Trier des cartes

Christophe Viroulaud

Première NSI

Trier un jeu de cartes est une opération qui trouve des applications en informatique.

Existe-t-il plusieurs méthodes pour trier des données ?

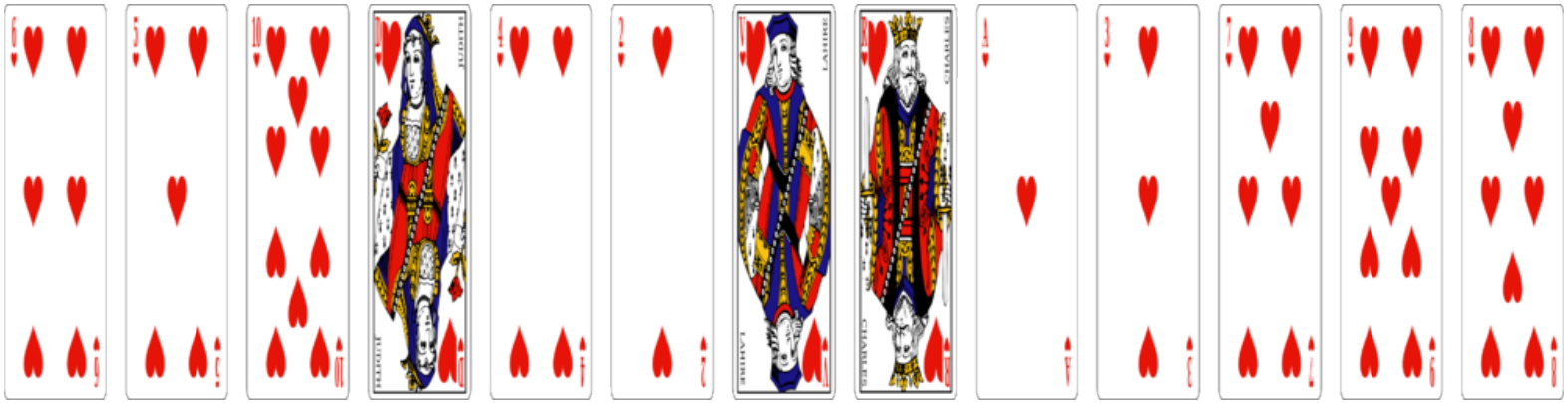


FIGURE – Cartes mélangées

Activité 1 :

1. Prendre le paquet de cartes mélangées et les étaler sur la table.
2. Trier les cartes.
3. Formaliser la méthode utilisée sous forme d'un algorithme.

Différentes méthodes

- Tri par sélection en place
- Tri par sélection dans un nouveau tableau
- Tri par insertion en place
- Tri par insertion dans un nouveau tableau

Tri par sélection en place

```
1 Pour chaque carte du tas
2     Trouver la plus petite carte dans la partie
          non triée.
3     Échanger cette carte avec la première de la
          partie non triée.
```

Code 1 – Tri par sélection (en place)

Retour menu

Tri par sélection en place

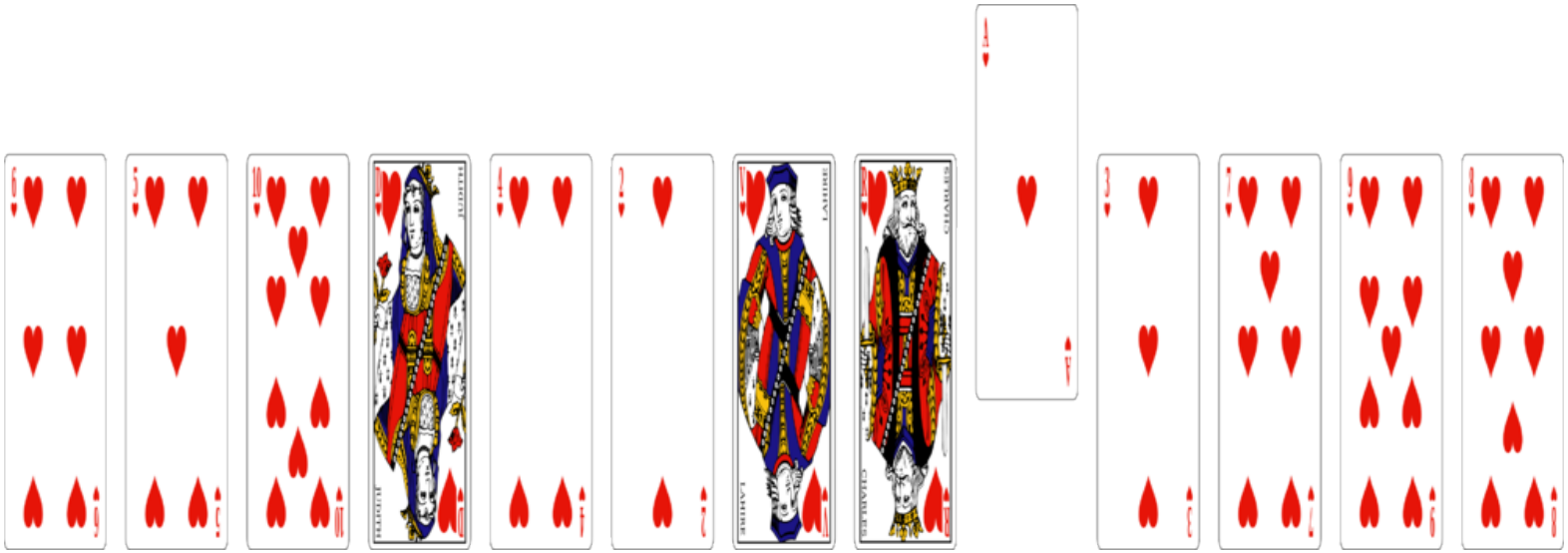


FIGURE – Sélectionne la plus petite du tas non trié

Retour menu

Tri par sélection en place

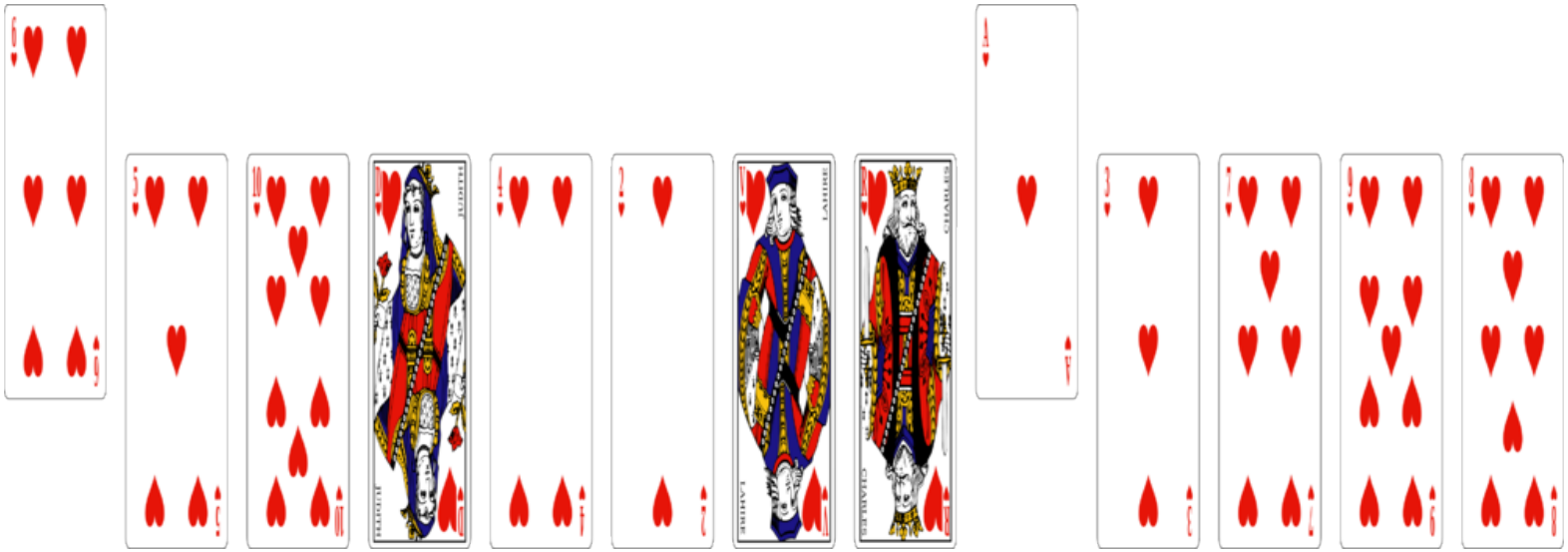


FIGURE – Échange avec la première carte du tas non trié

Retour menu

Tri par sélection en place

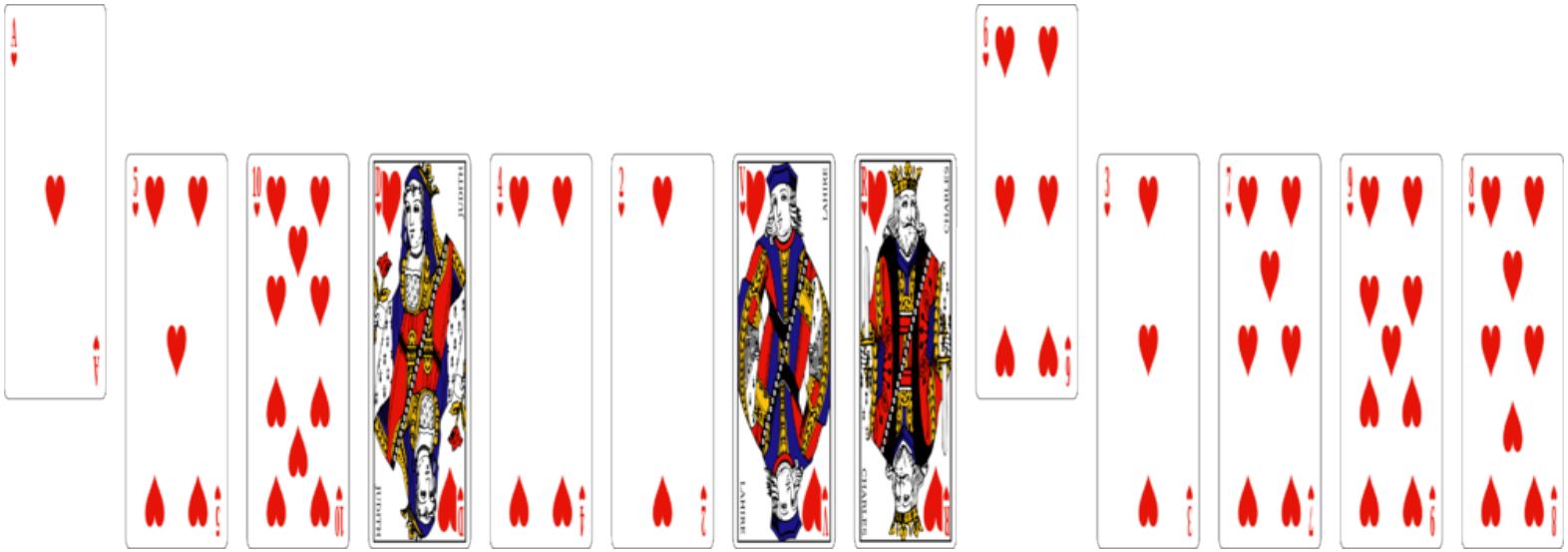


FIGURE – Échange avec la première carte du tas non trié

Retour menu

Tri par sélection en place

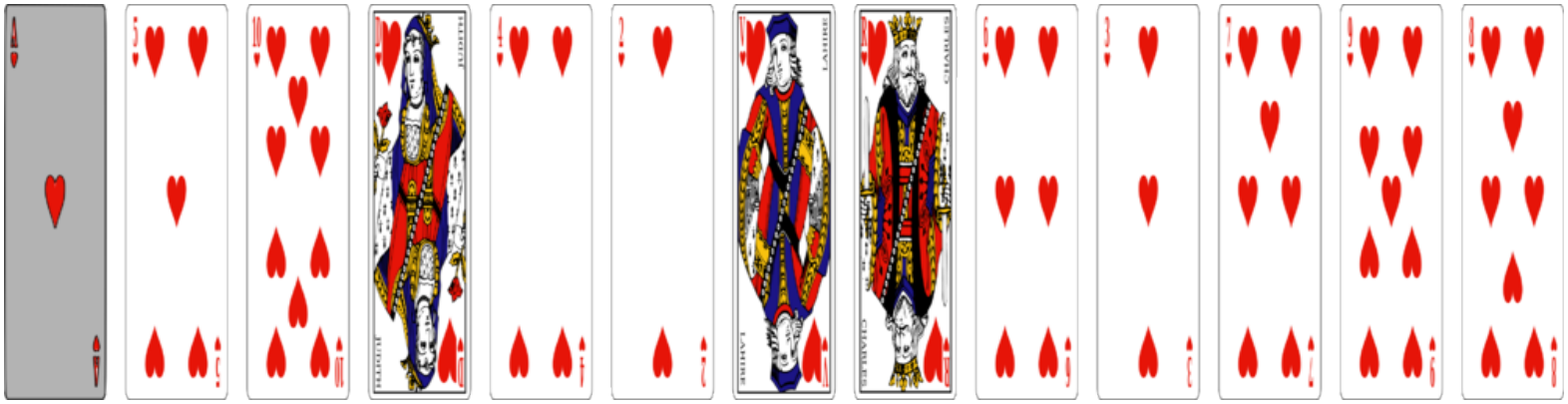


FIGURE – La carte est à sa place

Retour menu

Tri par sélection en place

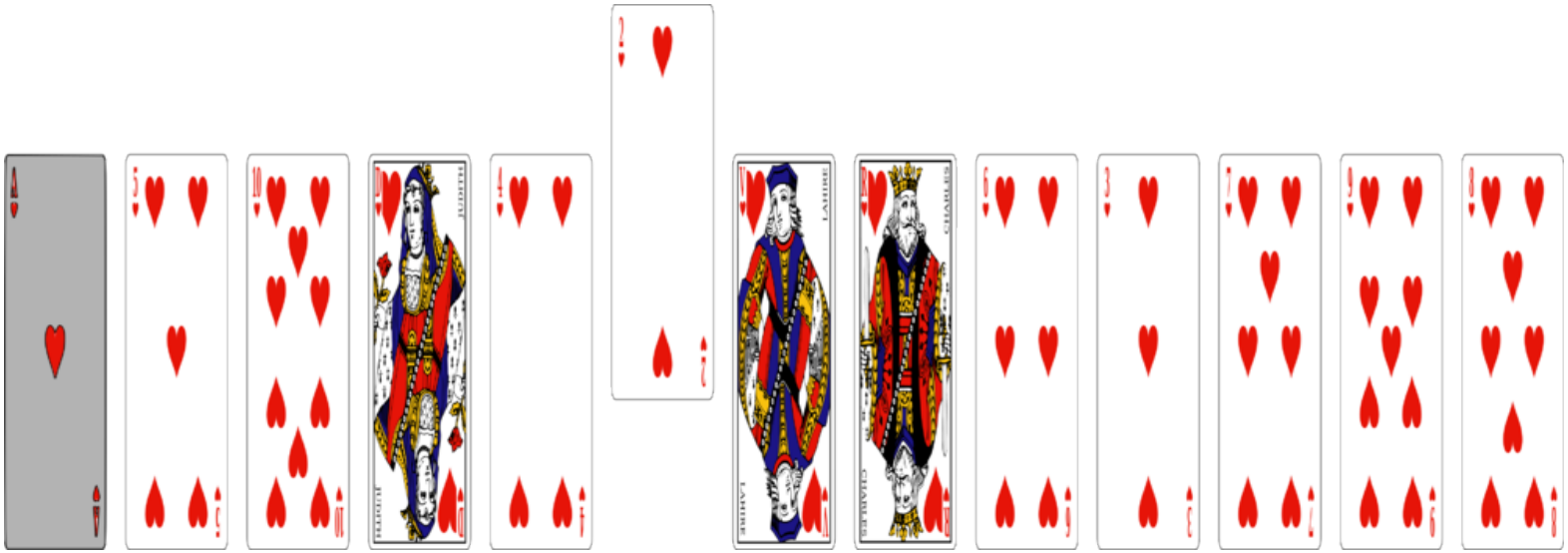


FIGURE – Sélectionne la plus petite du tas non trié

Retour menu

Tri par sélection en place

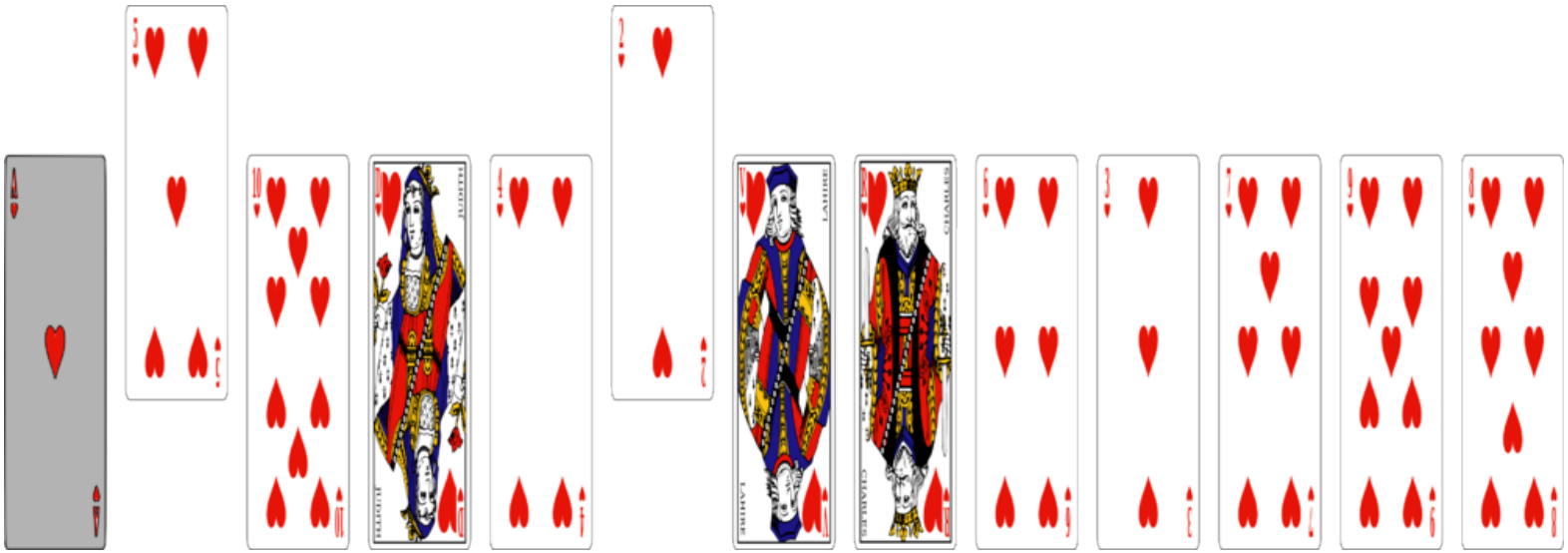


FIGURE – Échange avec la première carte du tas non trié

Retour menu

Tri par sélection en place

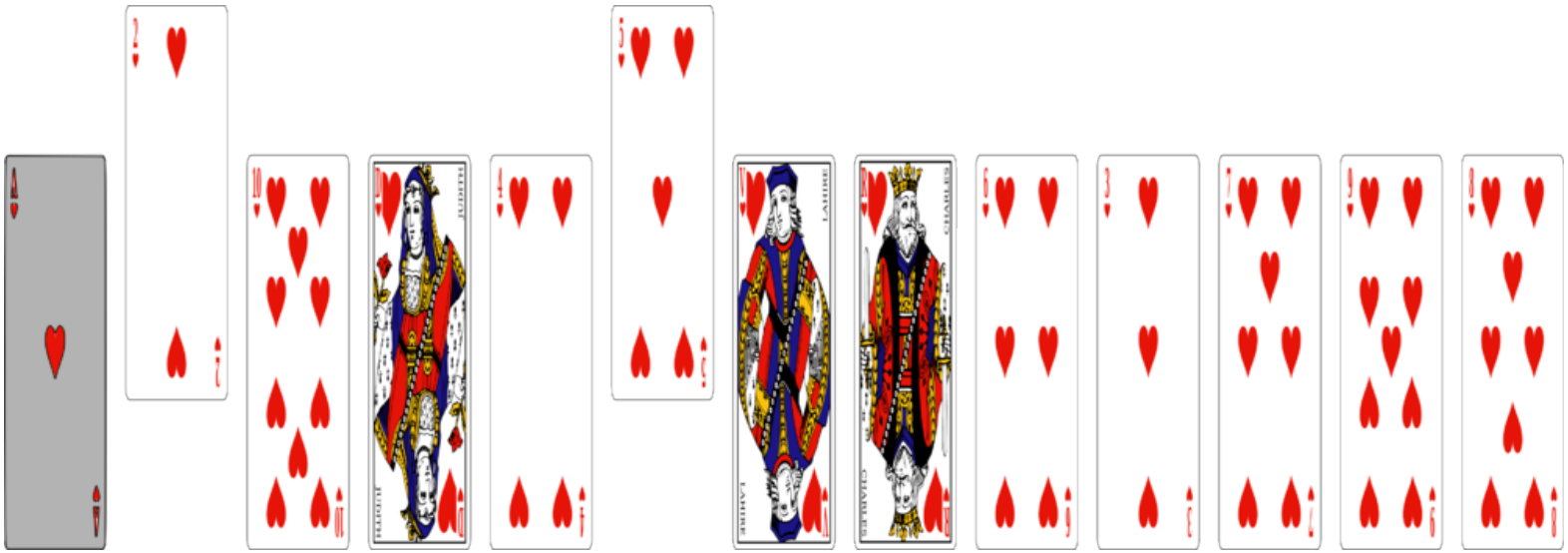


FIGURE – Échange avec la première carte du tas non trié

Retour menu

Tri par sélection en place

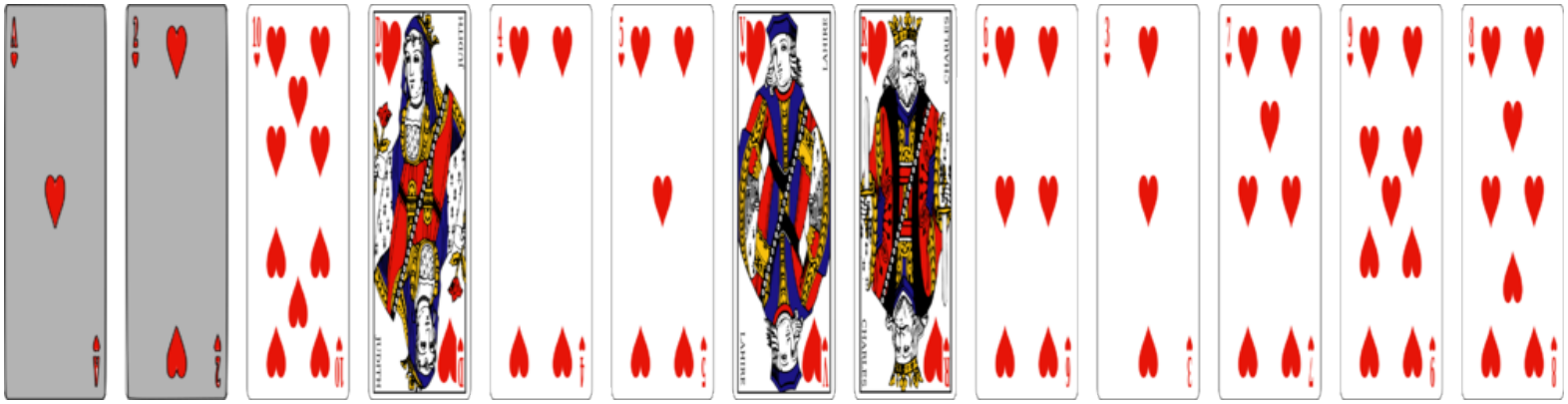


FIGURE – La carte est à sa place

Retour menu

Tri par sélection - nouveau tableau

```
1 Pour chaque carte du tas
2     Trouver la plus petite carte du tableau non trié.
3     La placer à la fin du tableau trié.
```

Code 2 – Tri par sélection dans un nouveau tableau

Retour menu

Tri par sélection - nouveau tableau

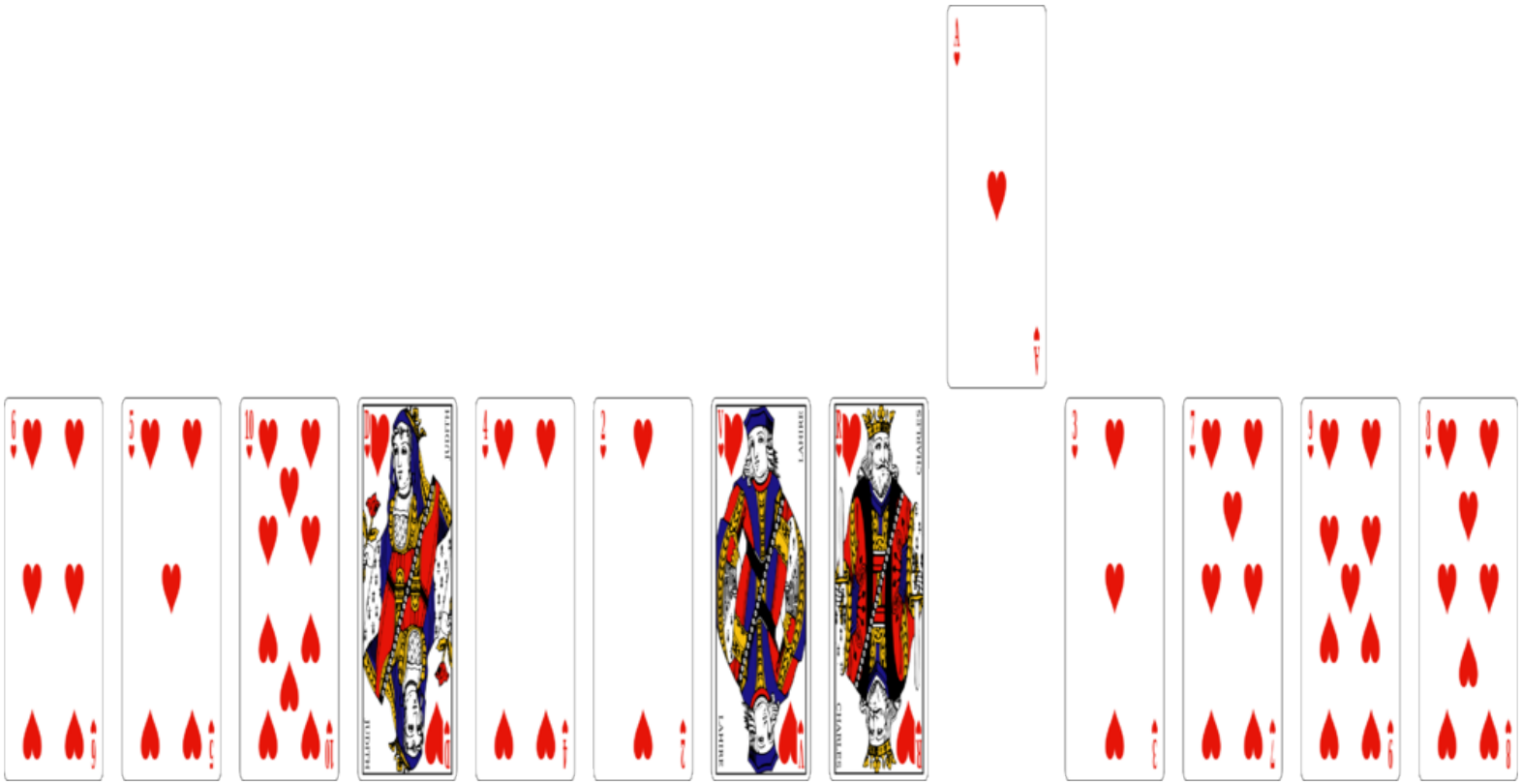


FIGURE – Trouve la plus petite du tas non trié

Retour menu

Tri par sélection - nouveau tableau

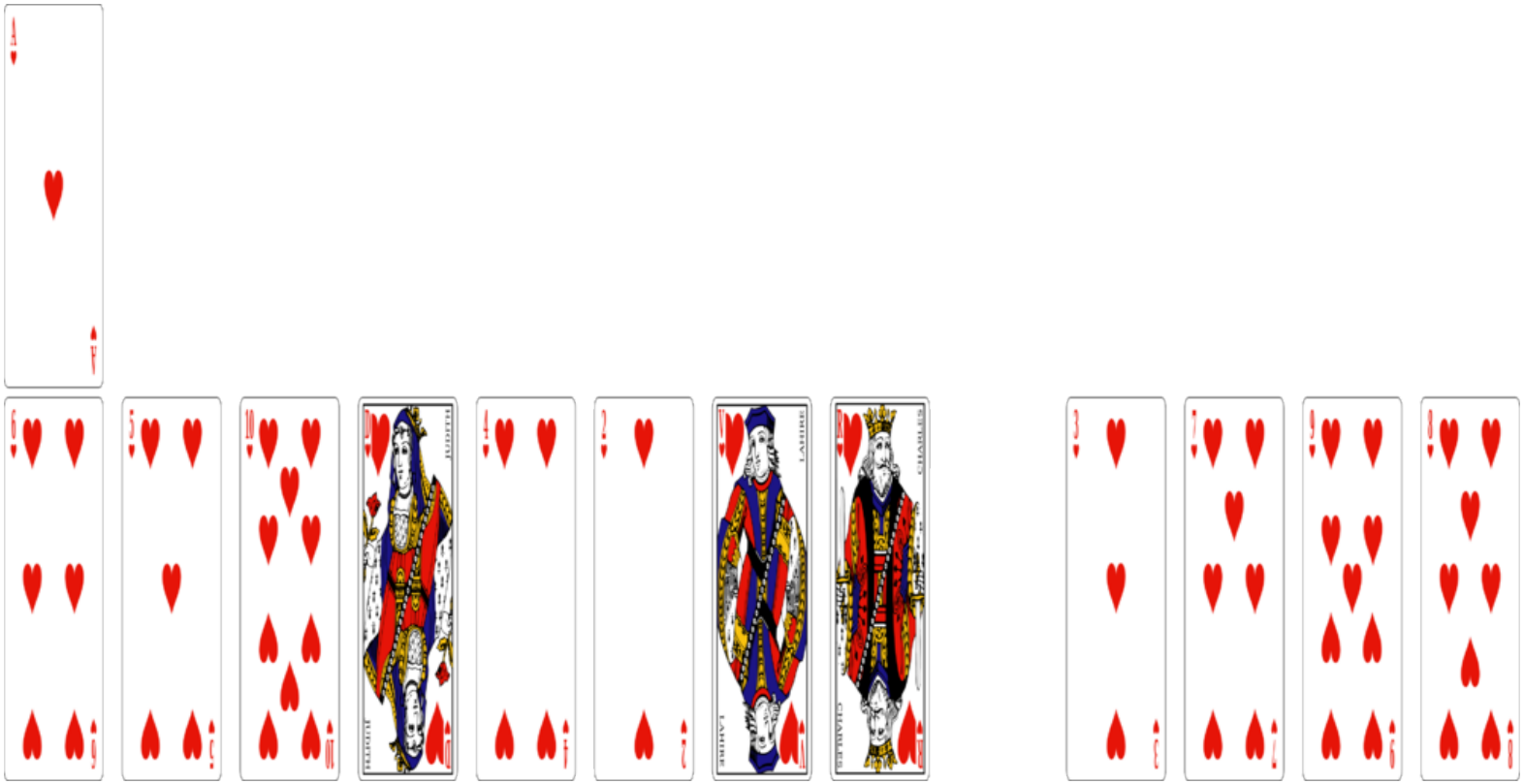


FIGURE – La place à la fin du tableau trié

Retour menu

Tri par sélection - nouveau tableau

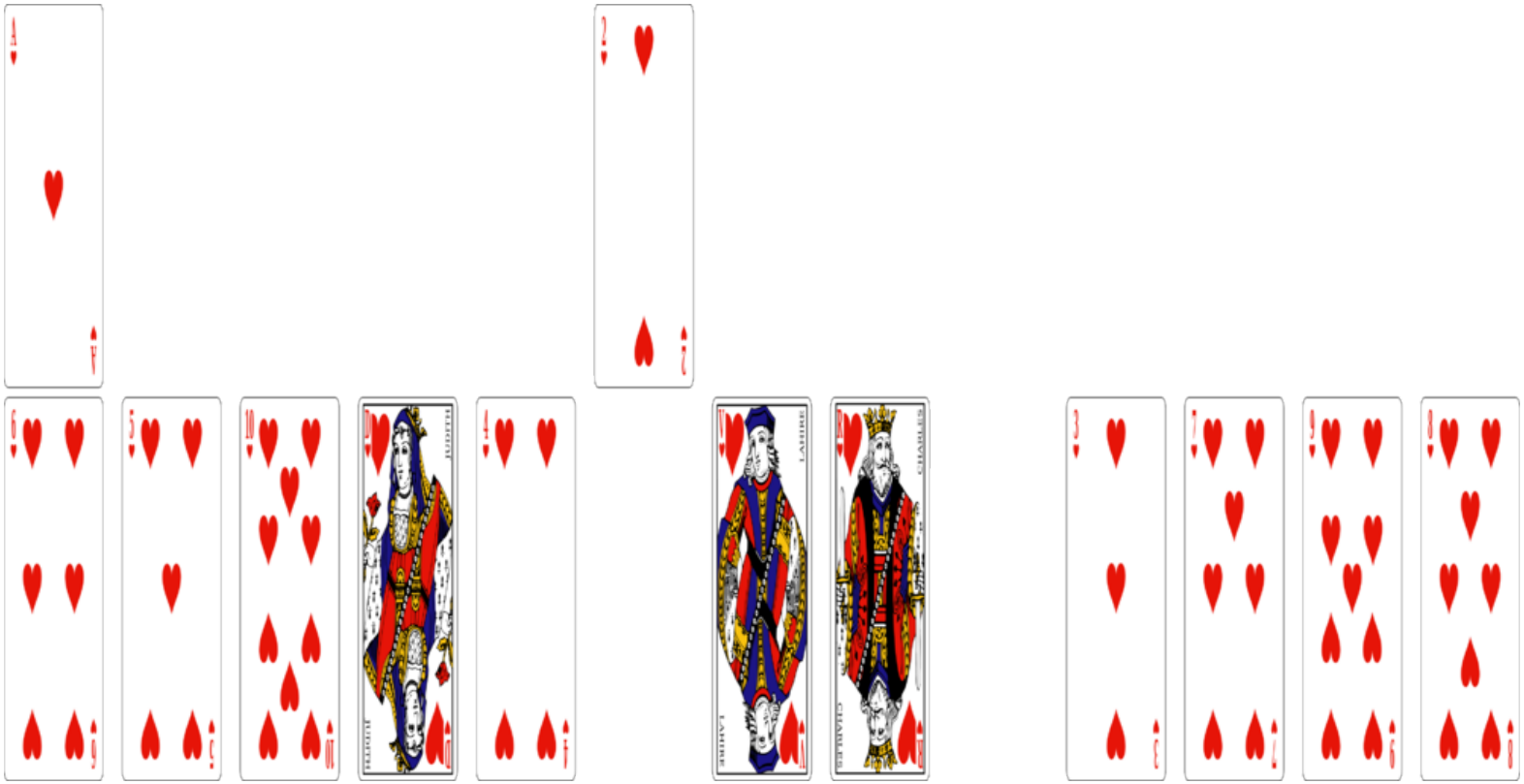


FIGURE – Trouve la plus petite du tas non trié

Retour menu

Tri par sélection - nouveau tableau

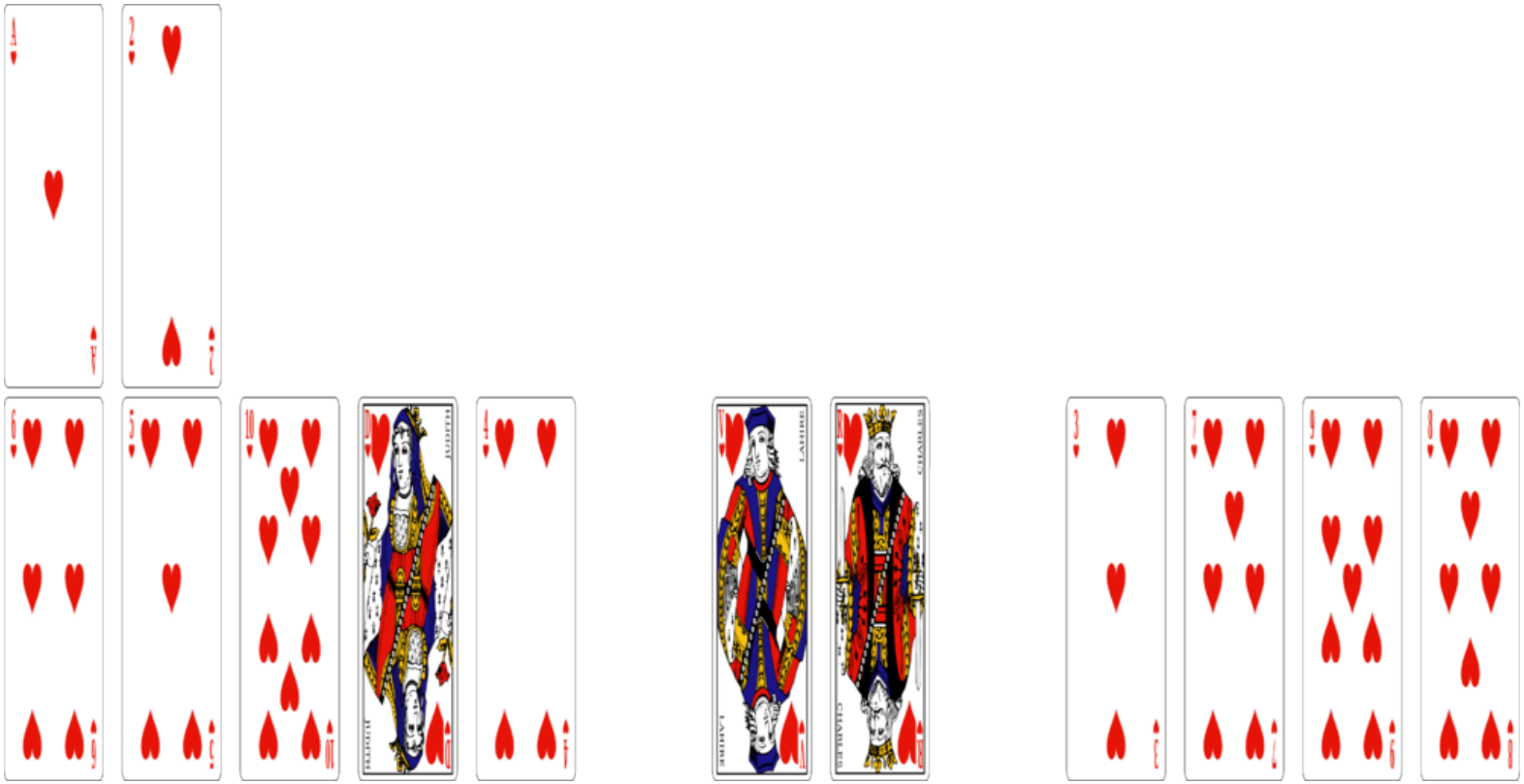


FIGURE – La place à la fin du tableau trié

Retour menu

Tri par insertion en place

```
1  Pour chaque carte du tas
2      Mémoriser la carte en cours
3      Décaler vers la droite toutes les cartes précé
           dentes, supérieures à la carte en cours.
4      Insérer la carte en cours dans l'espace vide.
```

Code 3 – Tri par insertion (en place)

Retour menu

Tri par insertion en place

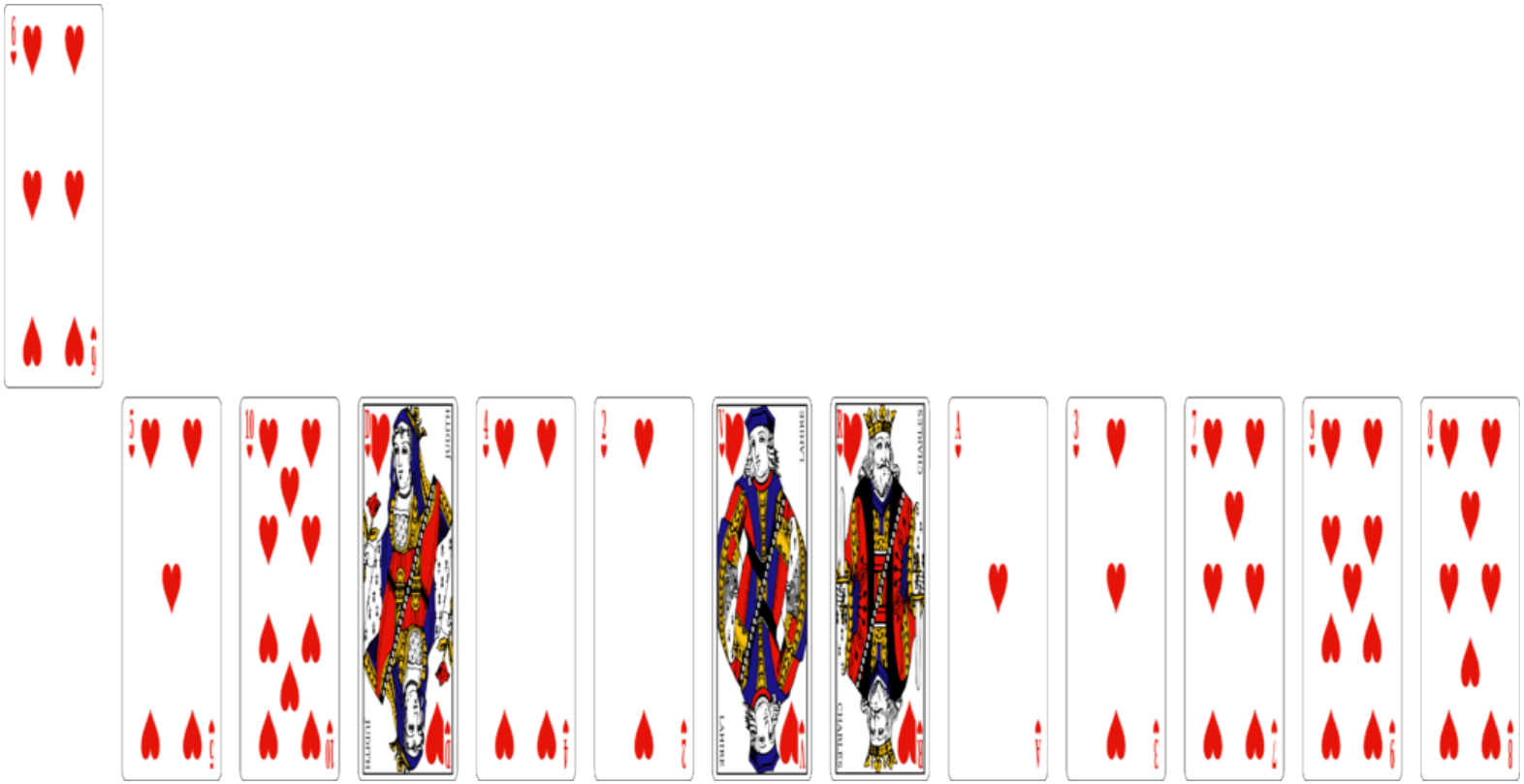


FIGURE – Mémoriser la première carte dans le tas non trié

Retour menu

Tri par insertion en place

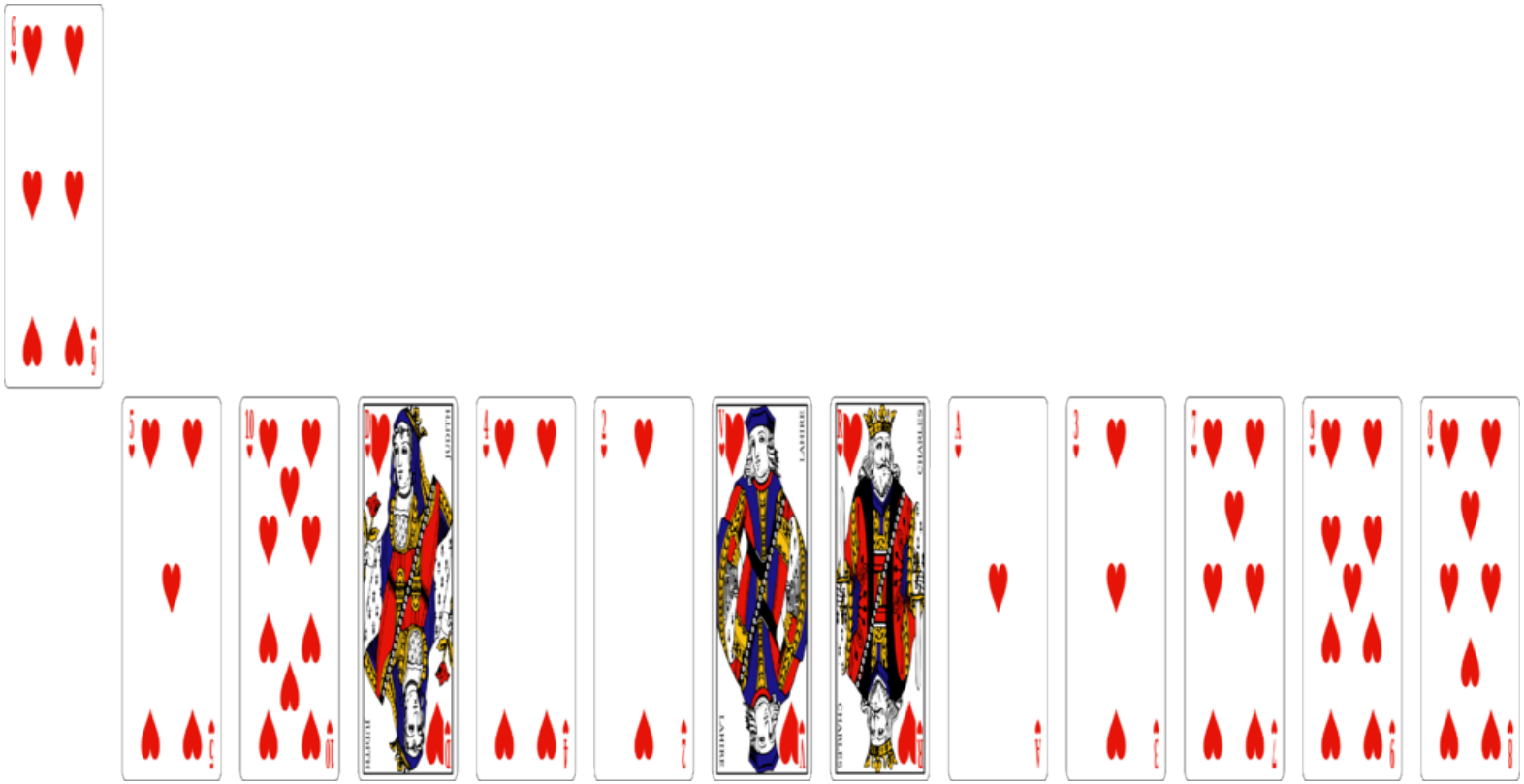


FIGURE – Décaler les cartes supérieures déjà triées

Retour menu

Tri par insertion en place

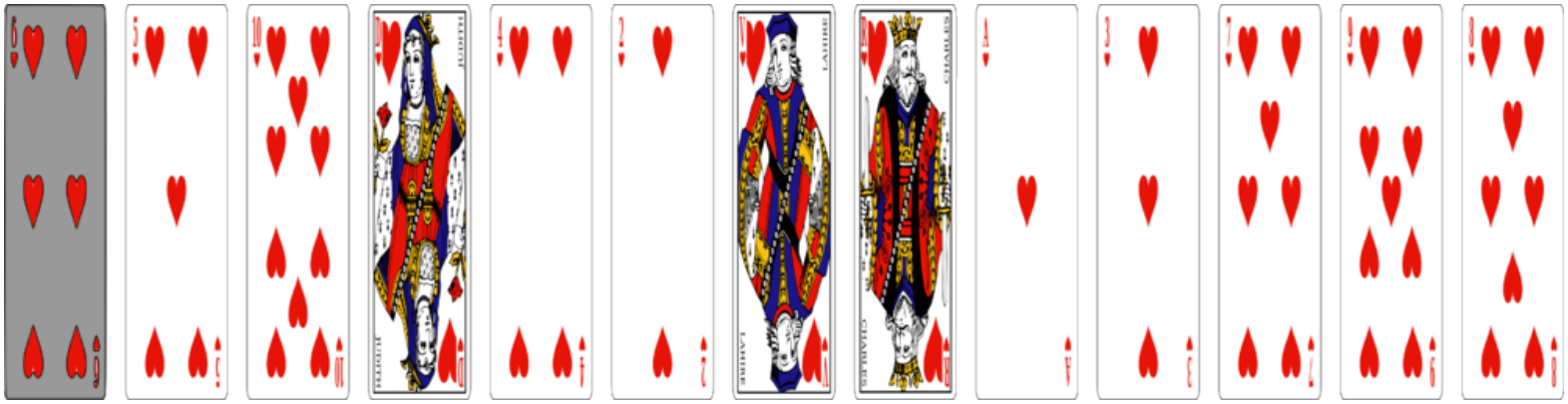


FIGURE – Replacer la carte dans l'espace

Retour menu

Tri par insertion en place

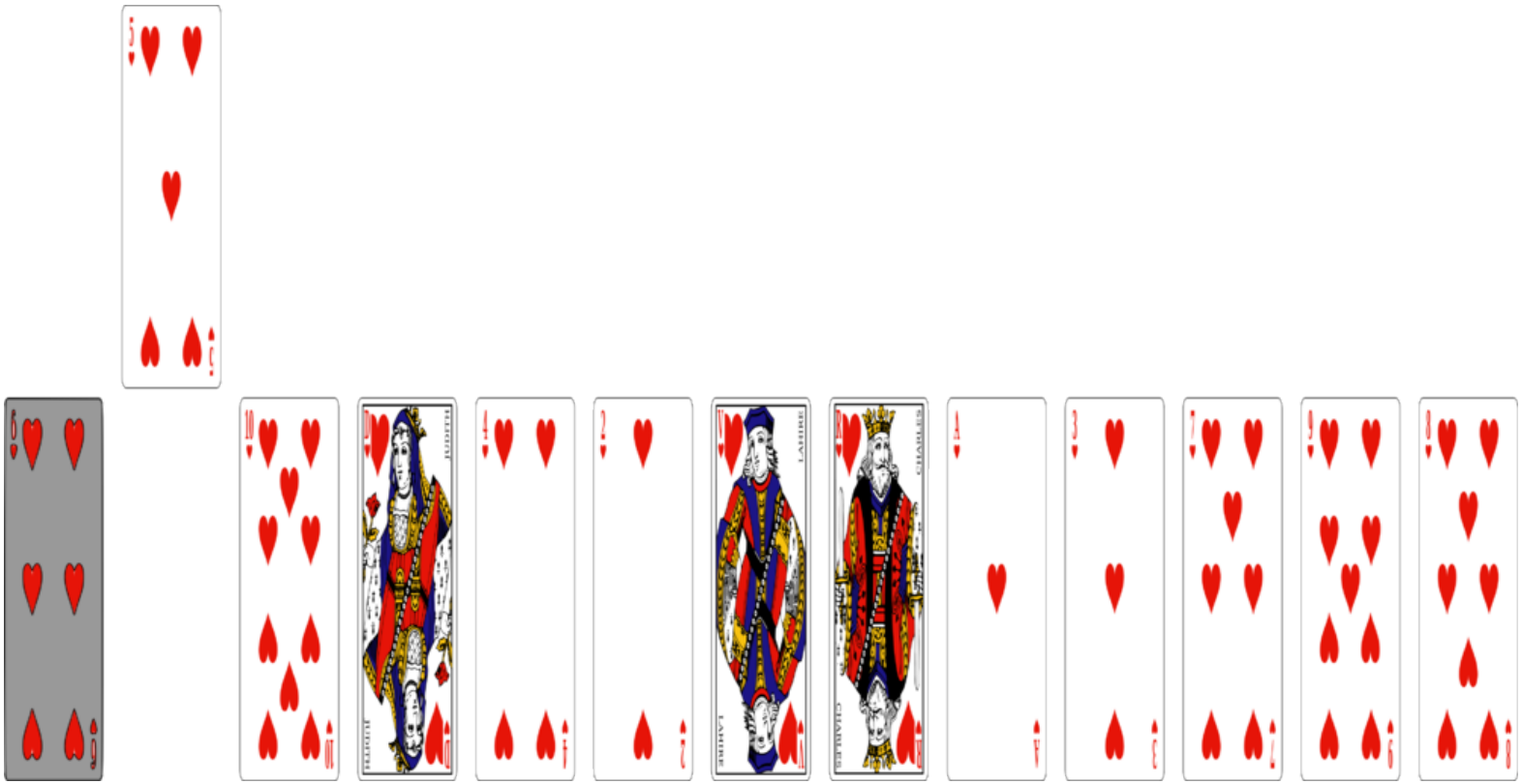


FIGURE – Mémoriser la première carte dans le tas non trié

Retour menu

Tri par insertion en place

FIGURE – Décaler les cartes supérieures déjà triées

Retour menu

Tri par insertion en place

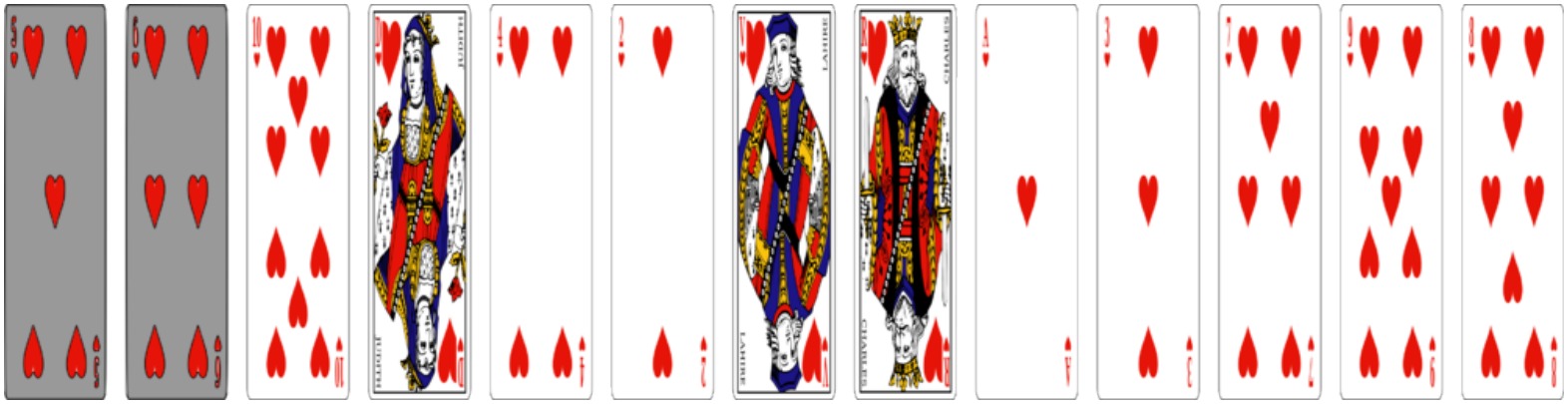


FIGURE – Replacer la carte dans l'espace

Retour menu

Tri par insertion en place

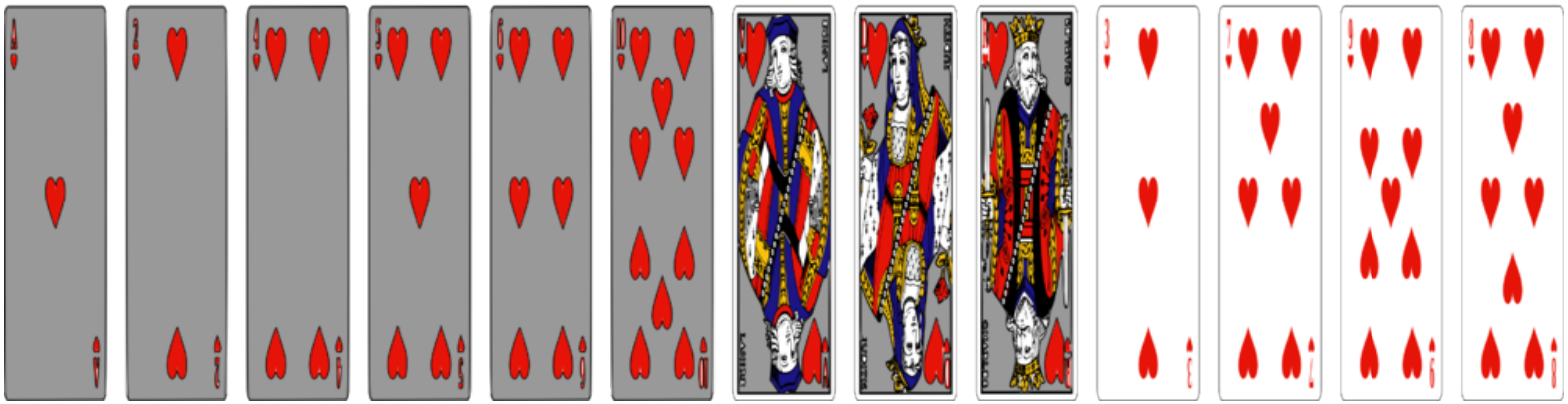


FIGURE – Après plusieurs itérations

Retour menu

Tri par insertion en place

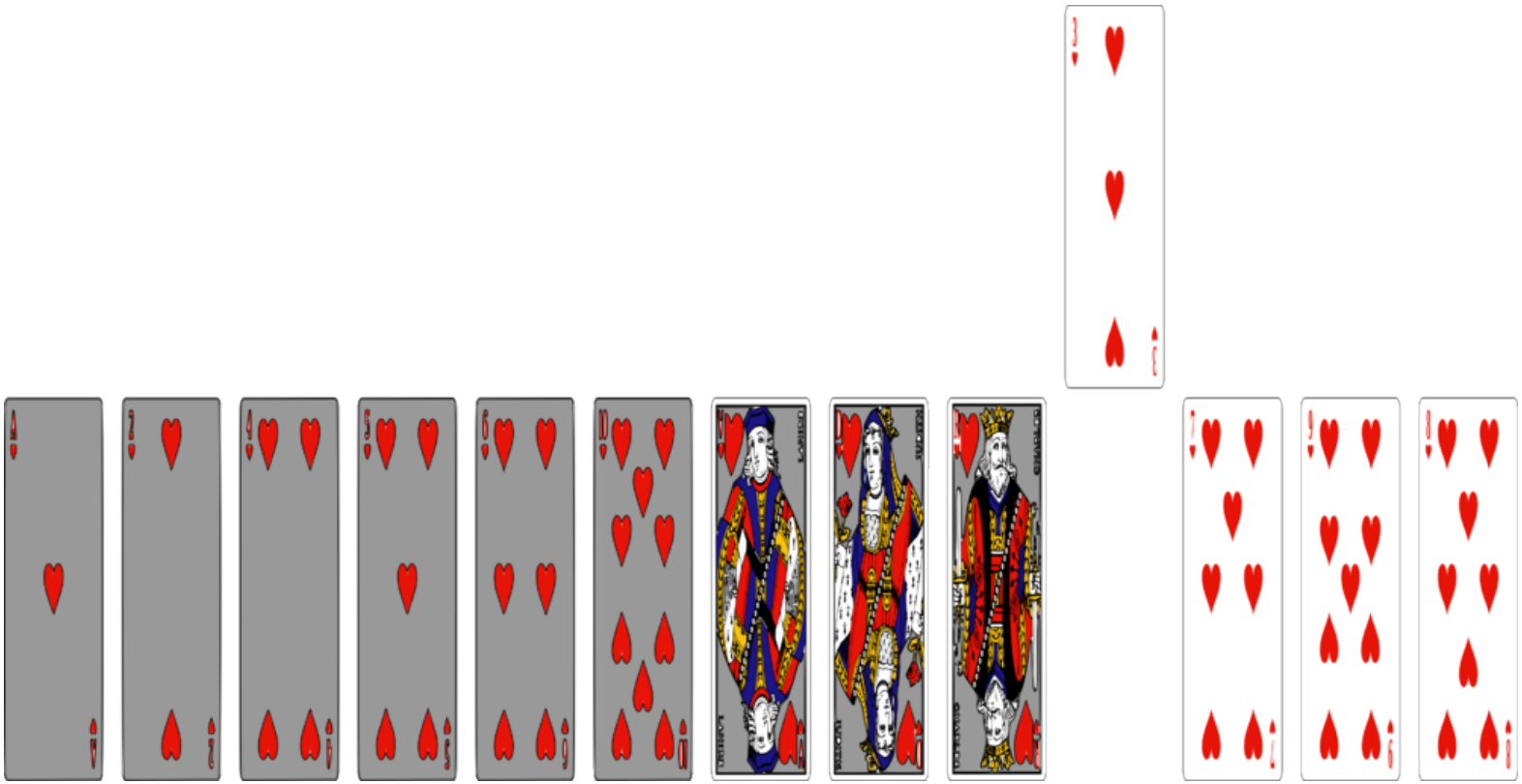


FIGURE – Mémoriser la première carte dans le tas non trié

Retour menu

Tri par insertion en place

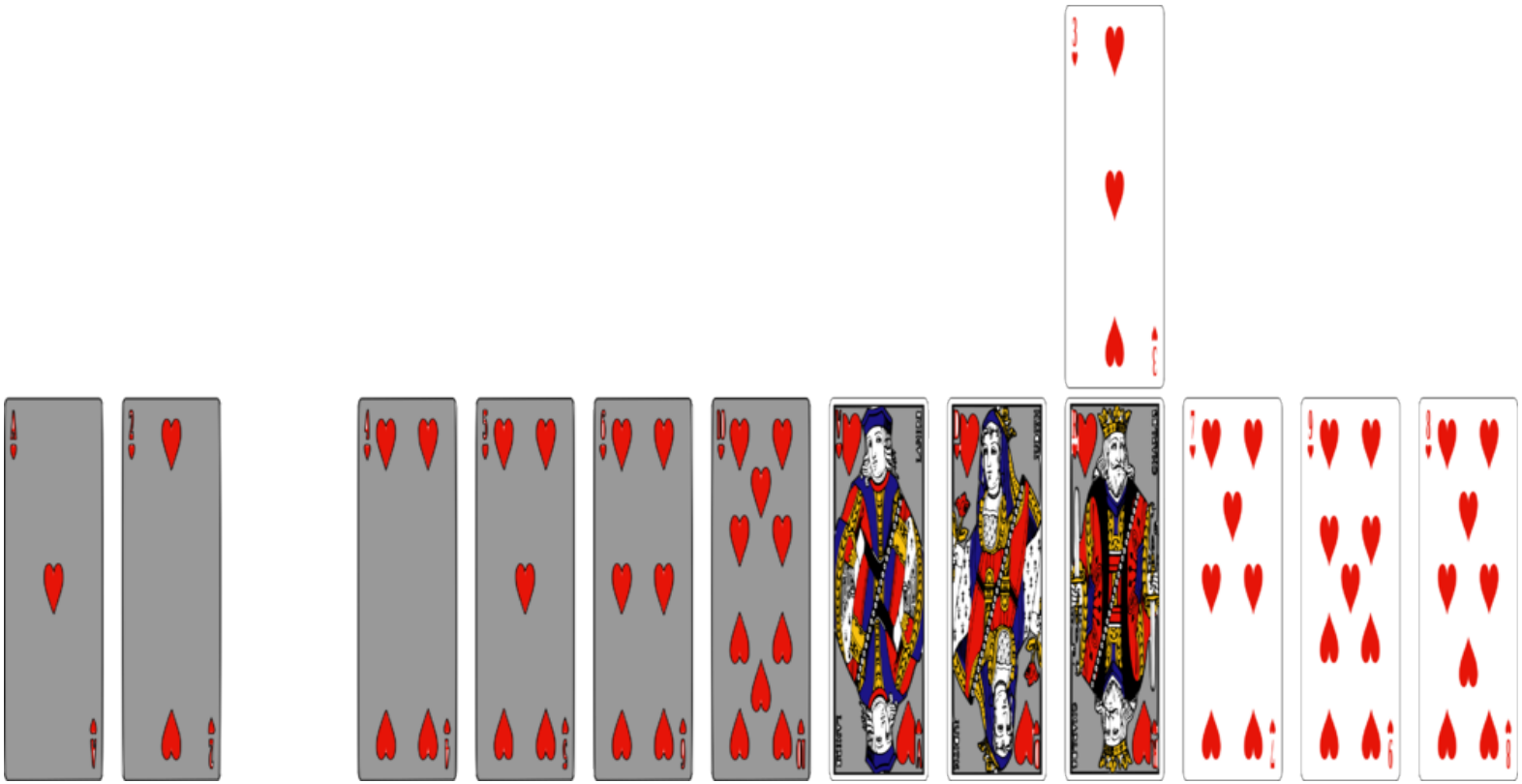


FIGURE – Décaler les cartes supérieures déjà triées

Retour menu

Tri par insertion en place

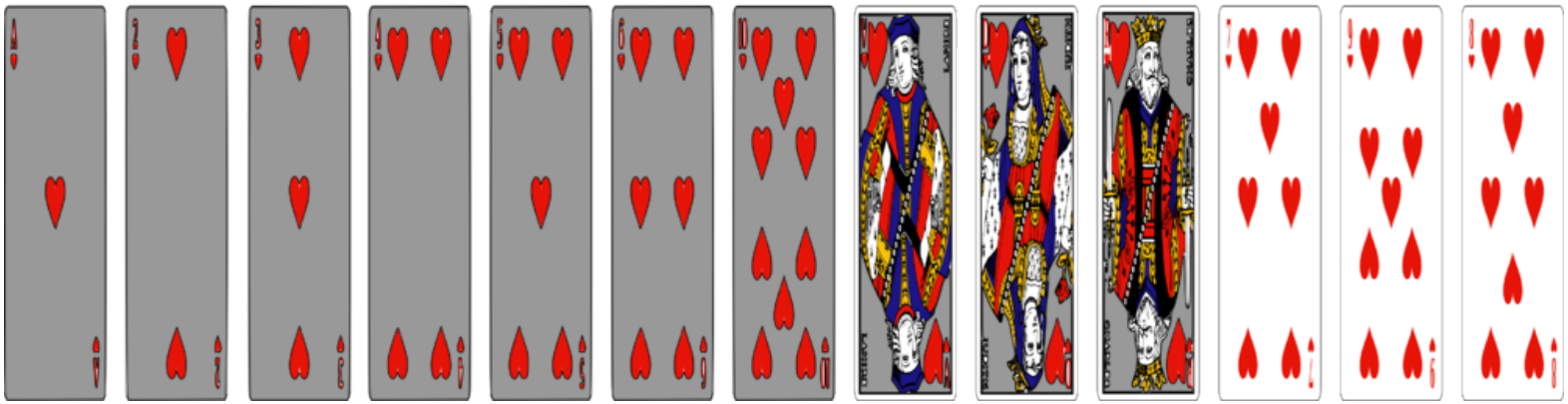


FIGURE – Insérer la carte dans l'espace

Retour menu

Tri par insertion - nouveau tableau

```
1  Pour chaque carte du tas
2      Prendre la première carte du tableau non trié.
3      Dans le tableau trié, décaler vers la droite
           toutes les cartes plus grandes.
4      Insérer la carte dans le tableau trié.
```

Code 4 – Tri par insertion dans un nouveau tableau

Retour menu

Tri par insertion - nouveau tableau

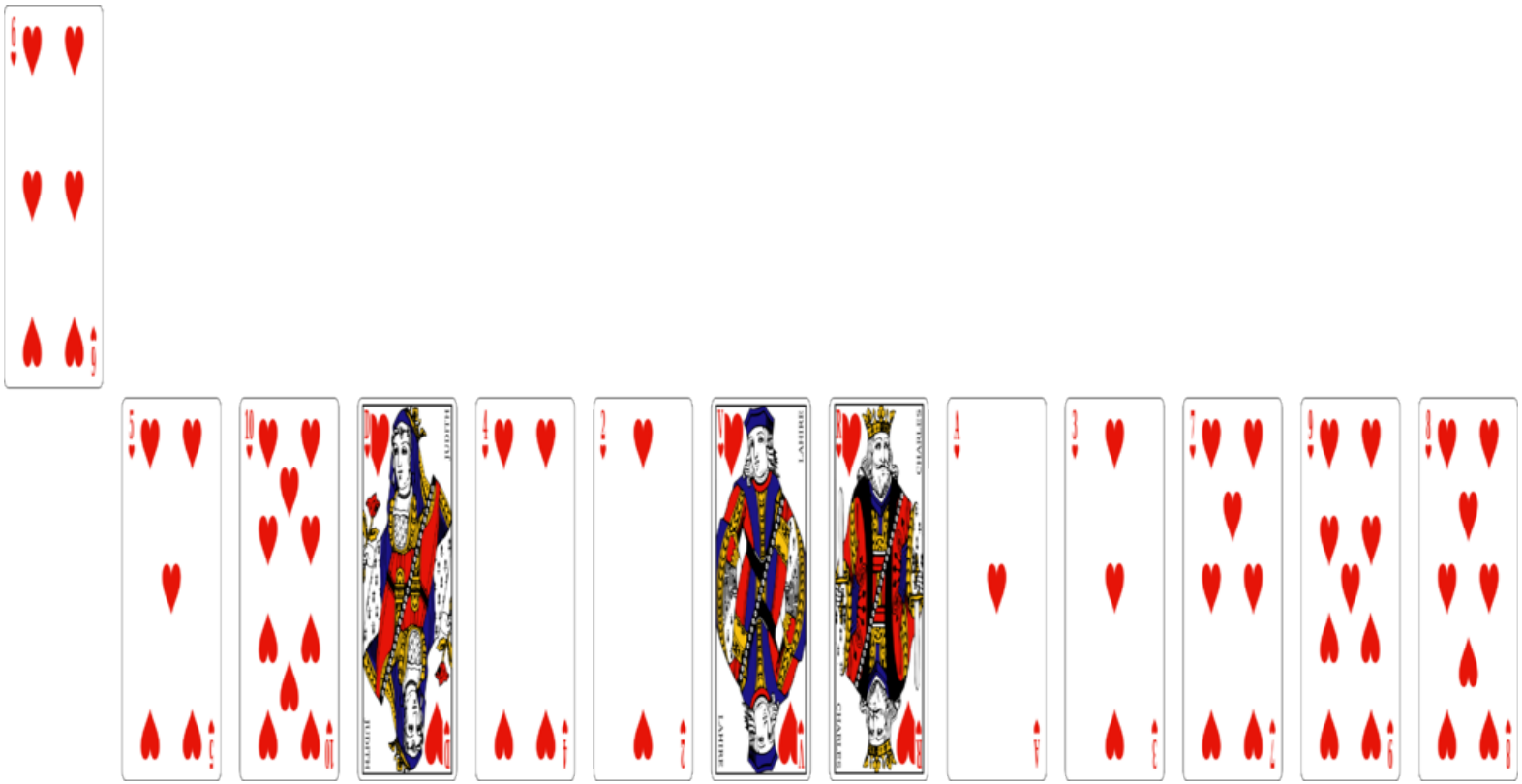


FIGURE – Prendre la première carte non triée

Retour menu

Tri par insertion - nouveau tableau

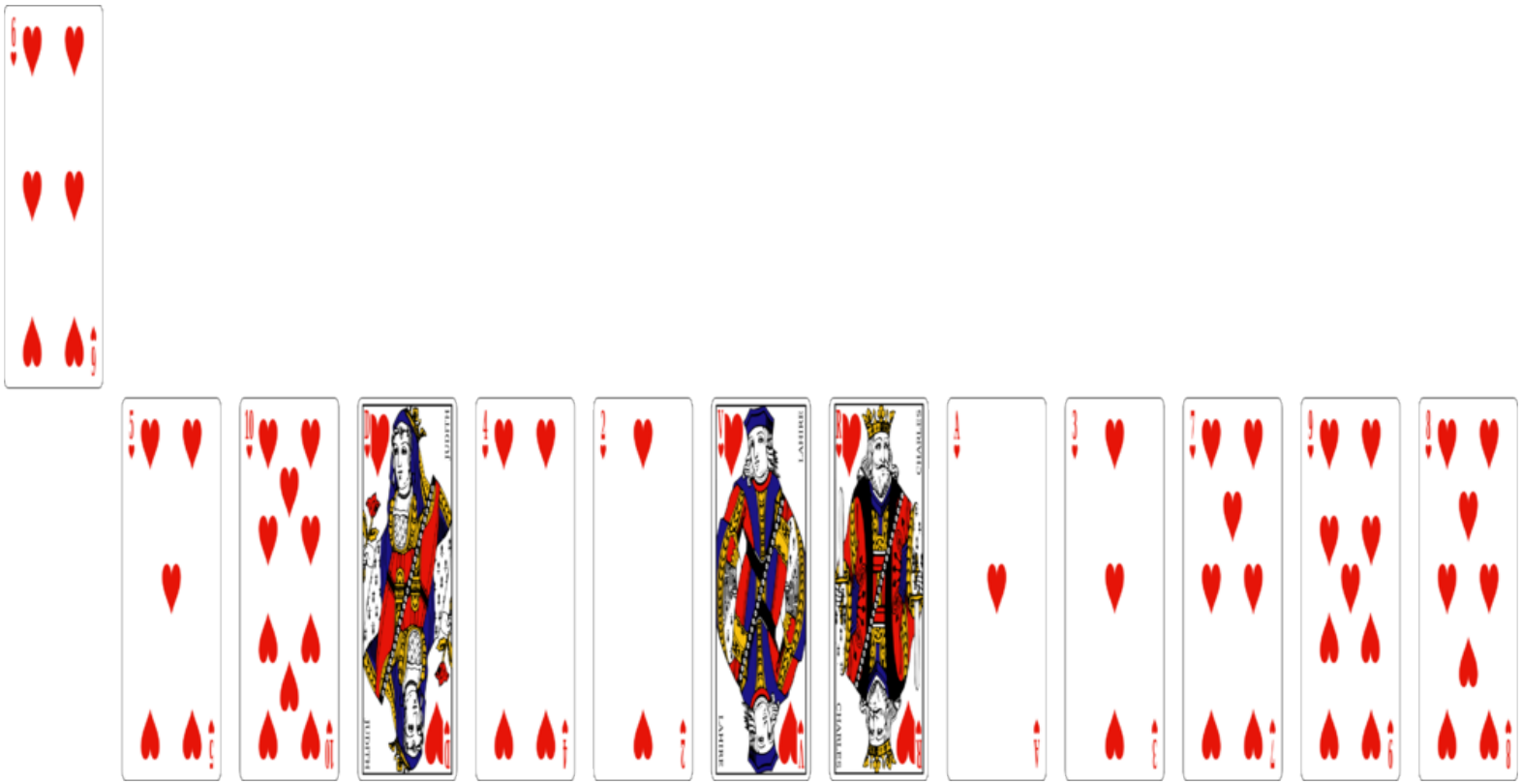


FIGURE – Décaler les cartes supérieures du tableau trié

Retour menu

Tri par insertion - nouveau tableau

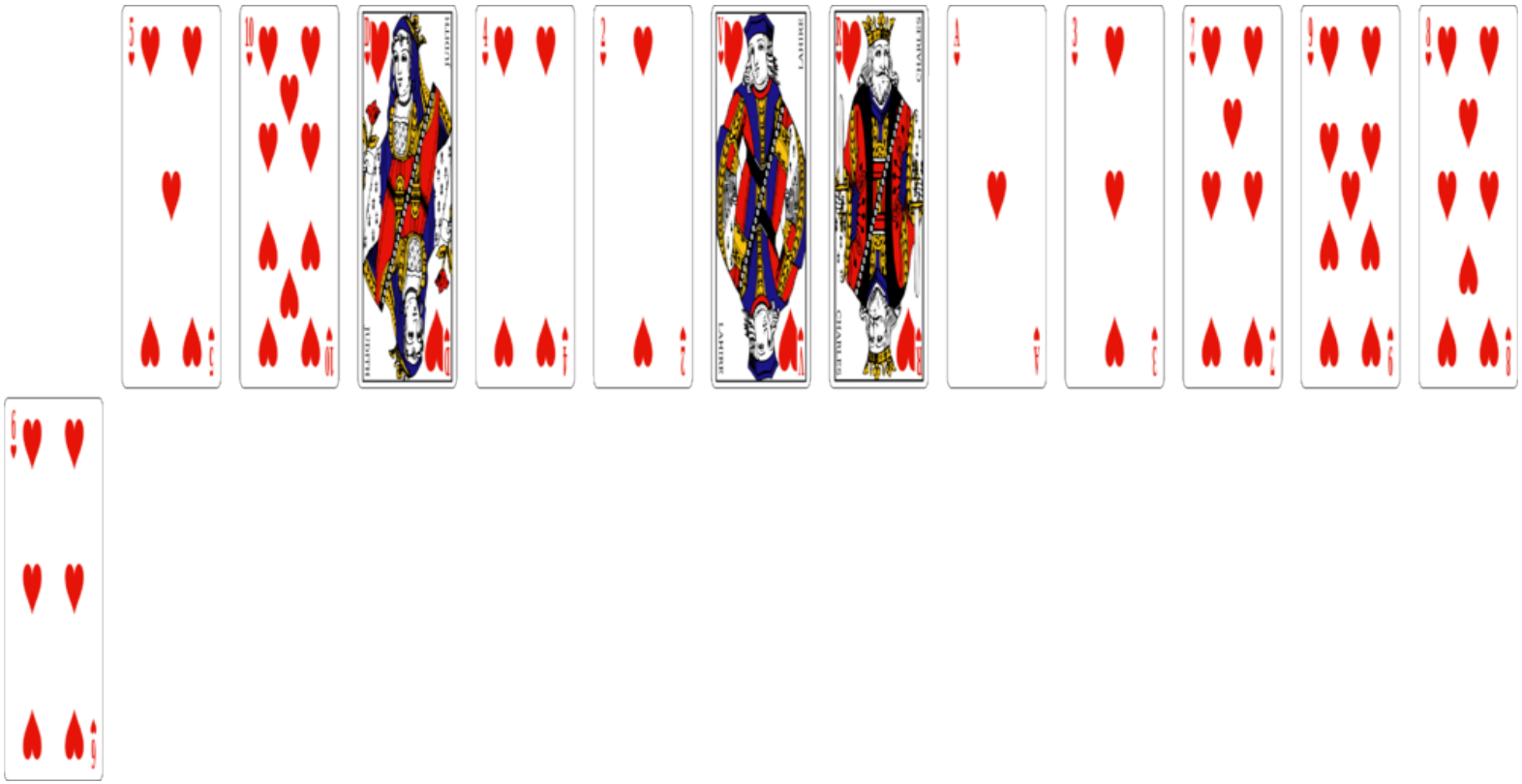


FIGURE – Insérer la carte dans le tableau triée

Retour menu

Tri par insertion - nouveau tableau

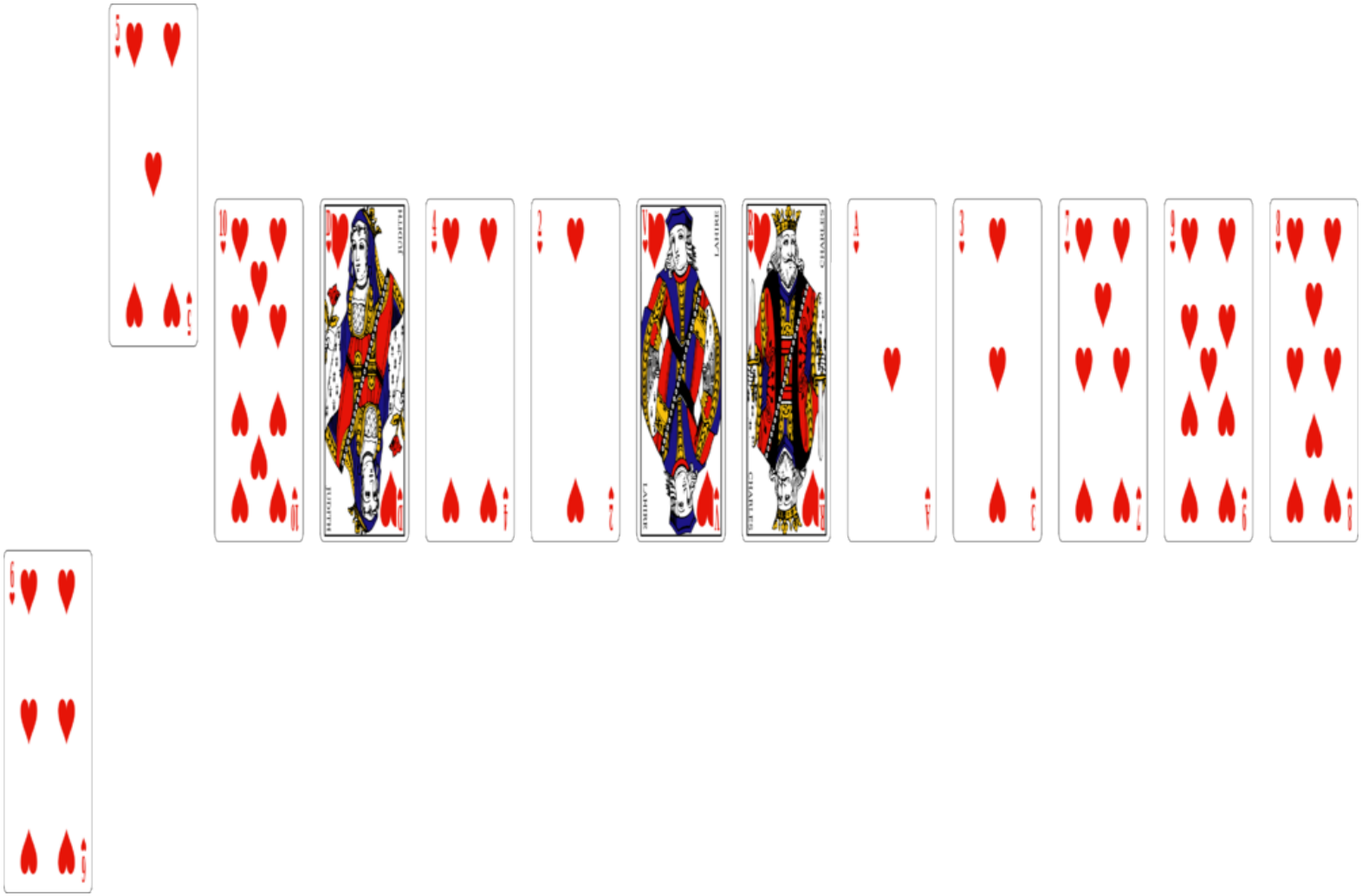


FIGURE – Prendre la première carte non triée

Retour menu

Tri par insertion - nouveau tableau

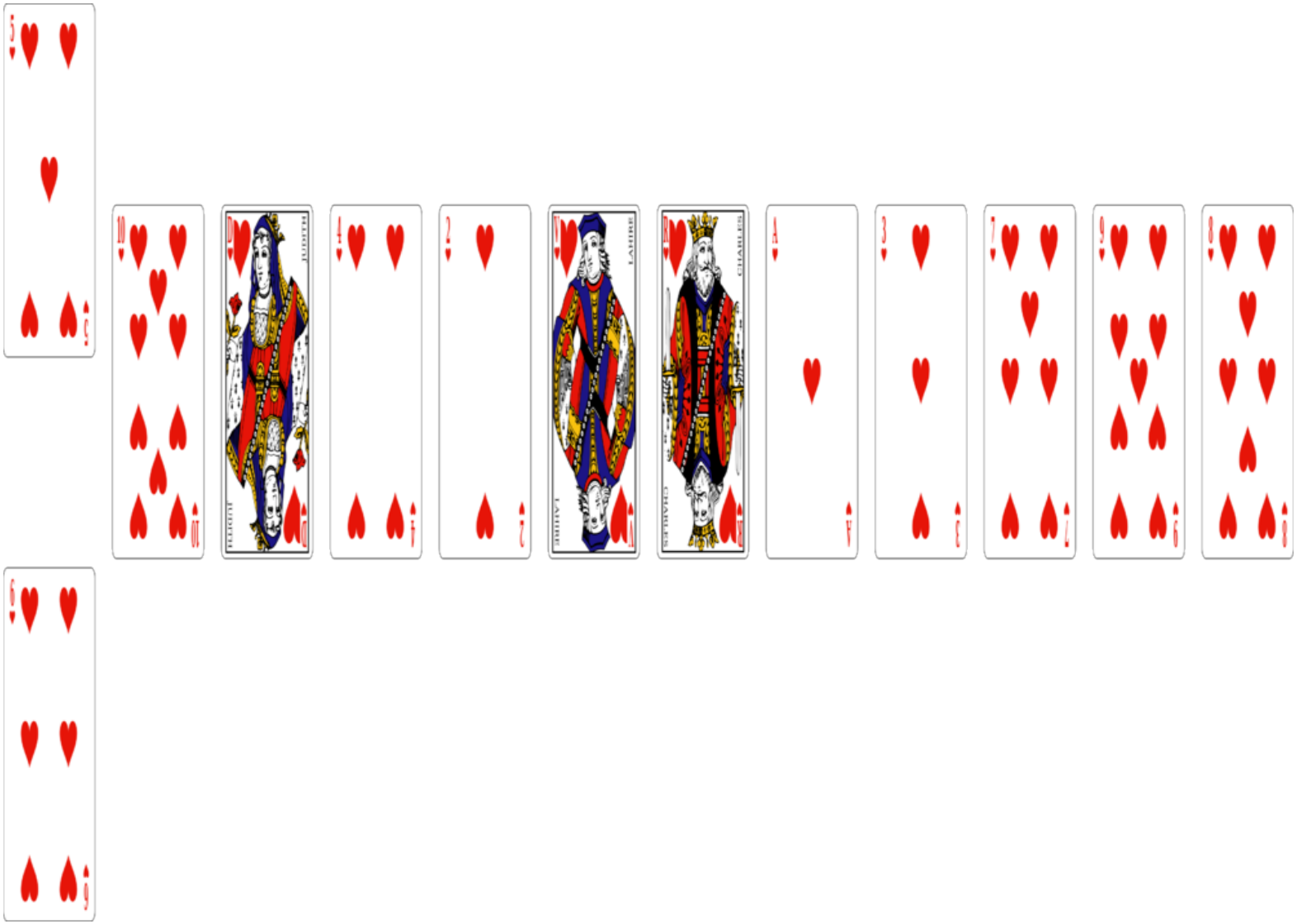


FIGURE – Décaler les cartes supérieures du tableau trié

Tri par insertion - nouveau tableau

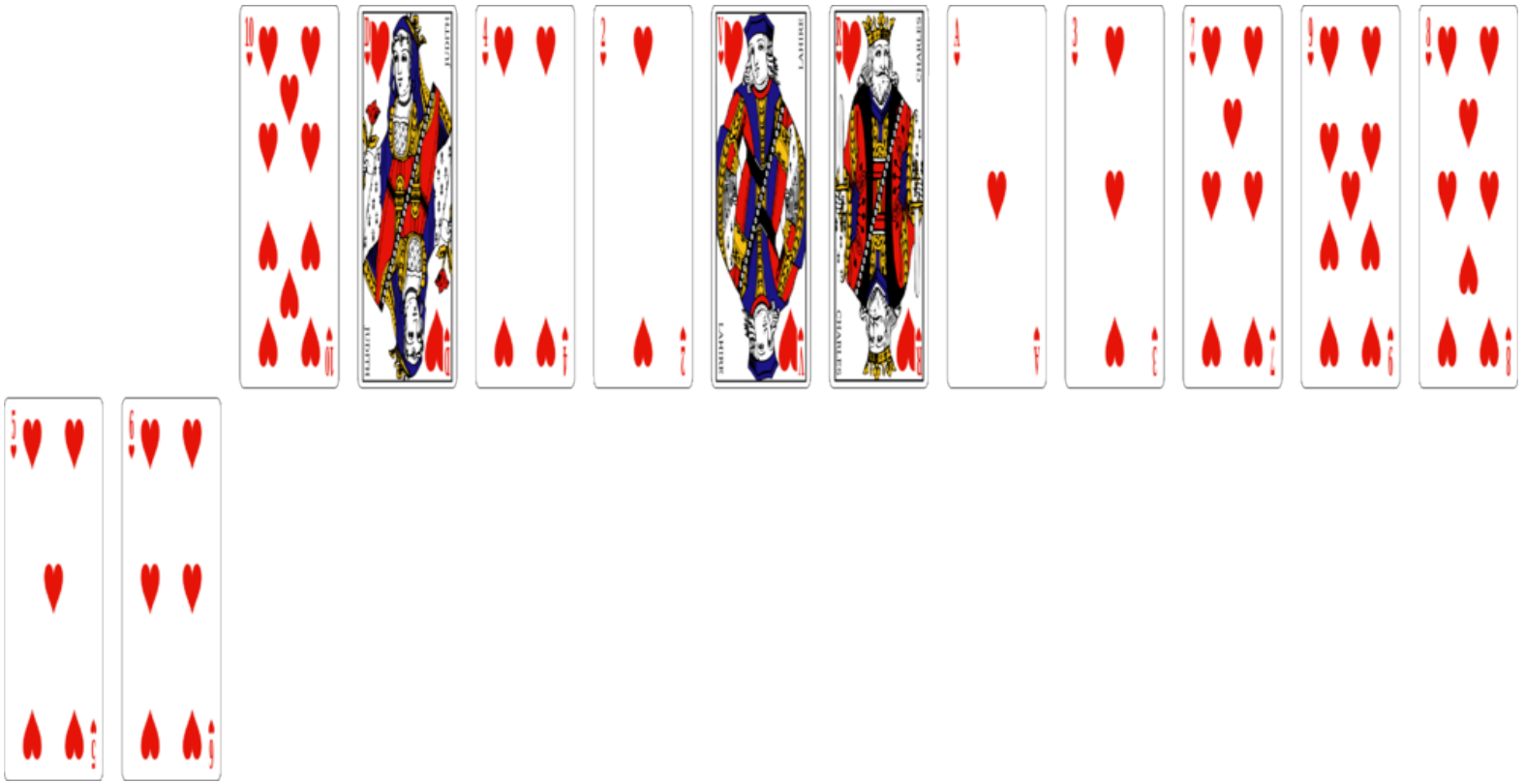


FIGURE – Insérer la carte dans le tableau triée

Retour menu

Quand on passe un tableau en argument à une fonction, cette dernière manipule le tableau original.

```
1 def ma_fonction(tab: list)->None:
2     tab[2] = 199
3
4 tab = [3, 8, 1, 10, 9]
5 ma_fonction(tab)
```

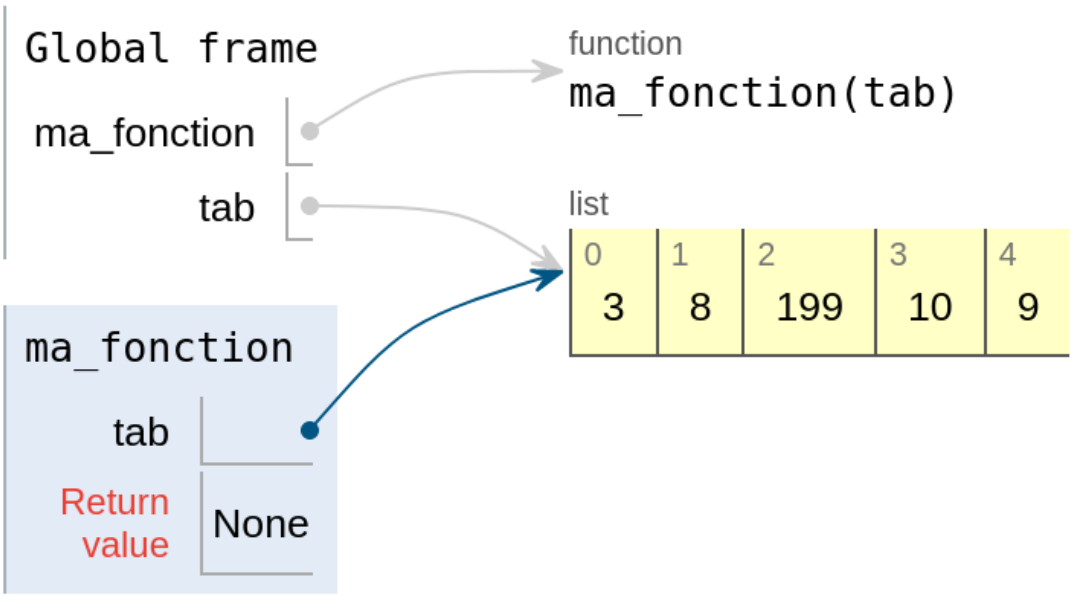


FIGURE – La fonction modifie le tableau original

Tri par sélection - question préparatoire

Activité 2 :

1. Écrire la fonction **trouver_mini(tab : list)** $\rightarrow$ **int** qui renvoie l'indice du plus petit élément de *tab*.

Correction

```
1  def trouver_mini(tab: list )->int:
2      """
3      Trouve l'indice du plus petit élément
4      """
5      i_mini = 0
6      for j in range(1, len(tab)):
7          if tab[j] < tab[i_mini] :
8              i_mini = j
9      return i_mini
```

Tri par sélection - Implémentation

Activité 2 :

2. Adapter la fonction précédente pour renvoyer l'indice du plus petit élément de *tab*, compris entre l'indice *i_depart* et la fin du tableau. La signature de la fonction deviendra **trouver_mini(tab : list, i_depart : int) → int**.
3. Écrire la fonction **echanger(tab : list, i : int, j : int) → None** qui échange les éléments d'indice *i* et *j* du tableau *tab*.
4. Écrire la fonction **tri_selection(tab : list) → None** qui effectue un tri par sélection sur *tab*.

Correction

```
1  def trouver_mini(tab: list, i_depart: int) -> int:
2      """
3      renvoie l'indice du plus petit élément entre
4      i_depart et la fin du tableau
5      """
6      i_mini = i_depart
7      for j in range(i_depart+1, len(tab)):
8          if tab[j] < tab[i_mini]:
9              i_mini = j
10     return i_mini
```

Correction

```
1  def echanger(tab: list, i: int, j: int) -> None:
2      """
3      inverse les élément d'indices i et j du tableau
4      """
5      tab[i], tab[j] = tab[j], tab[i]
```

Correction

```
1  def tri_selection(tab: list) -> None:
2      """
3      tri le tableau dans l'ordre croissant
4      """
5      for i in range(len(tab)):
6          position_du_mini = trouver_mini(tab, i)
7          echanger(tab, i, position_du_mini)
```

Tri par sélection - Tester

Activité 2 :

5. Construire par compréhension un tableau des entiers de 1 à 13.
6. Mélanger le tableau à l'aide de la méthode *shuffle* de la bibliothèque *random*.
7. Trier le tableau à l'aide de la fonction *tri_selection*.

Correction

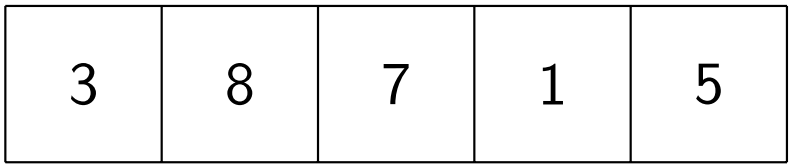
```
1 cartes = [i for i in range(1, 14)]
2 shuffle(cartes)
3 tri_selection(cartes)
```

Preuve de terminaison : variant de boucle

La terminaison de la fonction est triviale. Le tri est composé de deux boucles bornées donc qui terminent.

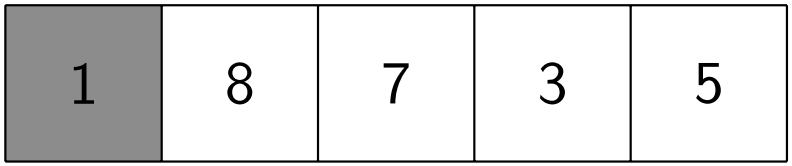
Preuve de correction : invariant de boucle

Avant chaque itération de la boucle externe, la partie gauche du tableau est triée.



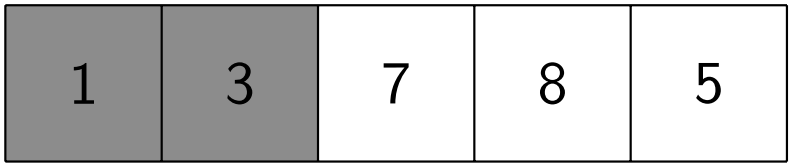
Code 5 – Avant la première itération, la partie gauche est vide, donc triée.

Preuve de correction : invariant de boucle



Code 6 – Avant la deuxième itération, la partie gauche est triée.

Preuve de correction : invariant de boucle



Code 7 – Avant la troisième itération, la partie gauche est triée.

Efficacité du tri

- à la première itération de *i*, la boucle de la fonction *trouver_mini* effectue *n-1* itérations.
- à la deuxième itération de *i*, la boucle de la fonction *trouver_mini* effectue *n-2* itérations.
- ...

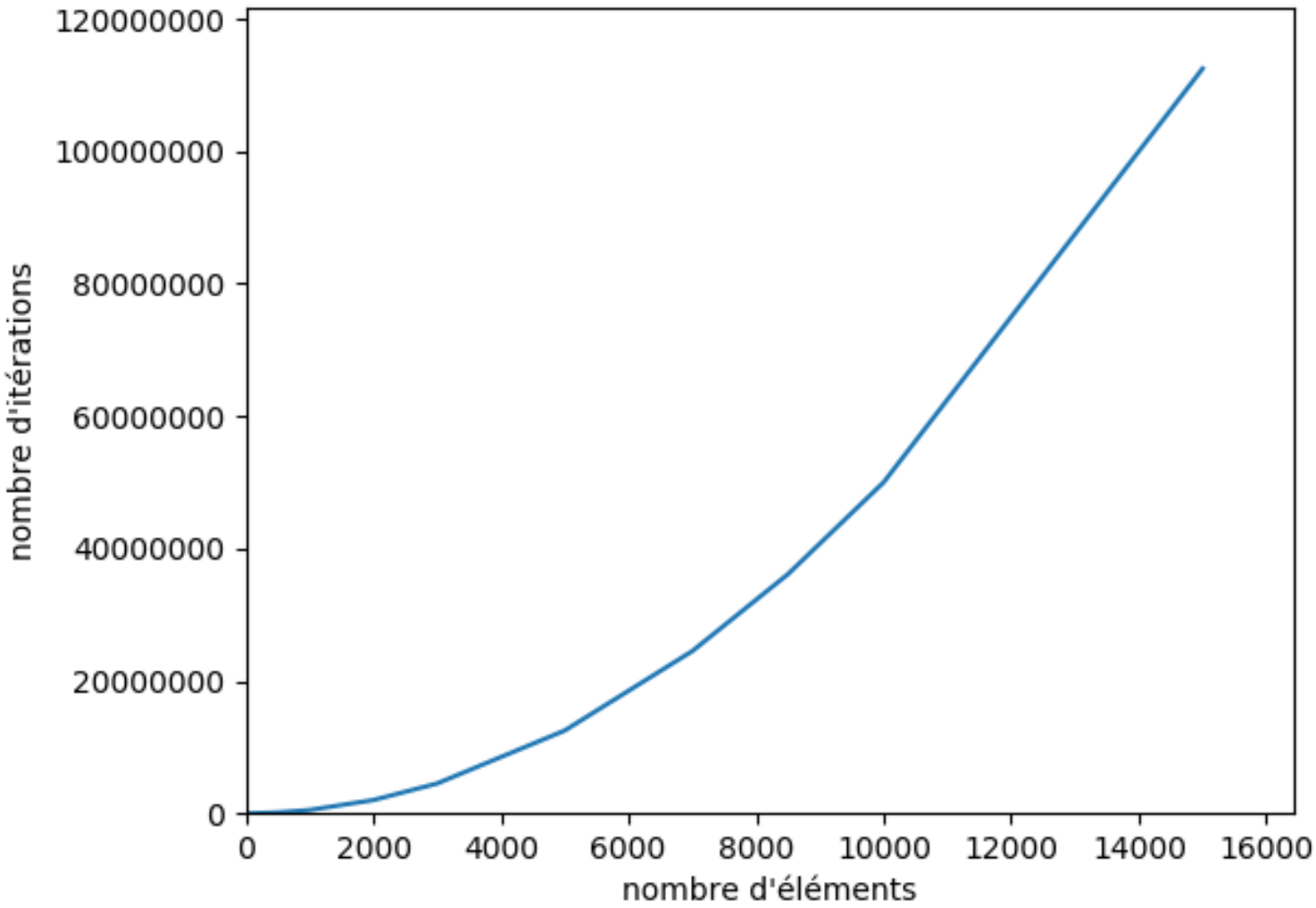
```
1 for j in range(i_depart+1, len(tab)):
2     if tab[j] < tab[i_mini]:
3         i_mini = j
```

$$\sum_{k=1}^{n-1} k = 1 + 2 + \cdots + n - 1 = \frac{n.(n-1)}{2}$$

À retenir

Le tri par sélection effectue $\frac{n.(n-1)}{2}$ opérations pour ordonner le tableau. Le nombre d'opérations dépend de n^2.

Évolution du nombre d'itérations



Tri par insertion

Activité 3 :

1. Écrire la fonction **tri_insertion(tab : list)** $\rightarrow$**None** en s'appuyant sur l'algorithme. Les indications suivantes permettront de construire les trois étapes :
   - Mémoriser : définir une variable *en_cours*, élément en cours de placement et *pos*, position actuelle de cet élément.
   - Décaler : utiliser une boucle non bornée pour décaler les éléments vers la droite.
   - Insérer : placer l'élément *en_cours* à la nouvelle position *pos*.
2. Tester la fonction sur un tableau.

Rappel de l'algorithme

```
1  Pour chaque carte du tas
2      Mémoriser la carte en cours
3      Décaler vers la droite toutes les cartes précé
           dentes, supérieures à la carte en cours.
4      Insérer la carte en cours dans l'espace vide.
```

Code 8 – Tri par insertion (en place)

Correction : boucle principale

```
1  def tri_insertion(tab: list) -> None:
2      """
3      tri le tableau dans l'ordre croissant
4      """
5      for i in range(len(tab)):
```

Correction : mémoriser

```
1        en_cours = tab[i]
2        pos = i
```

Correction : décaler

```
1        while pos > 0 and en_cours < tab[pos-1] :
2            tab[pos] = tab[pos-1]
3            pos = pos-1
```

Correction : insérer

```
1    tab[pos] = en_cours
```

Correction : code complet

```
1  def tri_insertion (tab: list ) -> None:
2      """
3      tri le tableau dans l'ordre croissant
4      """
5      for i in range(len(tab)):
6          # mémoriser
7          en_cours = tab[i]
8          pos = i
9          # décaler
10         while pos > 0 and en_cours < tab[pos-1] :
11             tab[pos] = tab[pos-1]
12             pos = pos-1
13         # insérer
14         tab[pos] = en_cours
```

Correction : tester

```
1 cartes = [i for i in range(1, 14)]
2 shuffle(cartes)
3 tri_insertion(cartes)
```

Il faut se focaliser sur la boucle interne, non bornée.

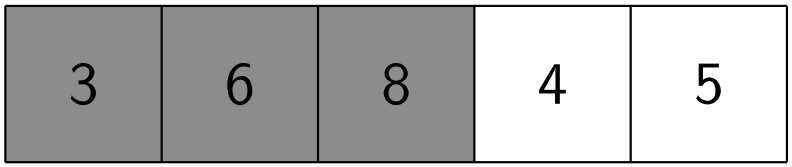
> **Activité 4 :** Déterminer un variant de la boucle, qui prouve la terminaison.

Correction

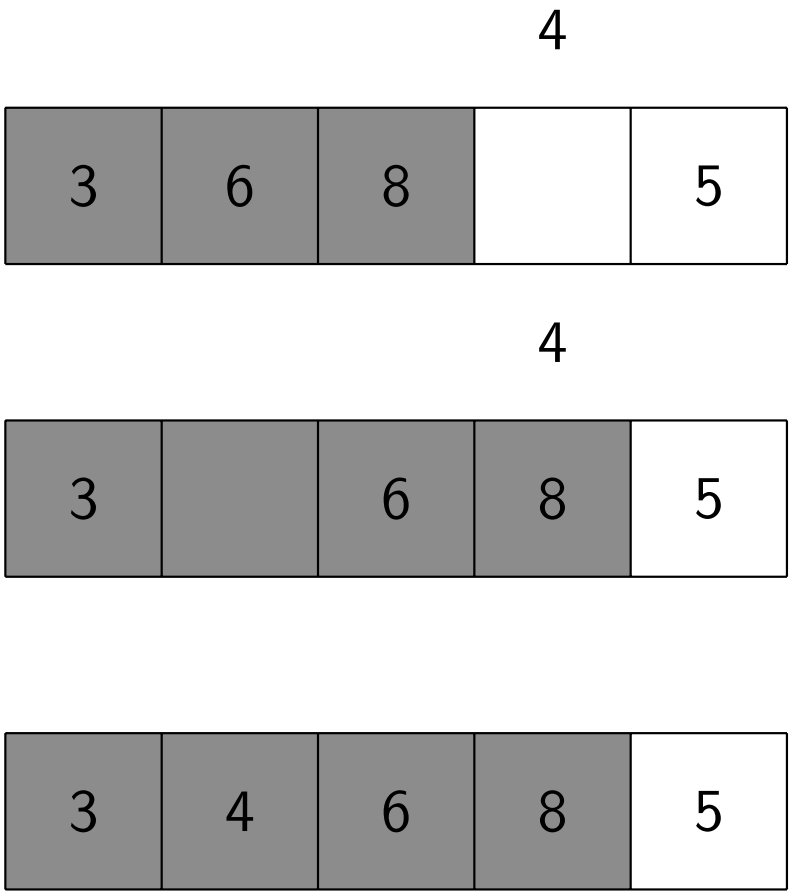
```
1  while pos > 0 and en_cours < tab[pos-1] :
2      tab[pos] = tab[pos-1]
3      pos = pos-1
```

pos est un variant de la boucle.

Comme pour le tri sélection, avant chaque itération de la boucle externe, la partie gauche du tableau est triée.



Code 9 – Insertion de l'élément 4



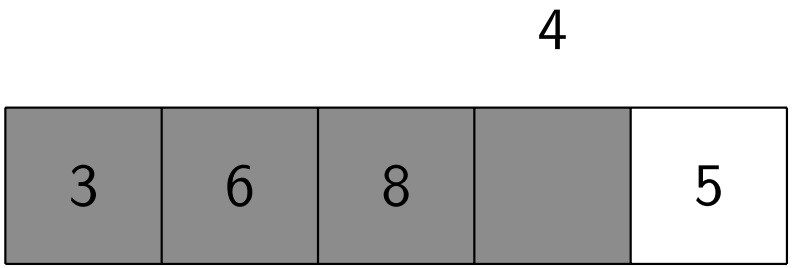
Code 10 – Insertion de l'élément 4

La boucle externe effectue n itérations dans tous les cas.

```
1  for i in range(len(tab)):
```

Cependant, le nombre d'itérations de la boucle interne peut varier.

```
1  while pos > 0 and en_cours < tab[pos-1] :
```



Code 11 – Insertion de l'élément 4

Activité 5 :

1. Compter le nombre d'itérations de la boucle interne si le tableau est déjà trié.
2. Compter le nombre d'itérations de la boucle interne si le tableau est trié dans l'ordre décroissant.

Correction

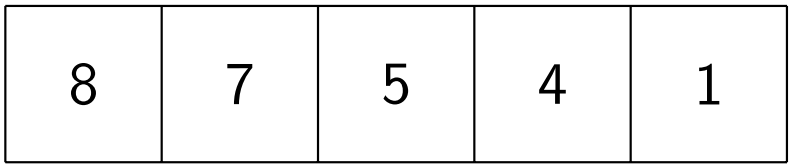
```
1  while pos > 0 and en_cours < tab[pos−1] :
```

1	4	5	7	8

Code 12 – Le tableau est déjà trié. La boucle interne n'effectue aucune itération.

Correction

```
1  for i in range(len(tab)):
2      en_cours = tab[i]
3      pos = i
4      while pos > 0 and en_cours < tab[pos-1] :
```



Code 13 – Le tableau est inversé. La boucle interne effectue i itérations.

À retenir

Le tri par insertion effectue un nombre moyen d'opérations qui dépend de n^2.