

3.3 Tri Rapide



Tri rapide - Quicksort



- Développé par Tony Hoare (*Communications of the ACM*, 1959), étudiant invité à Moscou (URSS) pour travailler sur un logiciel de traduction du russe à l'anglais.
- Comme pour le tri fusion, l'approche récursive, mais ...
- On manipule **avant** les appels récursifs
- La taille / position des 2 sous-tableaux à trier récursivement dépend des données

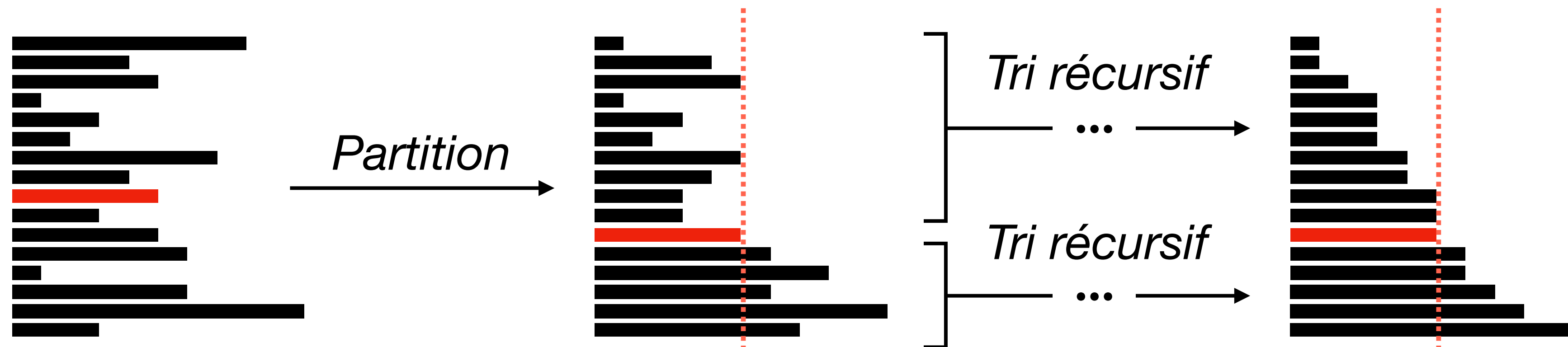


Sir Charles Antony Richard Hoare

Idée générale



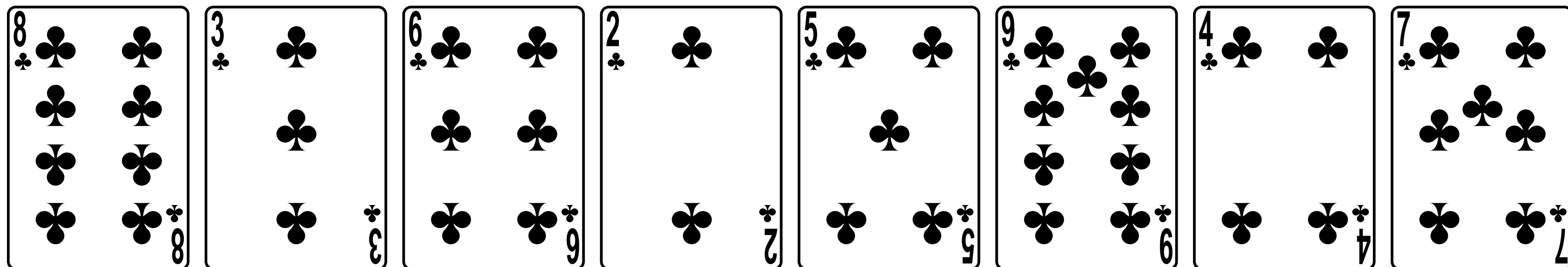
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



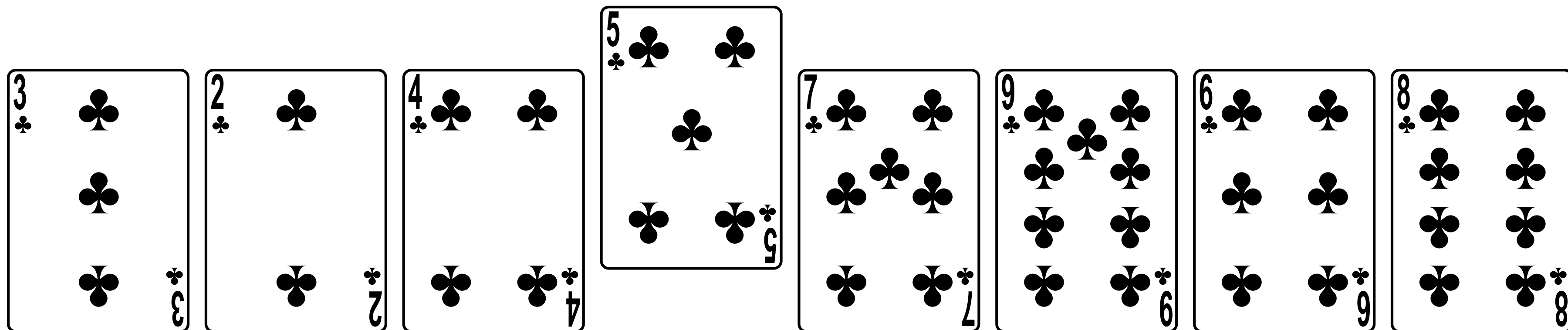
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



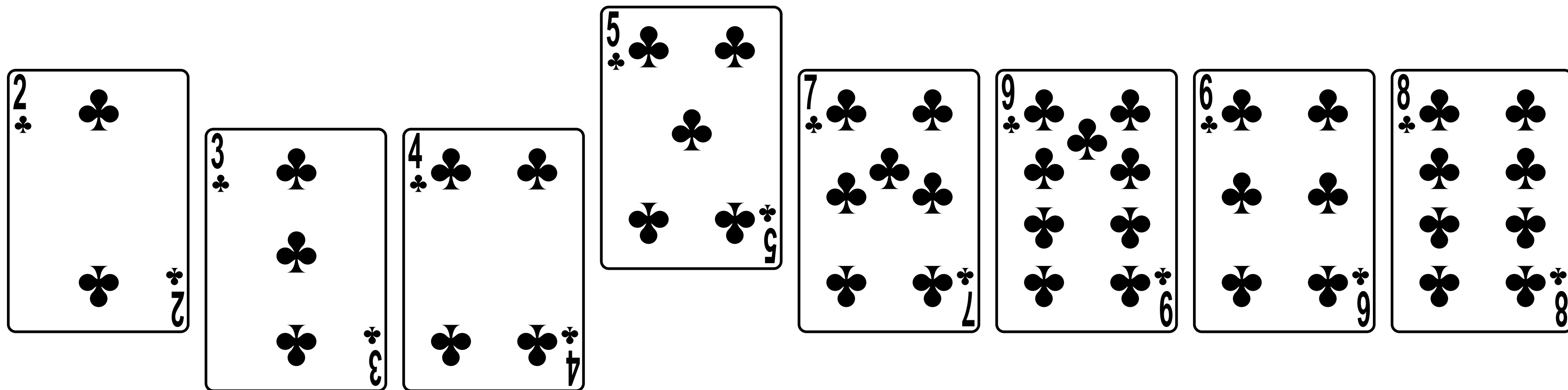
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



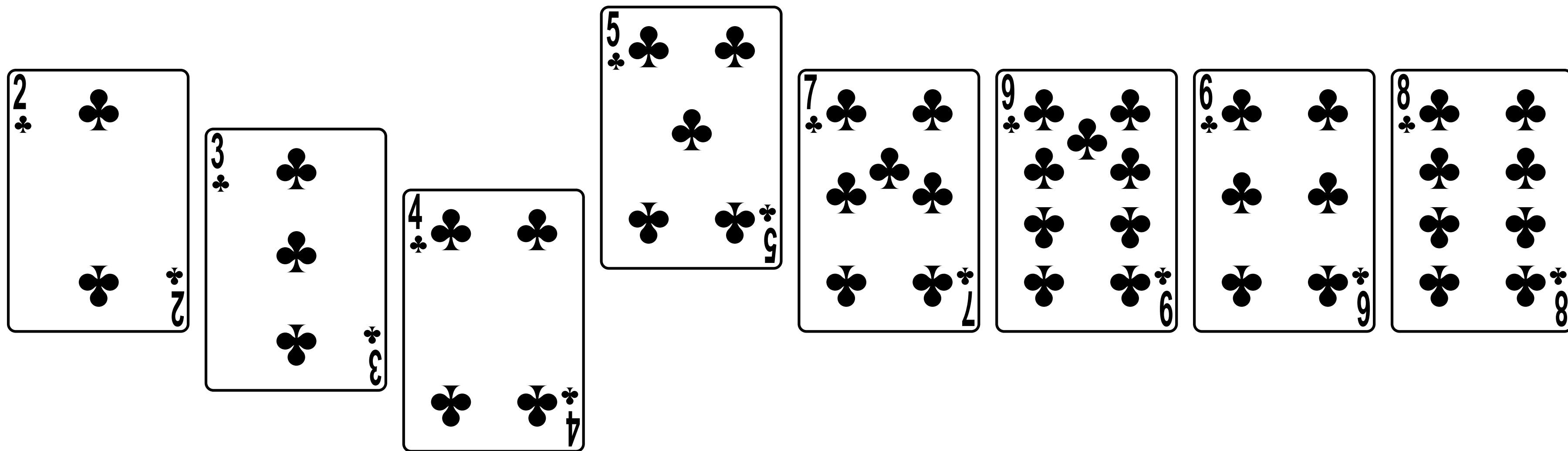
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



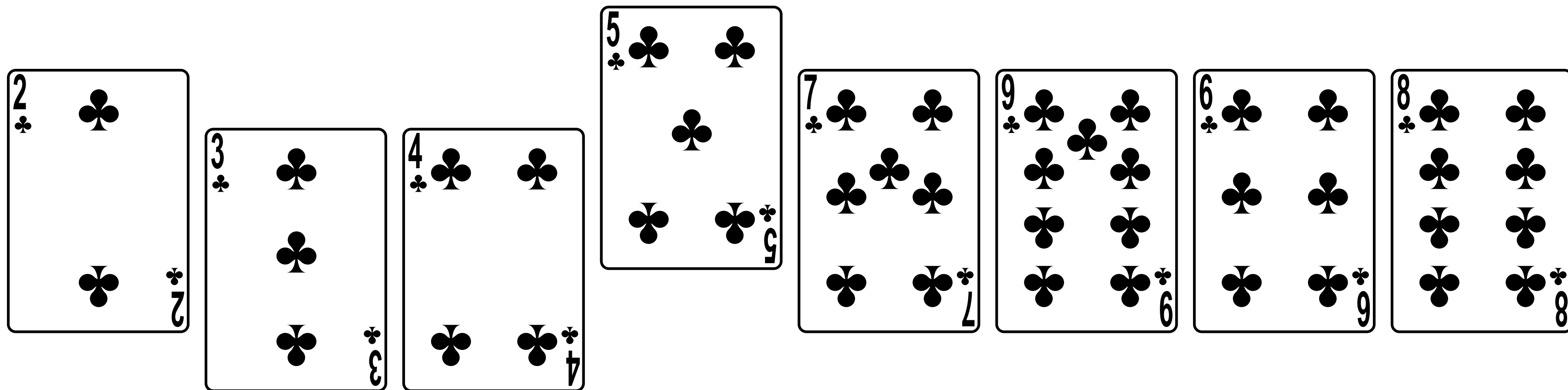
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



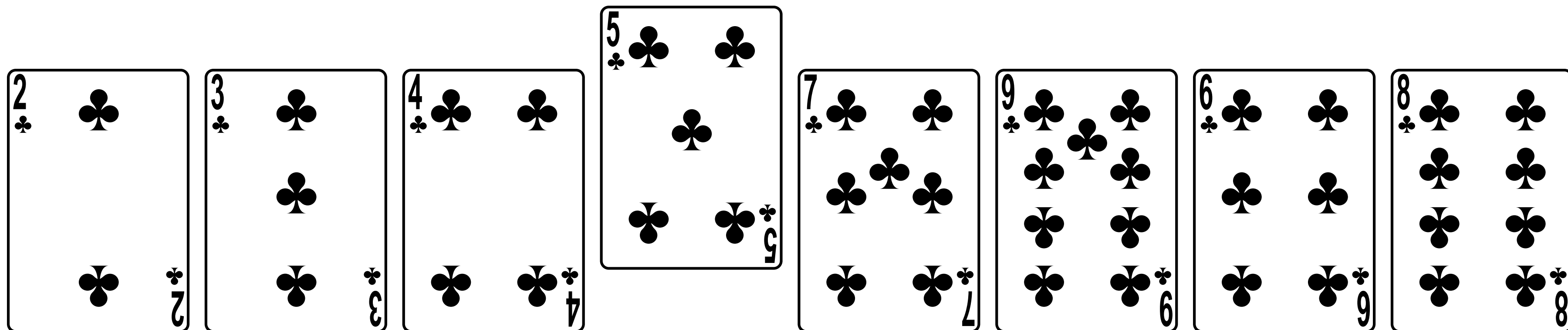
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



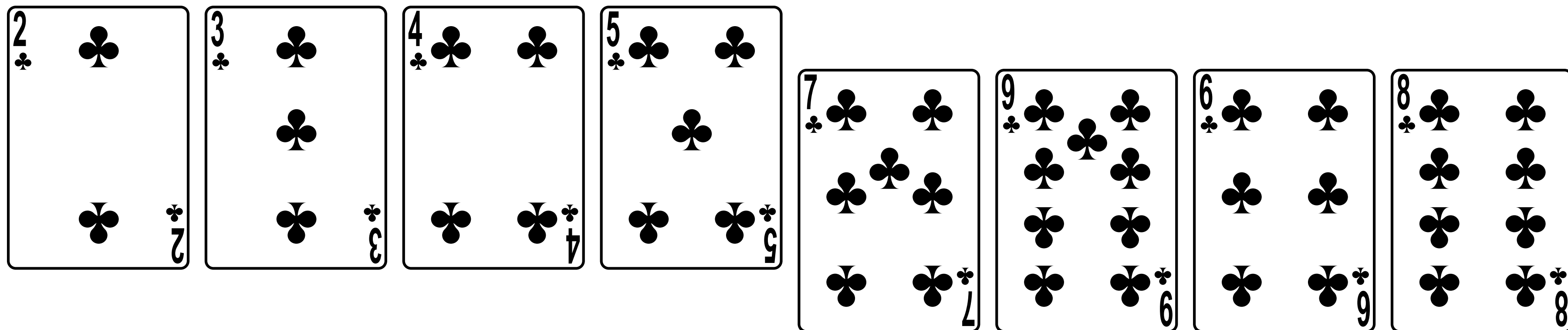
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



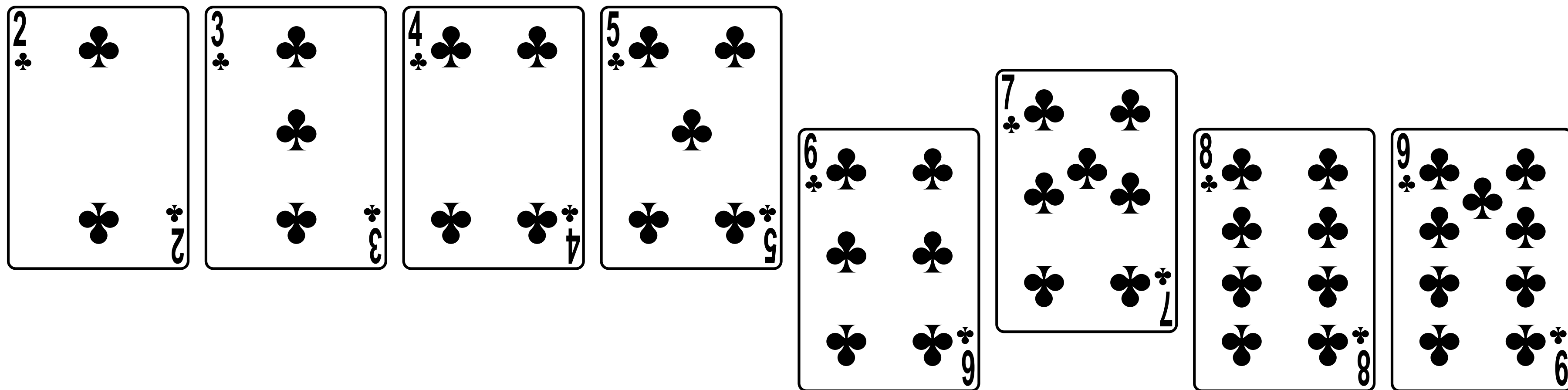
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



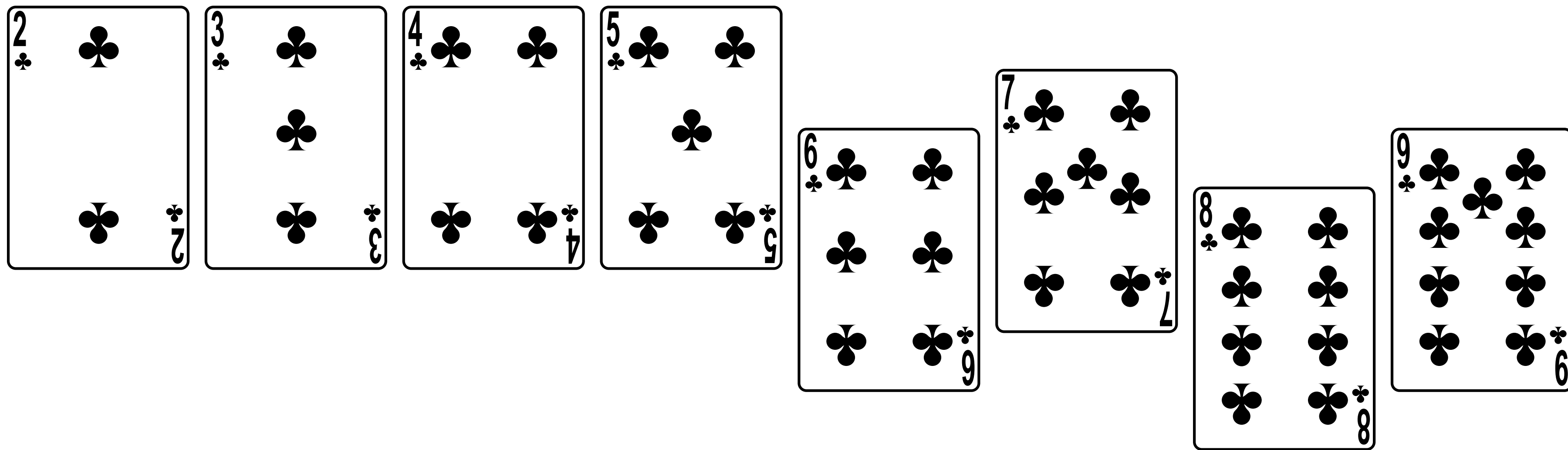
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



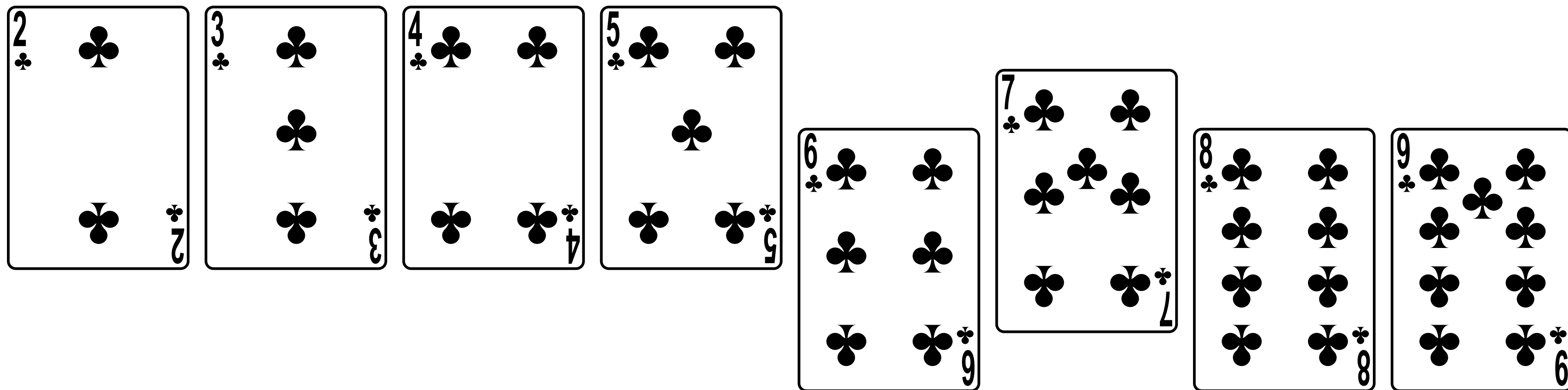
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



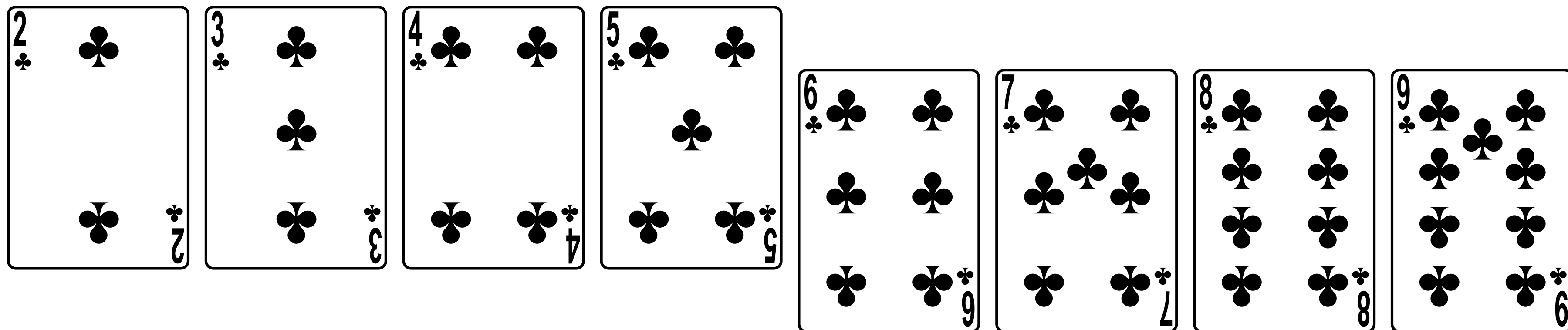
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



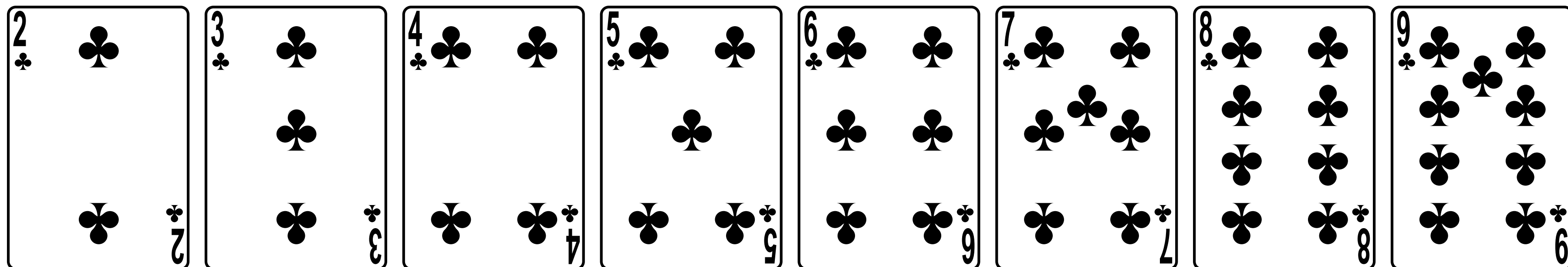
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale



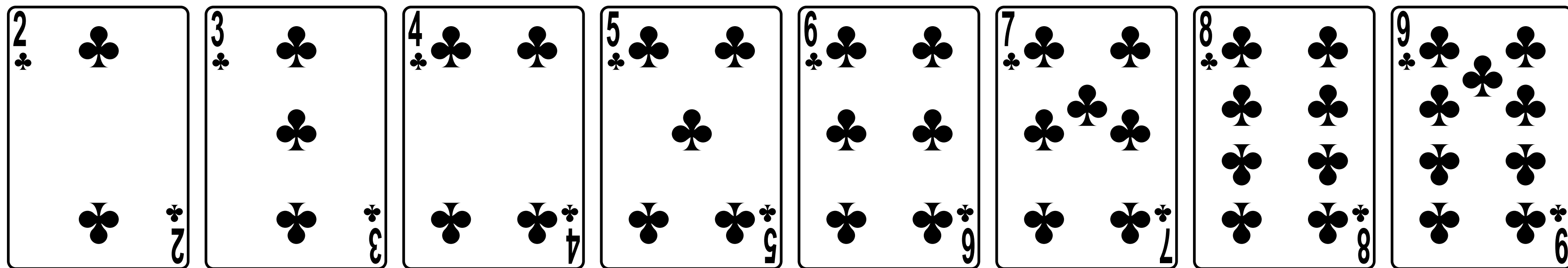
- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



Idée générale

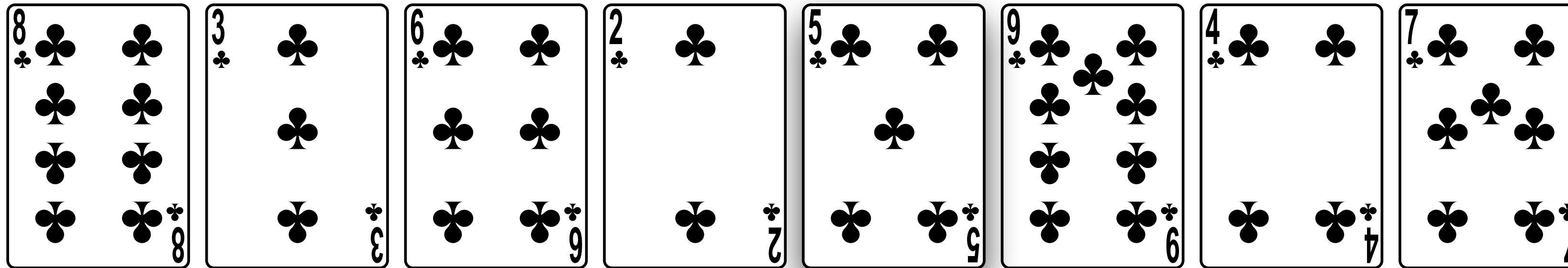


- Choisir un pivot p
- Partitionner les éléments $e \leq p$ à gauche et les $e \geq p$ à droite
- Trier récursivement les sous-tableaux à gauche et droite de p



- Comment partitionner ?

Partition en place

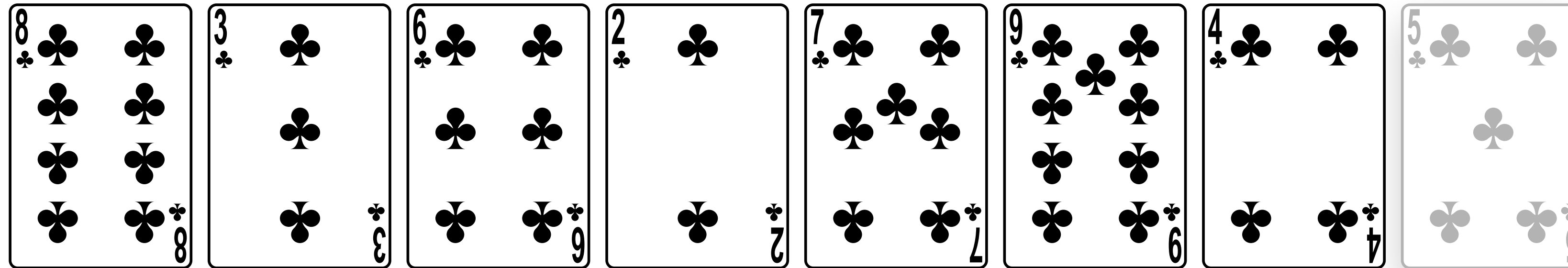


- Placer **p** en fin de tableau **A** pour ne pas qu'il gêne

Partition en place



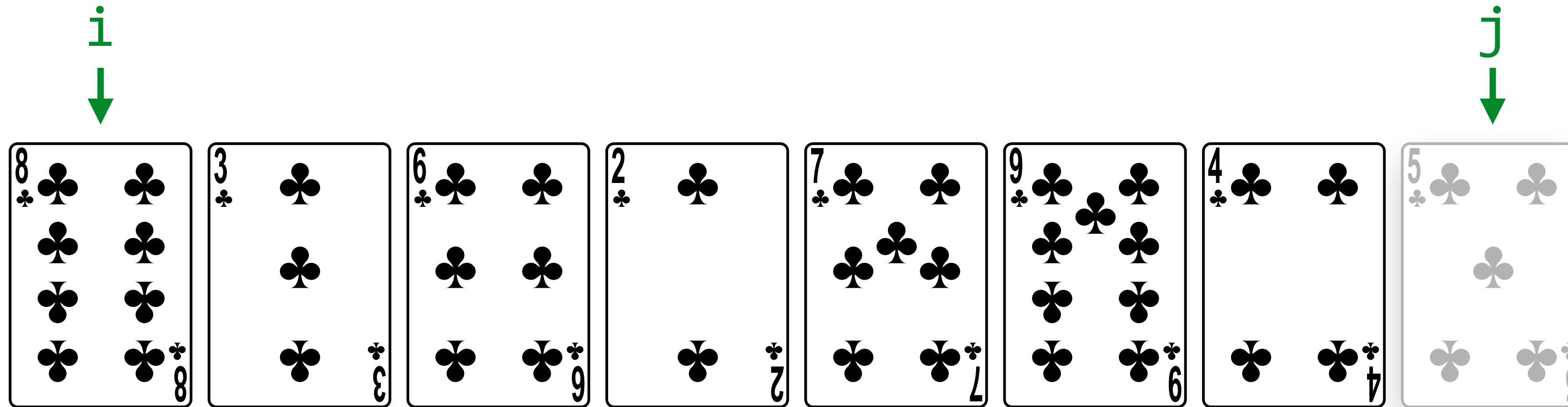
i
↓



j
↓

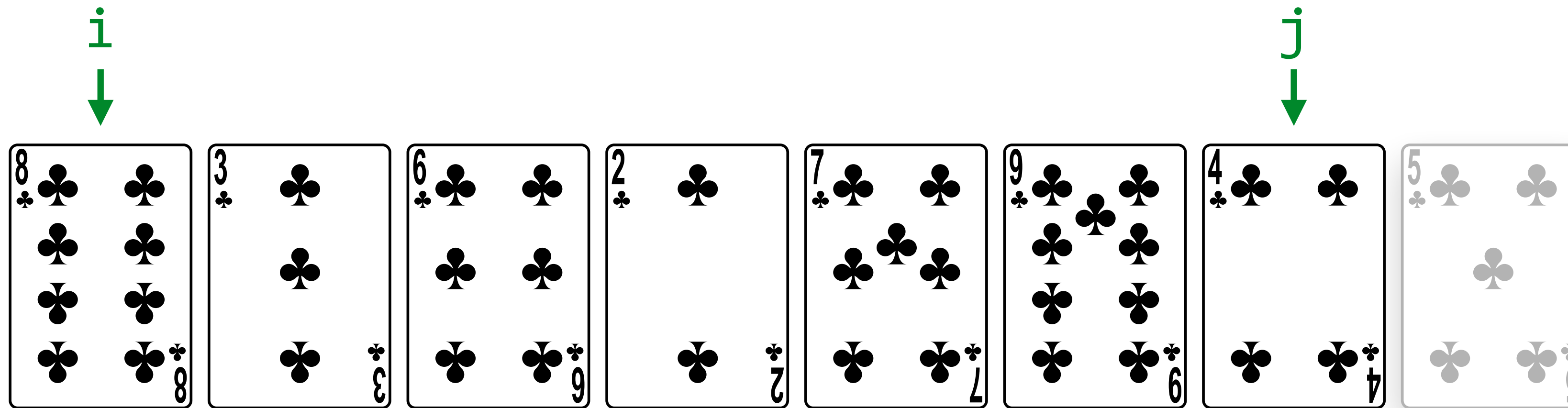
- Placer **p** en fin de tableau **A** pour ne pas qu'il gêne
- **i** et **j** commencent en dehors des deux extrémités

Partition en place



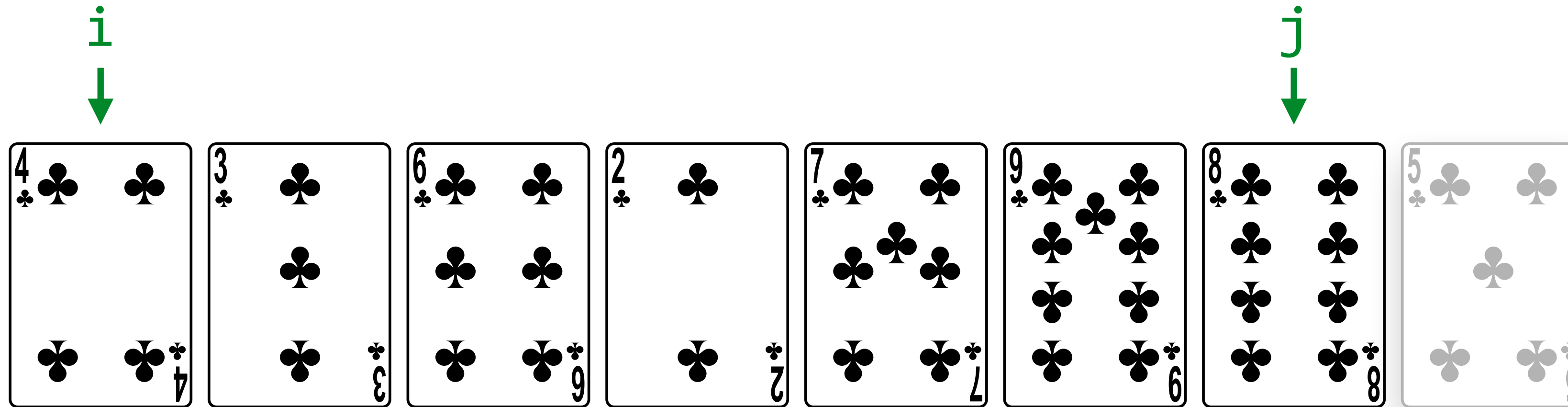
- Placer p en fin de tableau A pour ne pas qu'il gêne
- i et j commencent en dehors des deux extrémités
- Tant que i et j ne se croisent pas
 - i cherche le prochain plus grand que p (*il n'est pas du bon côté*)

Partition en place



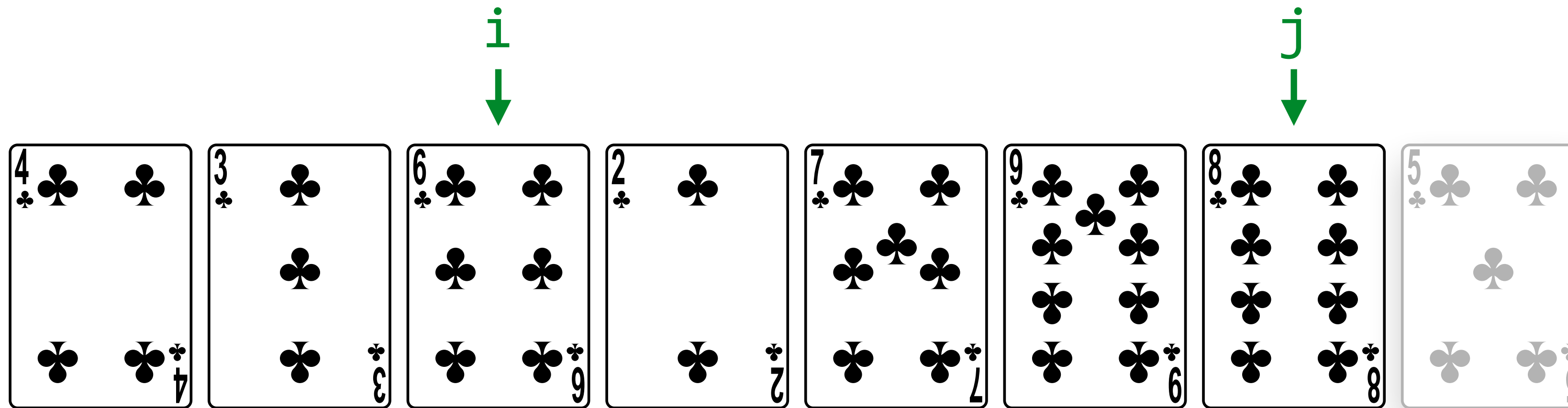
- Placer p en fin de tableau A pour ne pas qu'il gêne
- i et j commencent en dehors des deux extrémités
- Tant que i et j ne se croisent pas
 - i cherche le prochain plus grand que p (*il n'est pas du bon côté*)
 - j cherche le prochain plus petit que p (*il n'est pas du bon côté*)

Partition en place



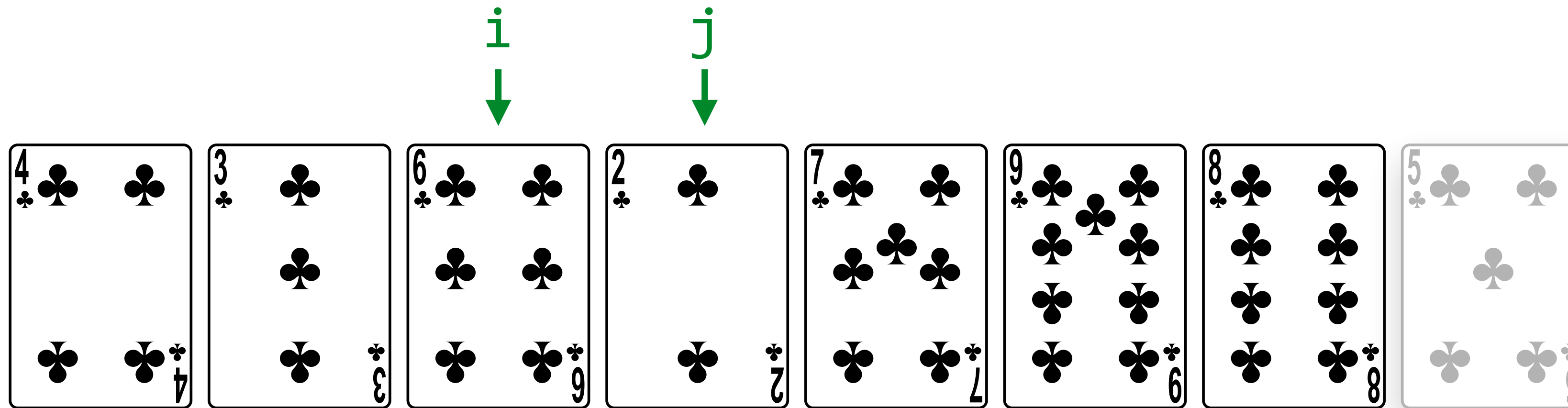
- Placer p en fin de tableau A pour ne pas qu'il gêne
- i et j commencent en dehors des deux extrémités
- Tant que i et j ne se croisent pas
 - i cherche le prochain plus grand que p (*il n'est pas du bon côté*)
 - j cherche le prochain plus petit que p (*il n'est pas du bon côté*)
 - Échanger $A(i)$ et $A(j)$ pour maintenir $A(i) \leq p$ et $A(j) \geq p$

Partition en place



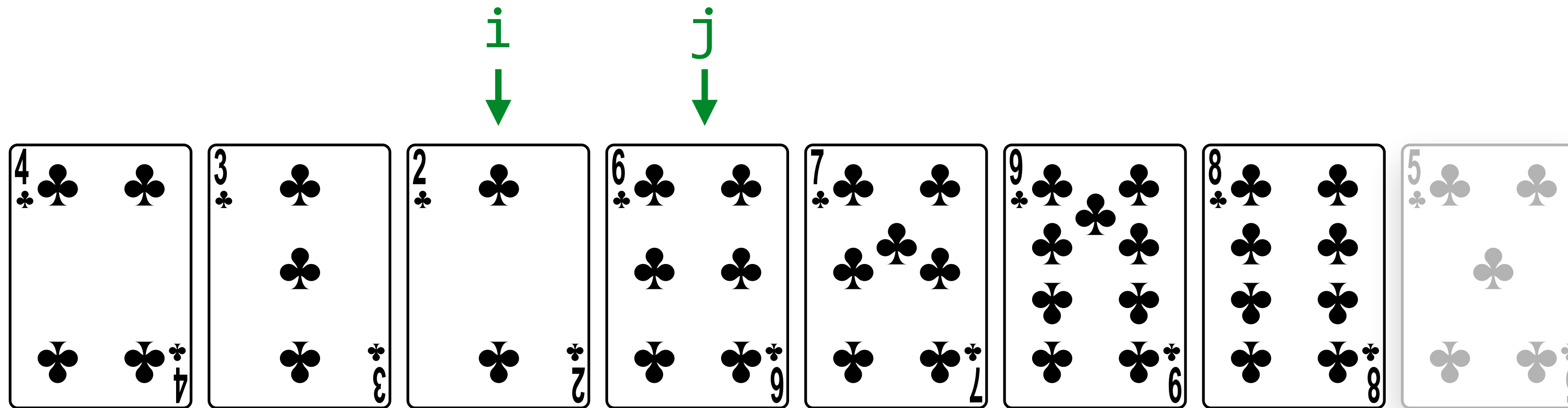
- Placer p en fin de tableau A pour ne pas qu'il gêne
- i et j commencent en dehors des deux extrémités
- Tant que i et j ne se croisent pas
 - i cherche le prochain plus grand que p (*il n'est pas du bon côté*)
 - j cherche le prochain plus petit que p (*il n'est pas du bon côté*)
 - Échanger $A(i)$ et $A(j)$ pour maintenir $A(i) \leq p$ et $A(j) \geq p$

Partition en place



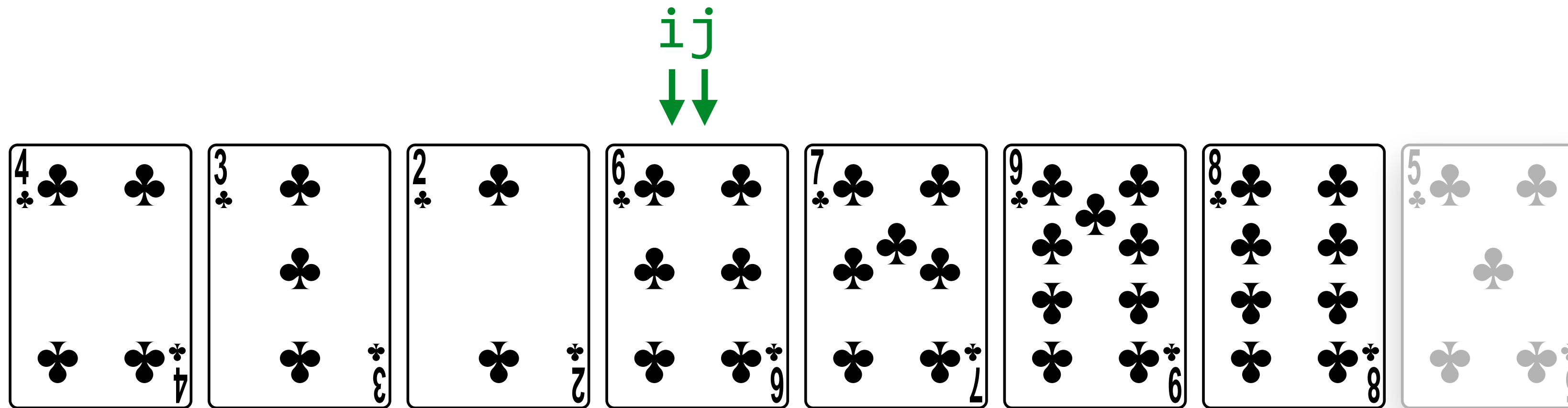
- Placer p en fin de tableau A pour ne pas qu'il gêne
- i et j commencent en dehors des deux extrémités
- Tant que i et j ne se croisent pas
 - i cherche le prochain plus grand que p (*il n'est pas du bon côté*)
 - j cherche le prochain plus petit que p (*il n'est pas du bon côté*)
 - Échanger $A(i)$ et $A(j)$ pour maintenir $A(i) \leq p$ et $A(j) \geq p$

Partition en place



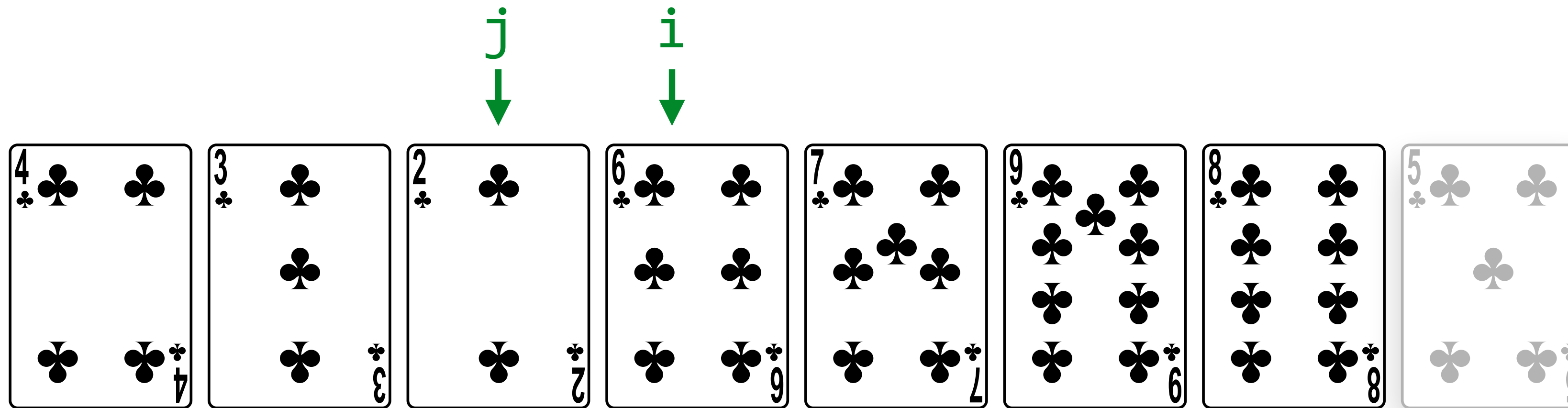
- Placer p en fin de tableau A pour ne pas qu'il gêne
- i et j commencent en dehors des deux extrémités
- Tant que i et j ne se croisent pas
 - i cherche le prochain plus grand que p (*il n'est pas du bon côté*)
 - j cherche le prochain plus petit que p (*il n'est pas du bon côté*)
 - Échanger $A(i)$ et $A(j)$ pour maintenir $A(i) \leq p$ et $A(j) \geq p$

Partition en place



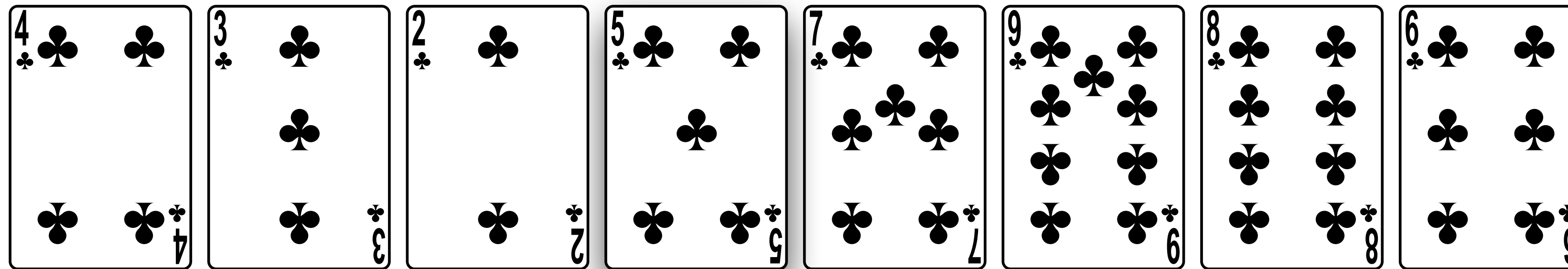
- Placer p en fin de tableau A pour ne pas qu'il gêne
- i et j commencent en dehors des deux extrémités
- Tant que i et j ne se croisent pas
 - i cherche le prochain plus grand que p (*il n'est pas du bon côté*)
 - j cherche le prochain plus petit que p (*il n'est pas du bon côté*)
 - Échanger $A(i)$ et $A(j)$ pour maintenir $A(i) \leq p$ et $A(j) \geq p$

Partition en place



- Placer p en fin de tableau A pour ne pas qu'il gêne
- i et j commencent en dehors des deux extrémités
- Tant que i et j ne se croisent pas
 - i cherche le prochain plus grand que p (*il n'est pas du bon côté*)
 - j cherche le prochain plus petit que p (*il n'est pas du bon côté*)
 - Échanger $A(i)$ et $A(j)$ pour maintenir $A(i) \leq p$ et $A(j) \geq p$

Partition en place



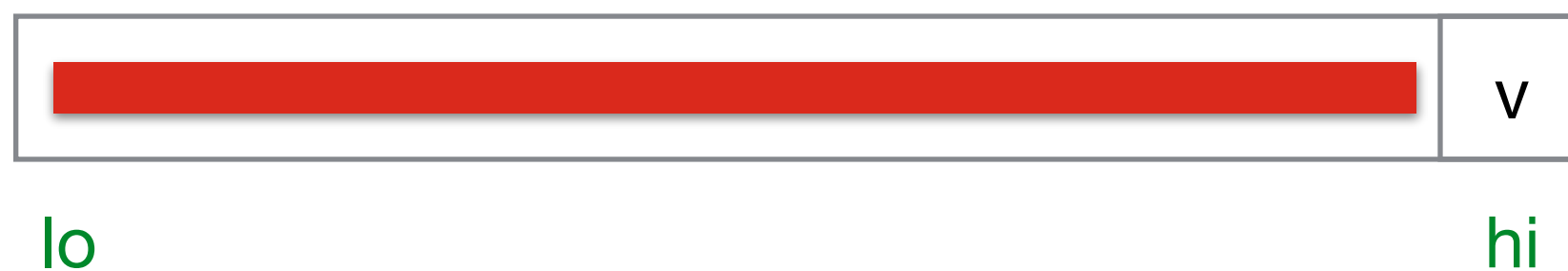
- Placer p en fin de tableau A pour ne pas qu'il gêne
- i et j commencent en dehors des deux extrémités
- Tant que i et j ne se croisent pas
 - i cherche le prochain plus grand que p (*il n'est pas du bon côté*)
 - j cherche le prochain plus petit que p (*il n'est pas du bon côté*)
 - Échanger $A(i)$ et $A(j)$ pour maintenir $A(i) \leq p$ et $A(j) \geq p$
- Placer p entre les deux partitions

Tri rapide : algorithme de partition



- Effectue la partition des éléments du tableau A de l'indice lo à l'indice hi en utilisant $A(hi)$ comme pivot.
- Retourne la position du pivot dans le tableau partitionné.

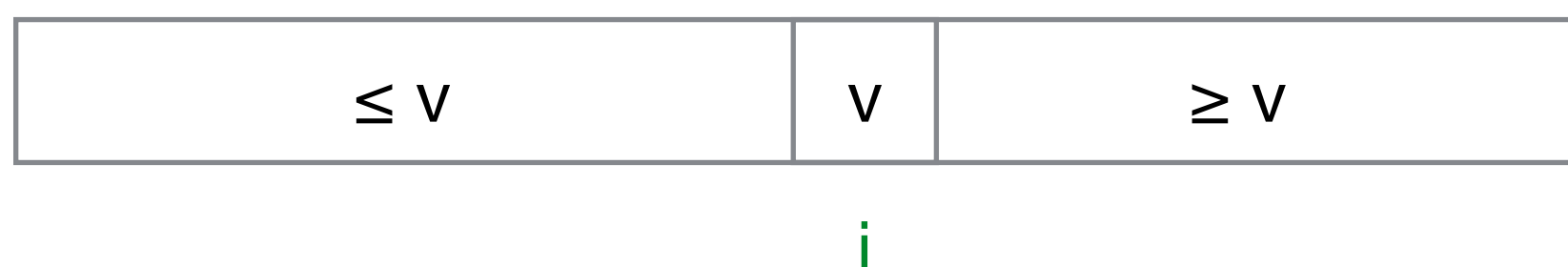
Avant



Pendant



Après



fonction Partition(A, lo, hi)

$i \leftarrow lo-1, j \leftarrow hi$

boucler

répéter incrémenter i
tant que $A(i) < A(hi)$

répéter décrémenter j
tant que $j > lo$ et $A(hi) < A(j)$

si $i \geq j$, alors sortir boucle
fin si

permuter $A(i)$ et $A(j)$

fin boucler

permuter $A(i)$ et $A(hi)$
retourner i

Partition avec pivot I



E	X	E	M	P	L	E	D	E	T	R	<u>I</u>
E	X	E	M	P	L	E	D	E	T	R	<u>I</u>
E	X	E	M	P	L	E	D	E	T	R	<u>I</u>
E	X	E	M	P	L	E	D	E	T	R	<u>I</u>
E	X	E	M	P	L	E	D	E	T	R	<u>I</u>
E	X	E	M	P	L	E	D	E	T	R	<u>I</u>
E	X	E	M	P	L	E	D	E	T	R	<u>I</u>
E	E	E	M	P	L	E	D	X	T	R	<u>I</u>
E	E	E	M	P	L	E	D	X	T	R	<u>I</u>
E	E	E	M	P	L	E	D	X	T	R	<u>I</u>
E	E	E	M	P	L	E	D	X	T	R	<u>I</u>
E	E	E	D	P	L	E	M	X	T	R	<u>I</u>
E	E	E	D	P	L	E	M	X	T	R	<u>I</u>
E	E	E	D	P	L	E	M	X	T	R	<u>I</u>
E	E	E	D	E	L	P	M	X	T	R	<u>I</u>
E	E	E	D	E	L	P	M	X	T	R	<u>I</u>
E	E	E	D	E	L	P	M	X	T	R	<u>I</u>
E	E	E	D	E	<u>I</u>	P	M	X	T	R	L

```

i=0, j=12
++i = 1
++i = 2
--j = 11
--j = 10
--j = 9
echange(2,9)
++i = 3
++i = 4
--j = 8
echange(4,8)
++i = 5
--j = 7
echange(5,7)
++i = 6
--j = 6
--j = 5
echange(6,12)
    
```

Tri rapide: algorithme récursif



- Effectue le tri rapide des éléments du tableau A de l'indice lo à l'indice hi (compris).
- Choisit un pivot, le place en position hi , et appelle la fonction partition.
- S'appelle récursivement sur les partitions droite et gauche.

```
fonction TriRapide( $A, lo, hi$ )  
  si  $lo < hi$ , alors  
     $p \leftarrow$  choisir l'élément pivot  
    permuter  $A(hi)$  et  $A(p)$   
     $i \leftarrow$  Partition( $A, lo, hi$ )  
  
    TriRapide( $A, lo, i-1$ )  
    TriRapide( $A, i+1, hi$ )  
  fin si
```

Appels récurrents



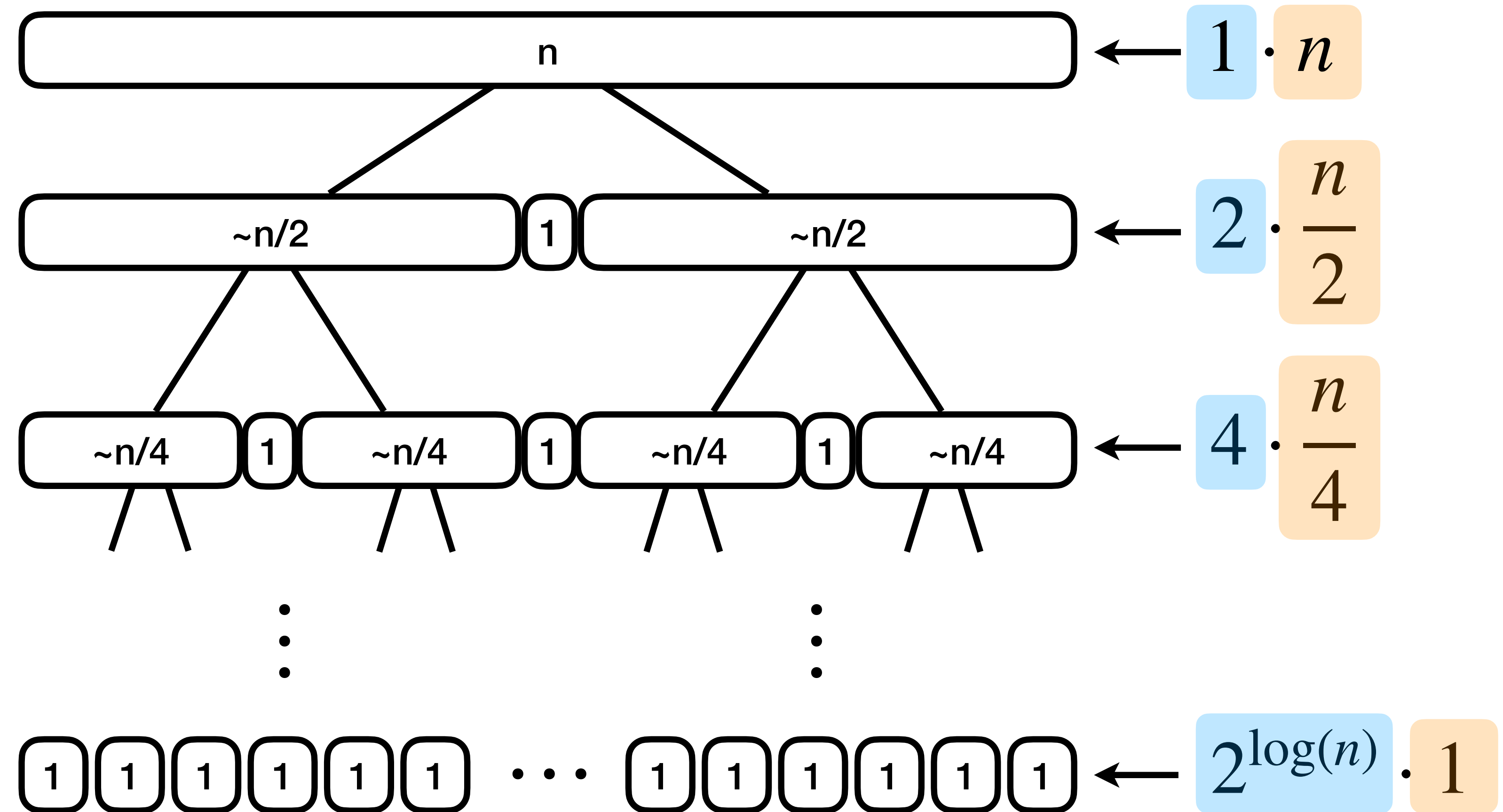
lo	i	hi	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>10</u>	<u>11</u>	<u>12</u>
			E	X	E	M	P	L	E	D	E	T	R	<u>I</u>
1	6	12	E	E	E	D	<u>E</u>	I	P	M	X	T	R	L
1	3	5	D	<u>E</u>	E	E	E	I	P	M	X	T	R	L
1	2	2	<u>D</u>	E	E	E	E	I	P	M	X	T	R	L
1		1	D	E	E	E	<u>E</u>	I	P	M	X	T	R	L
4	5	5	D	E	E	<u>E</u>	E	I	P	M	X	T	R	L
4		4	D	E	E	E	E	I	P	M	X	T	R	<u>L</u>
7	7	12	D	E	E	E	E	I	L	M	X	T	R	<u>P</u>
8	9	12	D	E	E	E	E	I	L	<u>M</u>	P	T	R	X
8		8	D	E	E	E	E	I	L	M	P	T	R	<u>X</u>
10	12	12	D	E	E	E	E	I	L	M	P	T	<u>R</u>	X
10	10	11	D	E	E	E	E	I	L	M	P	R	<u>T</u>	X
11		11	D	E	E	E	E	I	L	M	P	R	T	X
			D	E	E	E	E	I	L	M	P	R	T	X

Complexité : meilleur cas



- Pivot ?
- Médiane des éléments à partitionner
- Partition du tableau en 2 parts égales

appels récurrents *partition*

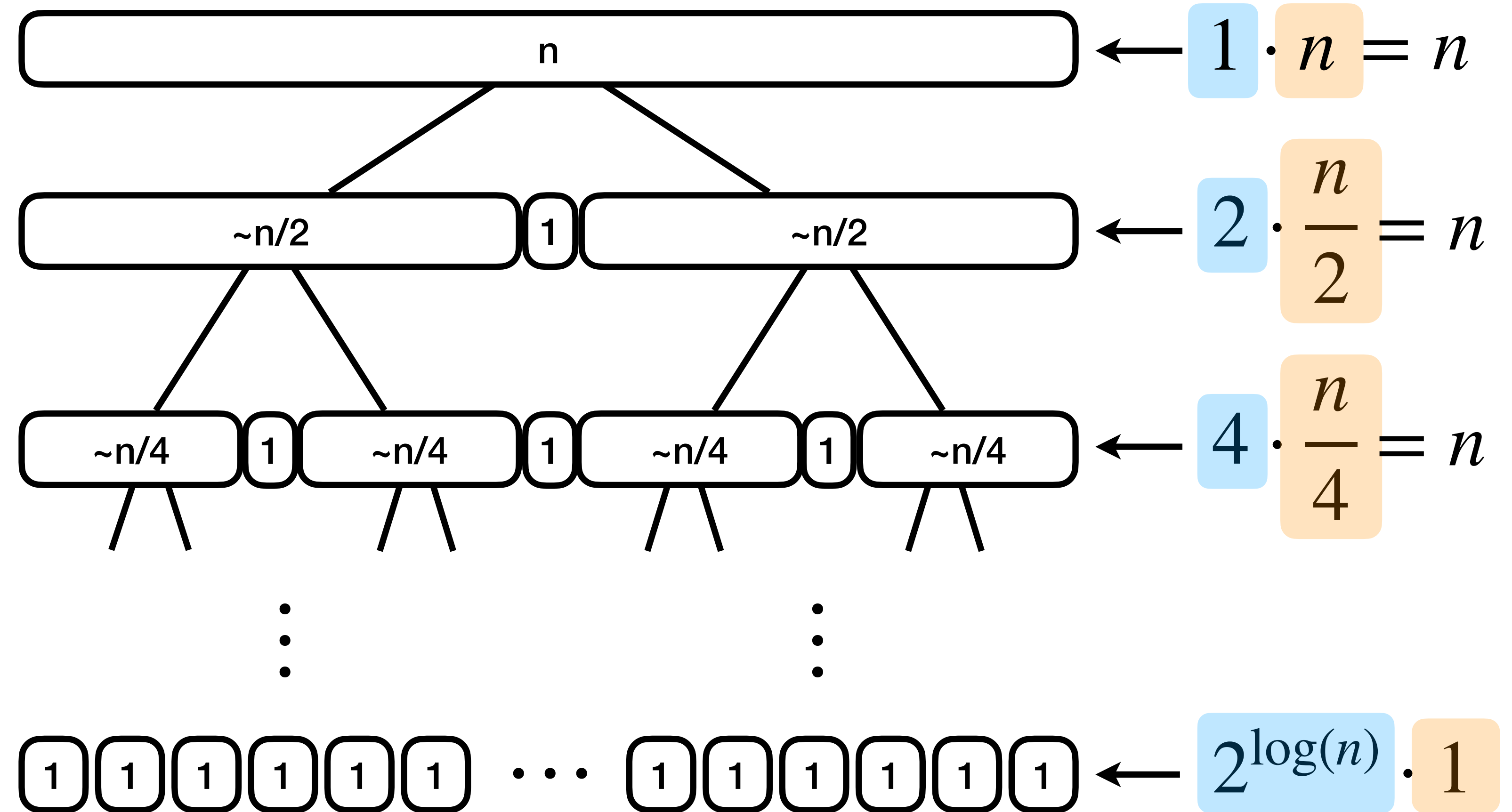


Complexité : meilleur cas



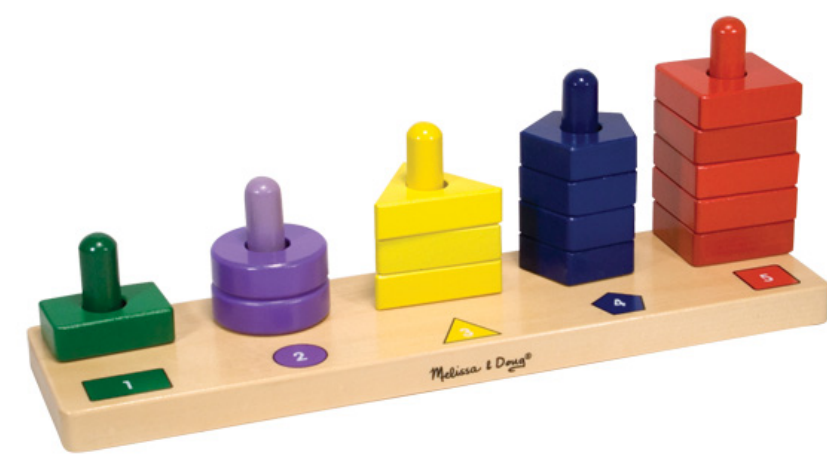
- Pivot ?
- Médiane des éléments à partitionner
- Partition du tableau en 2 parts égales
- Cout par profondeur : $O(n)$
- Profondeur de récursion : $\log_2(n)$
- Complexité en $O(n \cdot \log(n))$
- Comme pour le tri par fusion

appels récursifs *partition*



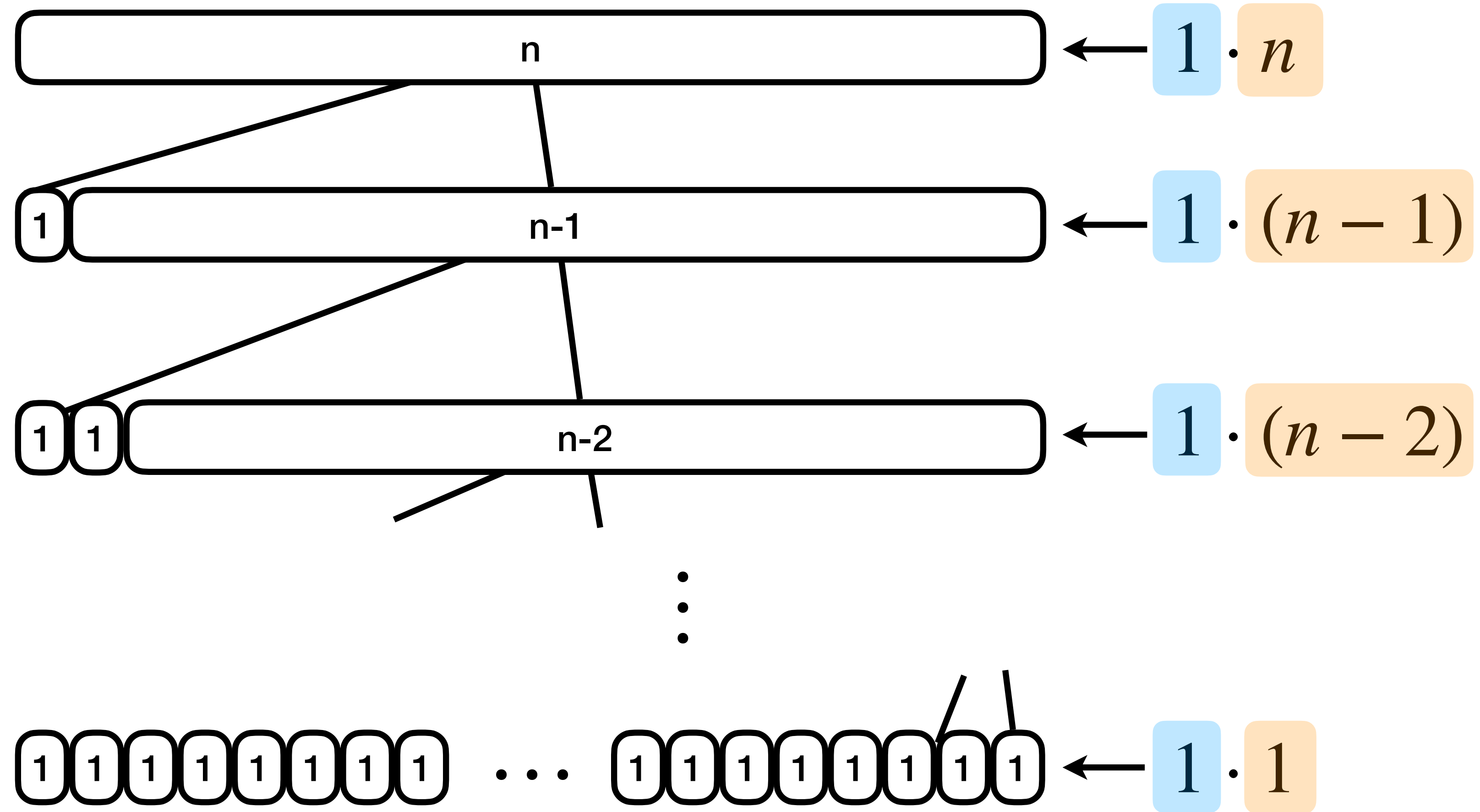
$$\Rightarrow O(n \log(n))$$

Complexité dans le pire cas

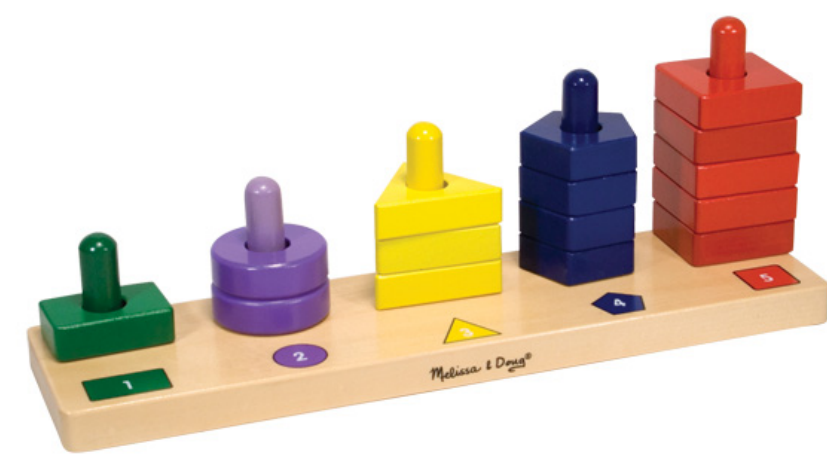


- Pivot ?
- Min ou max des éléments à partitionner
- Partition du tableau avec un côté vide et l'autre a $n-1$ éléments

appels récurifs *partition*

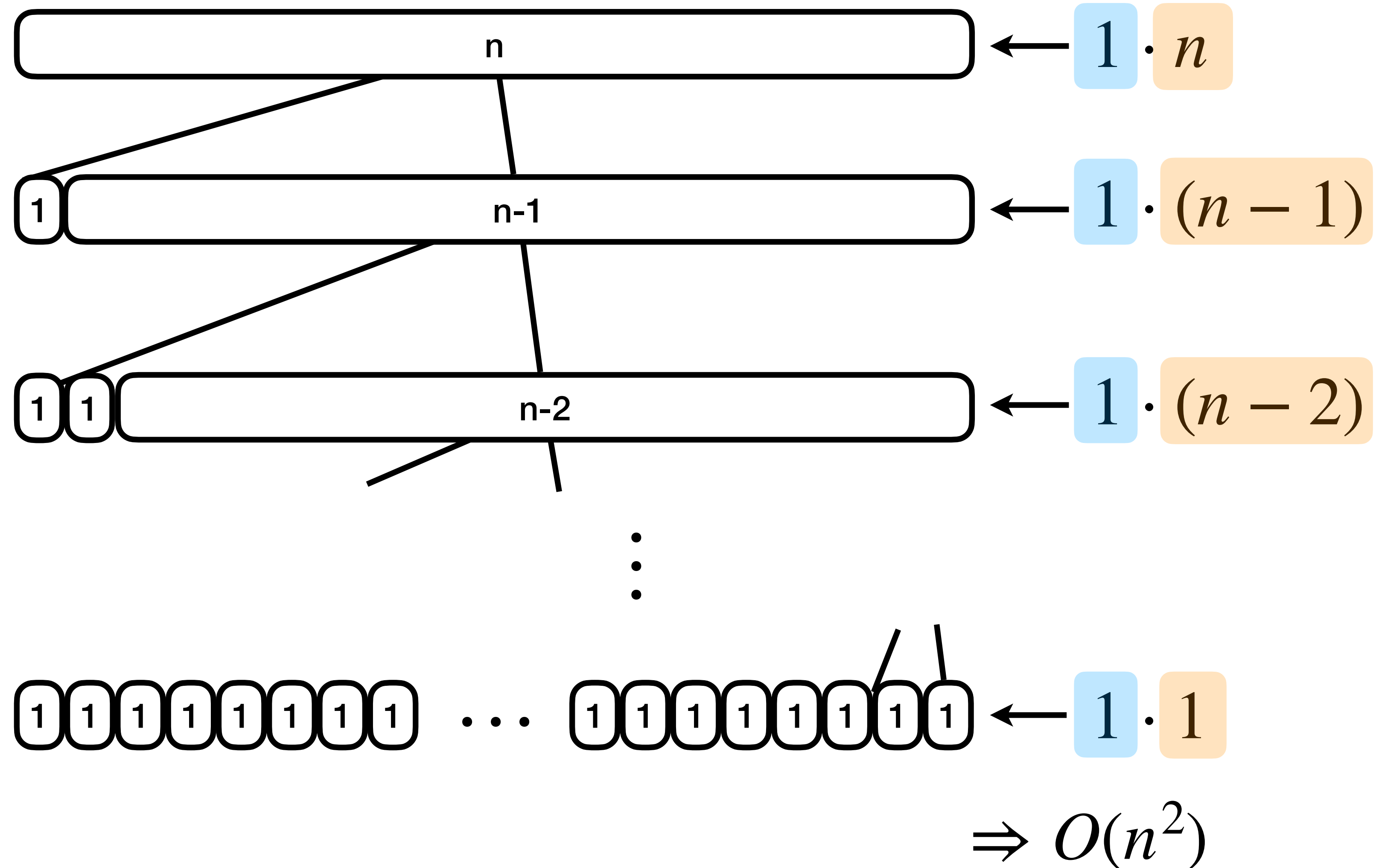


Complexité dans le pire cas



- Pivot ?
- Min ou max des éléments à partitionner
- Partition du tableau avec un côté vide et l'autre a $n-1$ éléments
- Profondeur de récursion : $n-1$
- Complexité en $O(n^2)$
- comme pour les tris simples

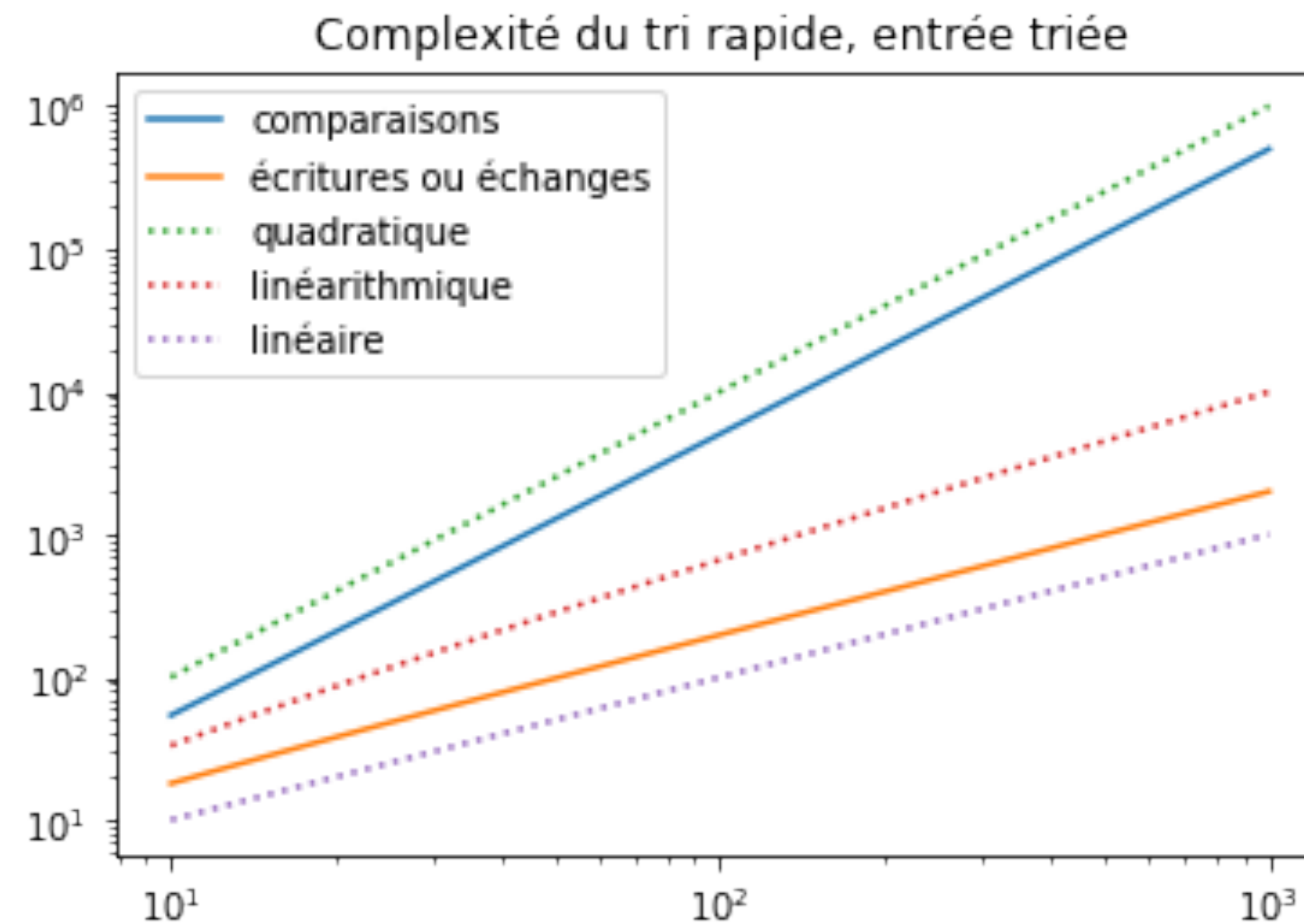
appels récurifs *partition*



Le pire cas est-il courant ?



- Cela dépend du choix du pivot
- Si pivot = dernier élément, une entrée triée est un pire cas



N	Comp.	Ecr.
10	54	18
19	189	36
37	702	72
71	2555	140
138	9590	274
268	36045	534
517	133902	1032
1000	500499	1998

Comment choisir le pivot?

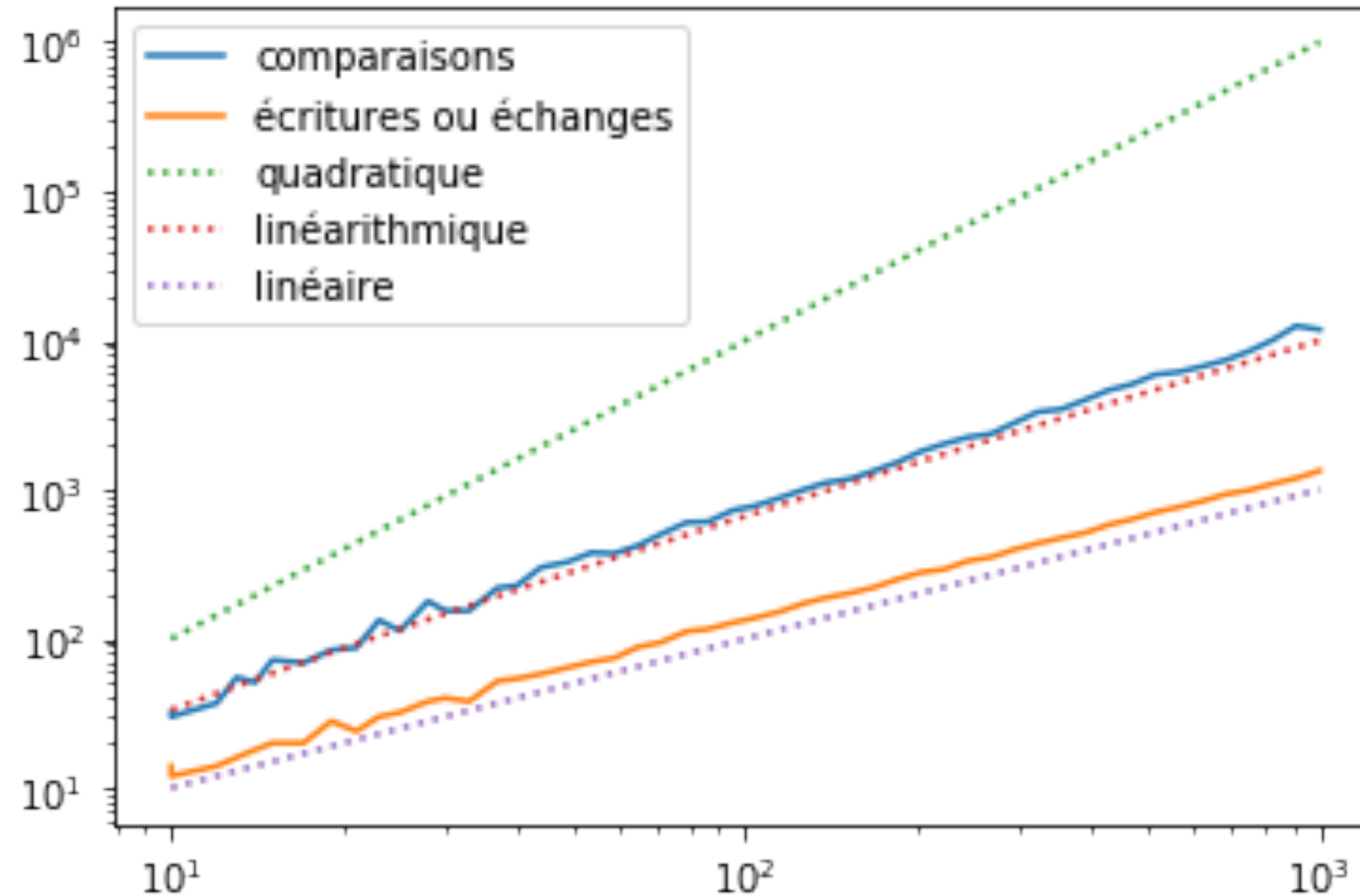


- $A(h_i)$
- $A(l_o)$
- $A(\text{mid} = l_o + (h_i - l_o)/2)$
- $A(\text{random})$?
- Médiane ($A(i)$ pour tous les i)
- Médiane ($A(h_i)$, $A(l_o)$, $A(\text{mid})$)
- Médiane (Médiane(...), Médiane(...), Médiane(...))
- Pire cas si A est trié
- Idem
- Idéal pour A trié, mais sinon ?
- Pas si mal...
- Trop cher, complexité $O(n)$
- Bonne idée

Pivot aléatoire



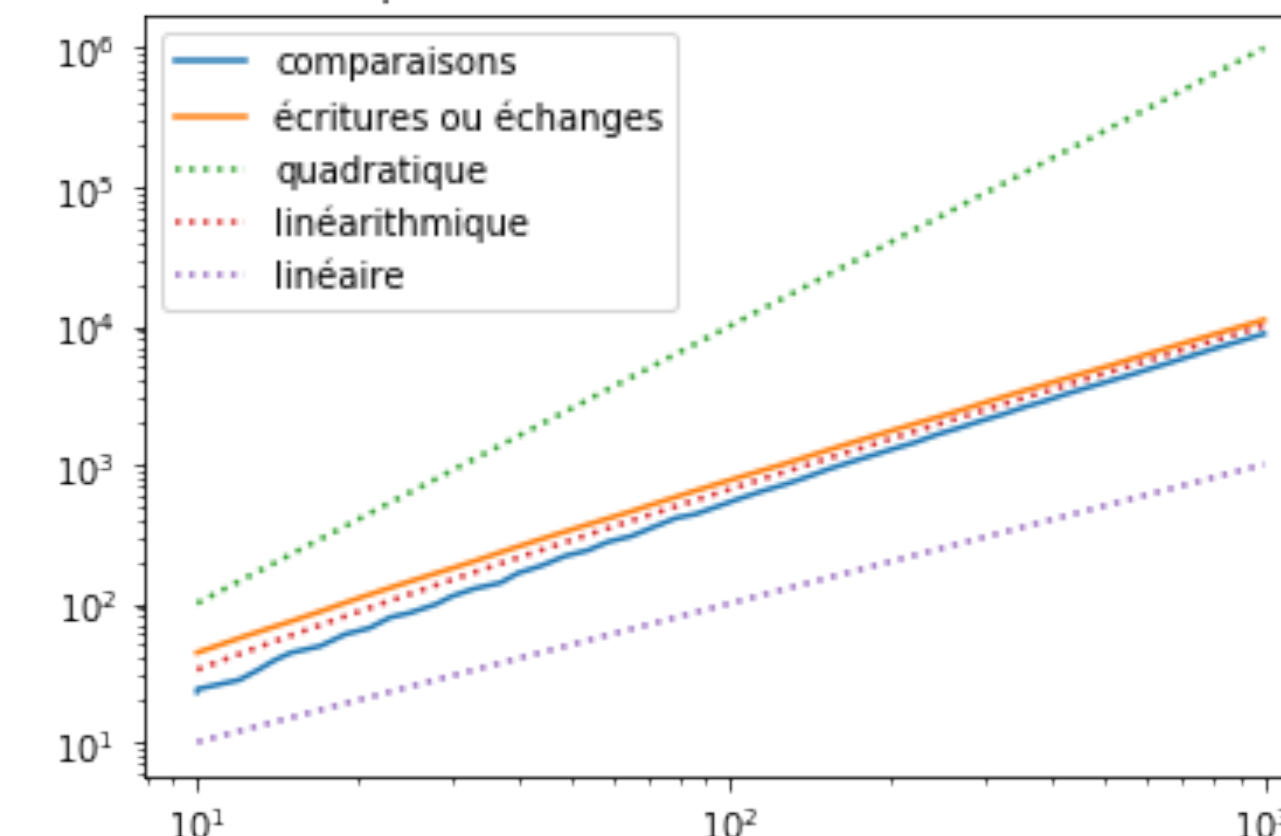
Complexité du tri rapide, entrée triée



N	Comp.	Ecr.
10	32	14
19	84	28
37	219	52
71	499	94
138	1110	188
268	2362	352
517	5895	704
1000	11811	1336

- Complexité linéarithmique $O(n \cdot \log(n))$
- Plus de comparaisons, moins d'écritures que le tri fusion

Complexité du tri fusion, tableau aléatoire



N	Comp.	Ecr.
10	23	44
19	60	101
37	140	232
71	348	511
138	802	1124
268	1830	2436
517	4013	5180
1000	8721	10976

Complexité moyenne, pivot aléatoire



Proposition - Le nombre moyen de comparaisons C_N pour trier rapidement un tableau de N éléments distincts est environ $2N \ln N$.

Preuve - C_N satisfait la récurrence $C_0=C_1=0$, et pour $N>1$,

$$C_N = (N+1) + \left(\frac{C_0 + C_{N-1}}{N} \right) + \left(\frac{C_1 + C_{N-2}}{N} \right) + \dots + \left(\frac{C_{N-1} + C_0}{N} \right)$$

partition
gauche
droite

↓
↓
↓

↑

probabilité

à partir de cette relation, on peut calculer $NC_N - (N-1)C_{N-1} = 2N + 2C_{N-1}$

en divisant par $N(N+1)$, on trouve $\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1}$

Tri rapide: cas moyen (suite)



- En appliquant cette équation de manière répétée, on trouve

$$\begin{aligned}\frac{C_N}{N+1} &= \frac{C_{N-1}}{N} + \frac{2}{N+1} \\ &= \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1} \\ &= \frac{C_{N-3}}{N-2} + \frac{2}{N-1} + \frac{2}{N} + \frac{2}{N+1} \\ &= \frac{2}{3} + \frac{2}{4} + \frac{2}{5} + \dots + \frac{2}{N+1}\end{aligned}$$

- Ce qui donne, en approximant la somme par une intégrale,

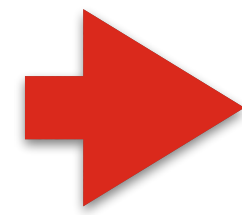
$$\begin{aligned}C_N &= 2(N+1)\left(\frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots + \frac{1}{N+1}\right) \\ &\approx 2(N+1) \int_3^{N+1} \frac{1}{x} dx \\ &\approx 2(N+1) \ln N \\ &\approx 1.39N \lg N\end{aligned}$$

Optimisation : tri hybride



- Pour des petits tableaux, le tri par insertion est plus rapide en évitant les appels récur­sifs. On a donc intérêt à arrêter la récur­sion avant d'atteindre des tableaux de 1 élément.

```
si  $hi \leq lo$ , alors  
    retourner  
fin si
```



```
si  $hi \leq lo + cutoff$ , alors  
    retourner TriInsertion( $A, lo, hi$ )  
fin si
```

- On peut aussi arrêter la récur­sion sans appeler TriInsertion, et puis appeler TriInsertion une seule fois à la fin.

Opt. : récursion terminale



- Avec un bon choix de pivot, le pire des cas est improbable, mais ...
 - dans le pire cas, profondeur de récursion de $n-1$
 - risque de saturer la pile d'appels
- Le deuxième appel récursif est en récursion terminale
 - On peut le remplacer par une itération
 - Pour éviter une récursion trop profonde dans le pire cas,
 - l'appel récursif restant traite la plus petite partition
 - l'itération traite la plus grande partition

```
fonction TriRapide(A, lo, hi)
    si lo < hi, alors
        p ← choisir l'élément pivot
        permuter A(hi) et A(p)
        i ← Partition(A, lo, hi)

        TriRapide(A, lo, i-1)
        TriRapide(A, i+1, hi)
    fin si
```

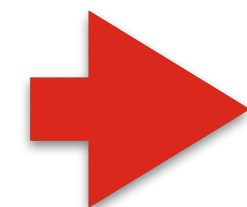
Semi-récurusif



```

fonction TriRapide(A,lo,hi)
  si lo < hi, alors
    p ← choisir l'élément pivot
    permuter A(hi) et A(p)
    i ← Partition(A,lo,hi)

    TriRapide(A,lo,i-1)
    TriRapide(A,i+1,hi)
  fin si
  
```



```

fonction TriRapide(A,lo,hi)
  tant que lo < hi
    p ← choisir l'élément pivot
    permuter A(hi) et A(p)
    i ← Partition(A,lo,hi)

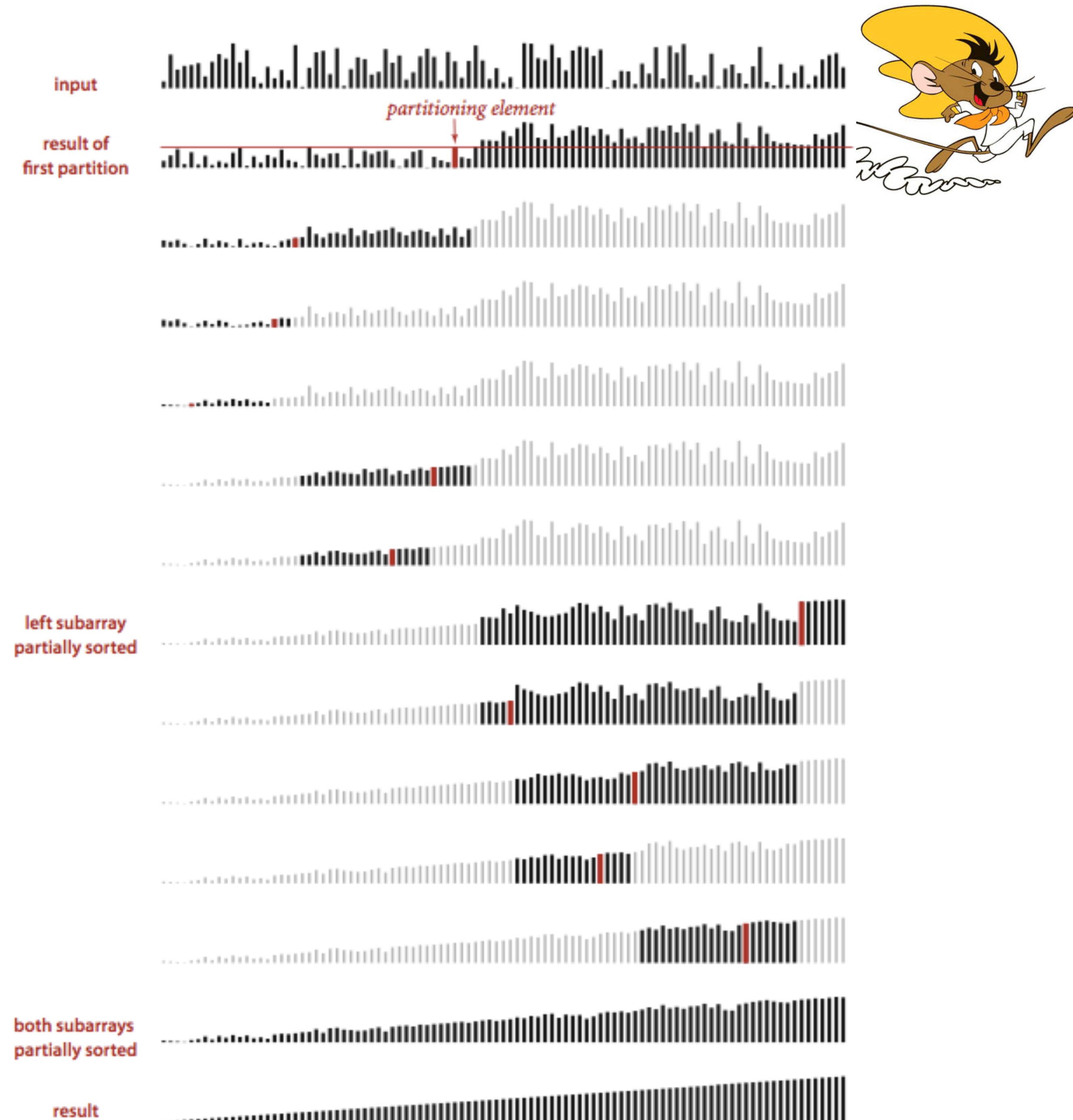
    si i-lo < hi-i, alors
      TriRapide(A,lo,i-1)
      lo ← i+1
    sinon,
      TriRapide(A,i+1,hi)
      hi ← i-1
    fin si
  fin tant que
  
```

*J'effectue le plus petit
coté récursivement...*

*...puis le plus grand en
recommençant la boucle.*

HE^{VD}IG Visualisation

- Pivot = médiane de 3 éléments
- Pas de partition si
$$hi - lo < cutoff$$
- Le résultat partiellement trié sera par insertion



Propriétés



- Complexité au pire : $O(n^2)$
 - Extrêmement rare avec un bon choix du pivot
- Complexité en moyenne : $O(n \log(n))$
- Temps de calcul en pratique : plus rapide que le tri fusion malgré 1,39 fois plus de comparaisons, car moins d'écritures
- Mémoire : s'effectue en place
- Stabilité : pas stable car la partition n'est pas stable
- Utilisation : Java sort (types primitifs), C qsort, C++ std::sort, ...

Éléments répétitifs



- Le tri rapide mal mis en oeuvre est sensible à la présence d'éléments répétés dans le tableau.
- Si on utilise $\leq$ plutôt que $<$ dans la partition, l'algorithme devient $O(n^2)$ en présence d'éléments répétés.
- Pourquoi ?

```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i  
    tant que A(i) < A(hi)
```

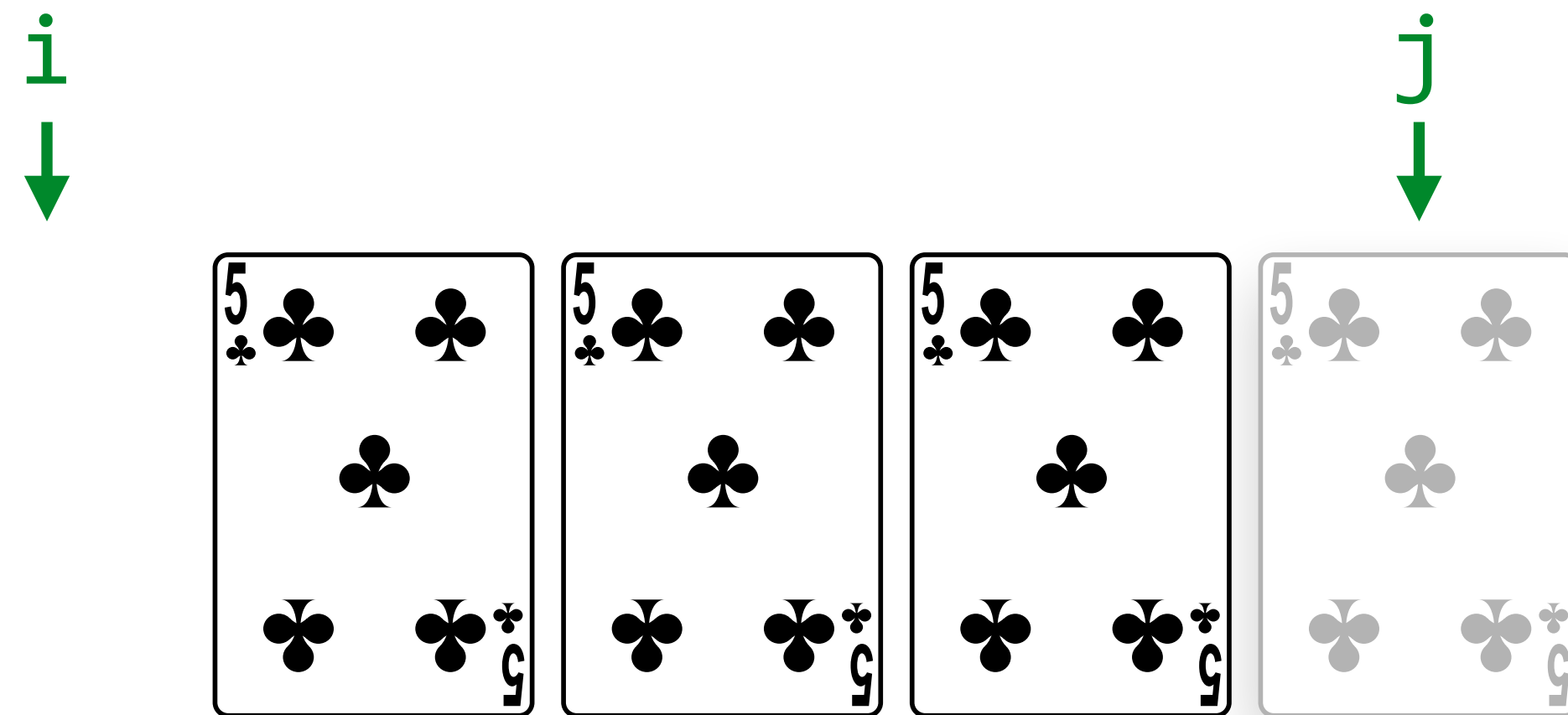
```
    répéter décrémenter j  
    tant que j > lo et A(hi) < A(j)
```

```
    si i ≥ j, alors sortir boucle  
  fin si
```

```
    permuter A(i) et A(j)  
  fin boucler
```

```
  permuter A(i) et A(hi)  
  retourner i
```

Partition en cas de doublons ?



Version correcte (avec <)

```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) < A(hi)$ 
```

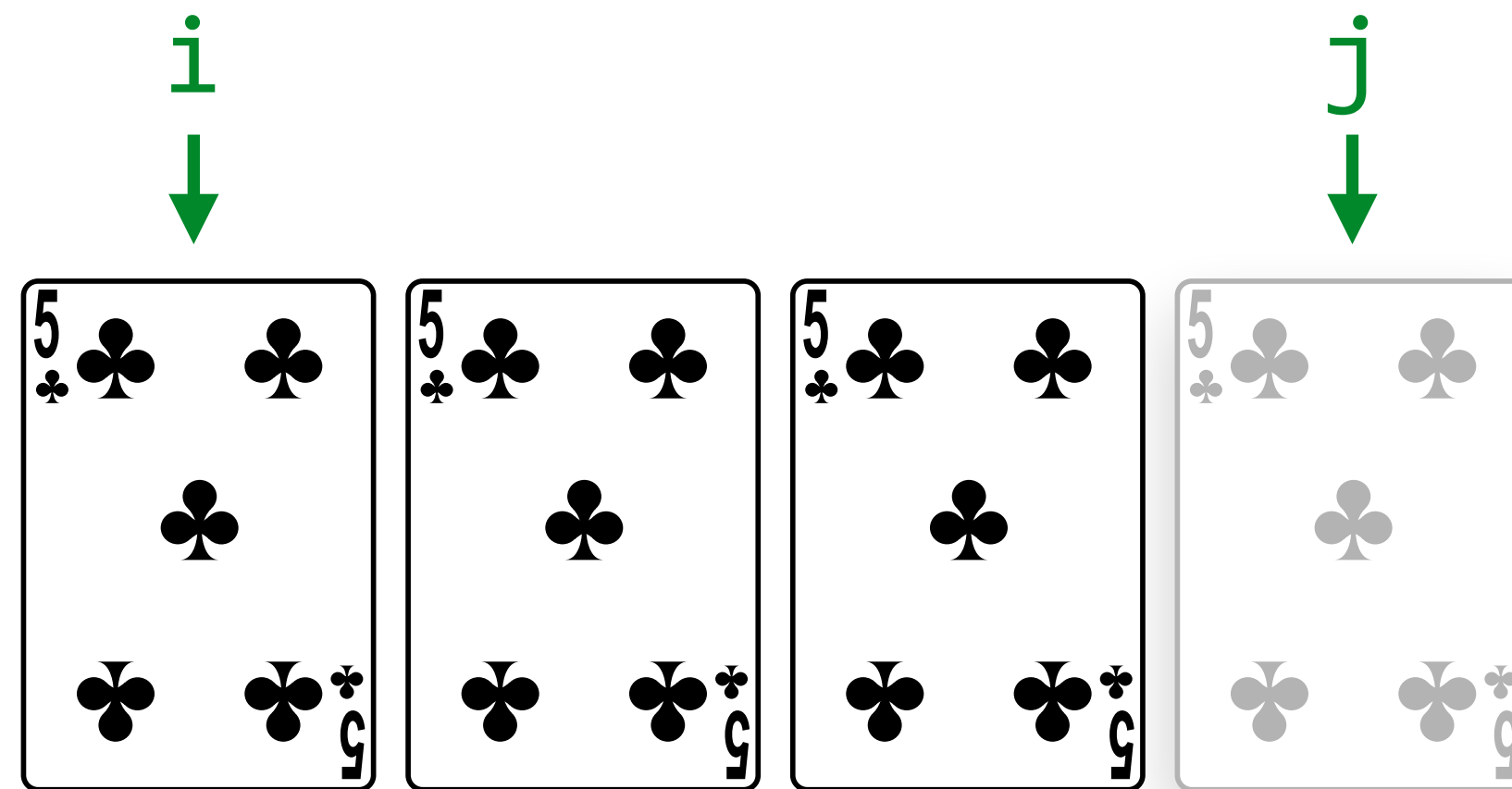
```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) < A(j)$ 
```

```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Partition en cas de doublons ?



Version correcte (avec <)

```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que A(i) < A(hi)
```

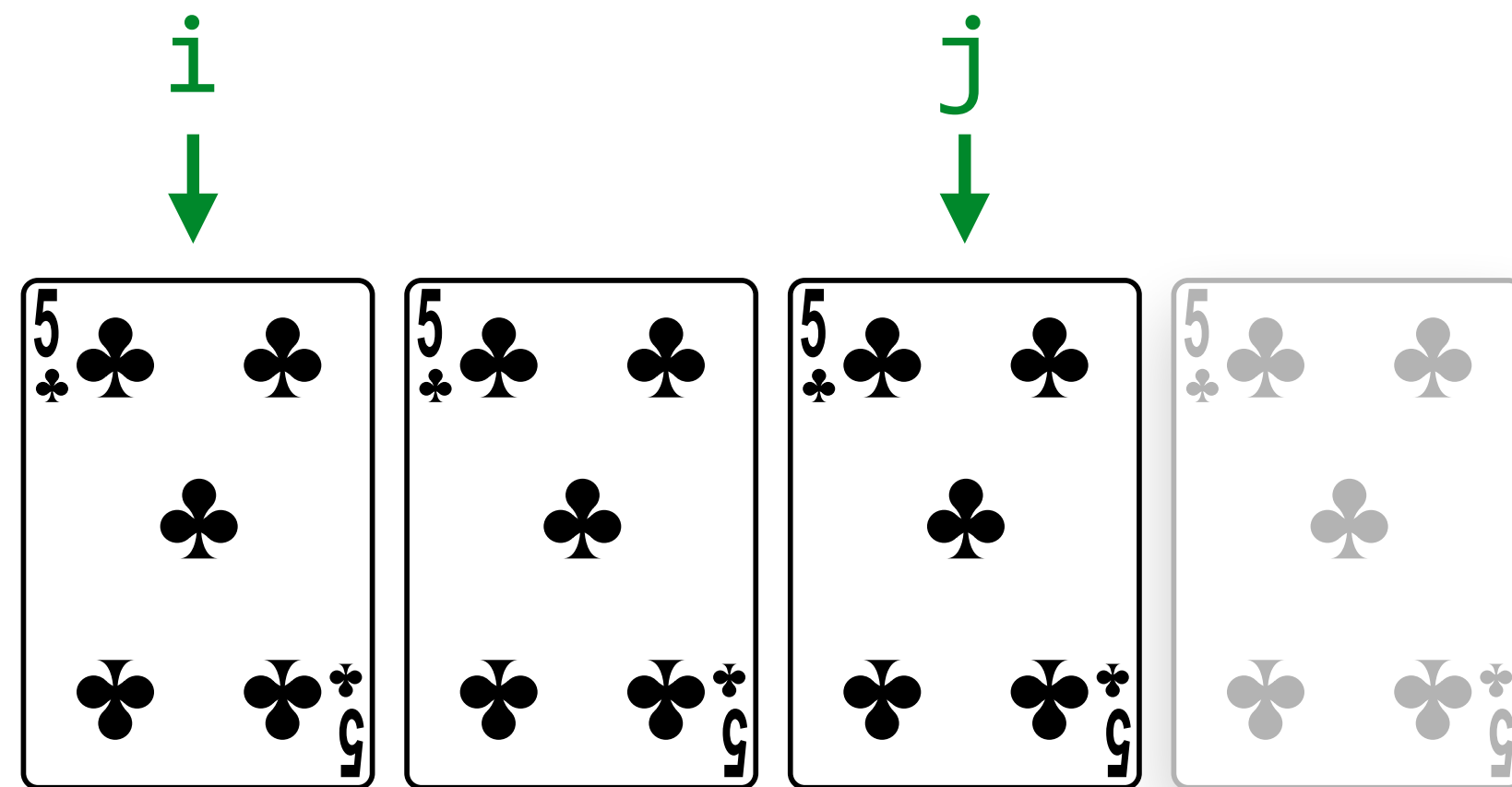
```
    répéter décrémenter j
    tant que j > lo et A(hi) < A(j)
```

```
  si i ≥ j, alors sortir boucle
  fin si
```

```
    permuter A(i) et A(j)
  fin boucler
```

```
  permuter A(i) et A(hi)
  retourner i
```

Partition en cas de doublons ?



```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) < A(hi)$ 
```

```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) < A(j)$ 
```

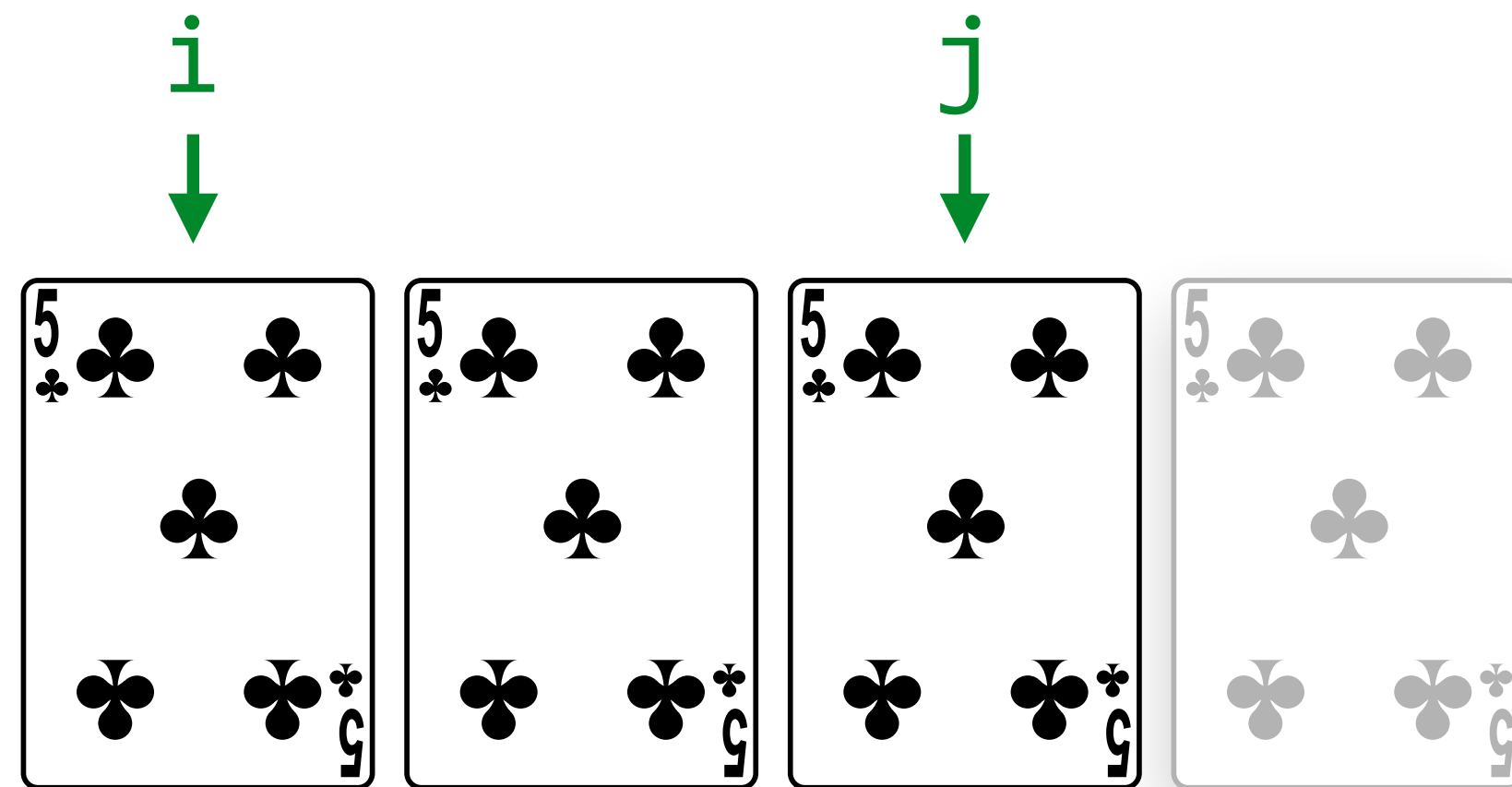
```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Version correcte (avec $<$)

Partition en cas de doublons ?



Version correcte (avec <)

```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) < A(hi)$ 
```

```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) < A(j)$ 
```

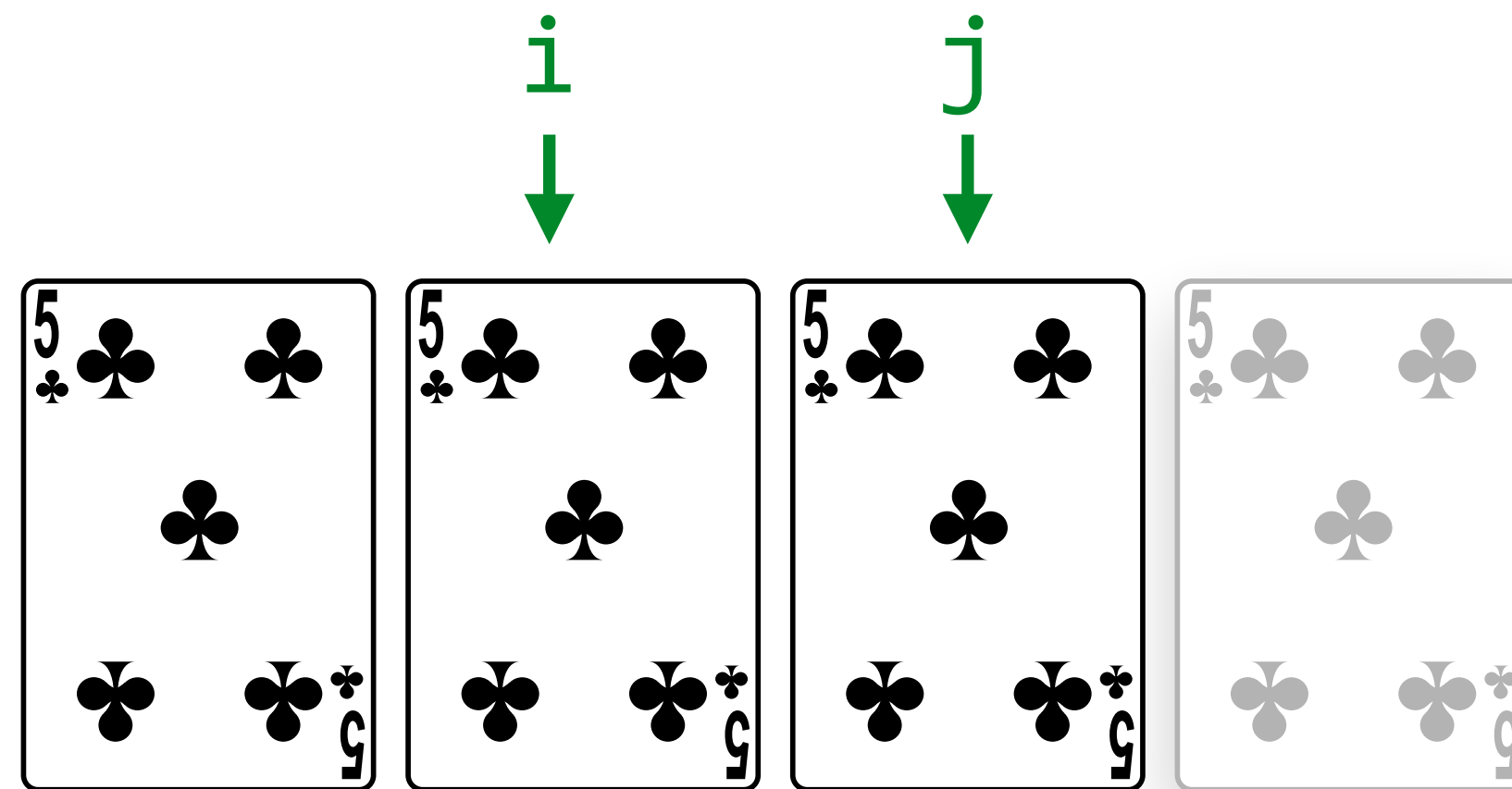
```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
```

```
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Partition en cas de doublons ?



Version correcte (avec <)

```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) < A(hi)$ 
```

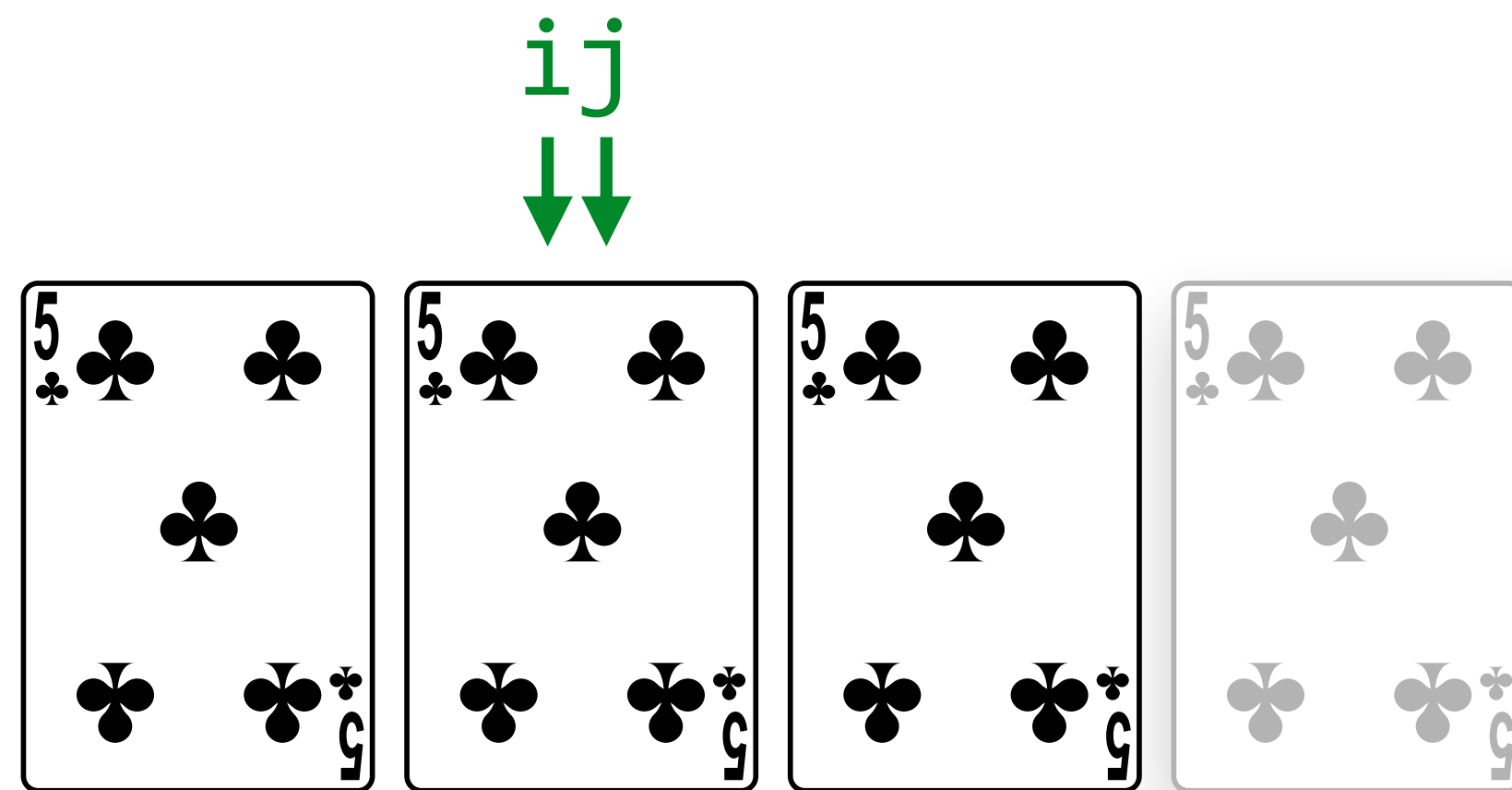
```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) < A(j)$ 
```

```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Partition en cas de doublons ?



```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) < A(hi)$ 
```

```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) < A(j)$ 
```

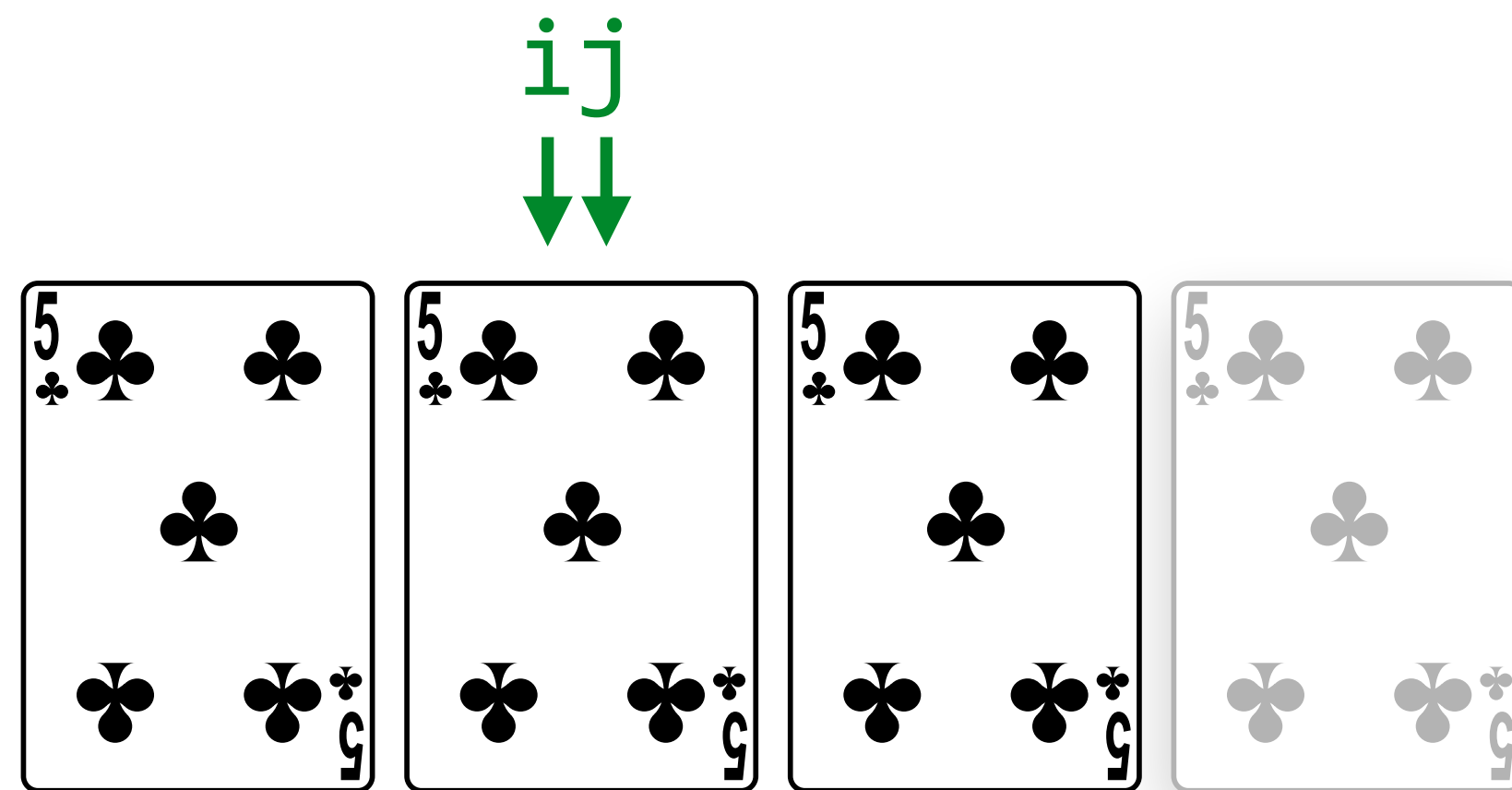
```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Version correcte (avec <)

Partition en cas de doublons ?



```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) < A(hi)$ 
```

```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) < A(j)$ 
```

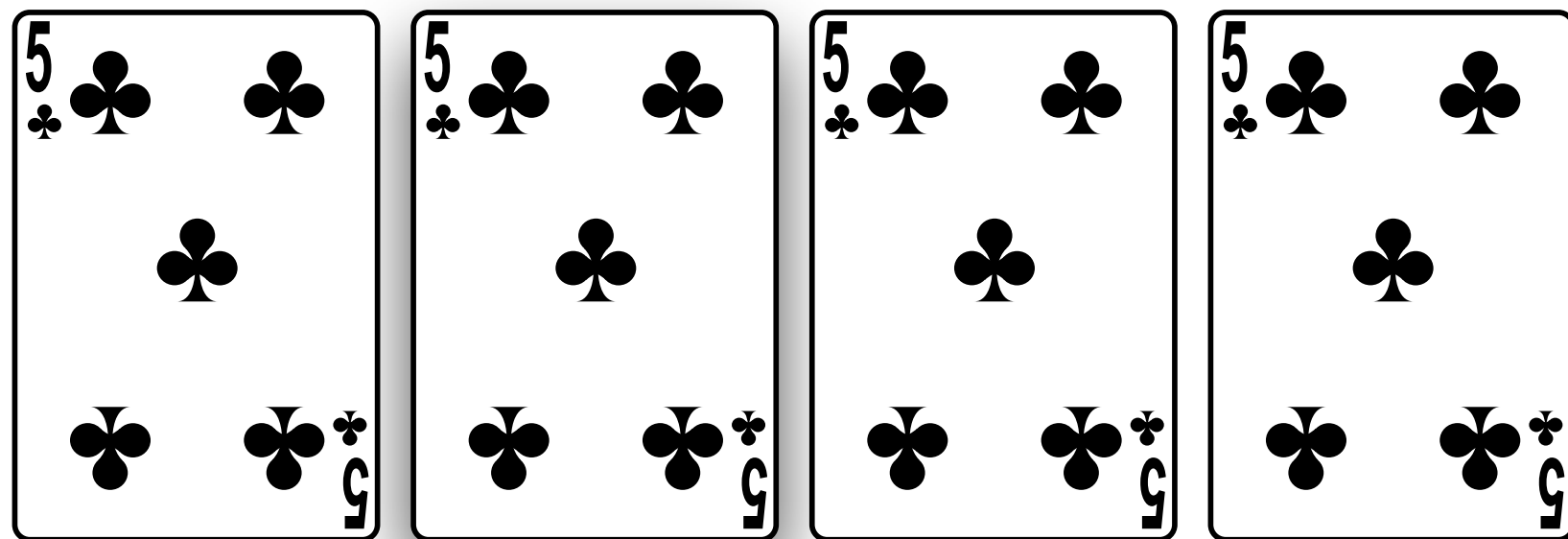
```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Version correcte (avec <)

Partition en cas de doublons ?



Version correcte (avec <)

```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que A(i) < A(hi)
```

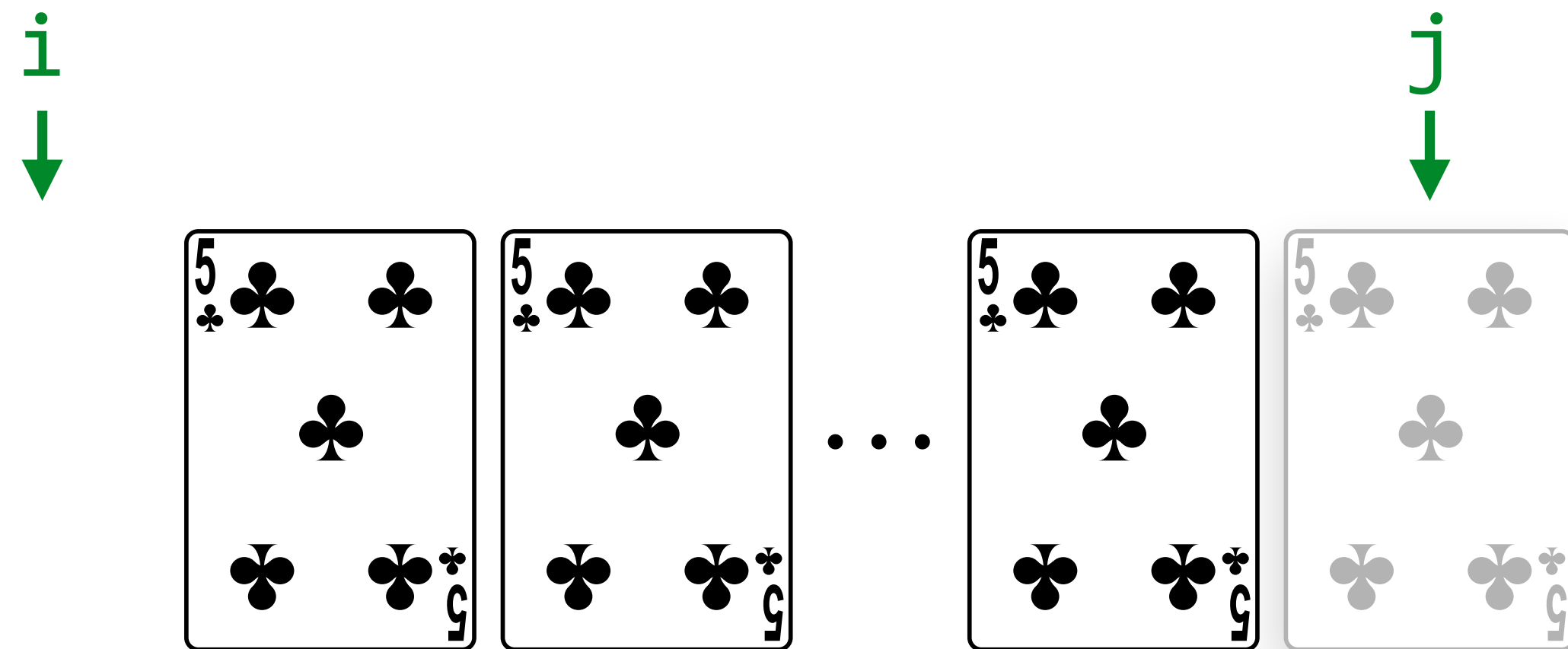
```
    répéter décrémenter j
    tant que j > lo et A(hi) < A(j)
```

```
  si i ≥ j, alors sortir boucle
  fin si
```

```
    permuter A(i) et A(j)
  fin boucler
```

```
  permuter A(i) et A(hi)
  retourner i
```

Partition en cas de doublons ?



Version incorrecte (avec $\leq$)

```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) \leq A(hi)$ 
```

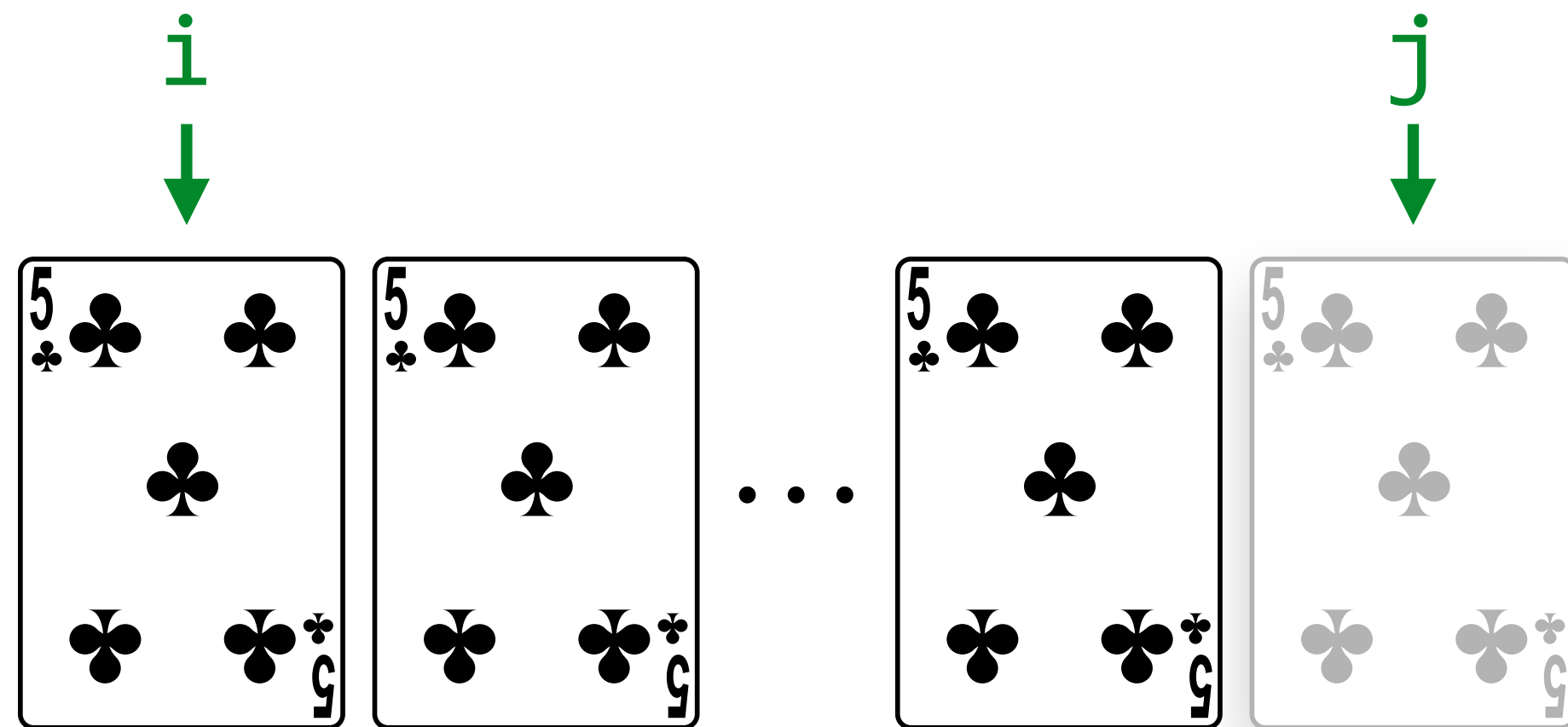
```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) \leq A(j)$ 
```

```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Partition en cas de doublons ?



Version incorrecte (avec $\leq$)

```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que A(i) ≤ A(hi)
```

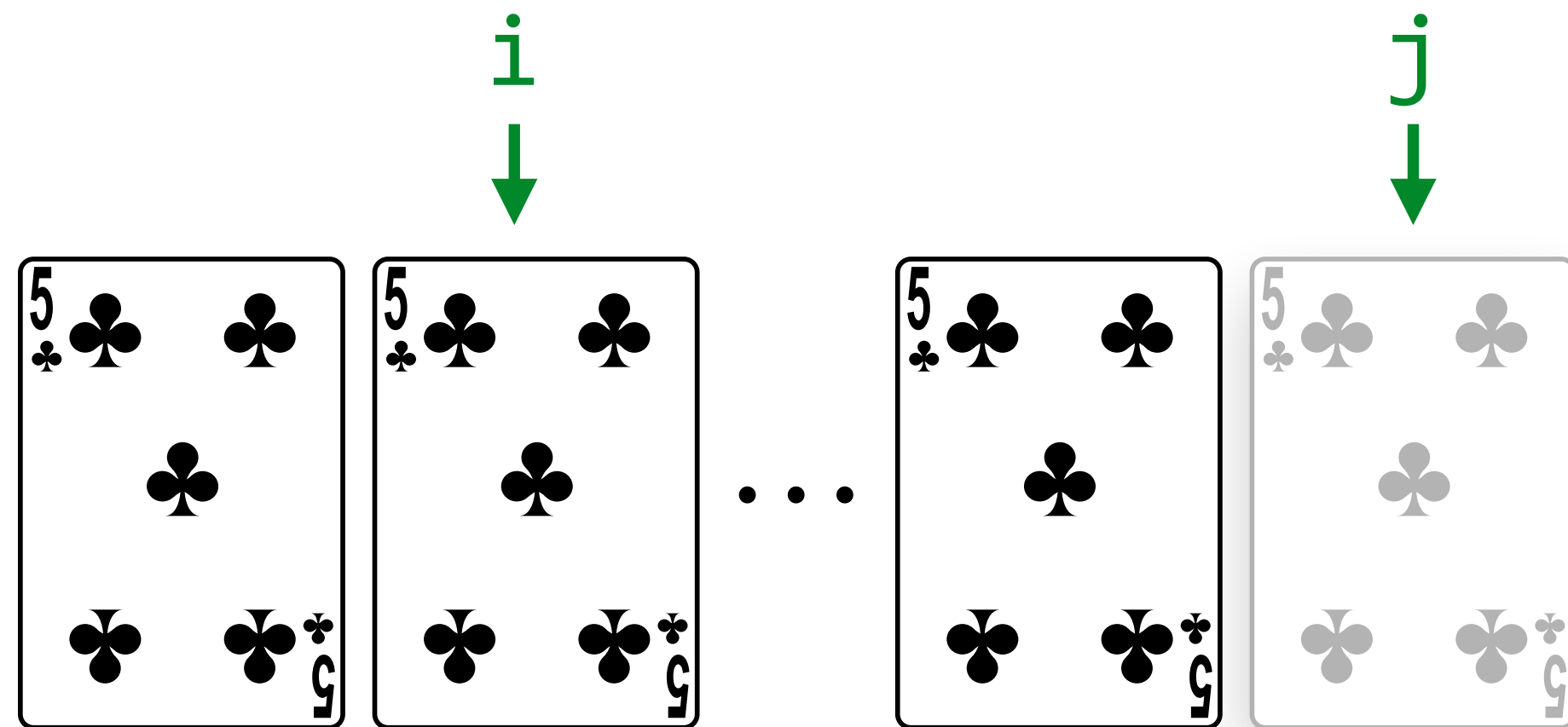
```
    répéter décrémenter j
    tant que j > lo et A(hi) ≤ A(j)
```

```
  si i ≥ j, alors sortir boucle
  fin si
```

```
    permuter A(i) et A(j)
  fin boucler
```

```
  permuter A(i) et A(hi)
  retourner i
```

Partition en cas de doublons ?



Version incorrecte (avec $\leq$)

```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que A(i) ≤ A(hi)
```

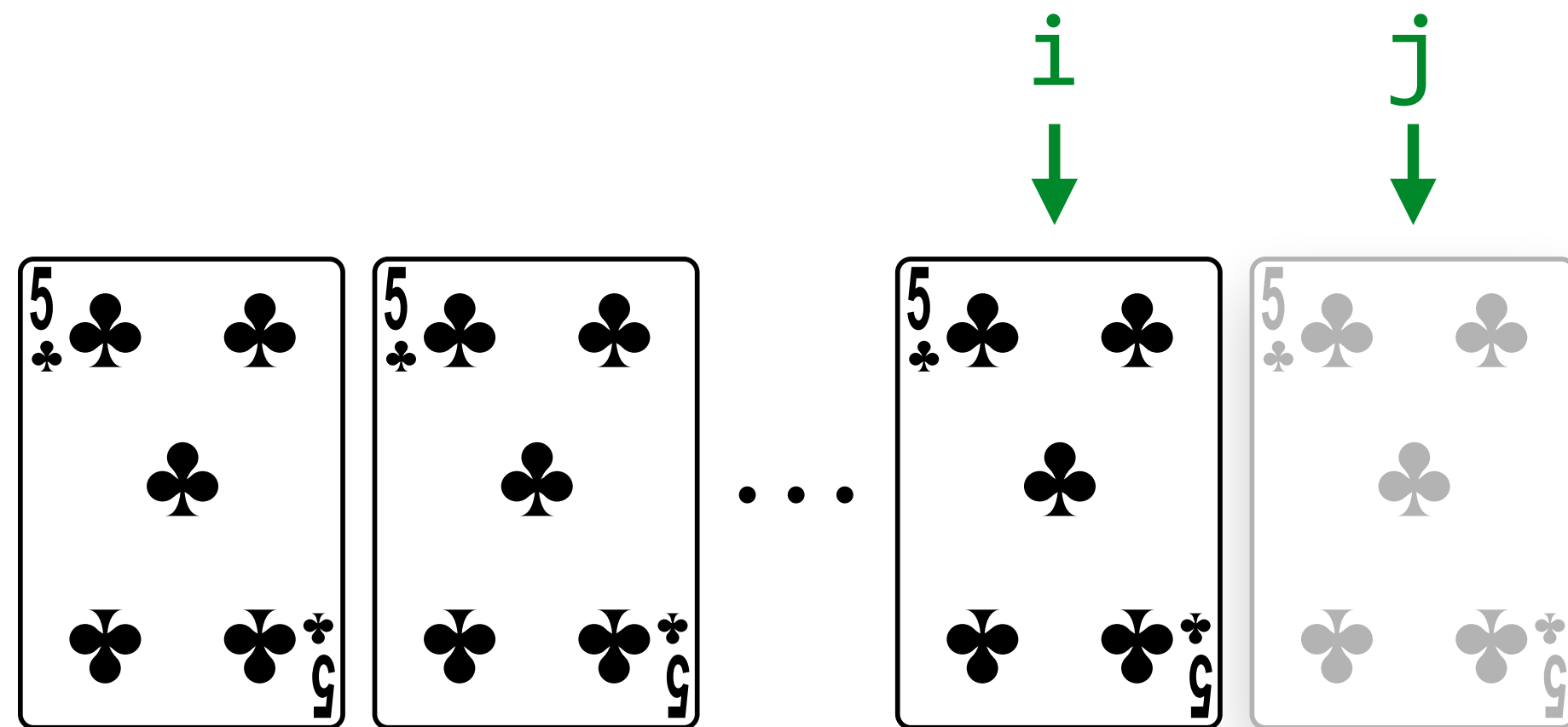
```
    répéter décrémenter j
    tant que j > lo et A(hi) ≤ A(j)
```

```
  si i ≥ j, alors sortir boucle
  fin si
```

```
    permuter A(i) et A(j)
  fin boucler
```

```
  permuter A(i) et A(hi)
  retourner i
```

Partition en cas de doublons ?



```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) \leq A(hi)$ 
```

```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) \leq A(j)$ 
```

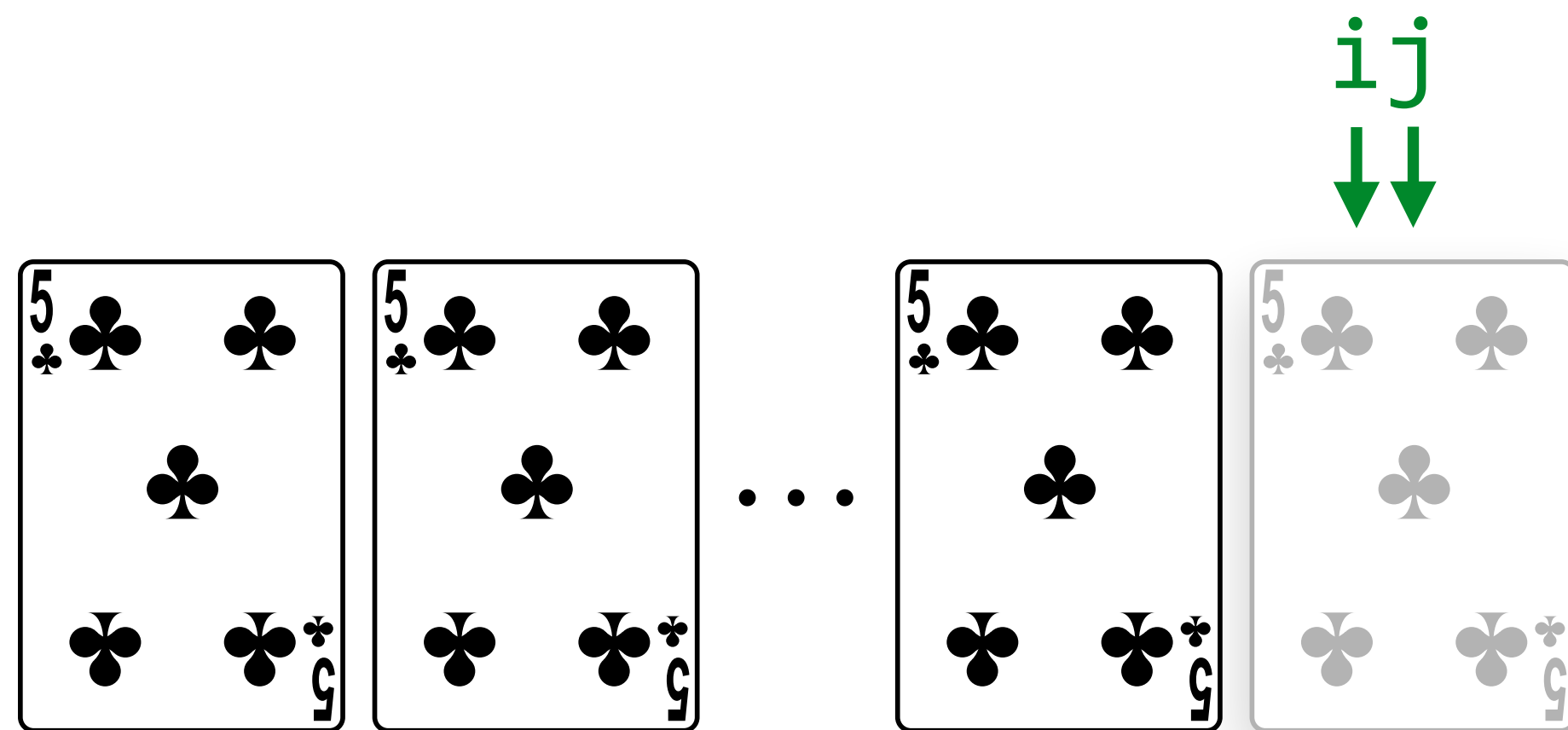
```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Version incorrecte (avec $\leq$)

Partition en cas de doublons ?



```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) \leq A(hi)$ 
```

```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) \leq A(j)$ 
```

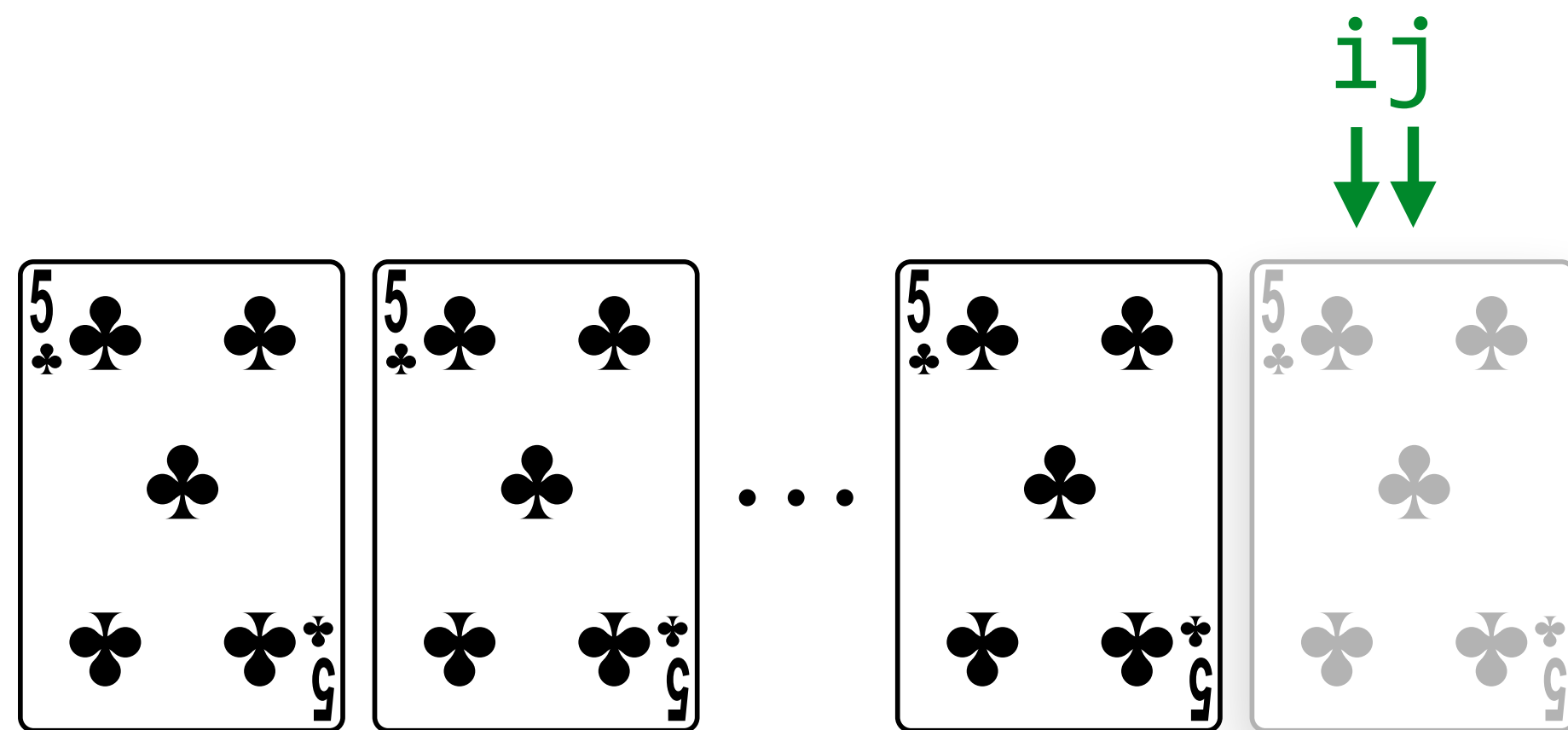
```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Version incorrecte (avec $\leq$)

Partition en cas de doublons ?



```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) \leq A(hi)$ 
```

```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) \leq A(j)$ 
```

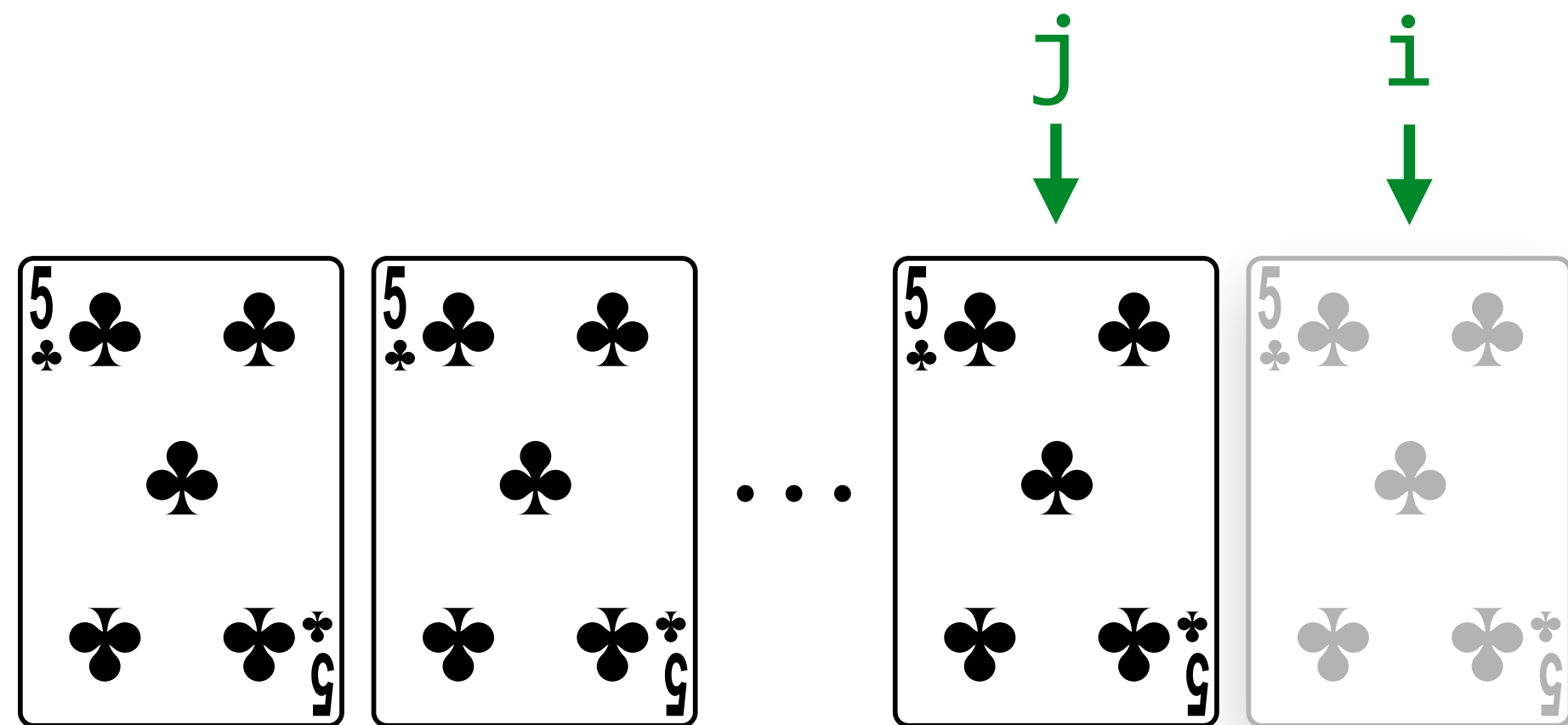
```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Version incorrecte (avec $\leq$)

Partition en cas de doublons ?



```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
```

```
    tant que  $A(i) \leq A(hi)$ 
```

```
    répéter décrémenter j
```

```
    tant que  $j > lo$  et  $A(hi) \leq A(j)$ 
```

```
  si  $i \geq j$ , alors sortir boucle
```

```
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
```

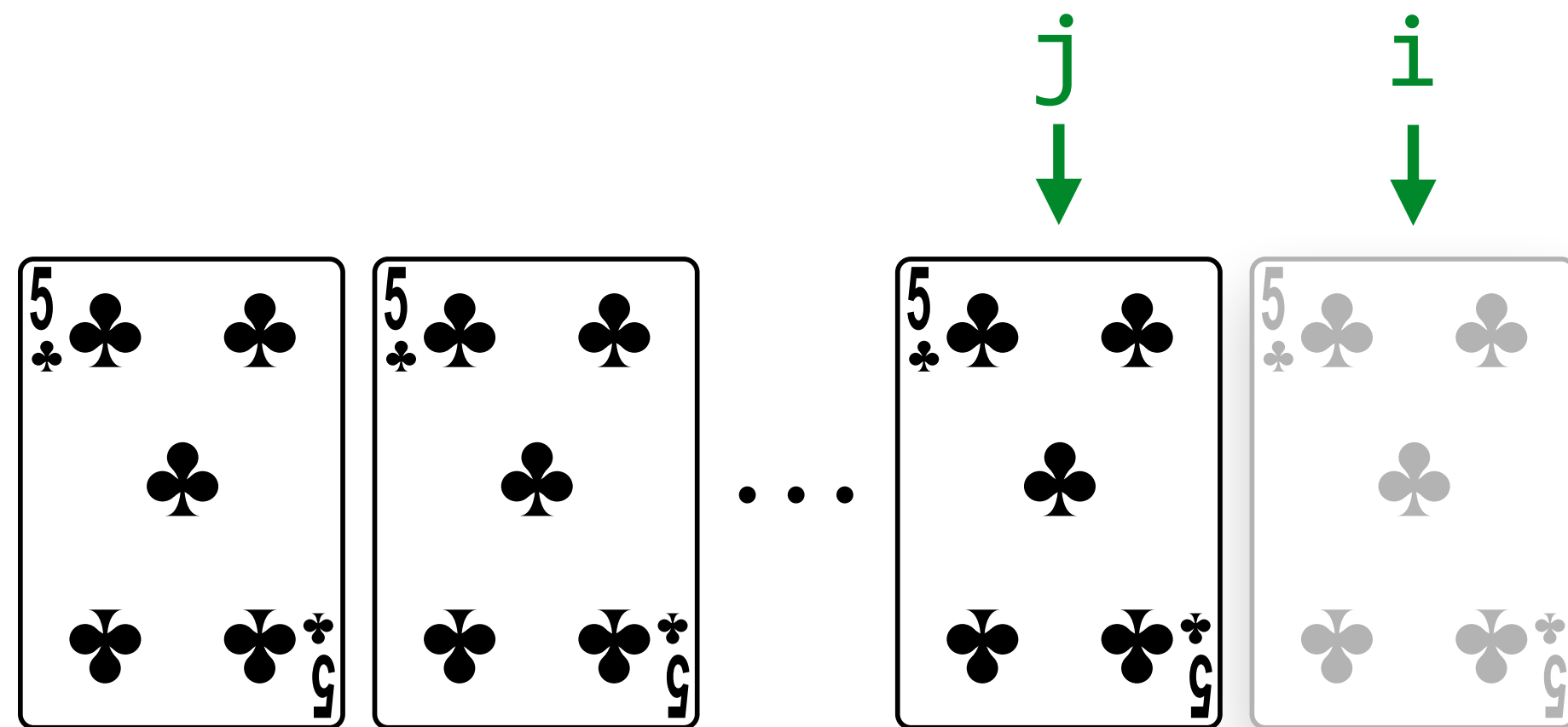
```
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
```

```
  retourner i
```

Version incorrecte (avec $\leq$)

Partition en cas de doublons ?



```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) \leq A(hi)$ 
```

```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) \leq A(j)$ 
```

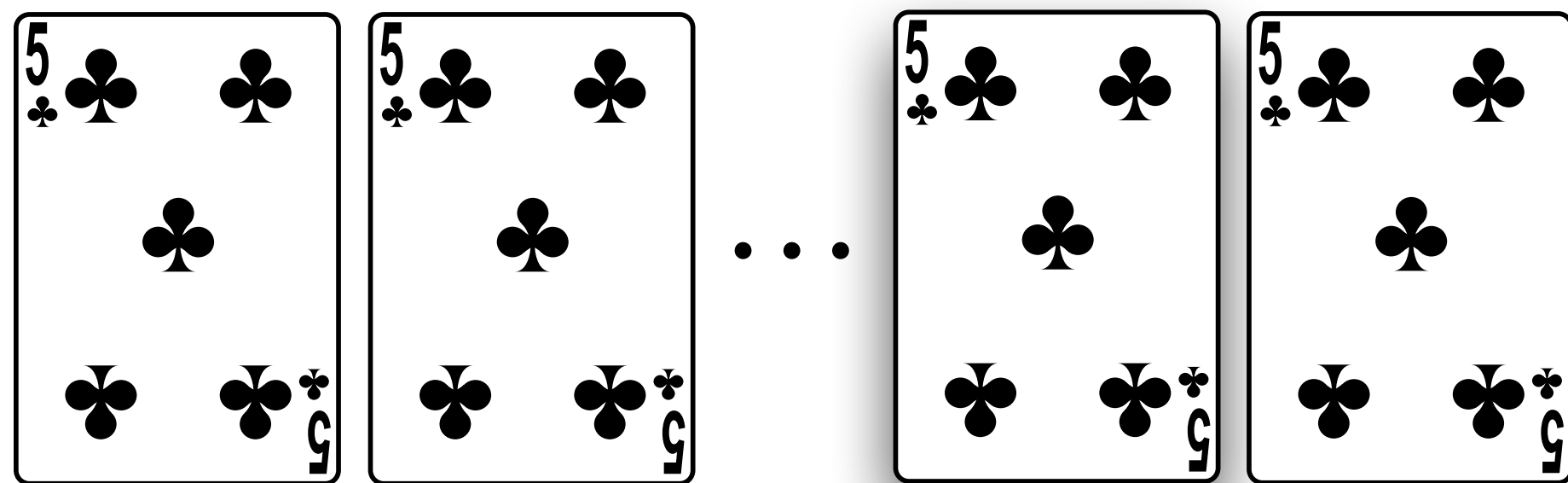
```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Version incorrecte (avec $\leq$)

Partition en cas de doublons ?



```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) \leq A(hi)$ 
```

```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) \leq A(j)$ 
```

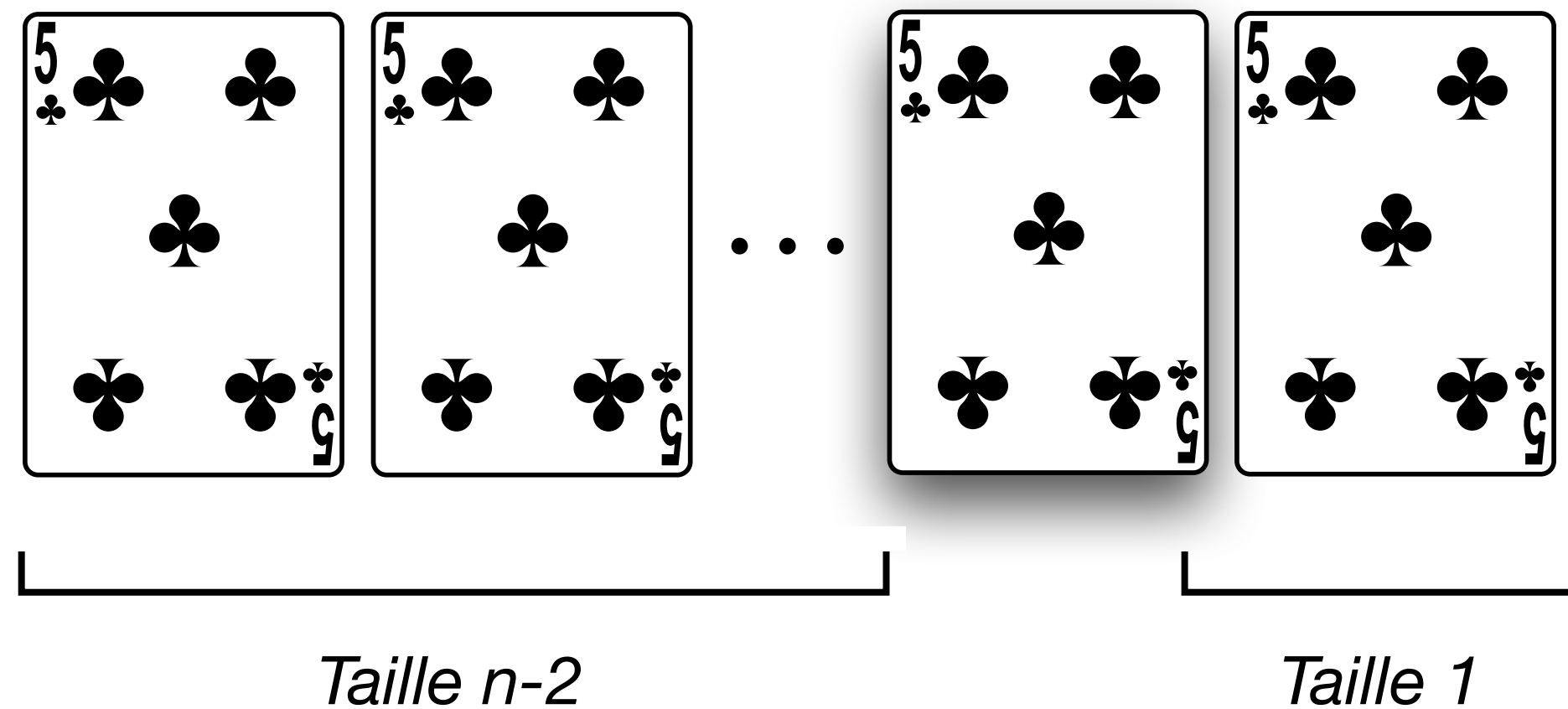
```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```

Version incorrecte (avec $\leq$)

Partition en cas de doublons ?



Crée la partition du pire cas.

Version incorrecte (avec $\leq$)

```
fonction Partition(A, lo, hi)
```

```
  i ← lo-1, j ← hi
```

```
  boucler
```

```
    répéter incrémenter i
    tant que  $A(i) \leq A(hi)$ 
```

```
    répéter décrémenter j
    tant que  $j > lo$  et  $A(hi) \leq A(j)$ 
```

```
  si  $i \geq j$ , alors sortir boucle
  fin si
```

```
    permuter  $A(i)$  et  $A(j)$ 
  fin boucler
```

```
  permuter  $A(i)$  et  $A(hi)$ 
  retourner i
```



- Si la partition utilise $\leq$ plutôt que $<$, la partition crée une plus grande partition gauche que droite.
- Dans le pire des cas, l'algorithme devient $O(n^2)$ sur une liste dont toutes les valeurs sont égales.
- La plupart des mises en oeuvre de la fonction `qsort()` de C avaient ce défaut jusqu'à sa découverte en 1991.
- On peut le résoudre plus proprement en partitionnant en 3

répéter incrémenter i
tant que $A(i) < A(hi)$

répéter décrémenter j
tant que $j > lo$ et $A(hi) < A(j)$

