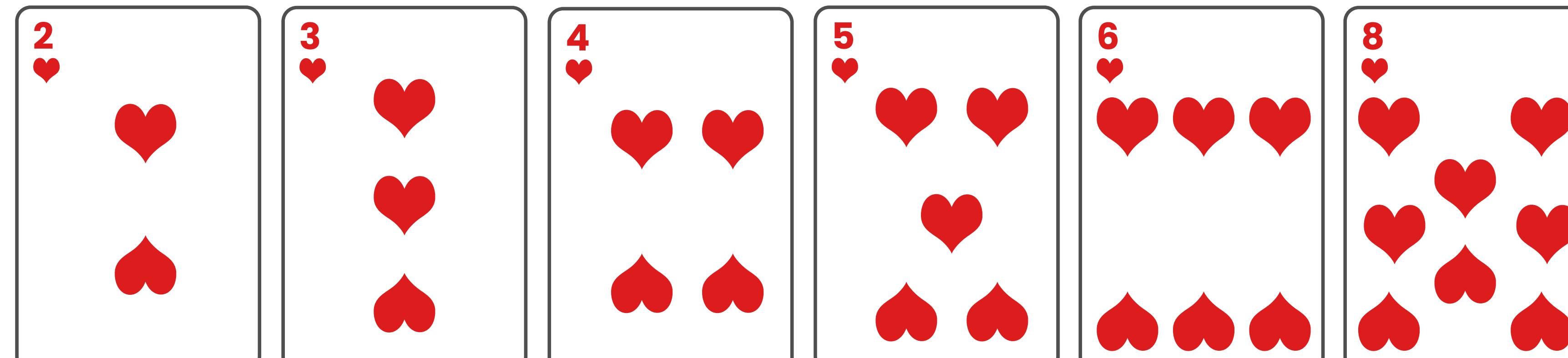
# Ordenació per inserció



## Ordenació per inserció



## Ordenació per inserció



Subseqüència parcialment ordenada

HEM ACABAT !

## Ordenació per inserció

### Algorisme

```
$ Suposem ordenació creixent
acció ordenacio_insercio (v: taula[ ] d'enter, n_elems: enter) és
var
    i, j : enter;
    elem : enter;
fvar
inici
    $ Començo a i=1
    per (i := 1; i < n_elems; i := i + 1) fer
        elem := v[i];
        j := i;
        mentre (v[j-1] > elem i j > 0) fer
            $ Desplaço un cap a la dreta
            v[j] := v[j-1];
            j := j - 1;
        fmentre
        $ Faig la inserció a la posició trobada
        v[j] := elem;
    fper
facció
```

## Ordenació per inserció

### Anàlisi de costos:

- En el **millor cas**, el vector ja està ordenat: { 1, 2, 3, 4, 5 }
- L'algorisme ha de recórrer una vegada tota la seqüència per adonar-se'n que cada element està al seu lloc.

## Ordenació per inserció

### Anàlisi de costos:

- En el **millor cas**, el vector ja està ordenat: { 1, 2, 3, 4, 5 }
- L'algorisme ha de recórrer una vegada tota la seqüència per adonar-se'n que cada element està al seu lloc.

**Best case =  $O(n)$**

## Ordenació per inserció

### Anàlisi de costos:

- En el **millor cas**, el vector ja està ordenat: { 1, 2, 3, 4, 5 }
- L'algorisme ha de recórrer una vegada tota la seqüència per adonar-se'n que cada element està al seu lloc.

**Best case =  $O(n)$**

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

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

<b>Exemple</b>	<b>Posició de la clau</b>	<b>Nombre de comparacions</b>	<b>Nombre de moviments</b>	<b>Total</b>
El 4	Posició 2	1	1	2
El 3	Posició 3	2	2	4
El 2	Posició 4	3	3	6
L'1	Posició n	$n-1$	$n-1$	$2(n-1)$

En el pitjor cas s'ha de fer tot això, per tant:

$$2(1) + 2(2) + 3(2) + \dots + 2(n-1) = 2 (1+2+3+\dots+n) = 2n(n-1)/2$$

**Worst case = O( $n^2$ )**

## Ordenació per inserció

### Anàlisi de costos:

- En el **millor cas**, el vector ja està ordenat: { 1, 2, 3, 4, 5 }
- L'algorisme ha de recórrer una vegada tota la seqüència per adonar-se'n que cada element està al seu lloc.

**Best case =  $O(n)$**

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

**Worst case =  $O(n^2)$**

## Ordenació per inserció

### Anàlisi de costos:

- En el **millor cas**, el vector ja està ordenat: { 1, 2, 3, 4, 5 }
- L'algorisme ha de recórrer una vegada tota la seqüència per adonar-se'n que cada element està al seu lloc.

**Best case =  $O(n)$**

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

**Worst case =  $O(n^2)$**

- En el **cas mig**:

**Average case =  $O(n^2)$**