Introduction à l’informatique CM5

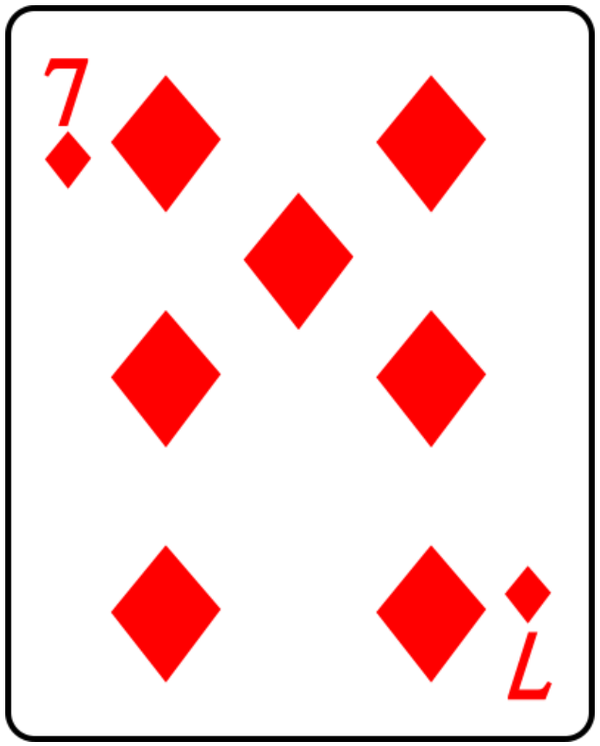
Antonio E. Porreca
aeporreca.org/introinfo

⚠️ Rappel ⚠️

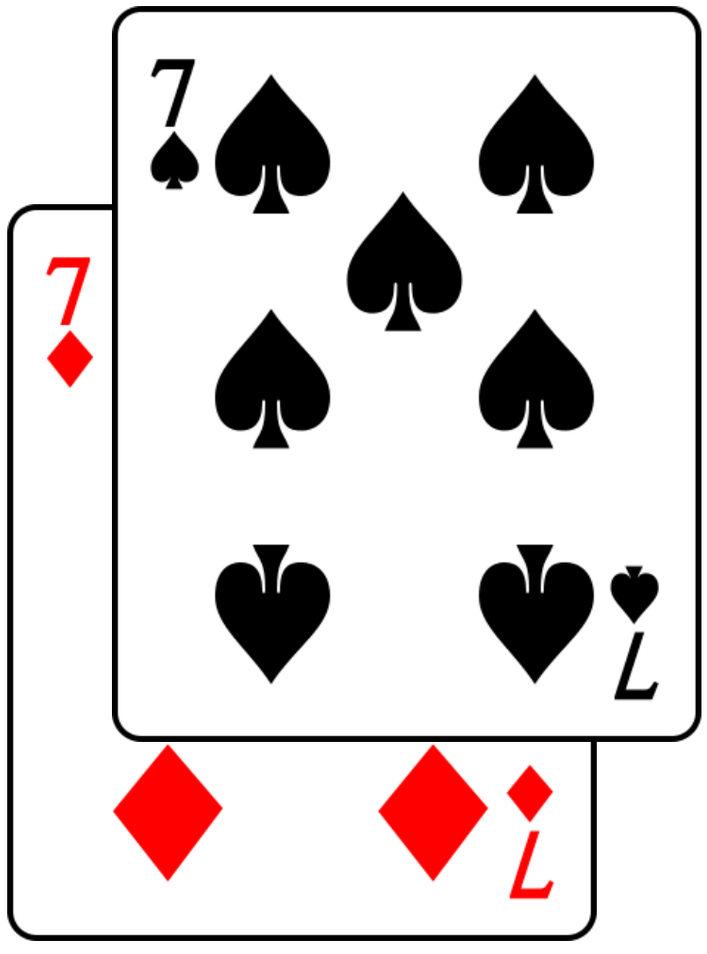
Vendredi 22 octobre (normalement à 14h) on a le partiel !

Tri par insertion

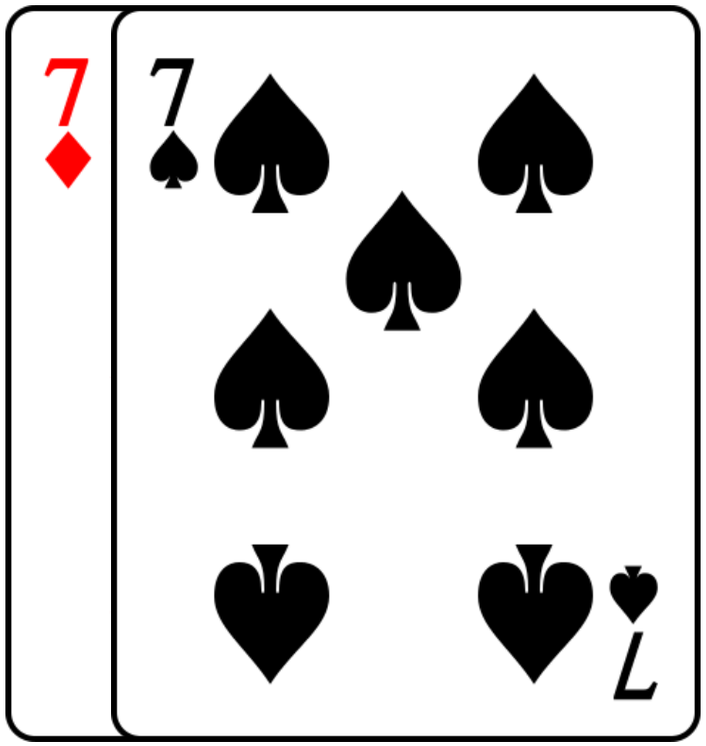
Tri par insertion



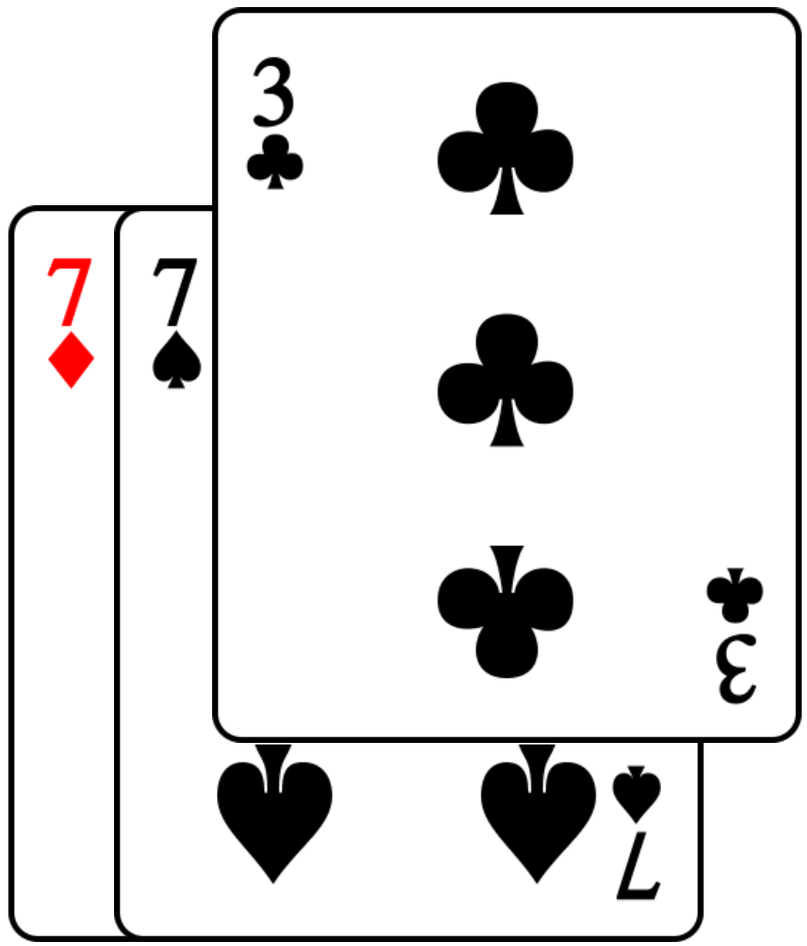
Tri par insertion



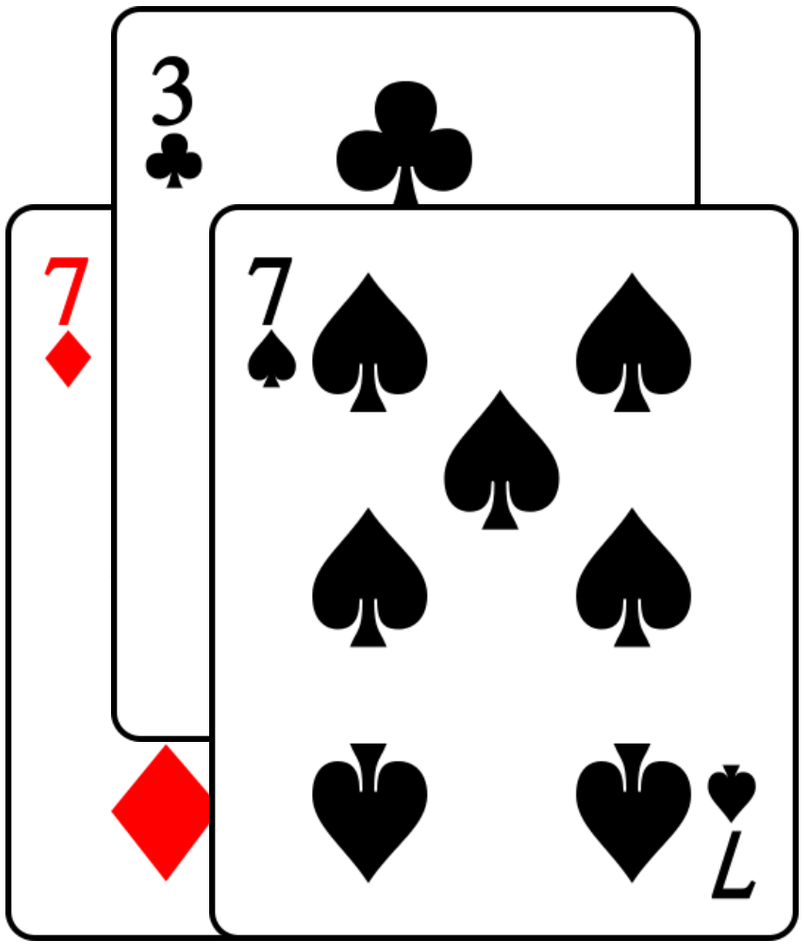
Tri par insertion



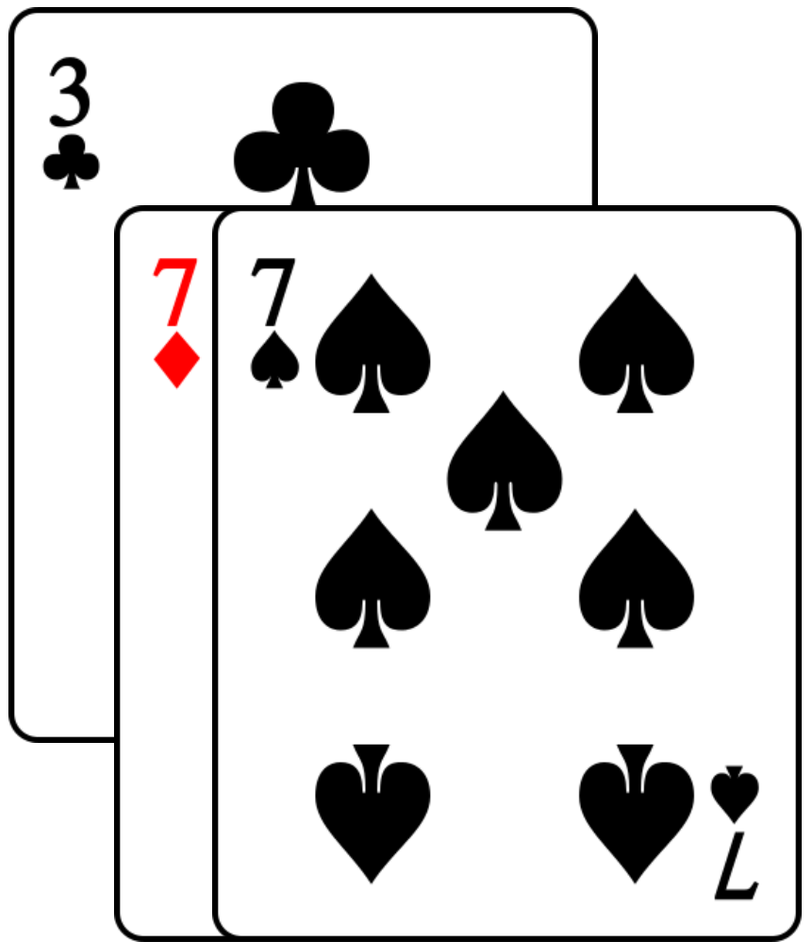
Tri par insertion



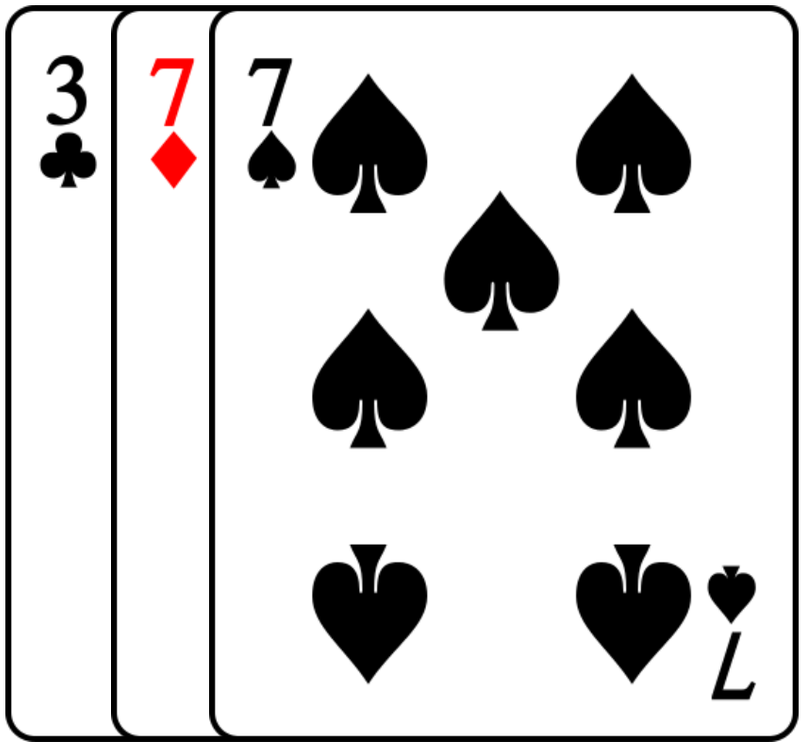
Tri par insertion



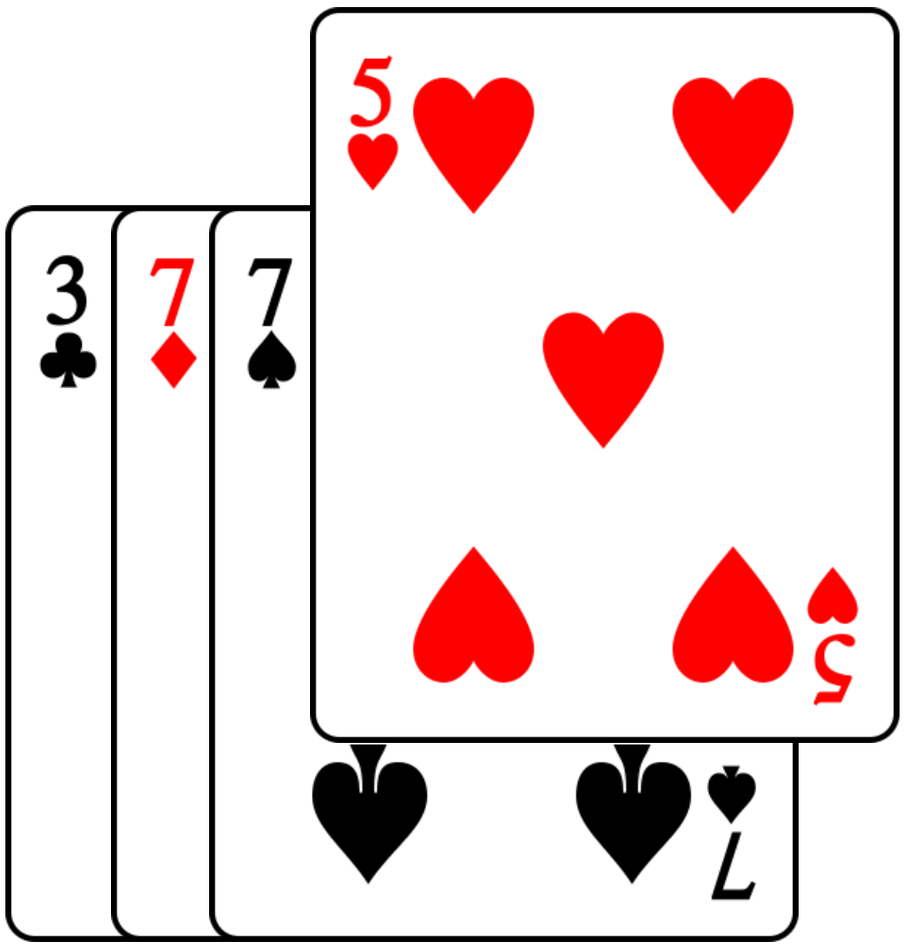
Tri par insertion



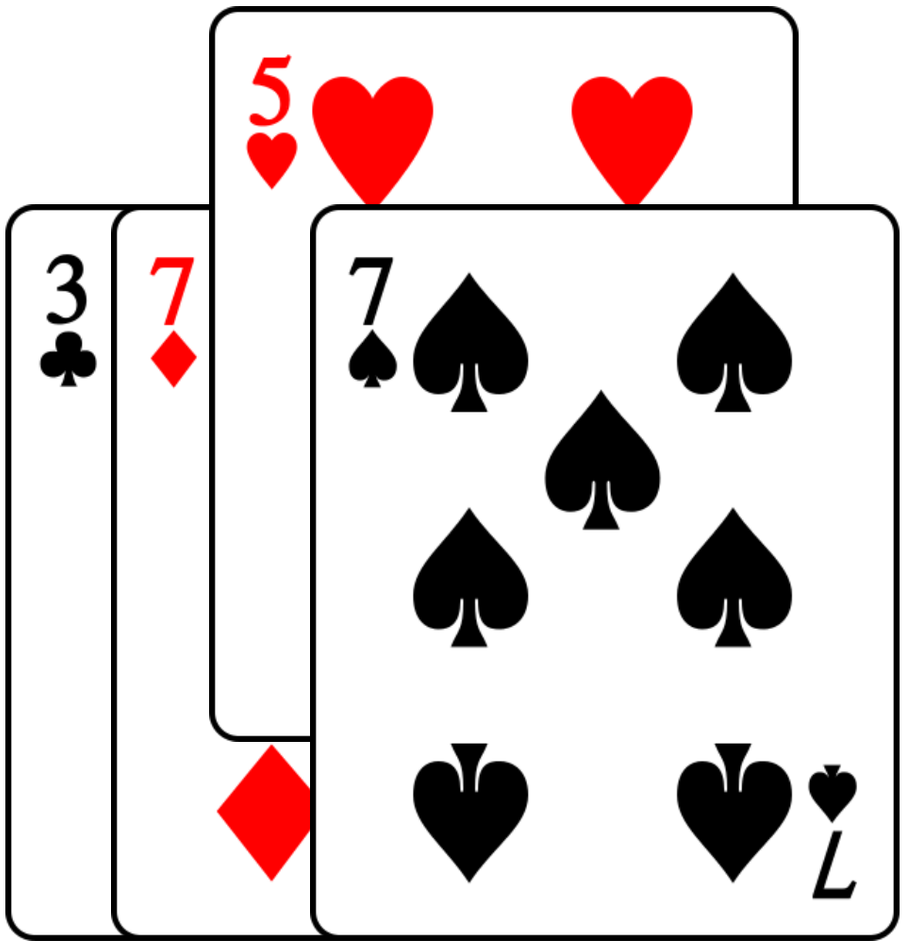
Tri par insertion



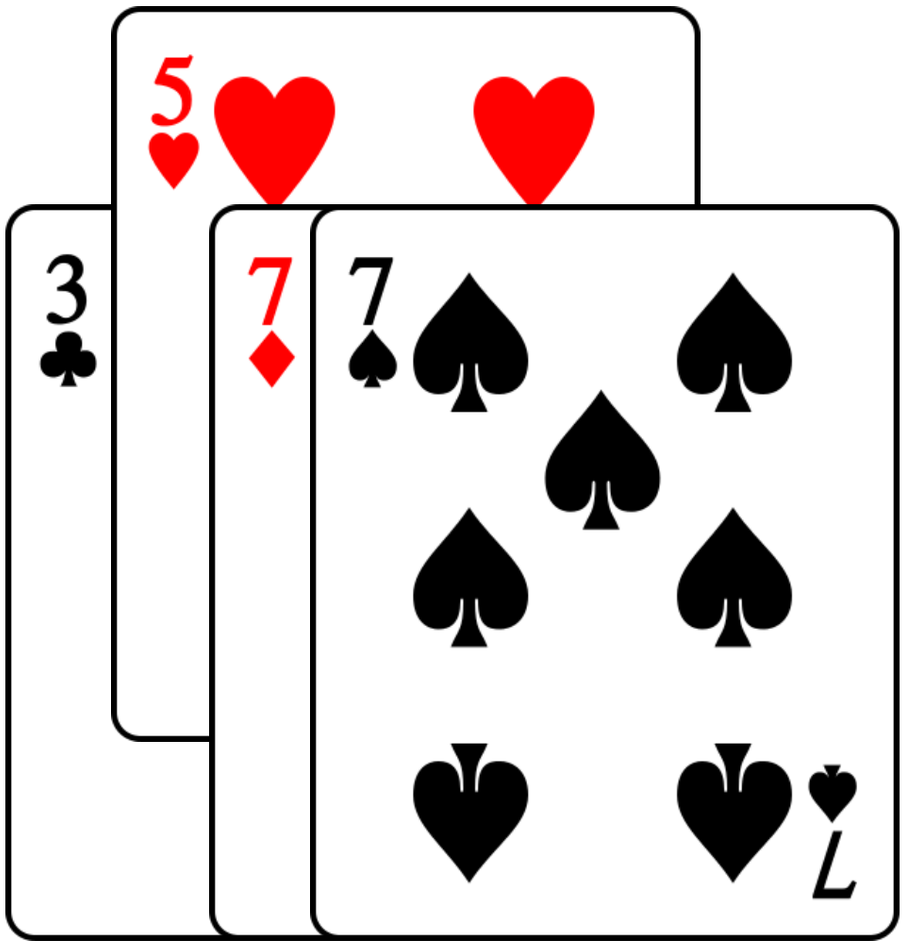
Tri par insertion



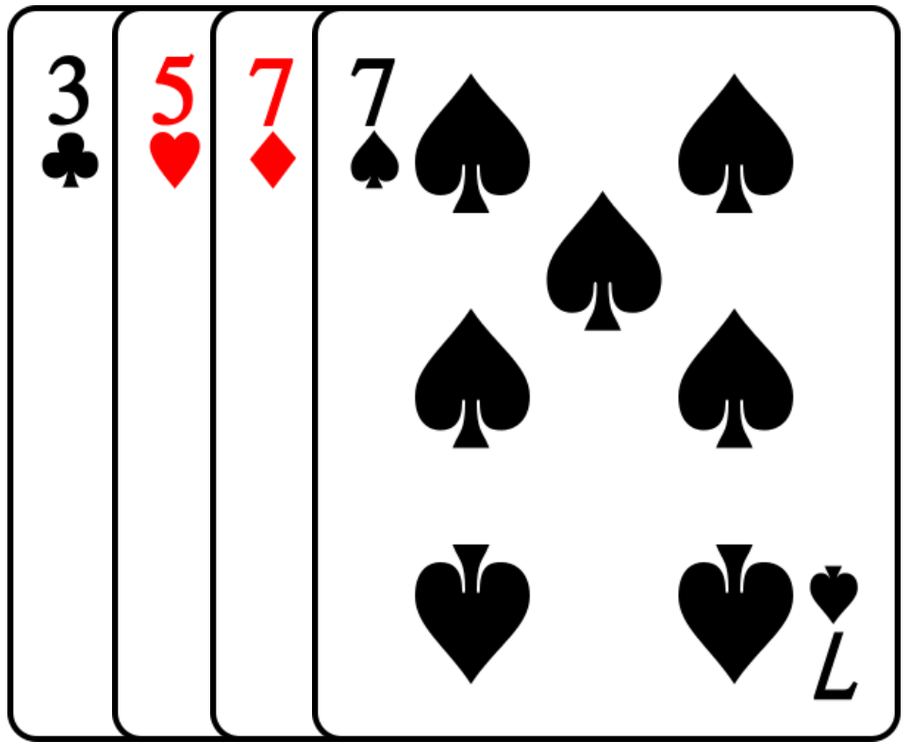
Tri par insertion



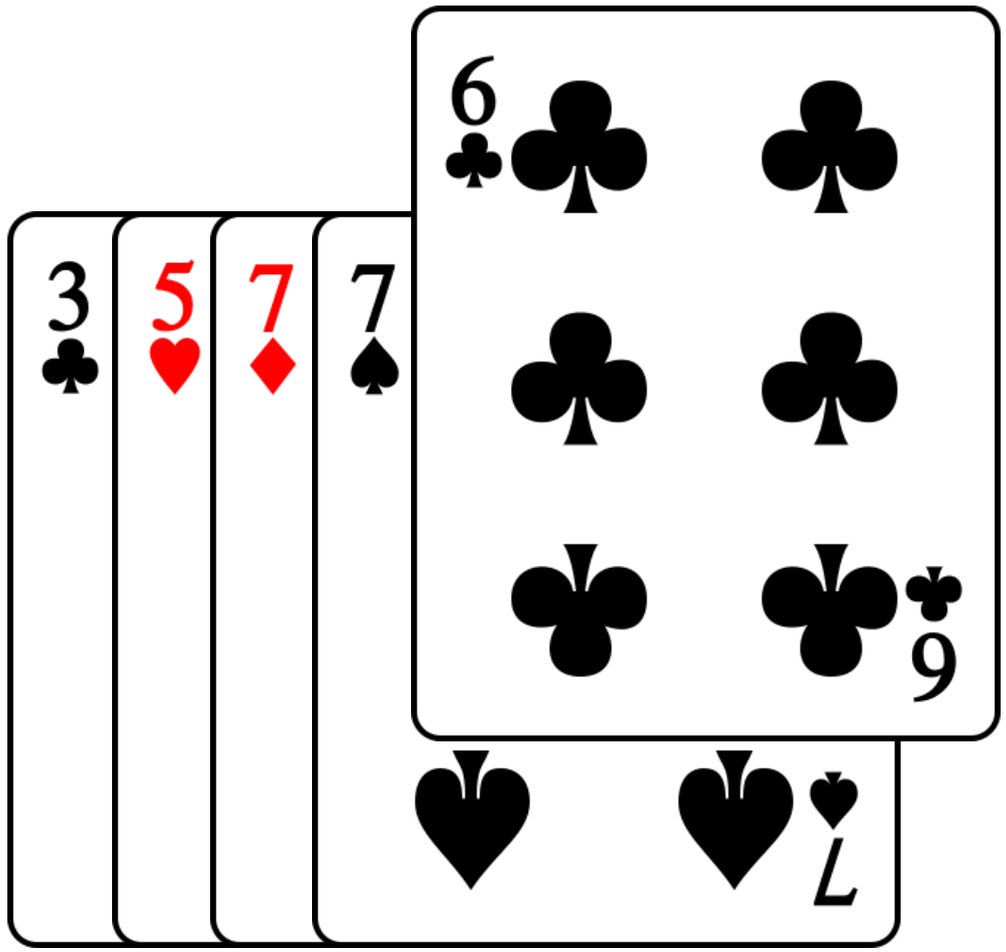
Tri par insertion



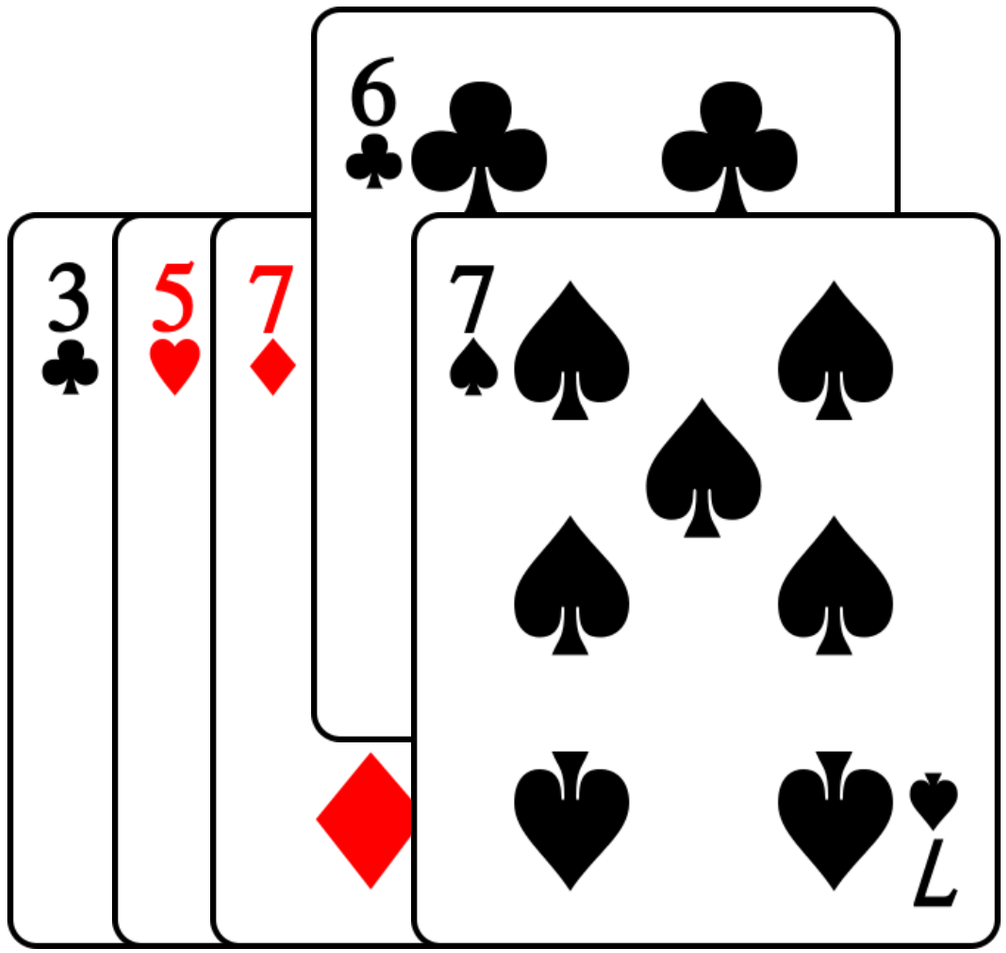
Tri par insertion



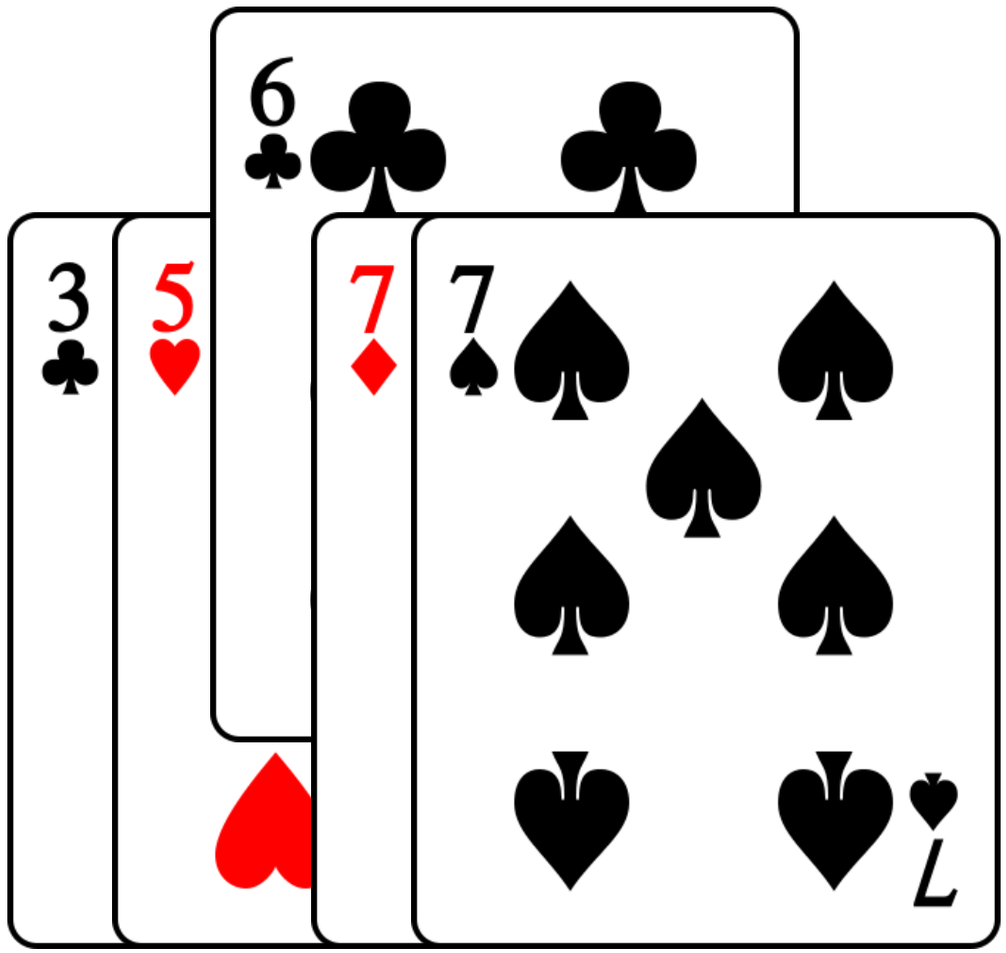
Tri par insertion



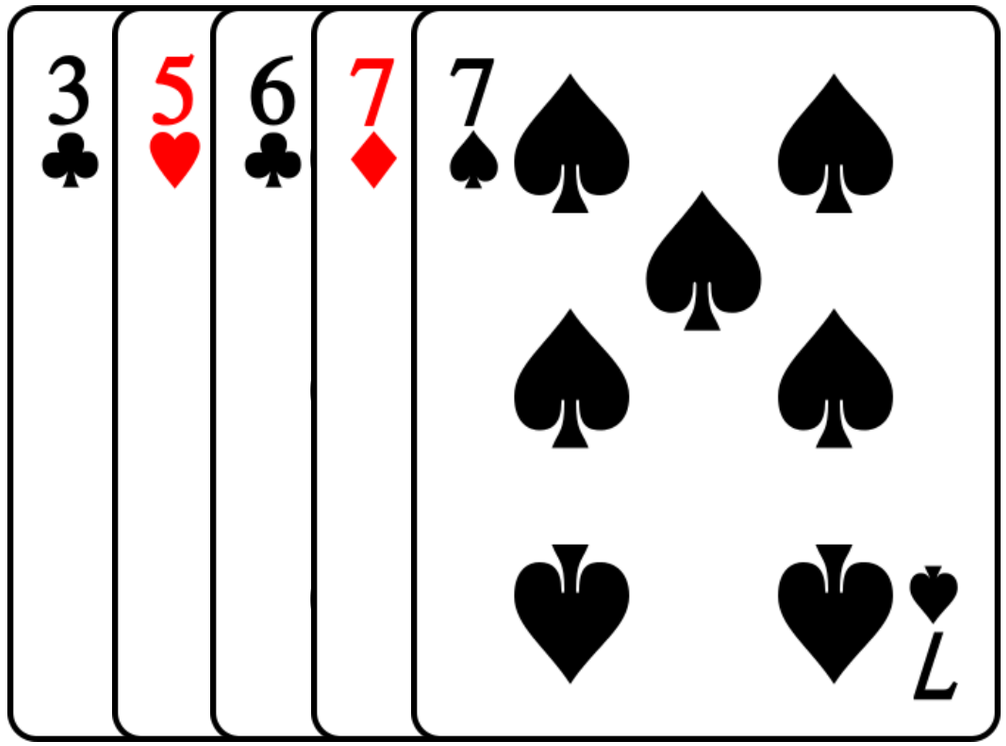
Tri par insertion



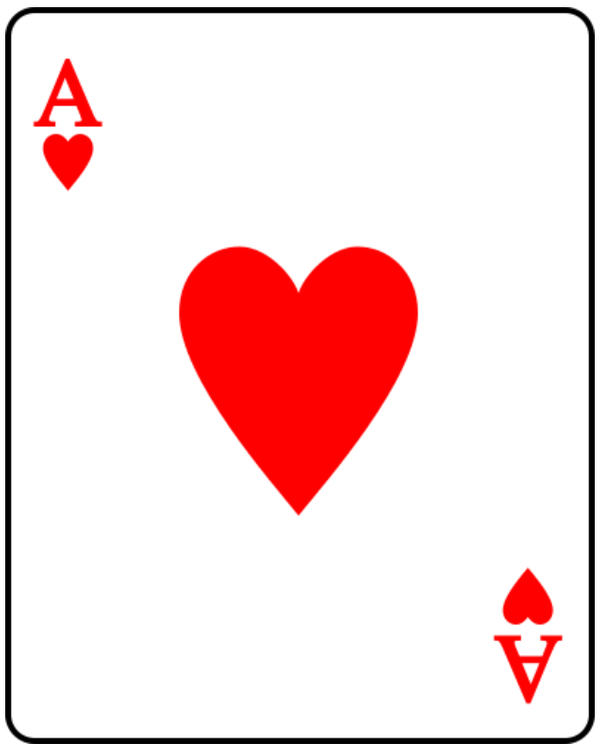
Tri par insertion



Tri par insertion

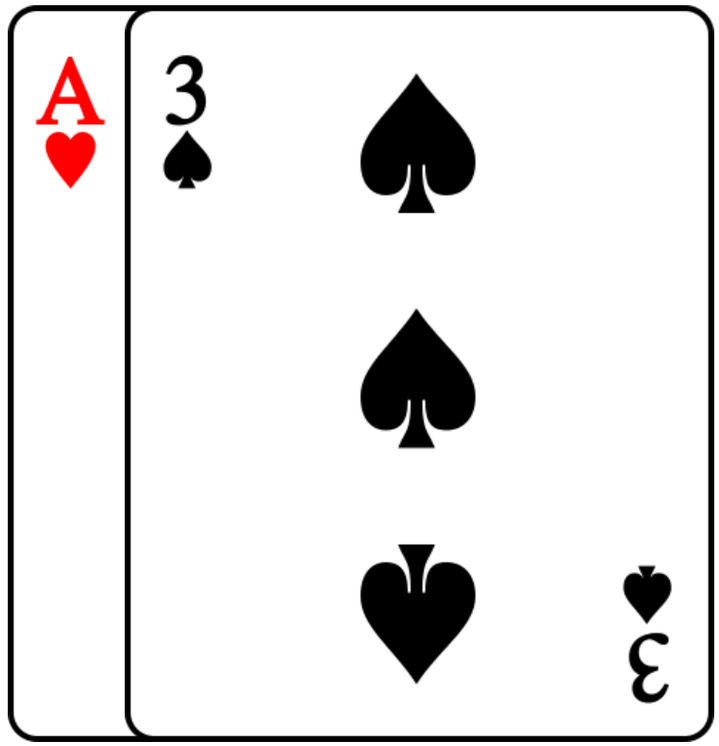


Le meilleur des cas



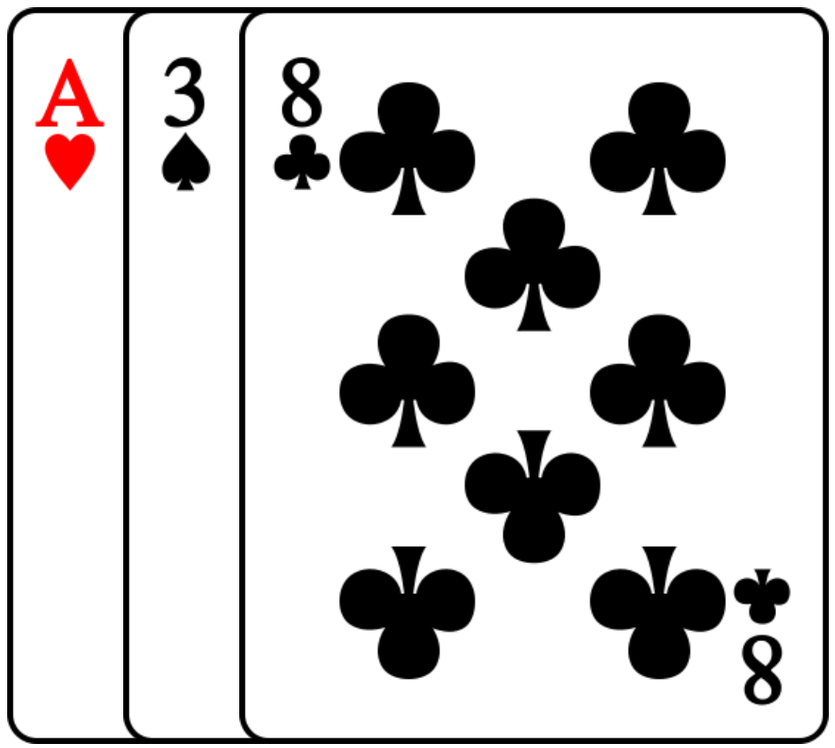
№ operations = 1

Le meilleur des cas



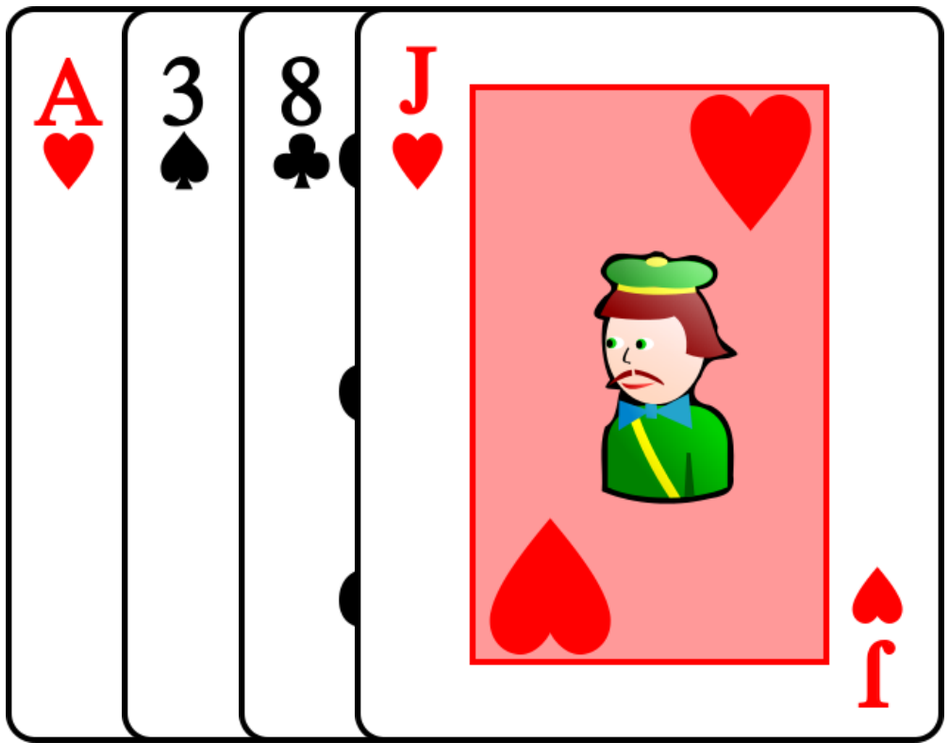
№ operations = 2

Le meilleur des cas



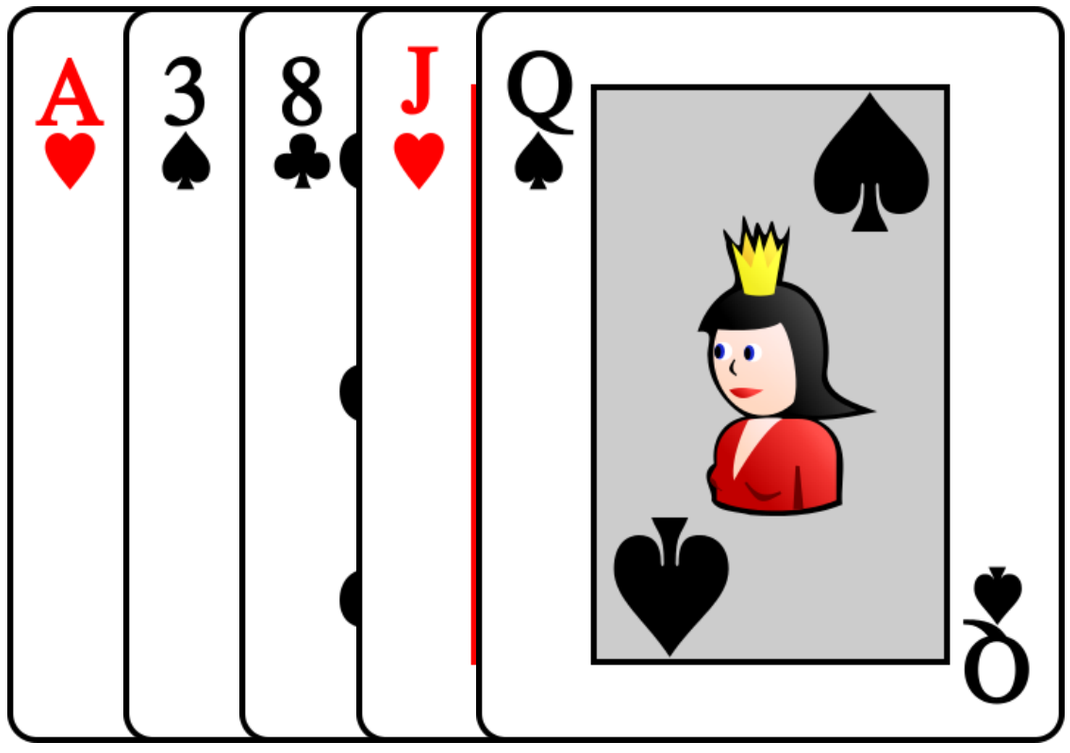
№ operations = 3

Le meilleur des cas



№ operations = 4

Le meilleur des cas



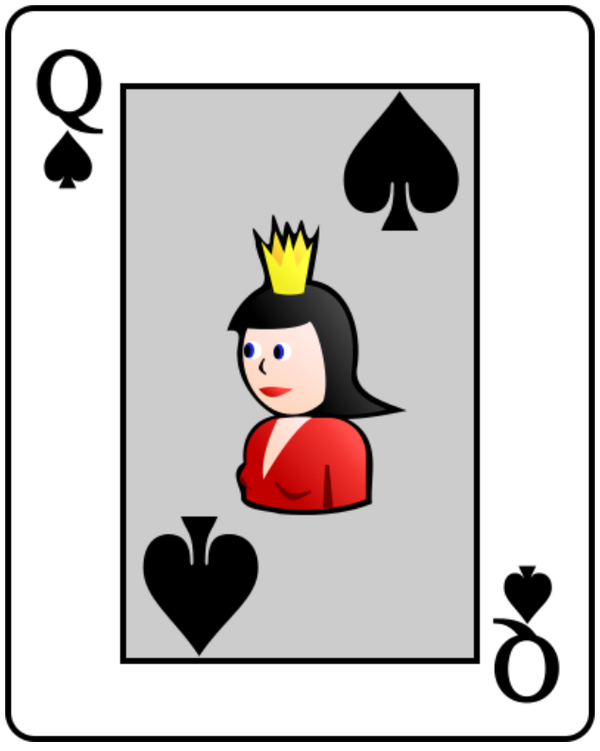
№ operations = 5

Le meilleur des cas

- Les cartes arrivent déjà triées

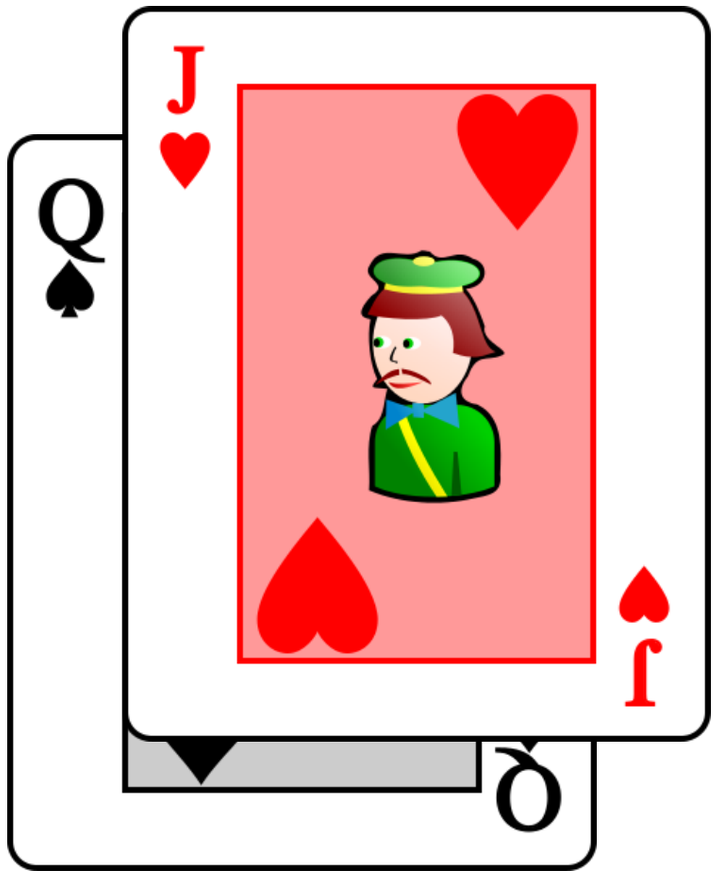
- On fait n opérations (déplacements de cartes)

Le pire des cas



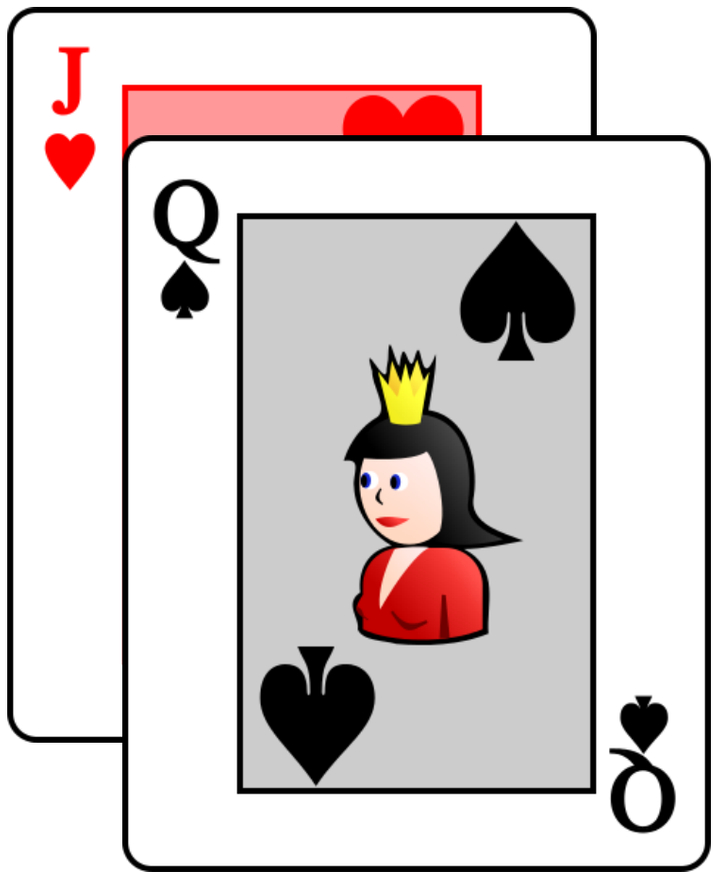
№ operations = 1

Le pire des cas



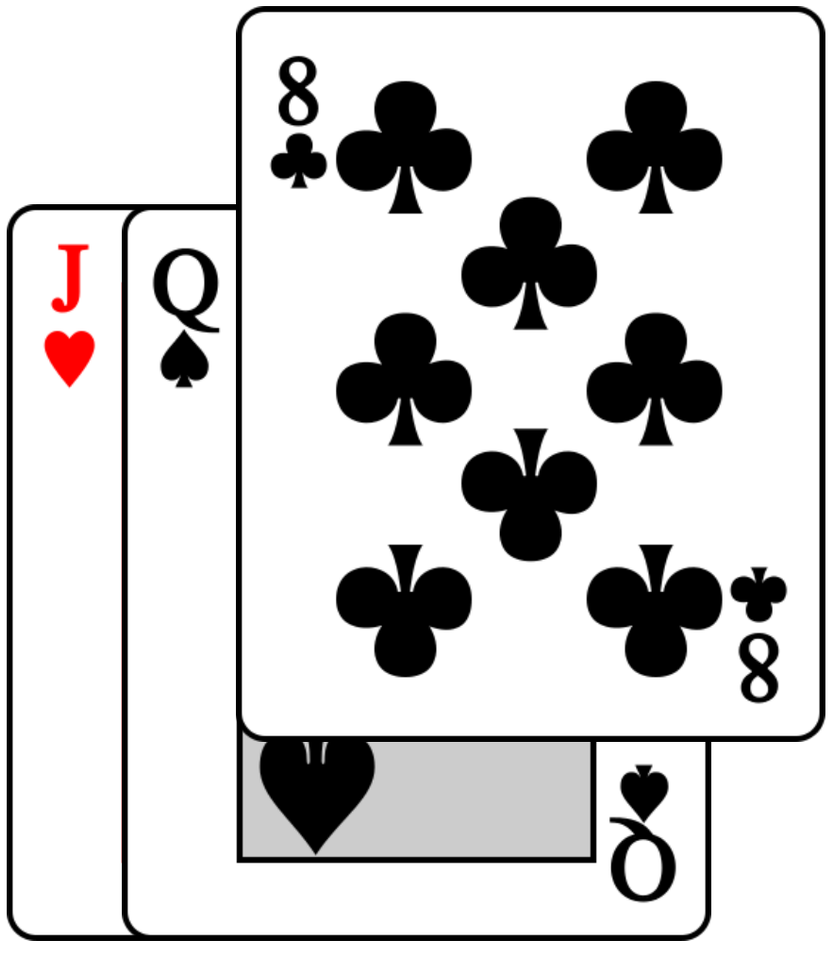
№ operations = 1 + 1

Le pire des cas



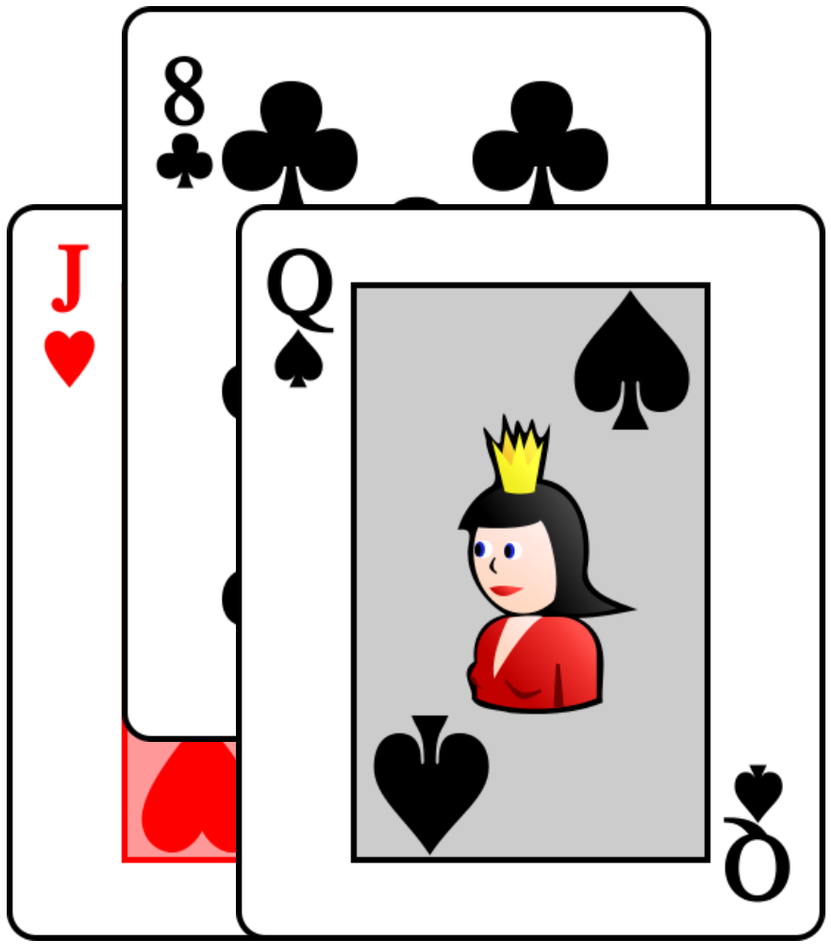
№ operations = 1 + 2

Le pire des cas



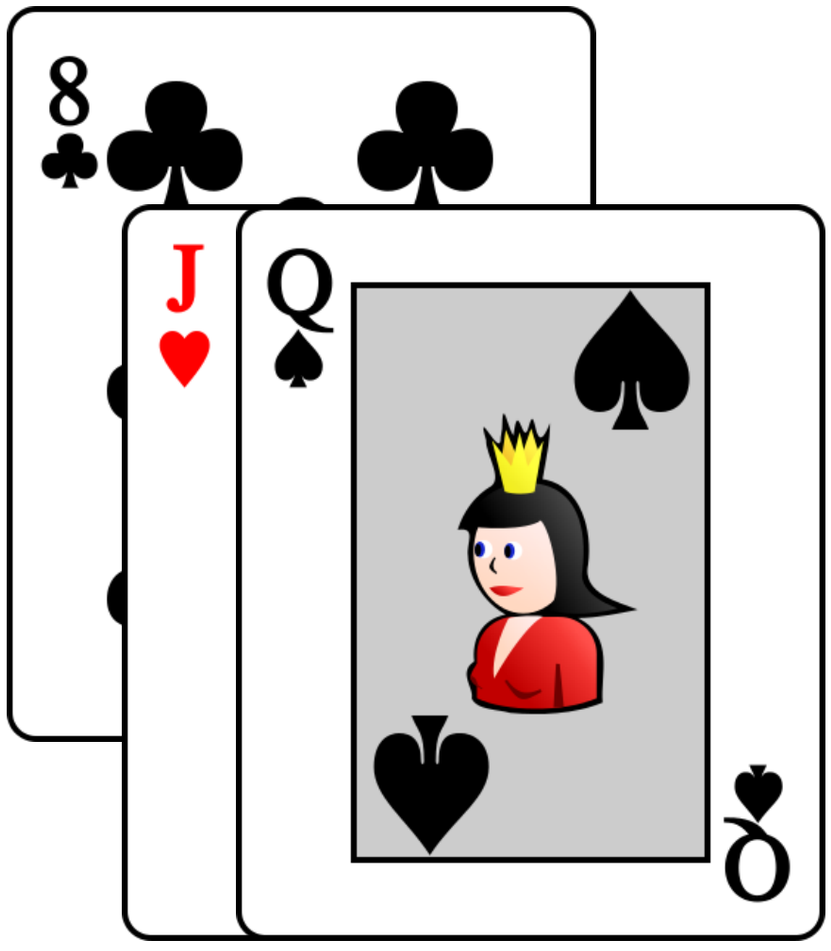
№ operations = 1 + 2 + 1

Le pire des cas



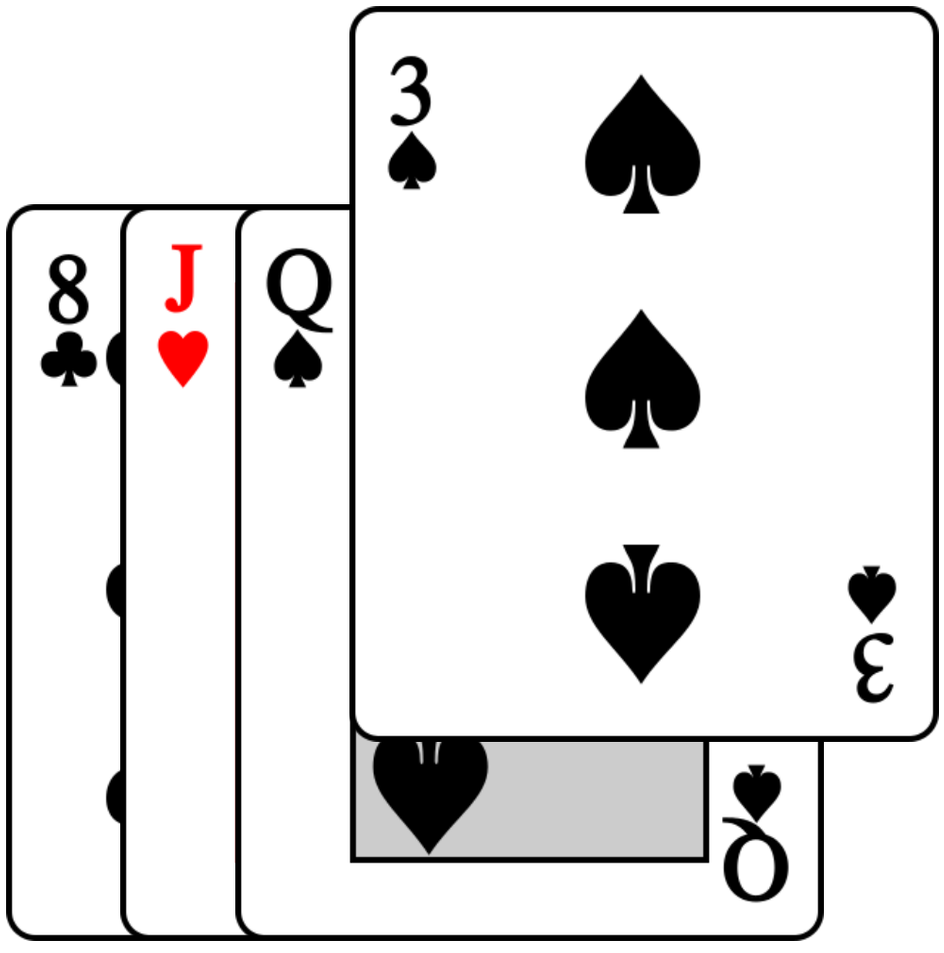
№ operations = 1 + 2 + 2

Le pire des cas



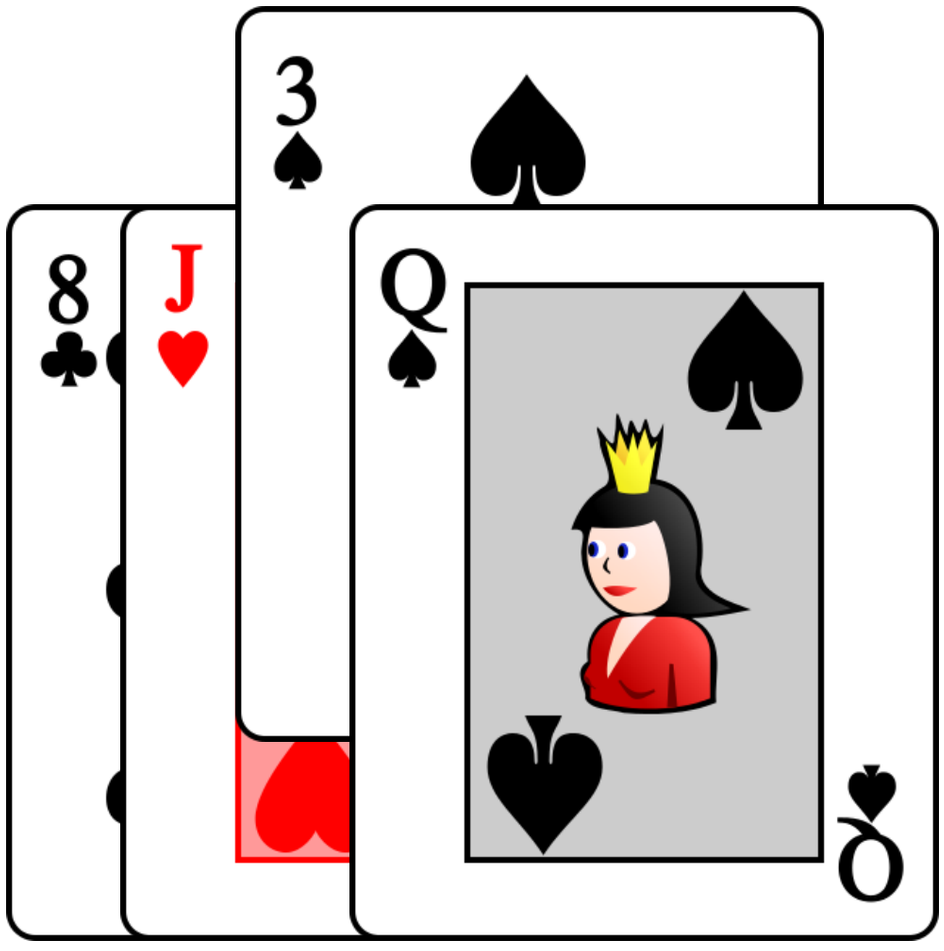
№ operations = 1 + 2 + 3

Le pire des cas



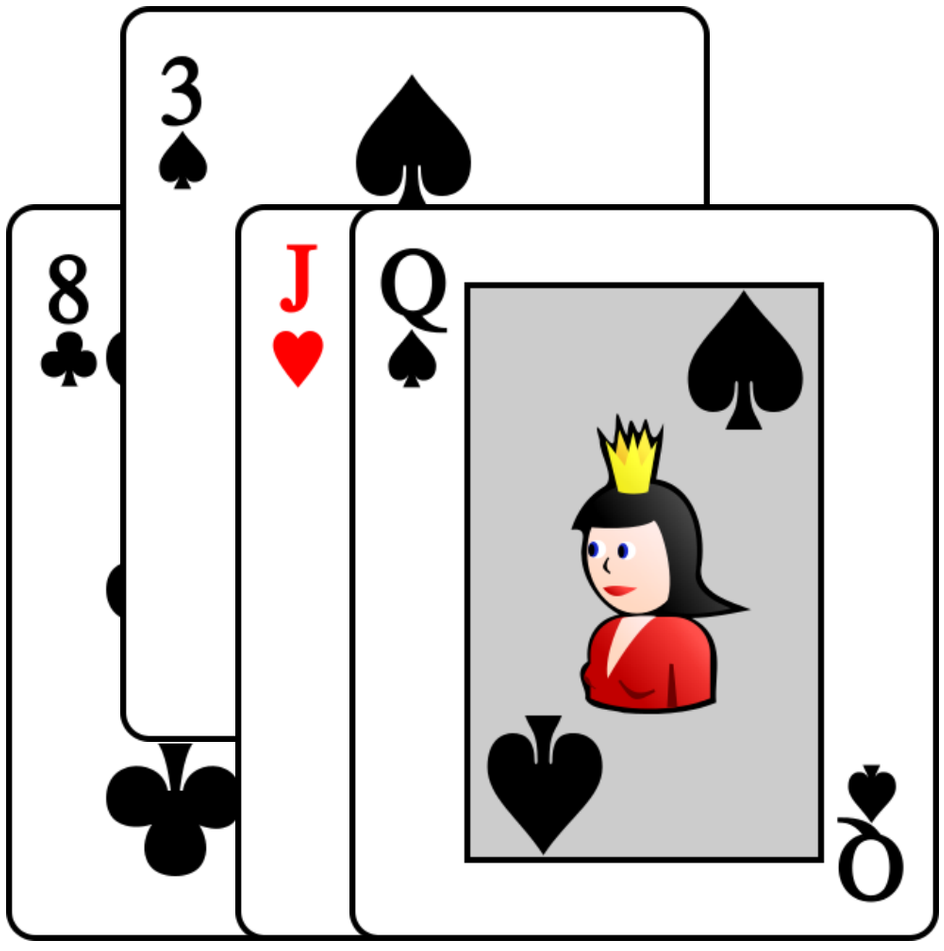
№ operations = 1 + 2 + 3 + 1

Le pire des cas



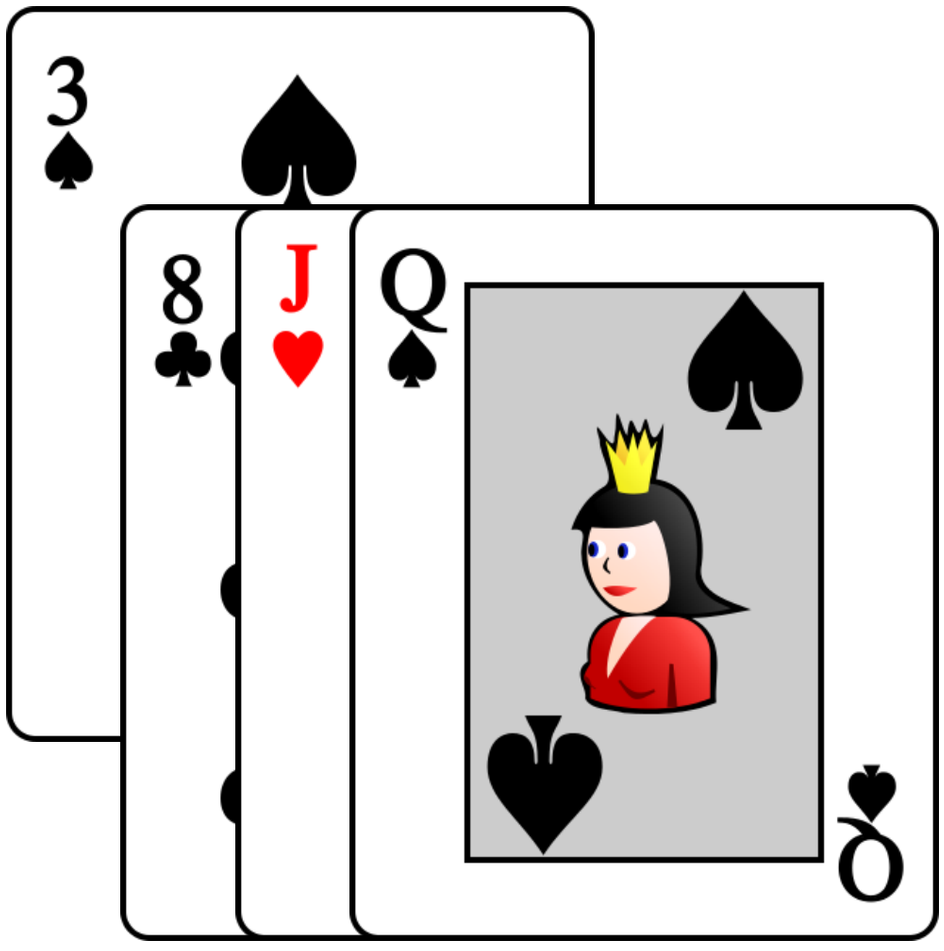
№ operations = 1 + 2 + 3 + 2

Le pire des cas



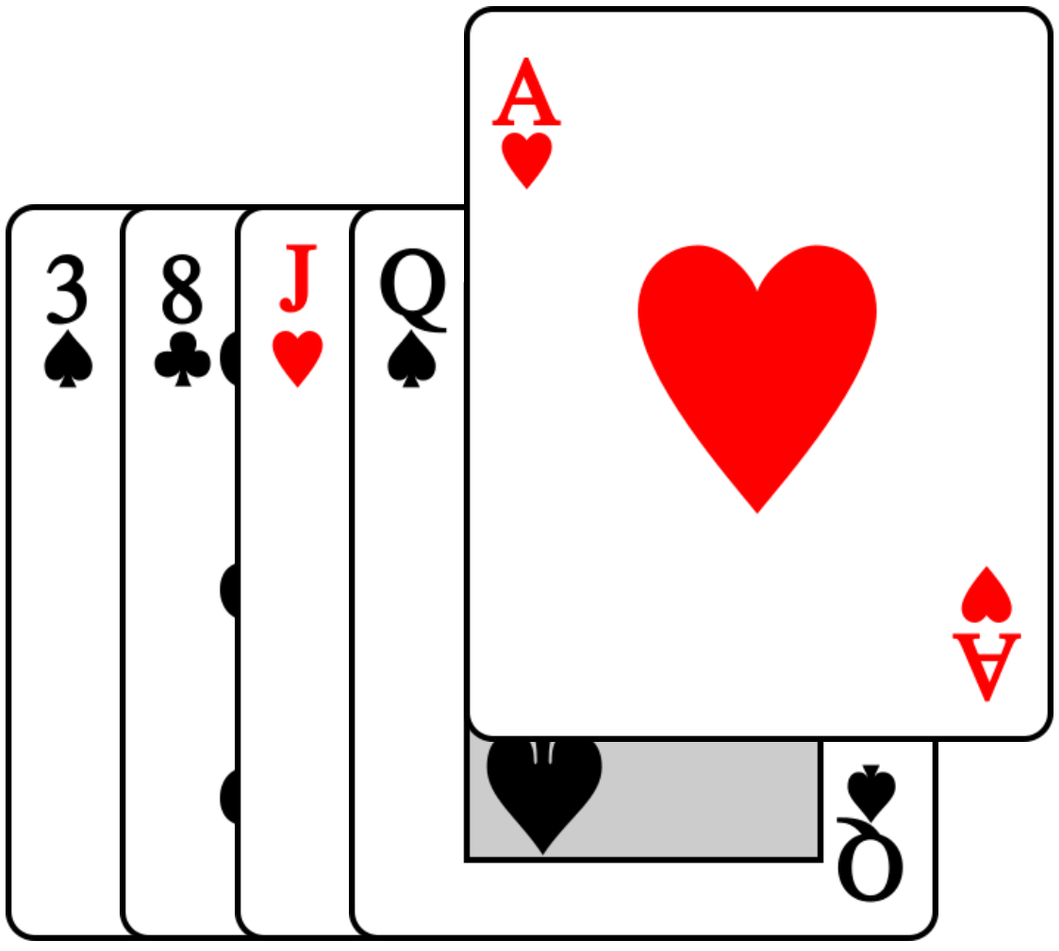
№ operations = 1 + 2 + 3 + 3

Le pire des cas



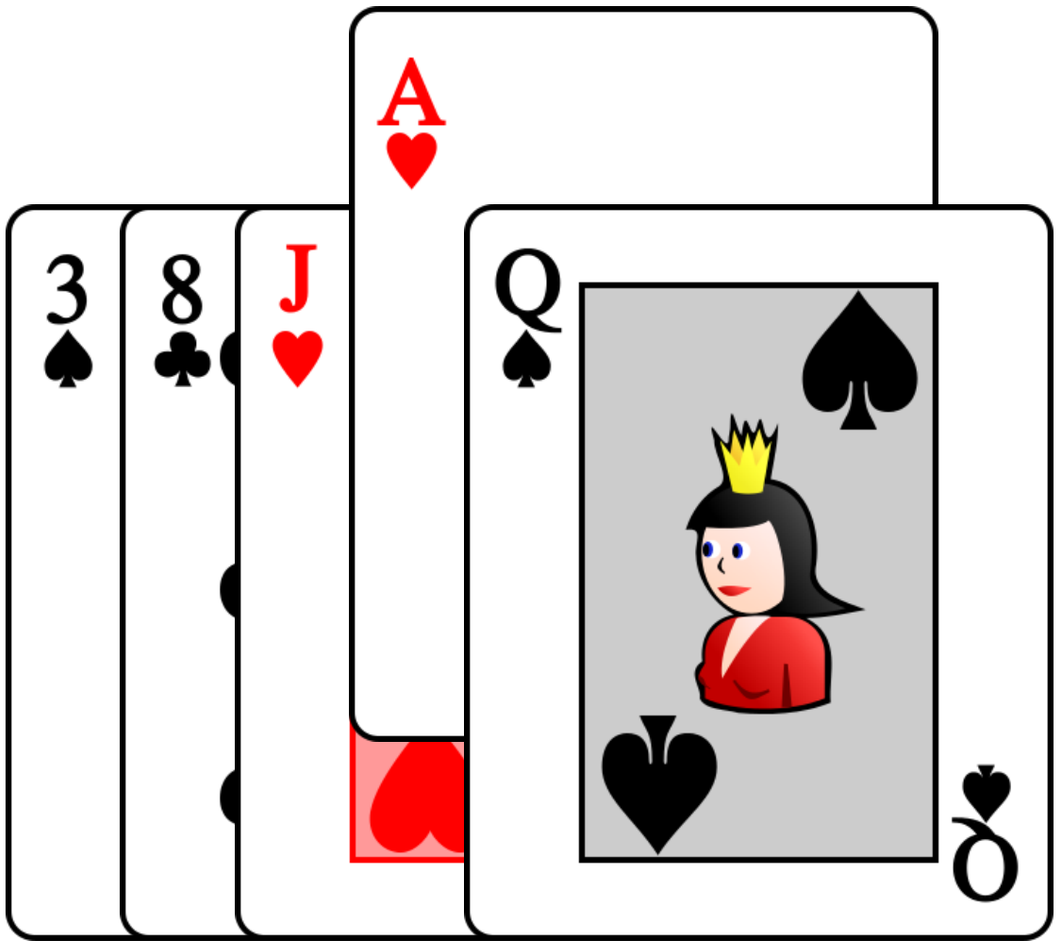
№ operations = 1 + 2 + 3 + 4

Le pire des cas



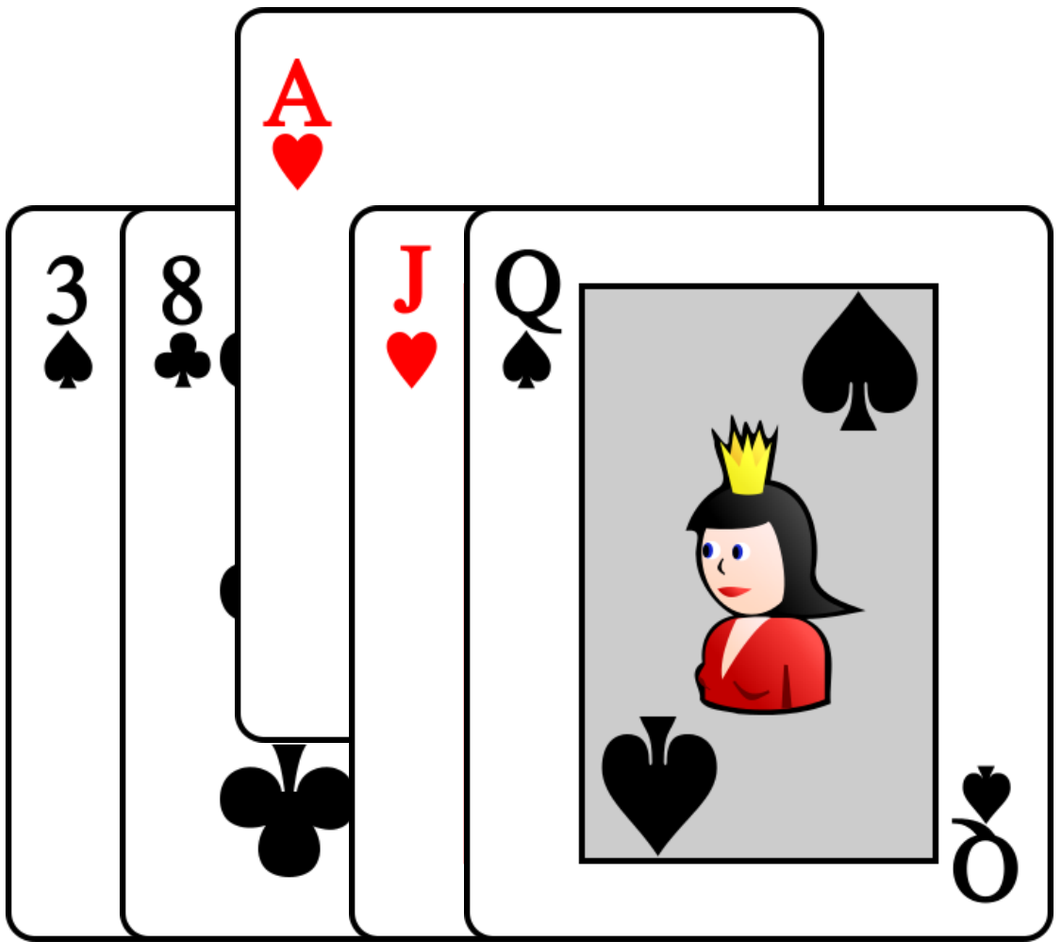
№ operations = 1 + 2 + 3 + 4 + 1

Le pire des cas



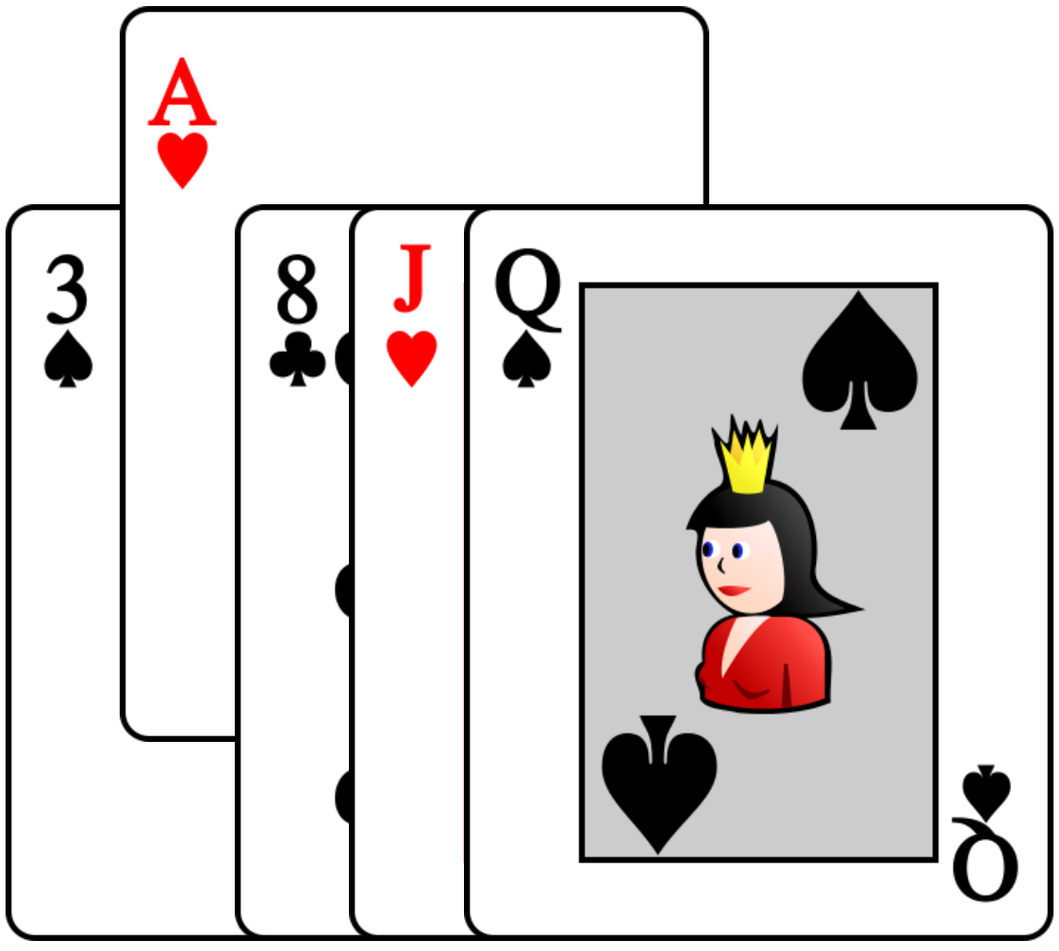
№ operations = 1 + 2 + 3 + 4 + 2

Le pire des cas



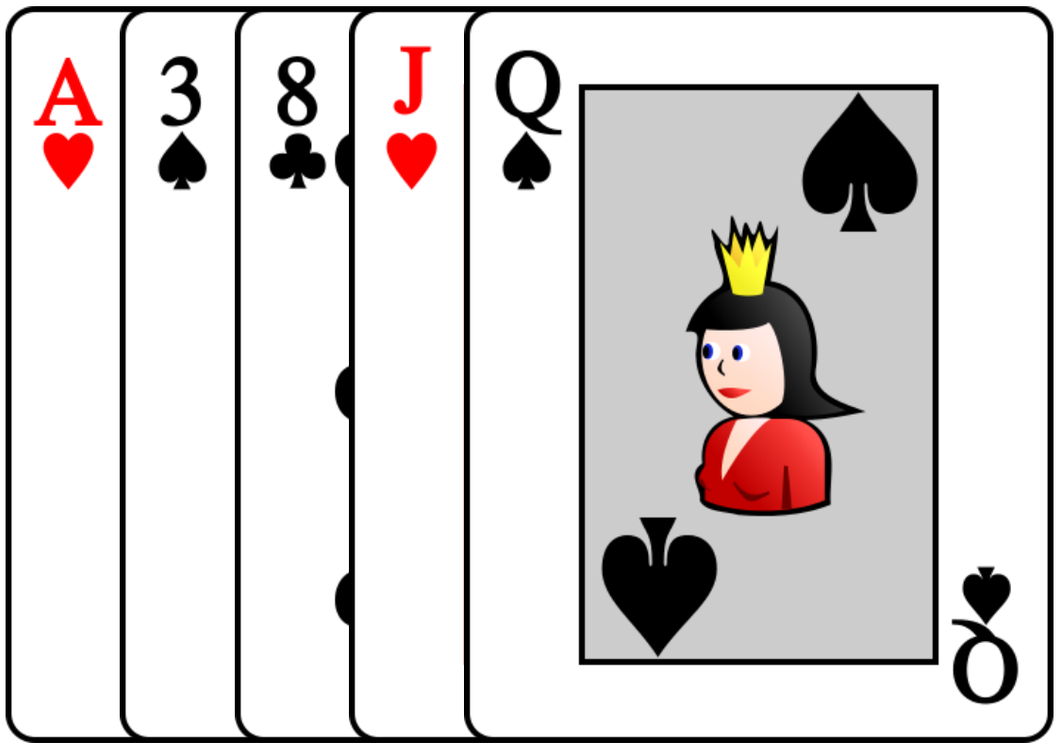
№ operations = 1 + 2 + 3 + 4 + 3

Le pire des cas



№ operations = 1 + 2 + 3 + 4 + 4

Le pire des cas



№ operations = 1 + 2 + 3 + 4 + 5

Le pire des cas

- Les cartes arrivent en ordre décroissant
- On fait i opérations pour la i-ème carte
- Le nombre totale est 1 + 2 + 3 + ⋯ + n

$$\sum_{i=1}^{n} i = \frac{1}{2}n(n+1) = \frac{1}{2}(n^2+n) = \frac{1}{2}n^2 + \frac{1}{2}n \in O(n^2)$$

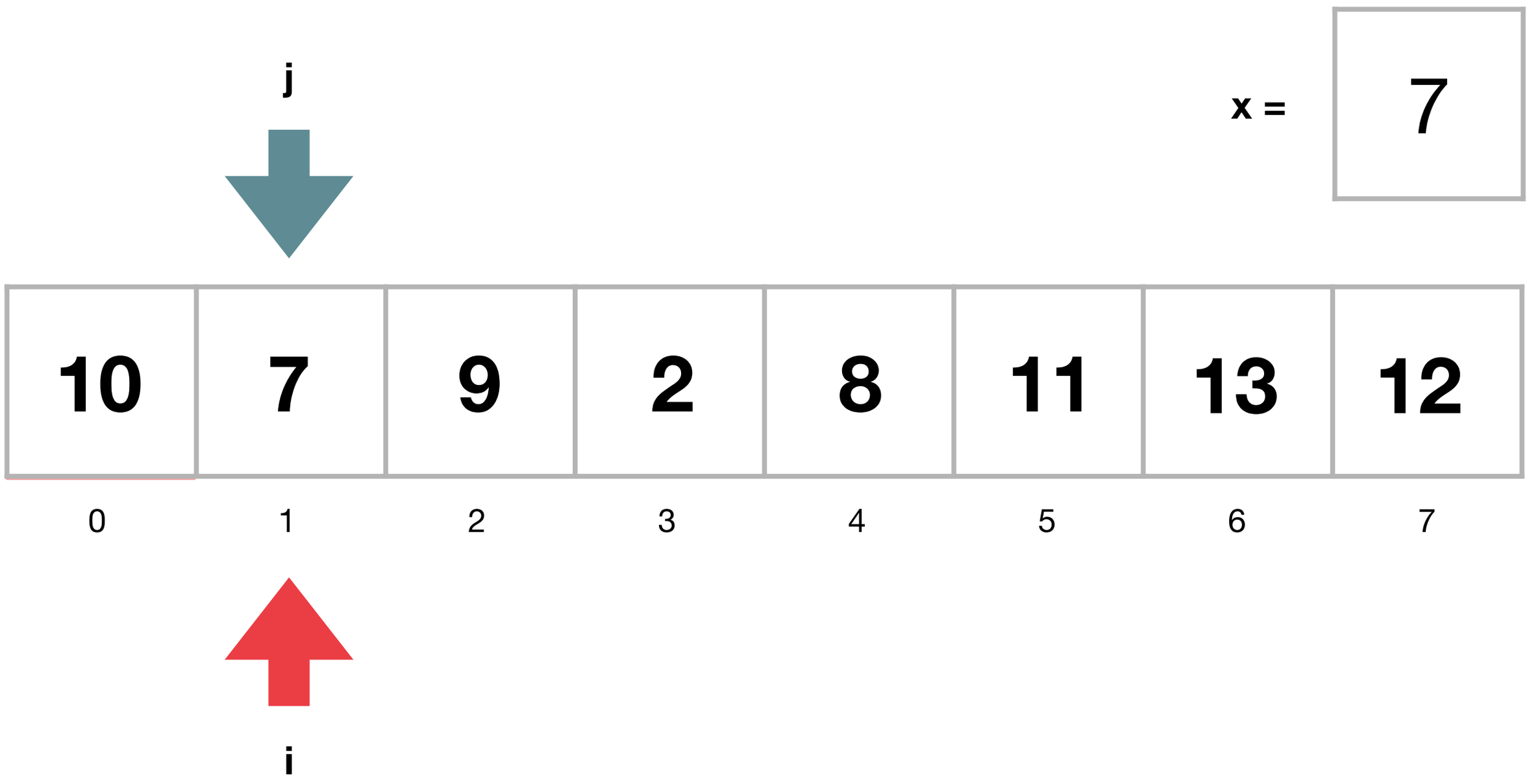
Tri d’un tableau par insertion

```
procedure trier-par-insertion(T)
    n ≔ longueur(T)
    pour i ≔ 1 à n – 1 faire (on saute le 0)
        x ≔ T[i]
        j ≔ i
        tant que j > 0 et x < T[j – 1] faire
            (décaler d’un élément)
            T[j] ≔ T[j – 1]
            j ≔ j – 1
        fin tant que
        (ici x ≥ T[j – 1] ou bien j = 0)
        T[j] ≔ x
    fin pour
fin procedure
```

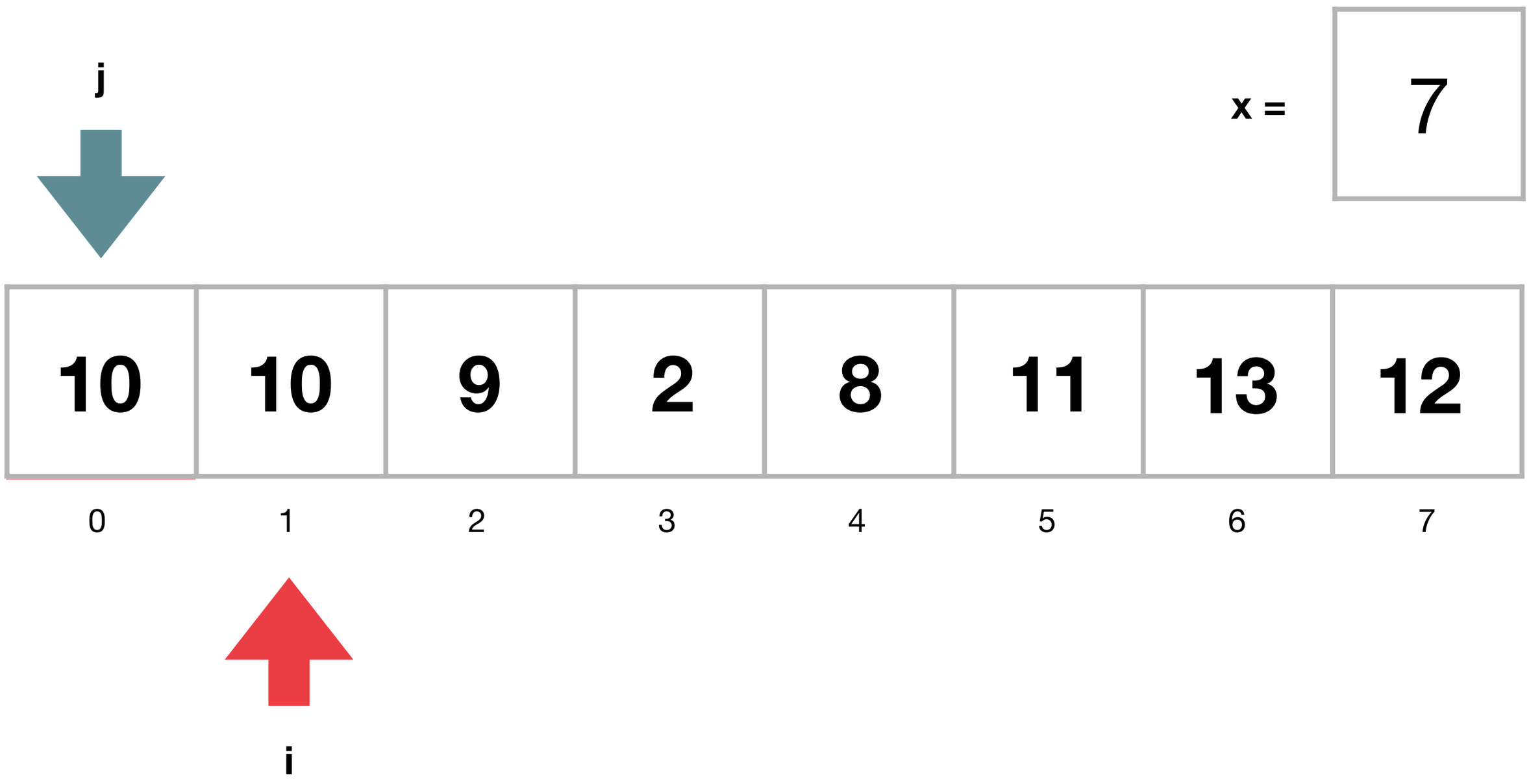
Tri par insertion

10	7	9	2	8	11	13	12
0	1	2	3	4	5	6	7

Tri par insertion



Tri par insertion



Tri par insertion

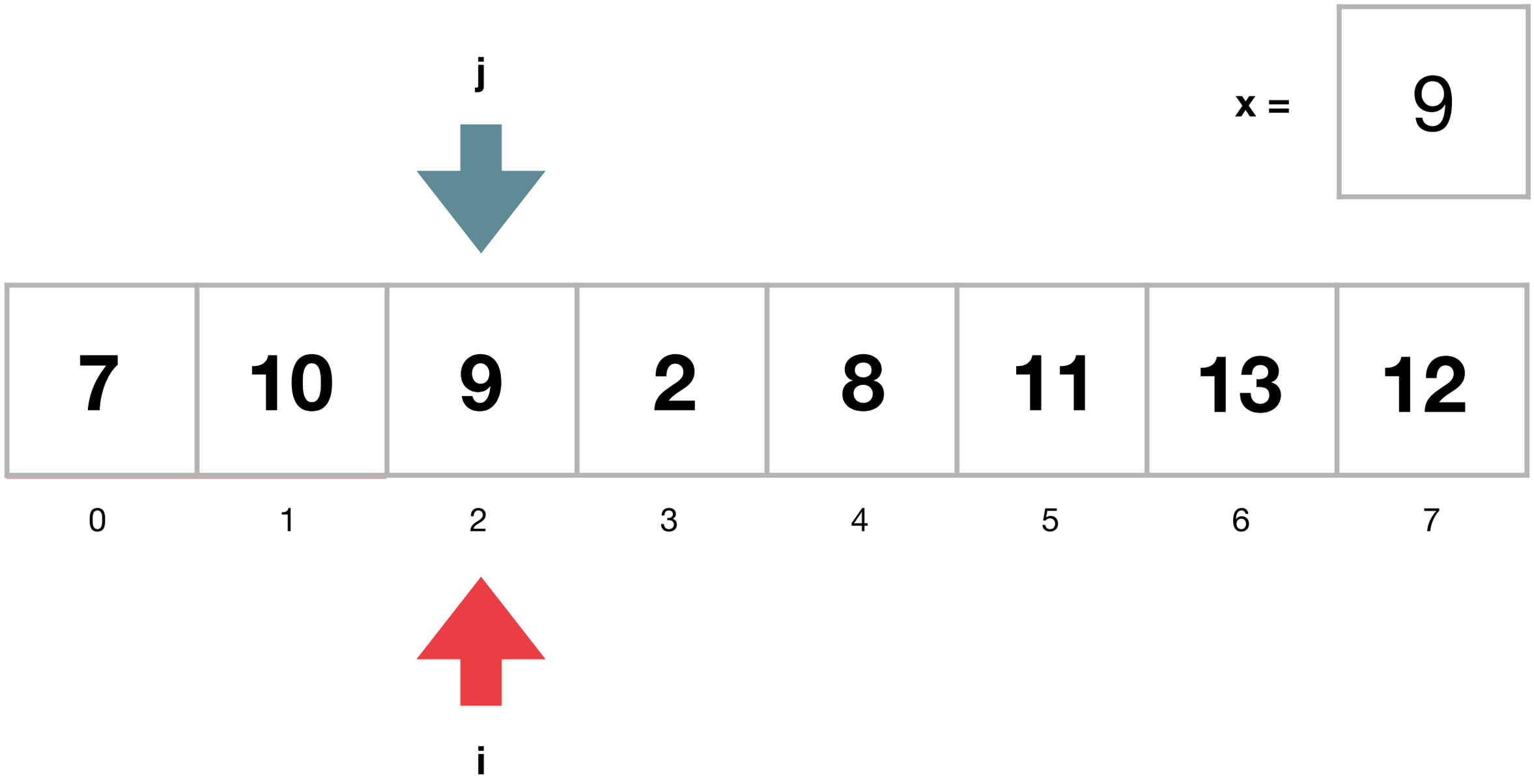
j

x = 7

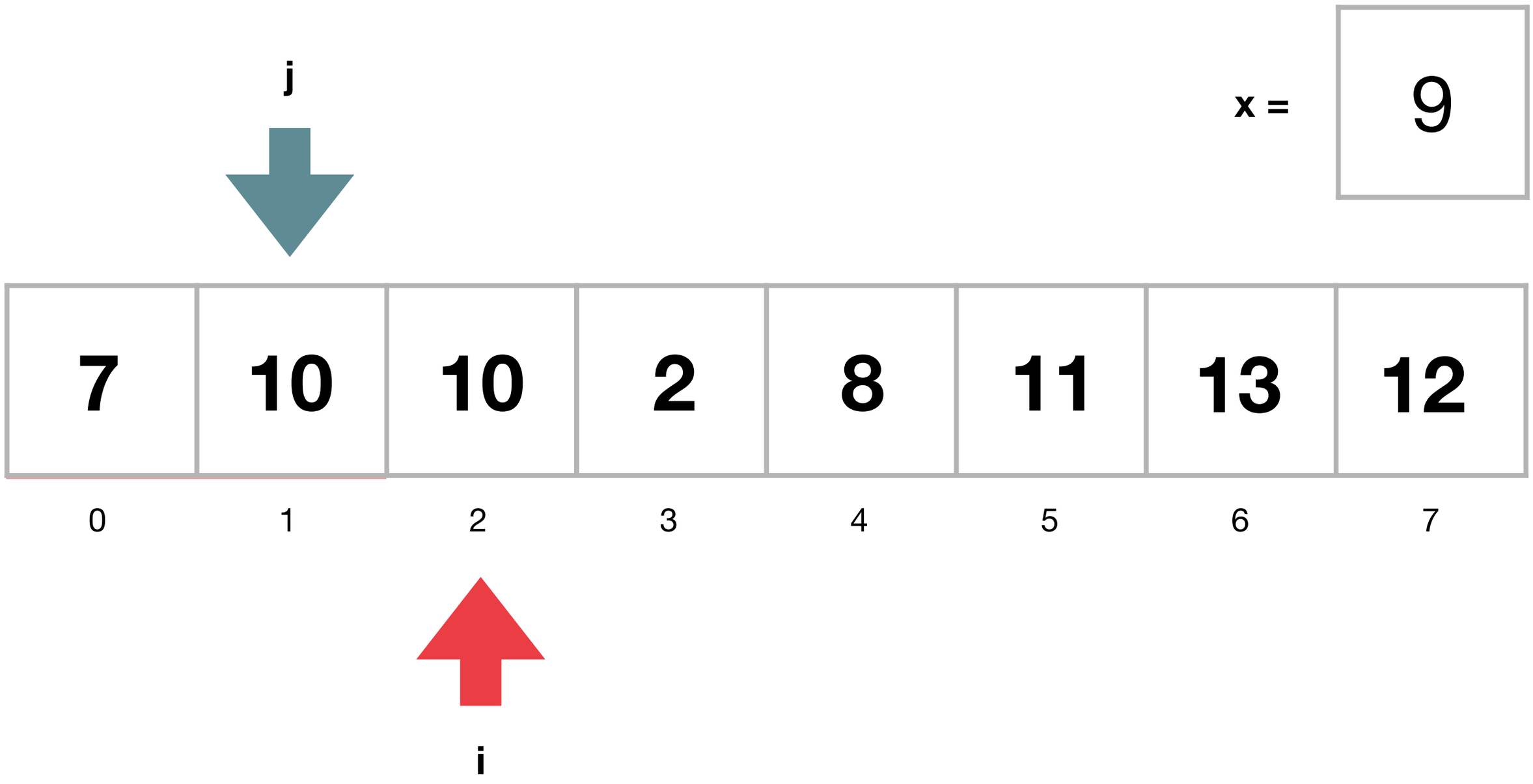
7	10	9	2	8	11	13	12
0	1	2	3	4	5	6	7

i

Tri par insertion



Tri par insertion



Tri par insertion

j

x = 9

7	9	10	2	8	11	13	12
0	1	2	3	4	5	6	7

i

Tri par insertion

j

x = 2

7	9	10	2	8	11	13	12
0	1	2	3	4	5	6	7

i

Tri par insertion

j

x = 2

7	9	10	10	8	11	13	12
0	1	2	3	4	5	6	7

i

Tri par insertion

j

x = 2

7	9	9	10	8	11	13	12
0	1	2	3	4	5	6	7

i

Tri par insertion

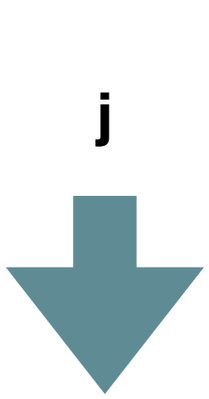
j

x = 2

7	7	9	10	8	11	13	12
0	1	2	3	4	5	6	7

i

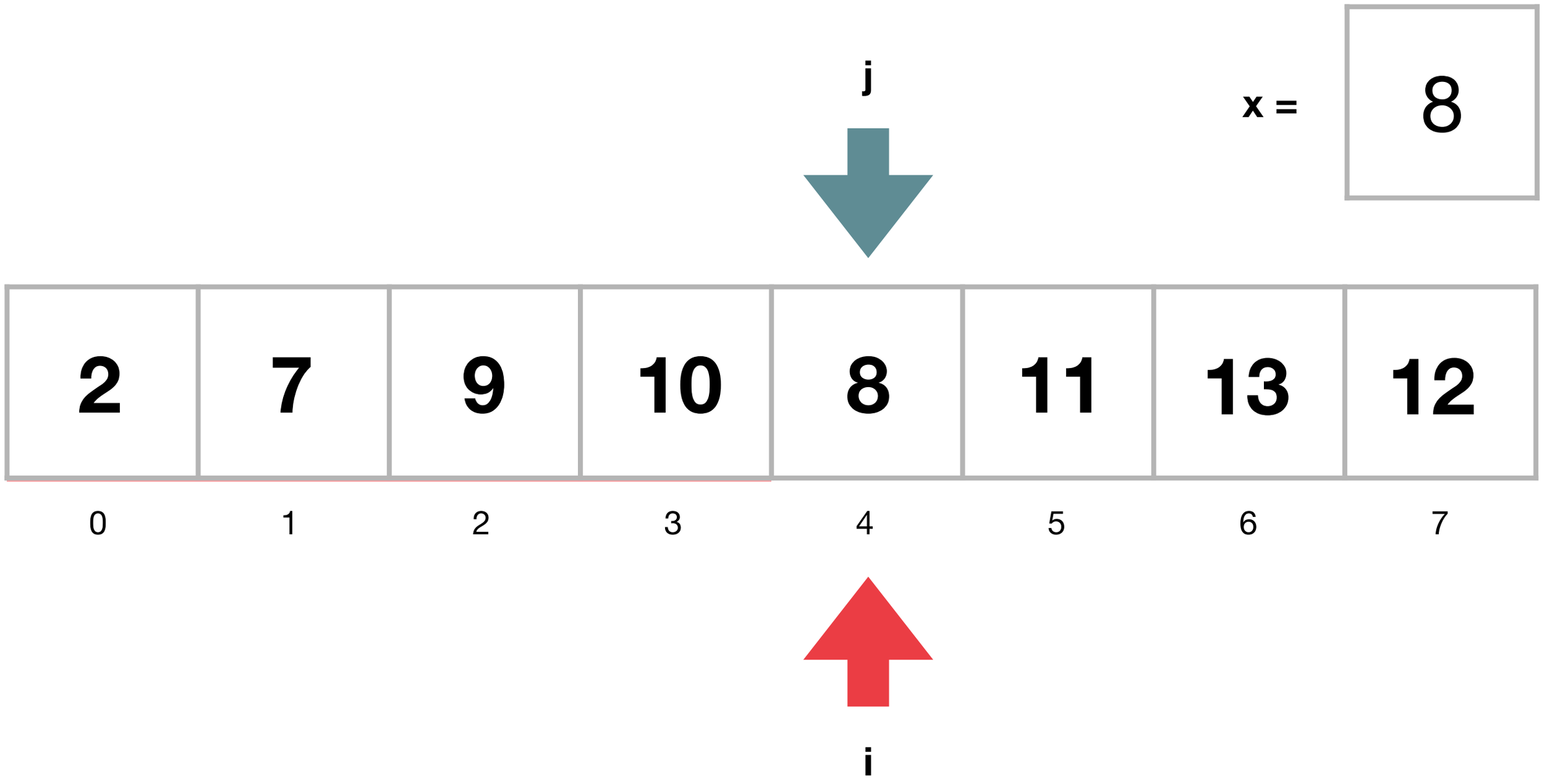
Tri par insertion



x = 2

2	7	9	10	8	11	13	12
0	1	2	3	4	5	6	7

Tri par insertion



Tri par insertion

j

x = 8

2	7	9	10	10	11	13	12
0	1	2	3	4	5	6	7

i

Tri par insertion

j

x = 8

2	7	9	9	10	11	13	12
0	1	2	3	4	5	6	7

i

Tri par insertion

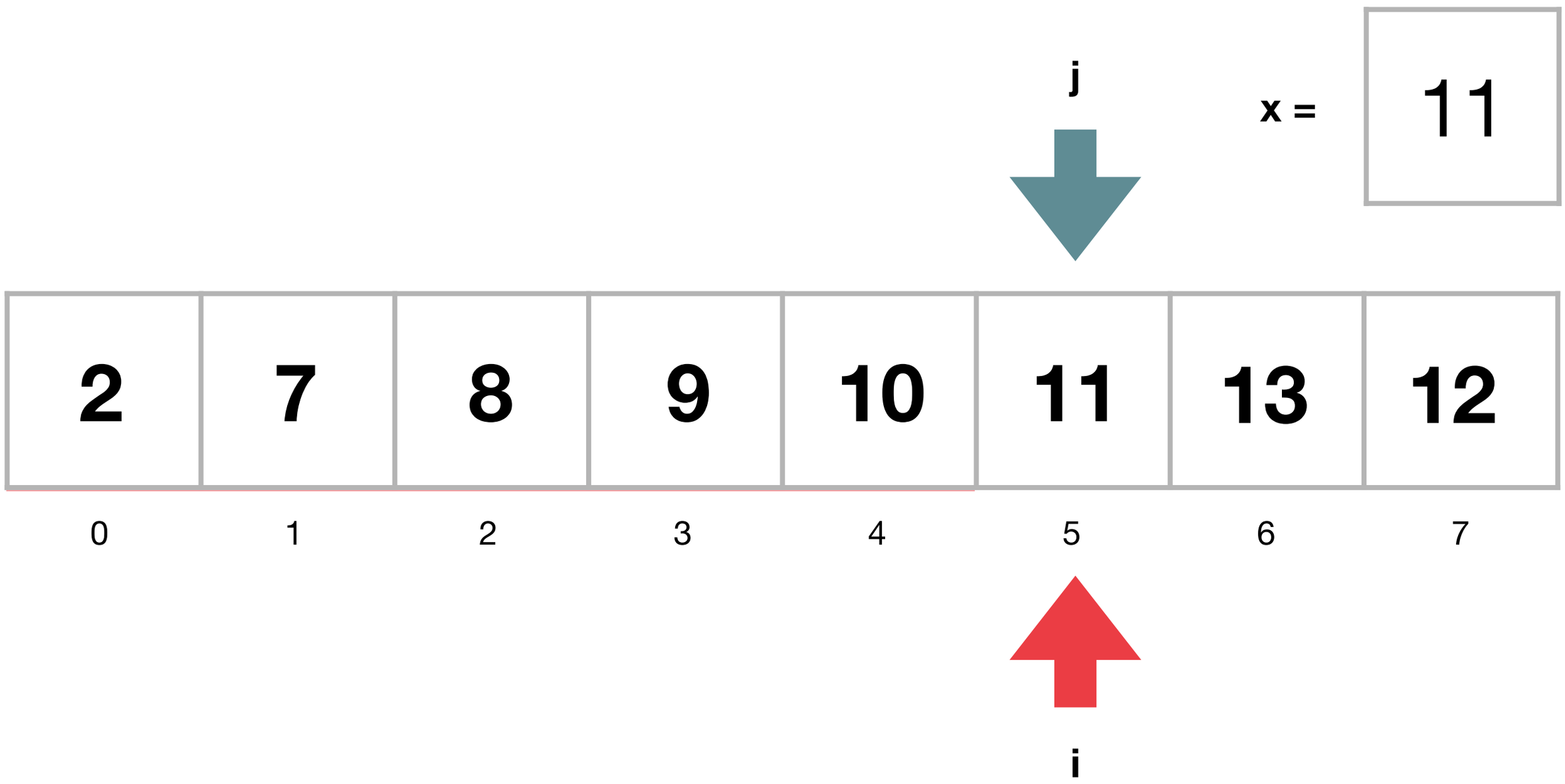
j

x = 8

2	7	8	9	10	11	13	12
0	1	2	3	4	5	6	7

i

Tri par insertion



Tri par insertion

x = 13

j

2	7	8	9	10	11	13	12
0	1	2	3	4	5	6	7

i

Tri par insertion

x = 12

j

2	7	8	9	10	11	13	12
0	1	2	3	4	5	6	7

i

Tri par insertion

x = 12

j

2	7	8	9	10	11	13	13
0	1	2	3	4	5	6	7

i

Tri par insertion

x = 12

j

2	7	8	9	10	11	12	13
0	1	2	3	4	5	6	7

i

Tri par insertion

x = 12

j

2	7	8	9	10	11	12	13
0	1	2	3	4	5	6	7

i

Terminaison

```
procedure trier-par-insertion(T)
    n ≔ longueur(T)
    pour i ≔ 1 à n – 1 faire (on saute le 0)
        x ≔ T[i]
        j ≔ i
        tant que j > 0 et x < T[j – 1] faire
            (décaler d'un élément)
            T[j] ≔ T[j – 1]
            j ≔ j – 1
        fin tant que
        (ici x ≥ T[j – 1] ou bien j = 0)
        T[j] ≔ x
    fin pour
fin procedure
```

- La boucle **pour** termine toujours
- La boucle **tant que** termine (au pire) quand j = 0

Correction

```
procedure trier-par-insertion(T)
    n ≔ longueur(T)
    pour i ≔ 1 à n – 1 faire (on saute le 0)
        x ≔ T[i]
        j ≔ i
        tant que j > 0 et x < T[j – 1] faire
            (décaler d'un élément)
            T[j] ≔ T[j – 1]
            j ≔ j – 1
        fin tant que
        (ici x ≥ T[j – 1] ou bien j = 0)
        T[j] ≔ x
    fin pour
fin procedure
```

- Le sous-tableau T[0, …, i – 1] est trié au début de la boucle **pour**

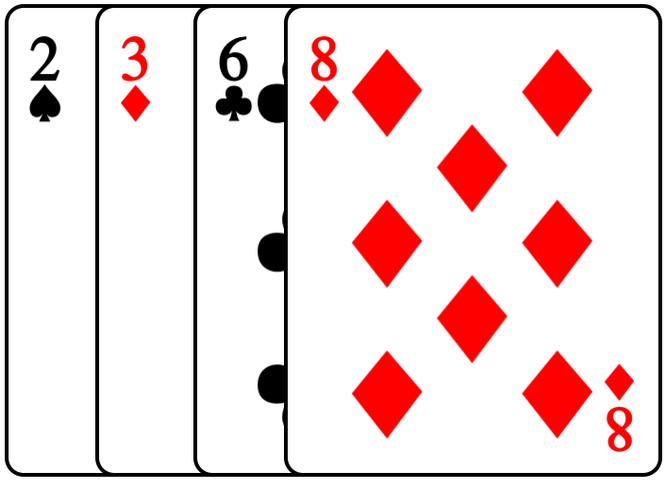
Efficacité

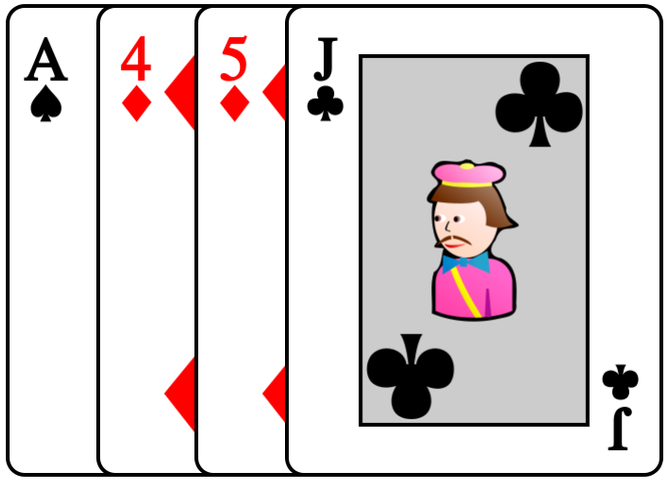
```
procedure trier-par-insertion(T)
    n ≔ longueur(T)
    pour i ≔ 1 à n – 1 faire (on saute le 0)
        x ≔ T[i]
        j ≔ i
        tant que j > 0 et x < T[j – 1] faire
            (décaler d'un élément)
            T[j] ≔ T[j – 1]
            j ≔ j – 1
        fin tant que
        (ici x ≥ T[j – 1] ou bien j = 0)
        T[j] ≔ x
    fin pour
fin procedure
```

- O(n) opérations dans le meilleur des cas
- $O(n^2)$ opérations dans le pire des cas

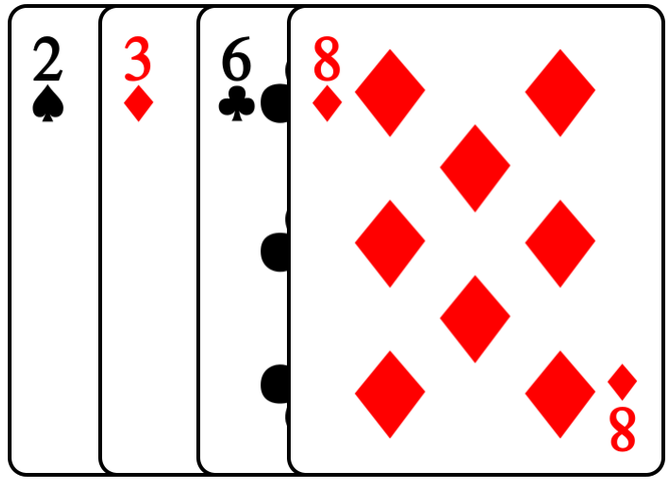
Est-ce qu’on peut faire mieux que ça ?

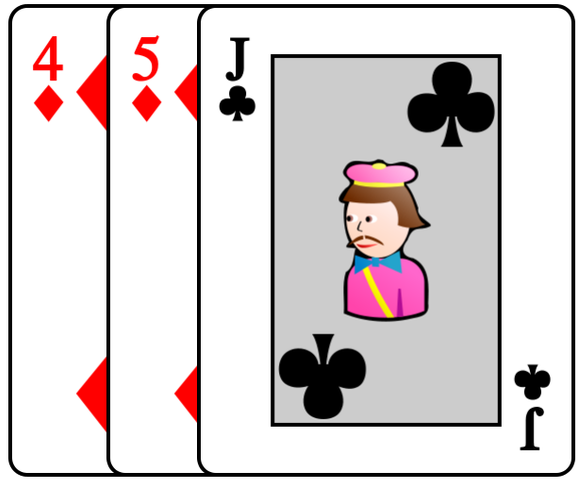
Fusion de tableaux triés

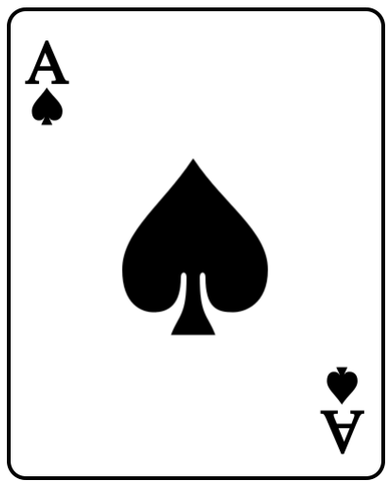




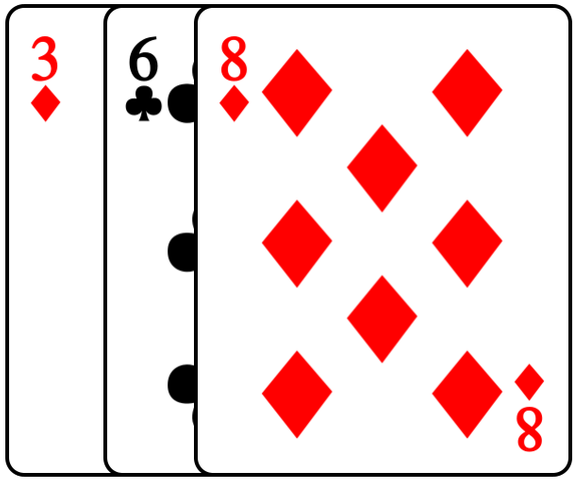
Fusion de tableaux triés

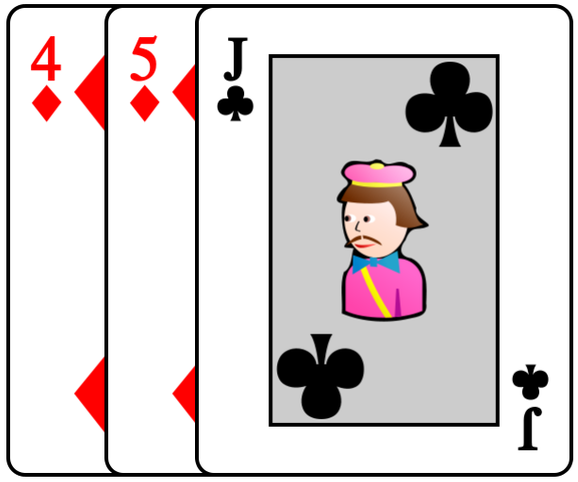


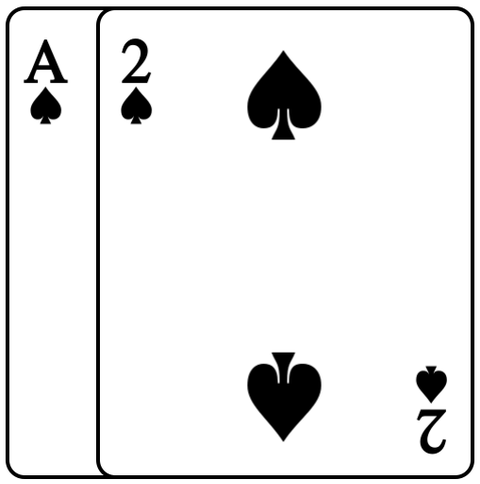




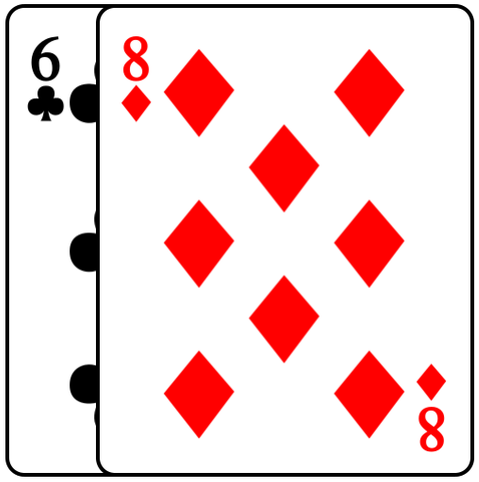
Fusion de tableaux triés

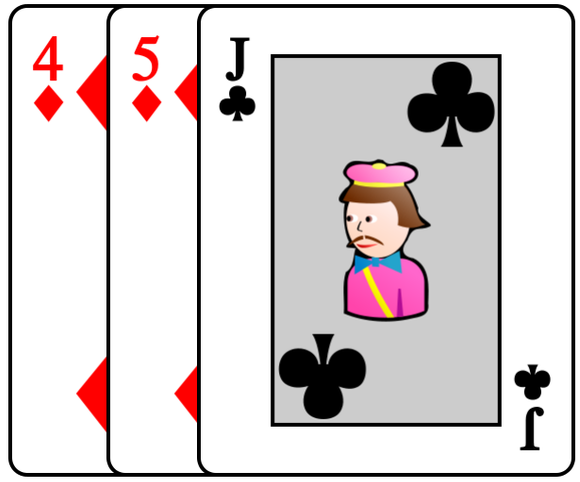


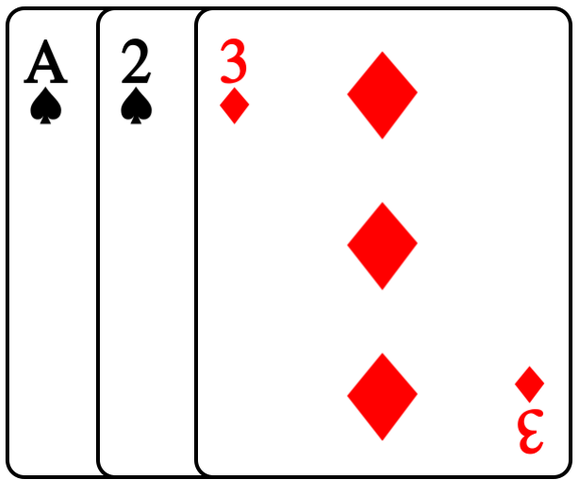




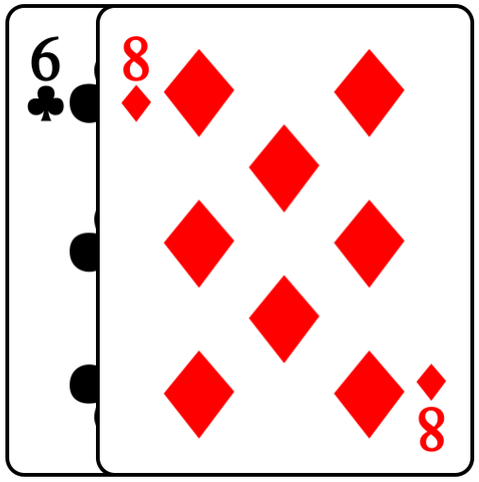
Fusion de tableaux triés

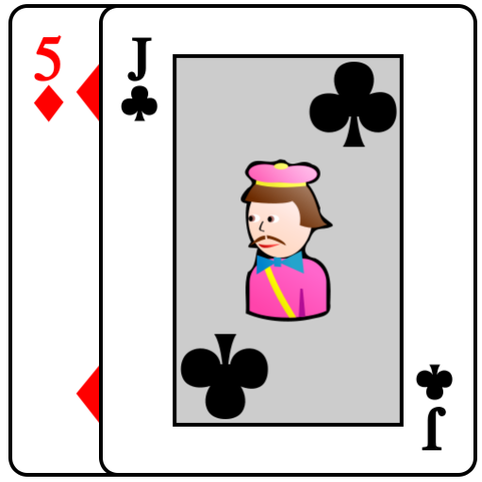


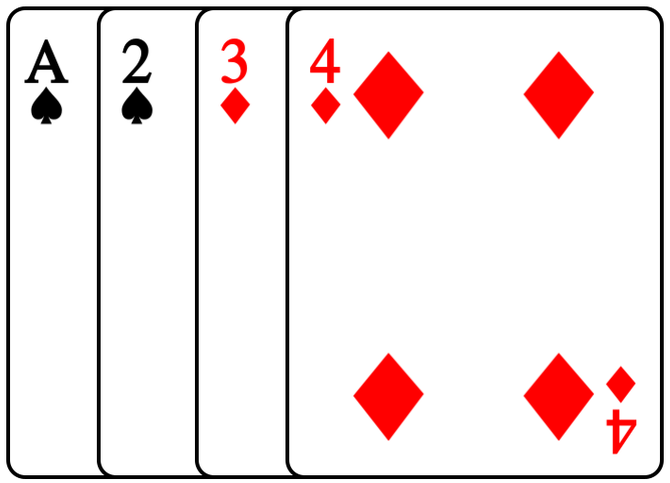




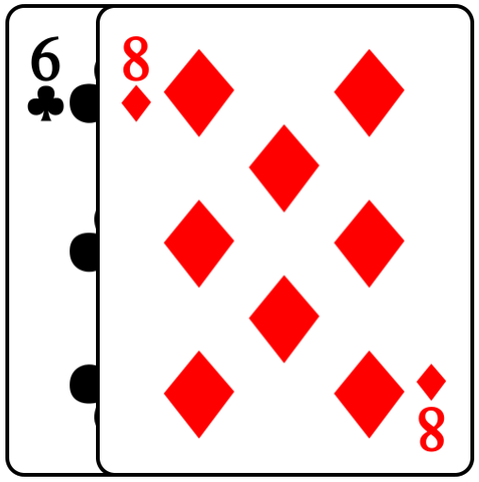
Fusion de tableaux triés

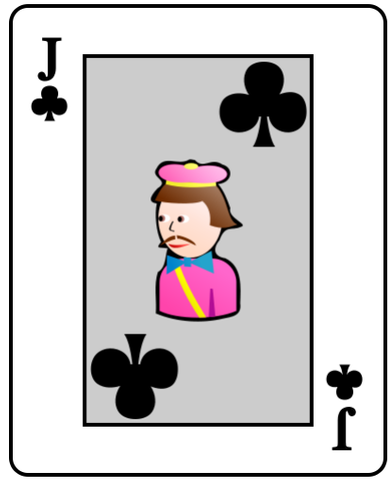


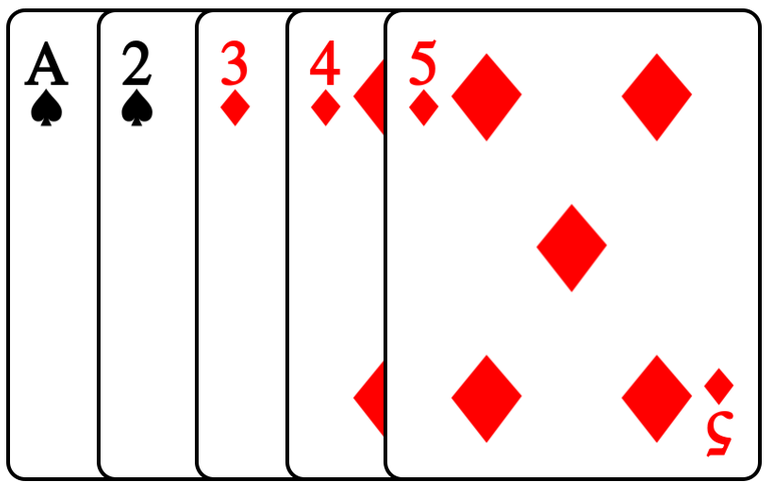




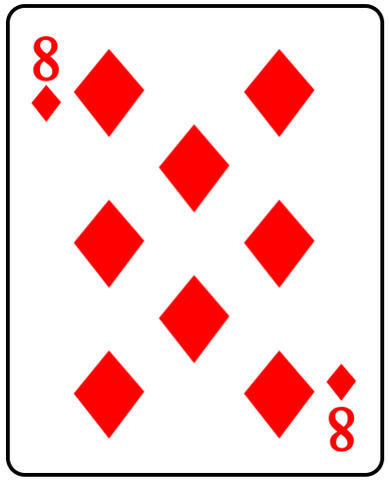
Fusion de tableaux triés

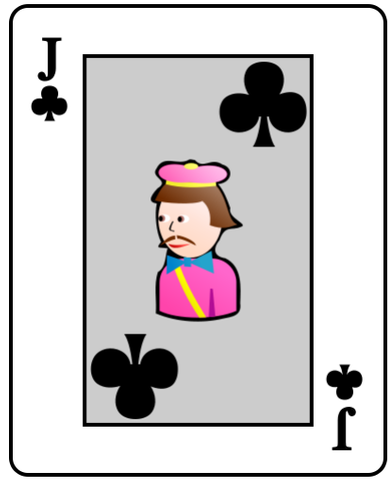


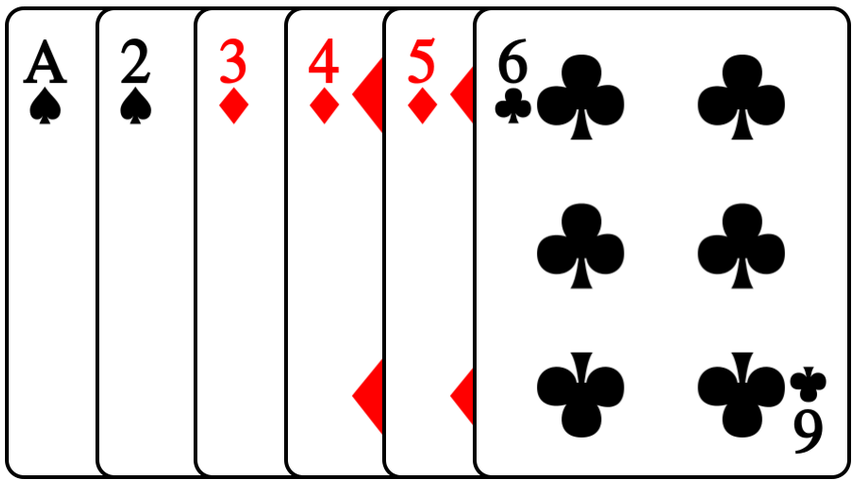




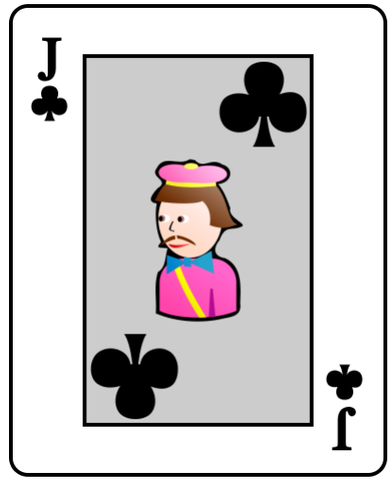
Fusion de tableaux triés

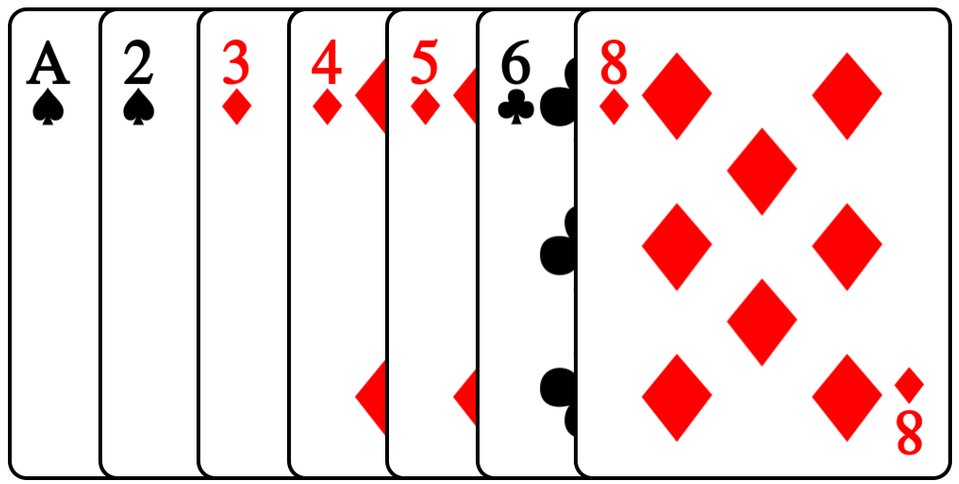




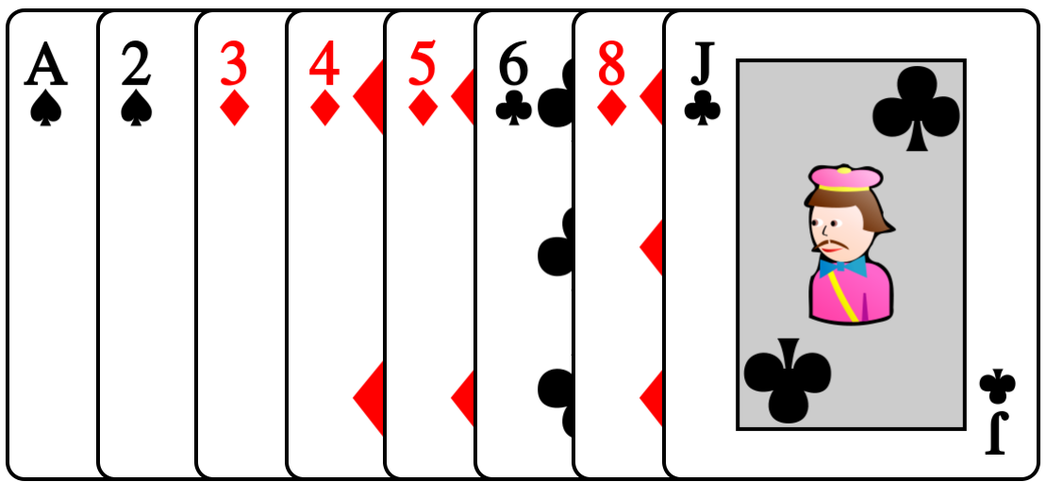


Fusion de tableaux triés

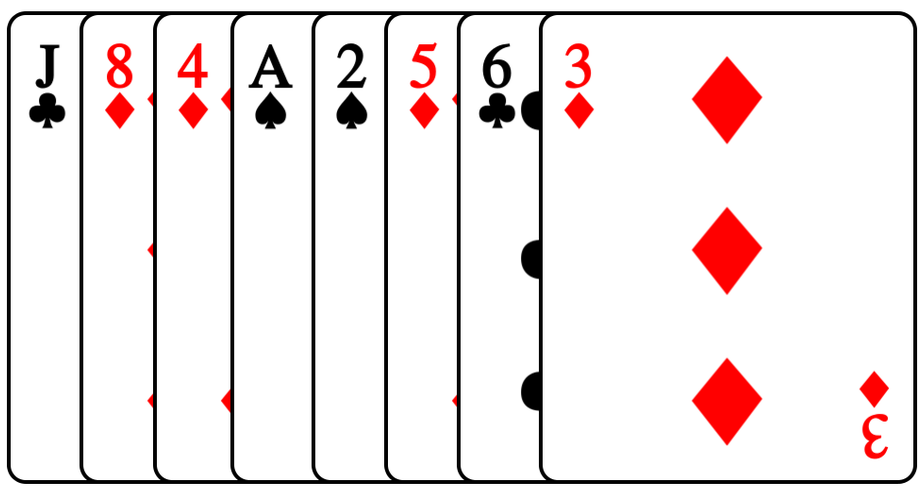




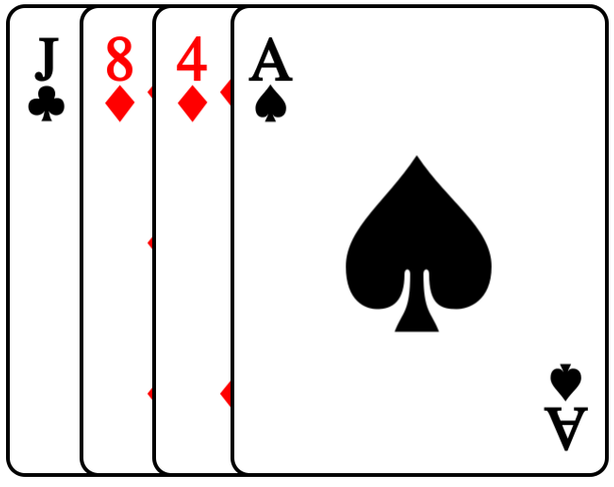
Fusion de tableaux triés

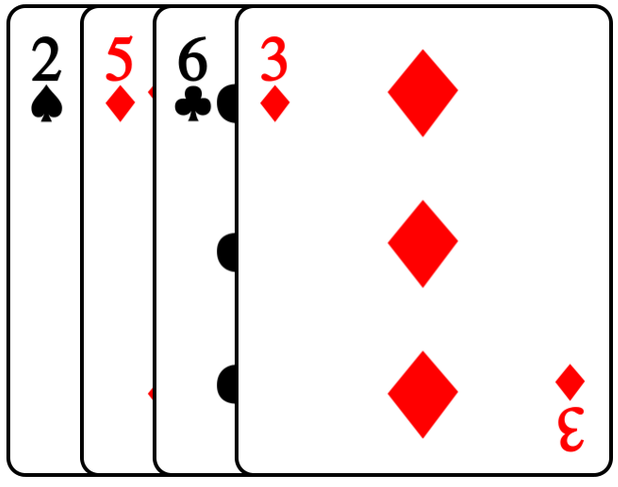


Tri fusion

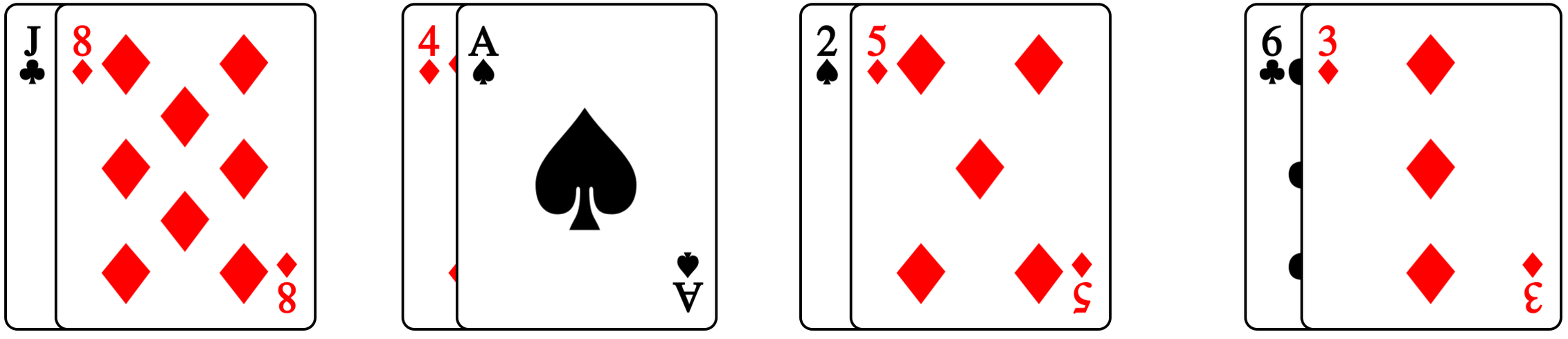


Diviser

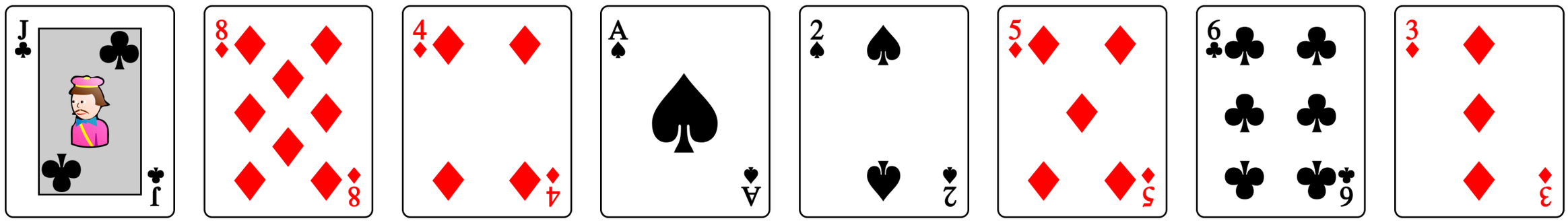




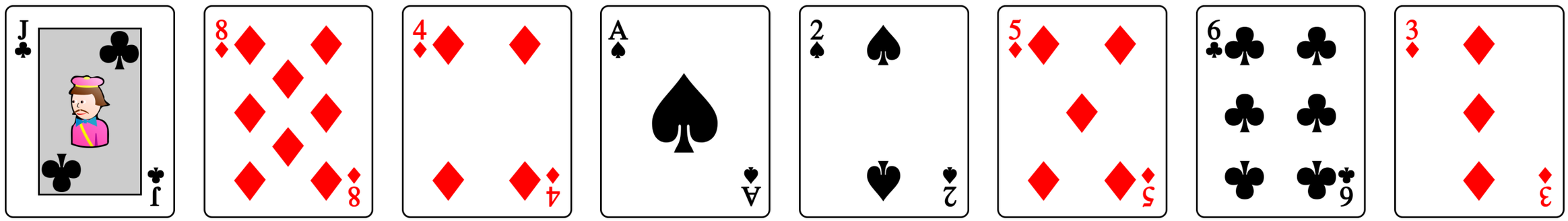
Diviser



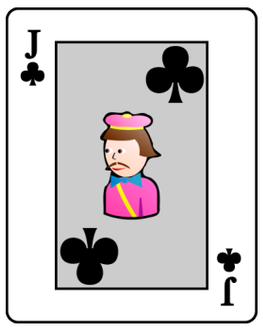
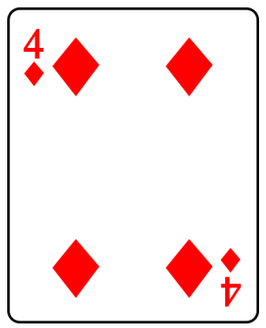
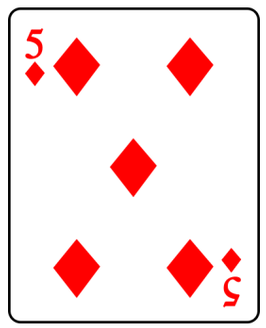
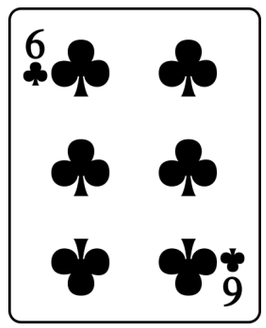
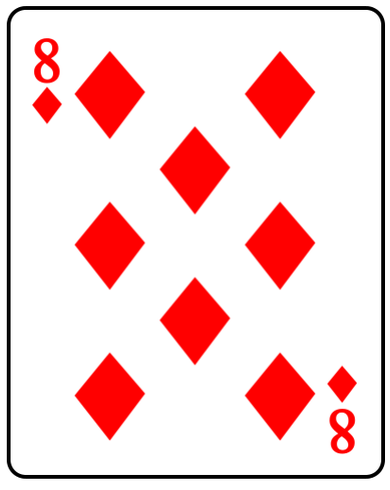
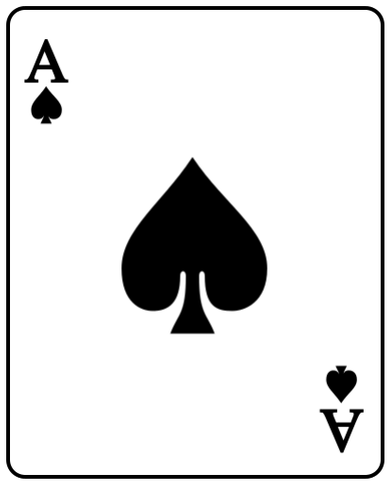
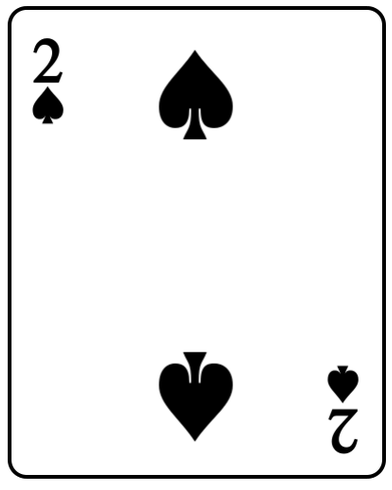
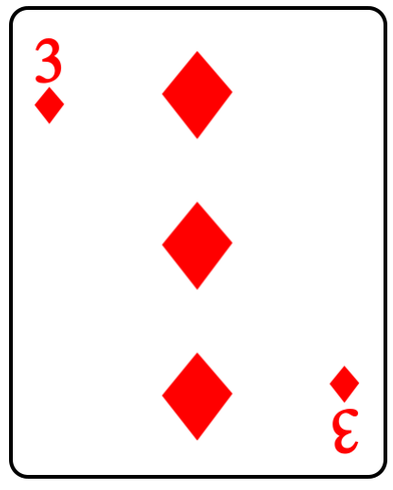
Diviser



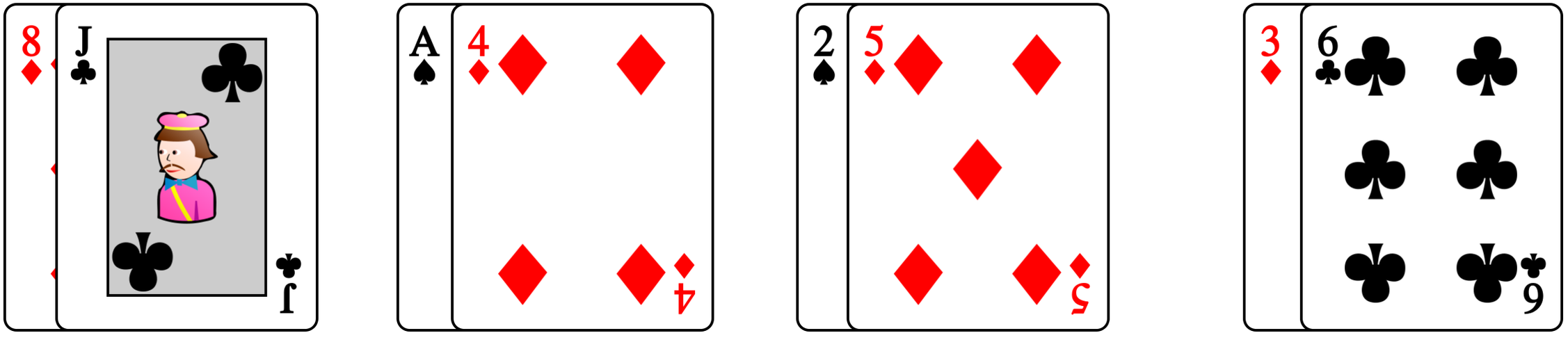
Tous les jeux sont triés !



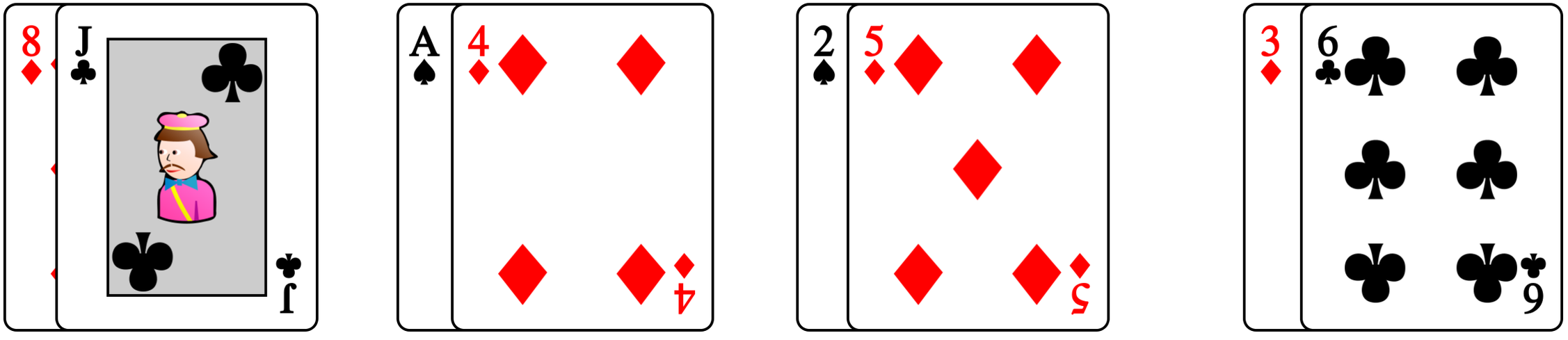
Fusionner

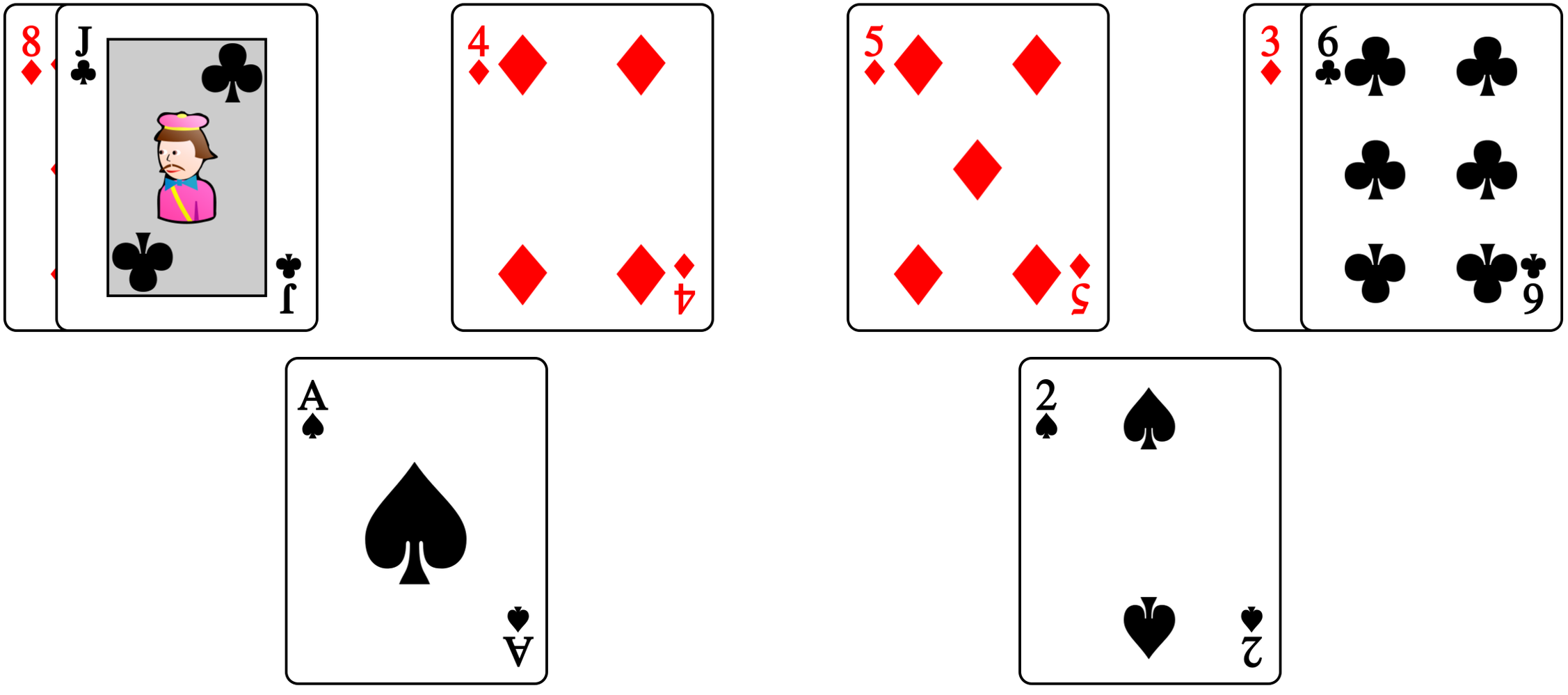
Fusionner



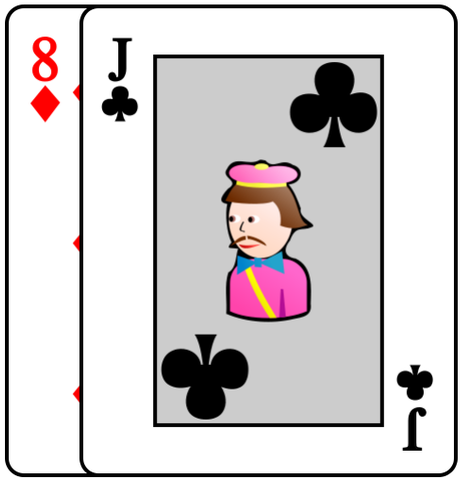
Tous les jeux sont triés !

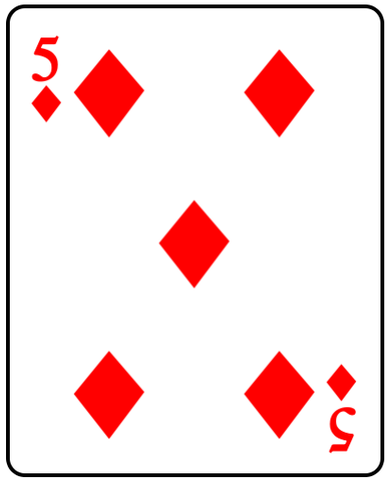


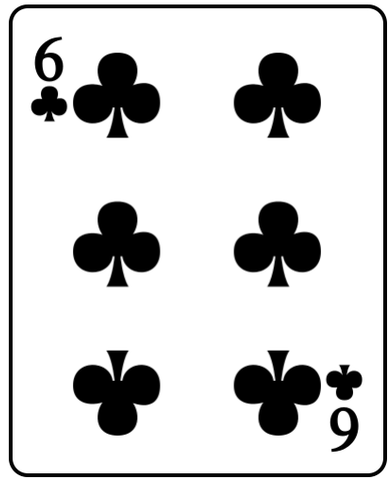
Fusionner

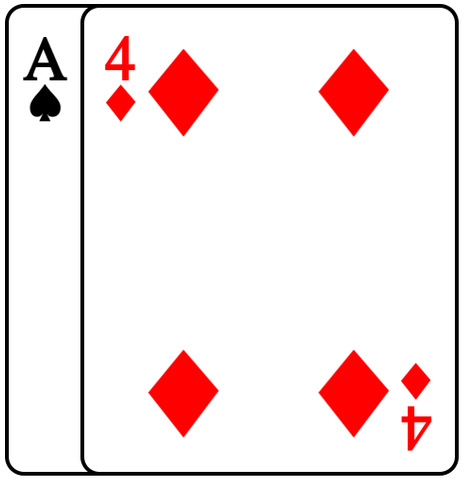


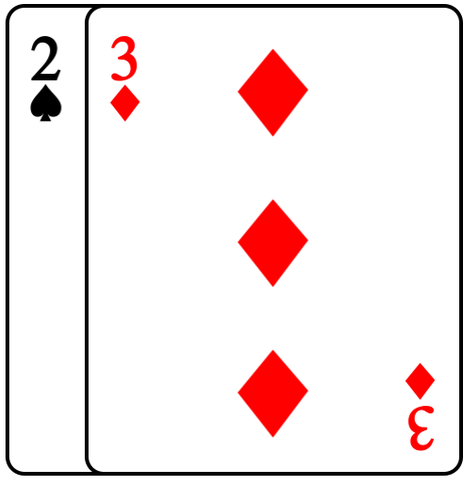
Fusionner



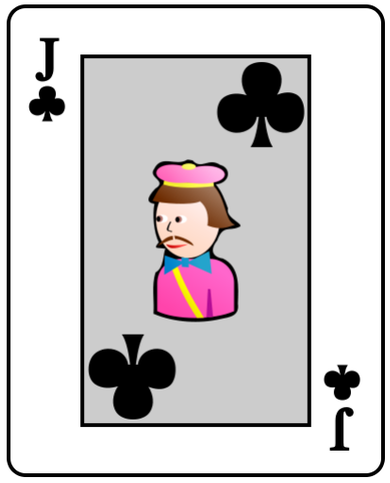


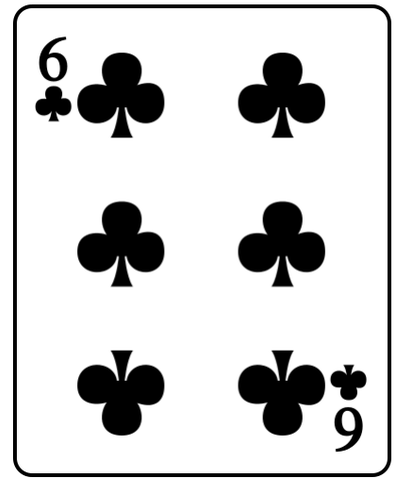


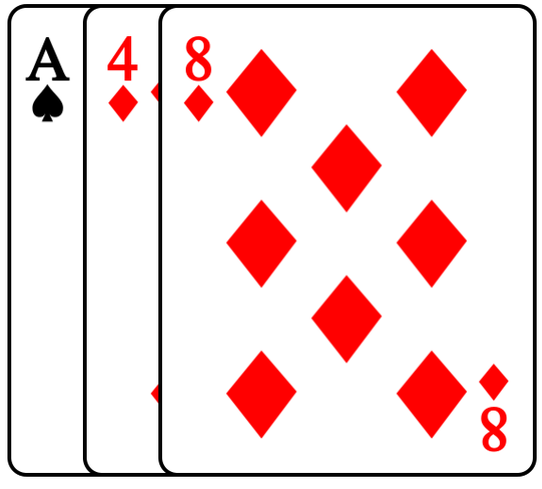


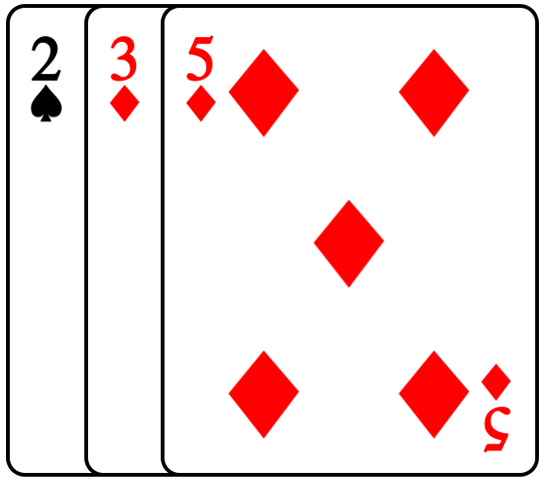


Fusionner

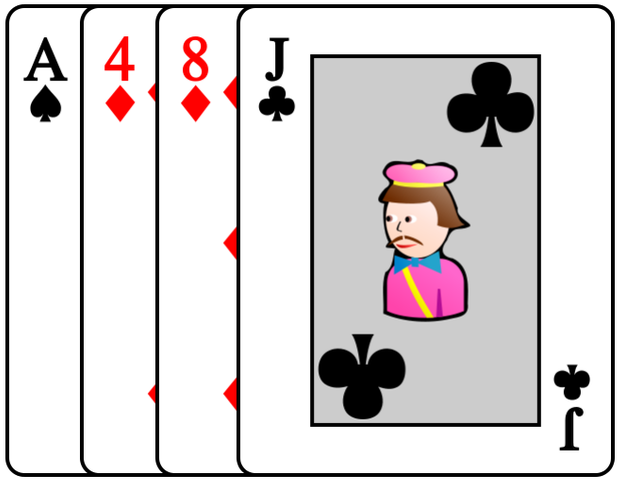


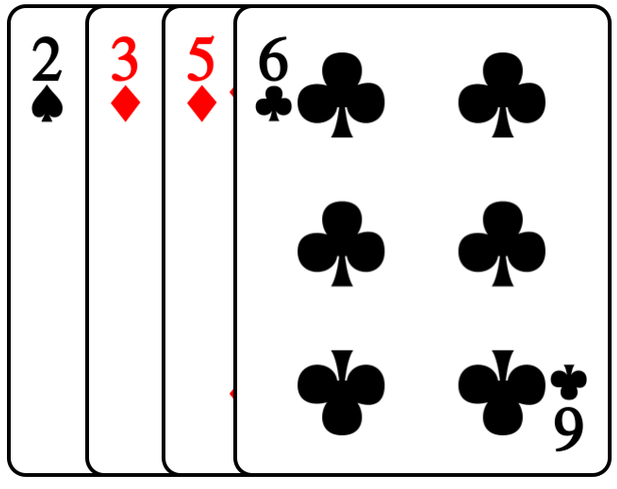




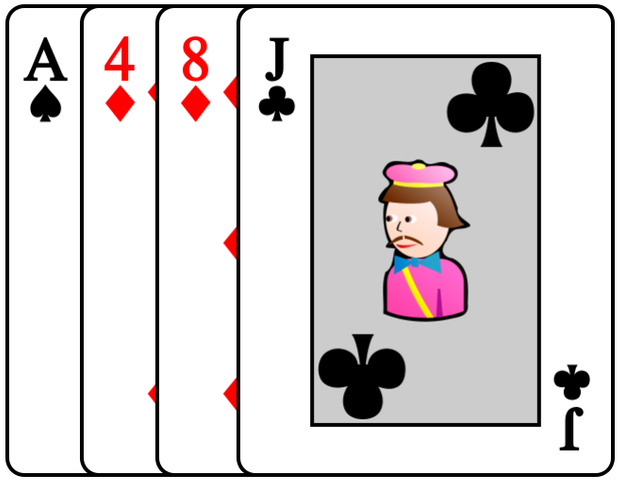


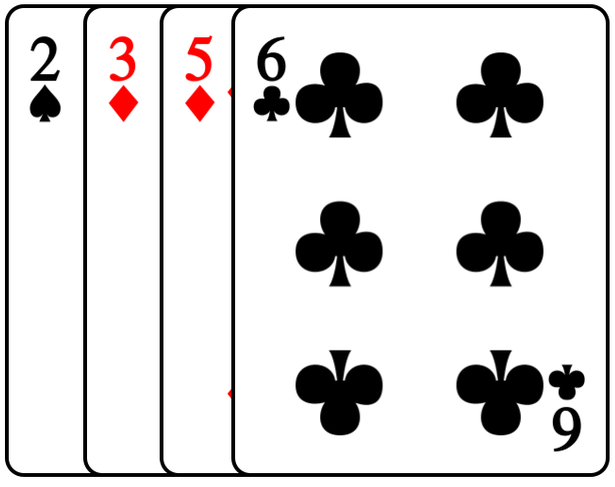
Fusionner



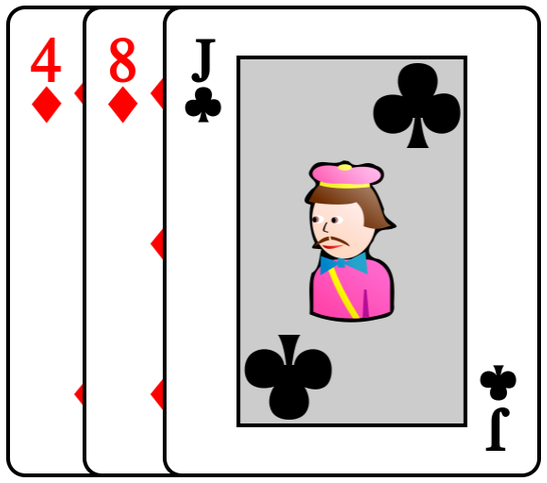


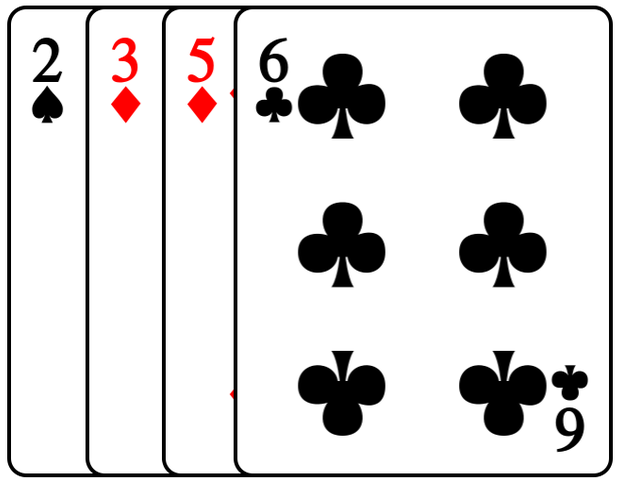
Tous les jeux sont triés !

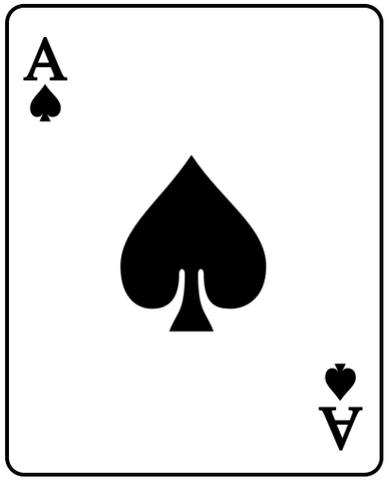




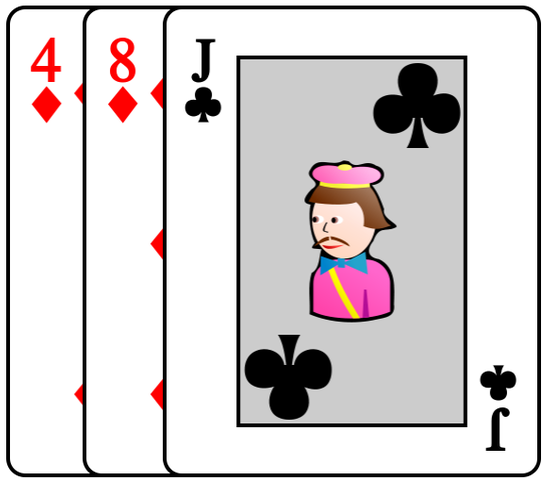
Fusionner

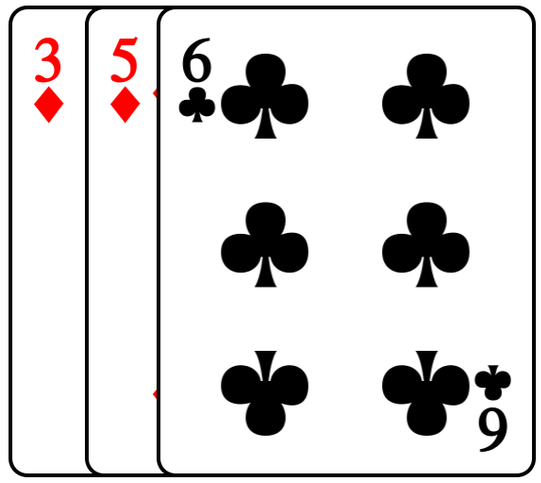


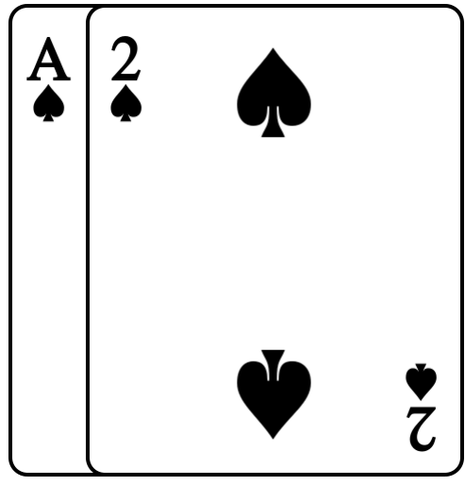




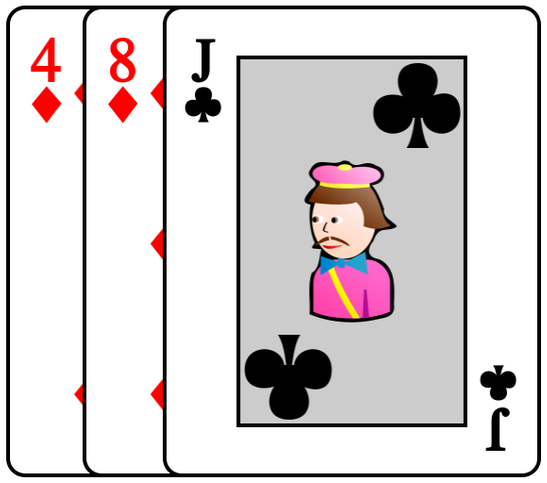
Fusionner

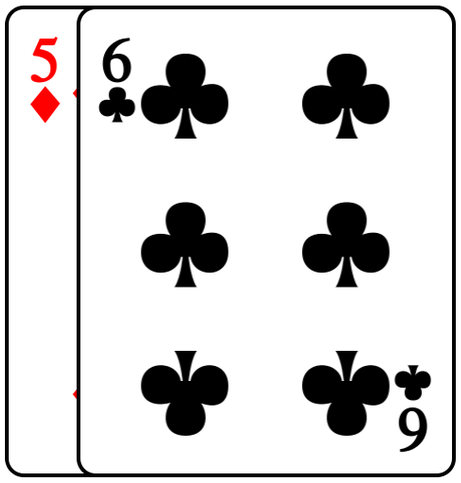


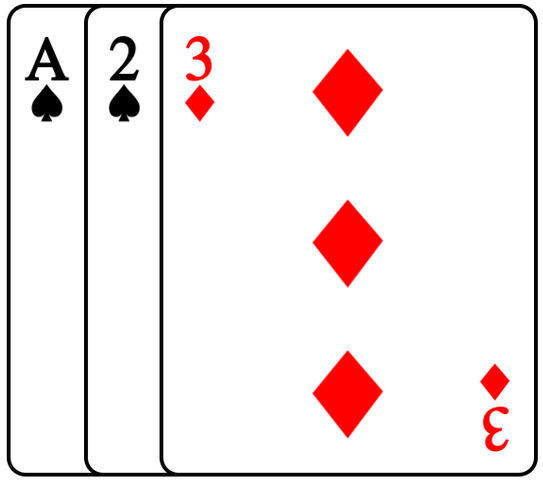




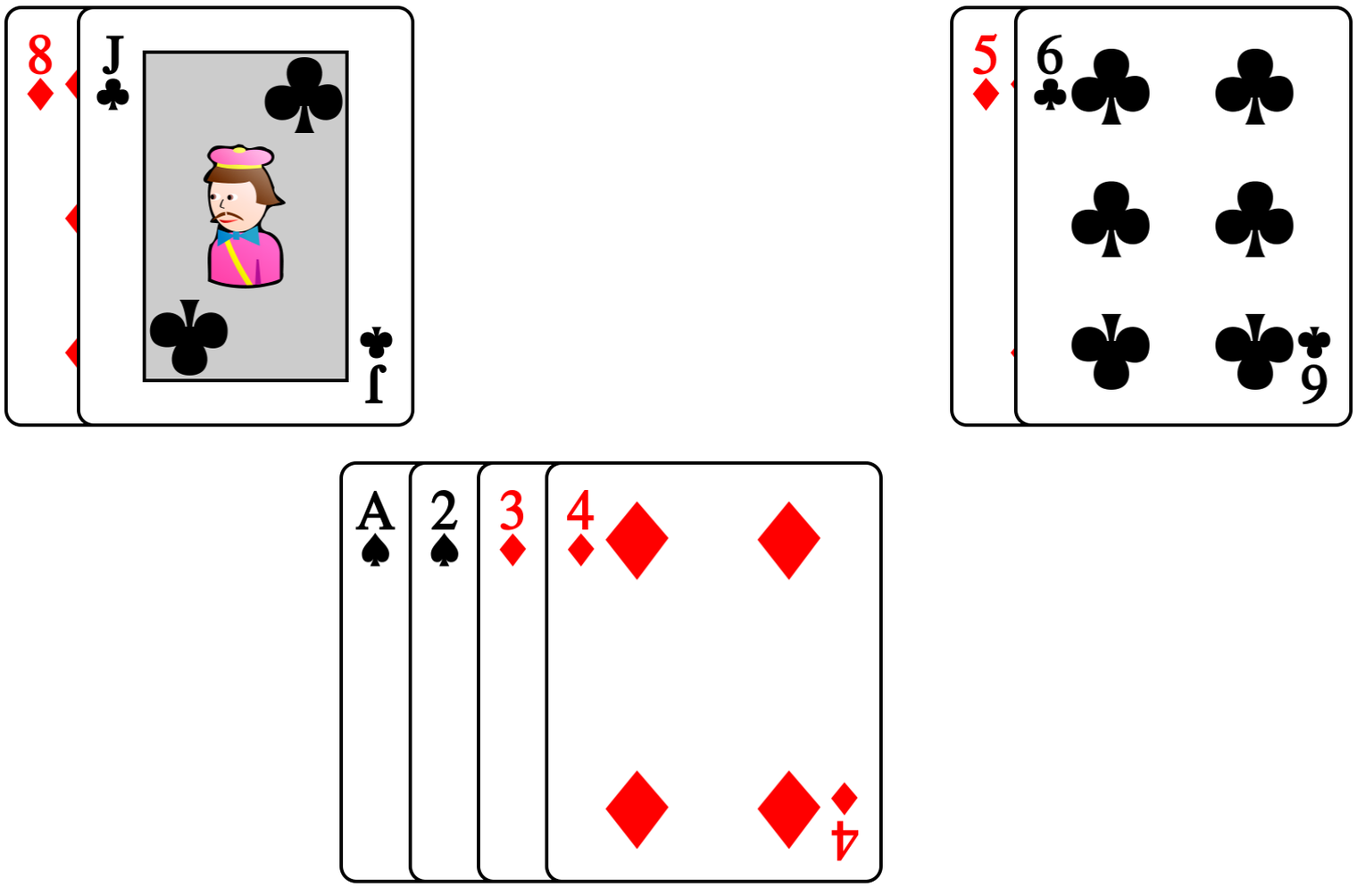
Fusionner



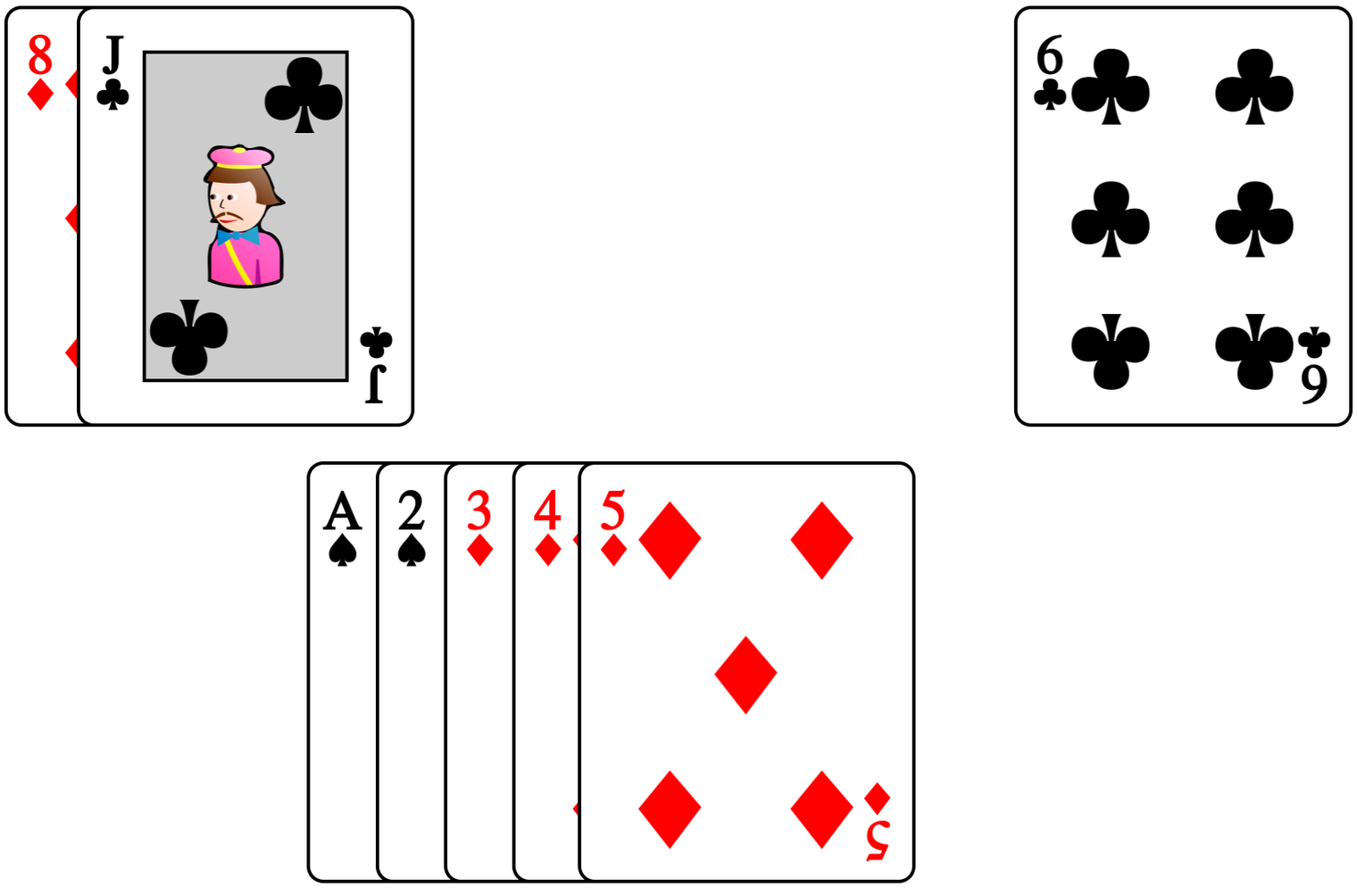




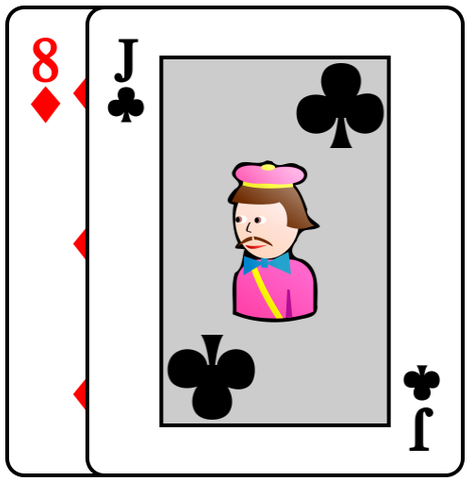
Fusionner

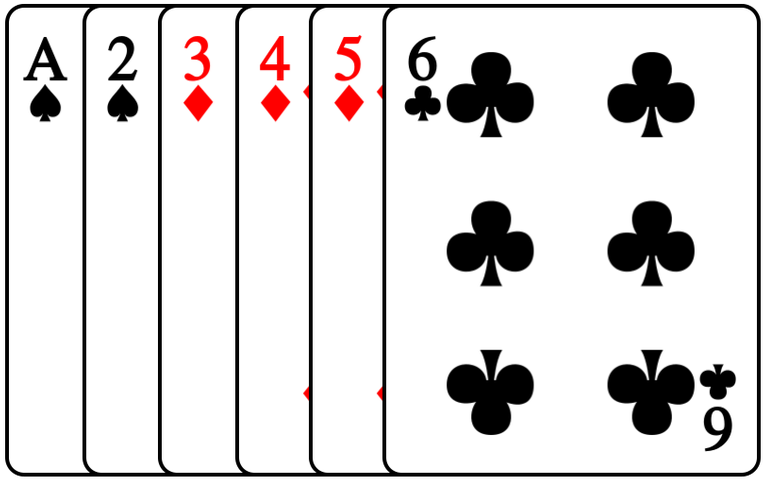


Fusionner

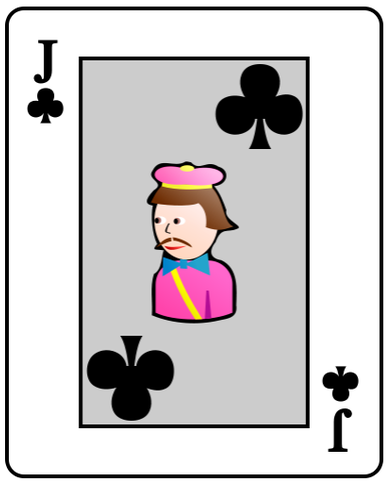


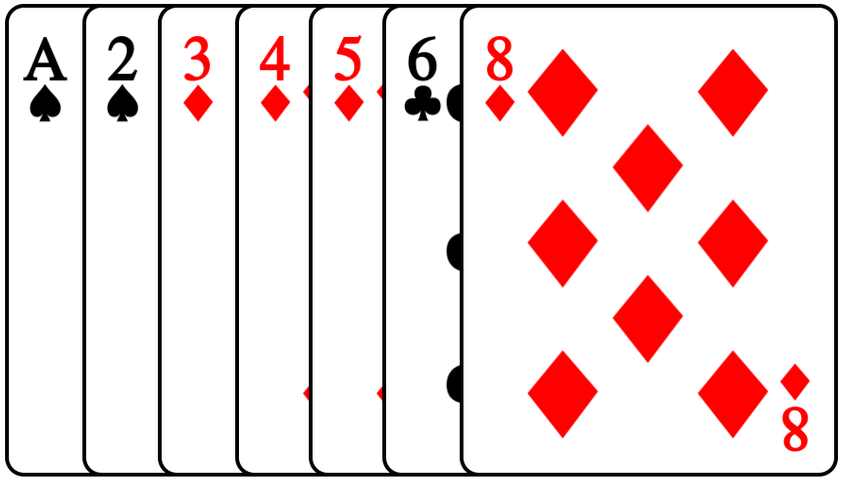
Fusionner



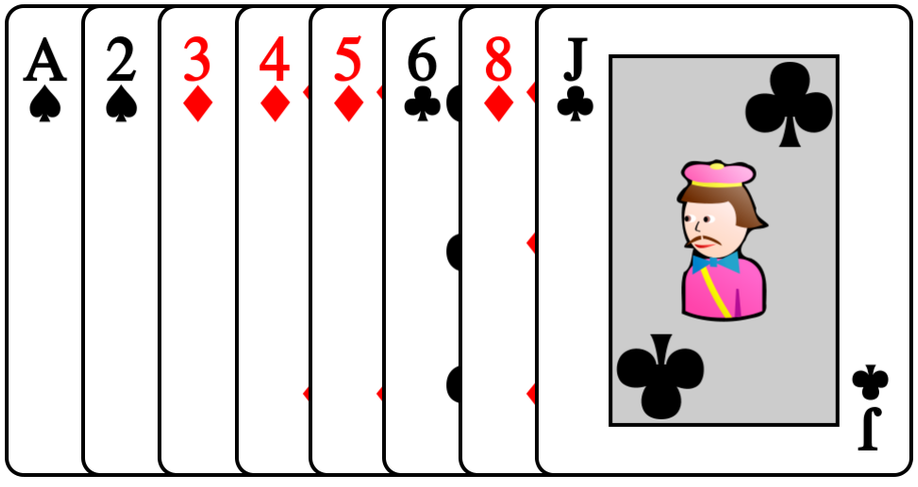


Fusionner

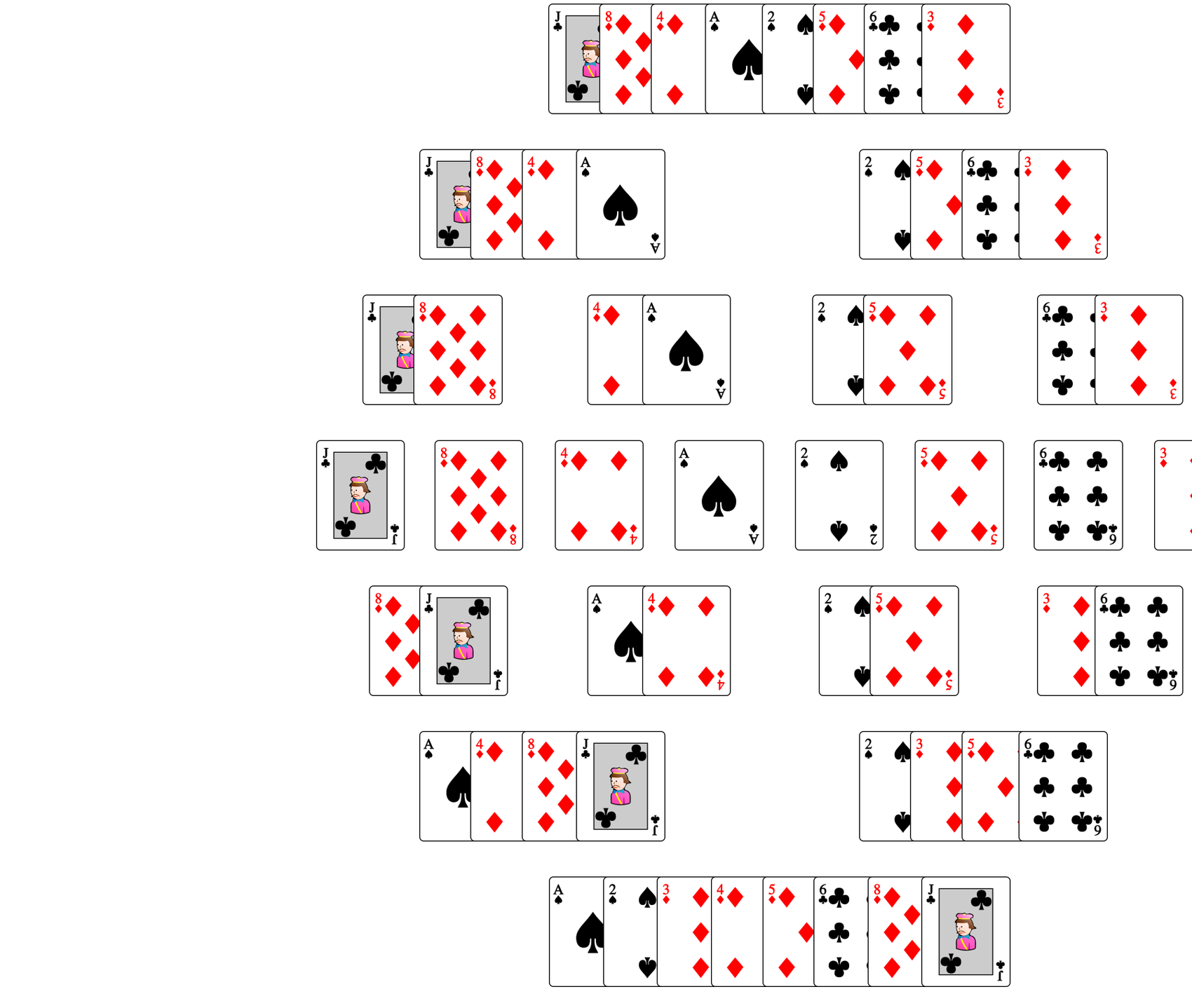




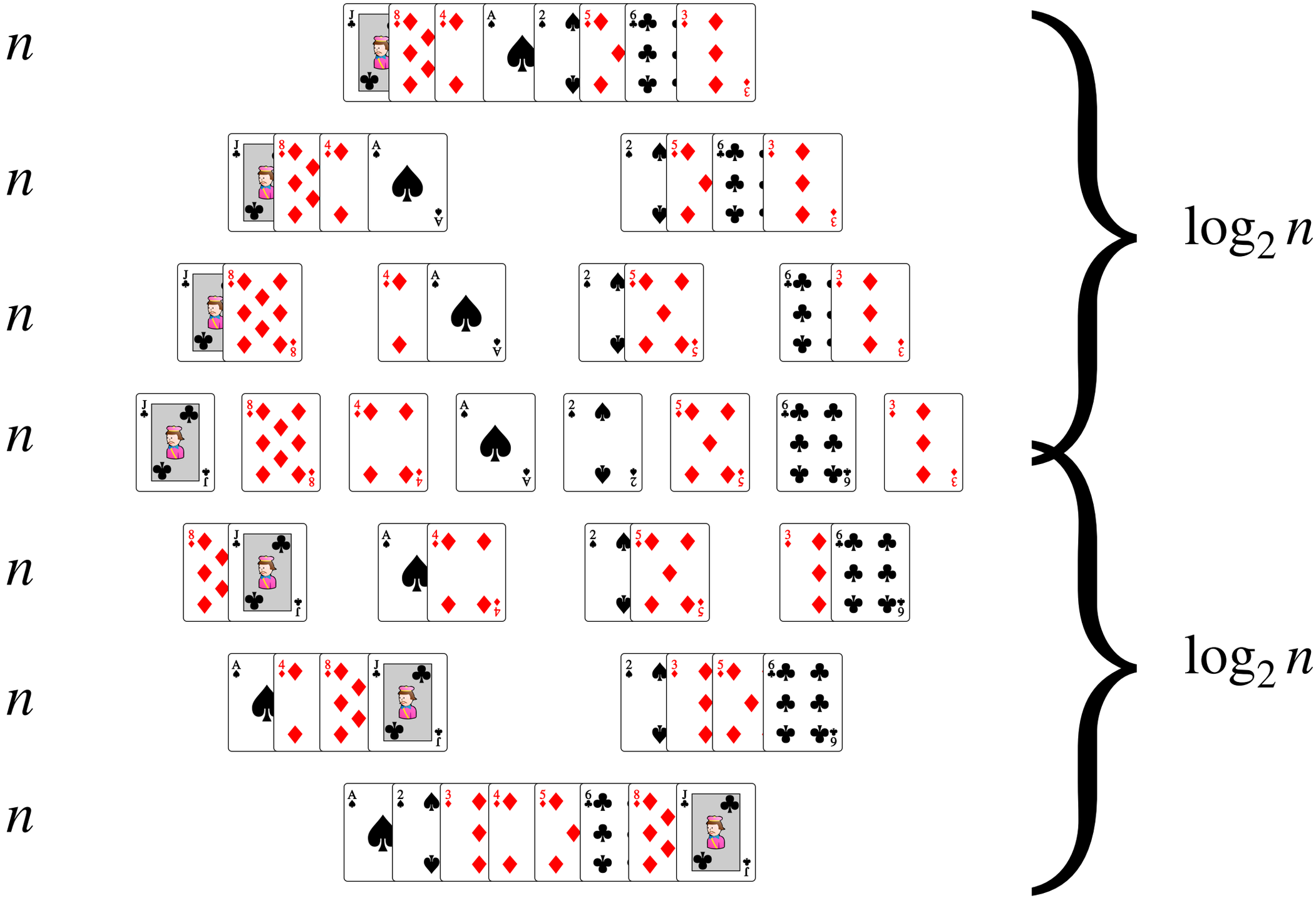
Le jeu est trié !



Efficacité du tri fusion



Efficacité du tri fusion



La complexité du tri fusion est

$$O(n \log_2 n)$$

Fusion

```
fonction fusionner(A, B)
    n ≔ longueur(A)
    m ≔ longueur(B)
    C ≔ tableau de longueur m + n
    i ≔ j ≔ k ≔ 0
    tant que i < n et j < m faire
        si A[i] < B[j] alors
            C[k] ≔ A[i]
            i ≔ i + 1
        sinon
            C[k] ≔ B[j]
            j ≔ j + 1
        fin si
        k ≔ k + 1
    fin tant que
    tant que i < n faire
        C[k] ≔ A[i]
        i ≔ i + 1
        k ≔ k + 1
    fin tant que
    tant que j < m faire
        C[k] ≔ B[j]
        j ≔ j + 1
        k ≔ k + 1
    fin tant que
    retourner C
fin fonction
```

encore d'éléments dans A et B

encore d'éléments dans A, mais B épuise

encore d'éléments dans B, mais A épuisé

A et B épuisés

Fusion

```
fonction fusionner(A, B)
    n ≔ longueur(A)
    m ≔ longueur(B)
    C ≔ tableau de longueur m + n
    i ≔ j ≔ k ≔ 0
    tant que i < n et j < m faire
        si A[i] < B[j] alors
            C[k] ≔ A[i]
            i ≔ i + 1
        sinon
            C[k] ≔ B[j]
            j ≔ j + 1
```

encore d'éléments dans A et B

Fusion

```
fonction fusionner(A, B)
    n ≔ longueur(A)
    m ≔ longueur(B)
    C ≔ tableau de longueur m + n
    i ≔ j ≔ k ≔ 0
    tant que i < n et j < m faire
        si A[i] < B[j] alors
            C[k] ≔ A[i]
            i ≔ i + 1
        sinon
            C[k] ≔ B[j]
            j ≔ j + 1
        fin si
        k ≔ k + 1
    fin tant que
    tant que i < n faire
        C[k] ≔ A[i]
        i ≔ i + 1
        k ≔ k + 1
    fin tant que
```

encore d'éléments dans A et B

encore d'éléments dans A, mais B épuise

```
        i ≔ i + 1
    sinon
        C[k] ≔ B[j]
        j ≔ j + 1
    fin si
    k ≔ k + 1
fin tant que
tant que i < n faire
    C[k] ≔ A[i]
    i ≔ i + 1
    k ≔ k + 1
fin tant que
tant que j < m faire
    C[k] ≔ B[j]
    j ≔ j + 1
    k ≔ k + 1
fin tant que
retourner C
fin fonction
```

encore d'éléments dans A et B

encore d'éléments dans A, mais B épuise

encore d'éléments dans B, mais A épuise

A et B épuisés

```
        k ≔ k + 1
    fin tant que
    tant que i < n faire
        C[k] ≔ A[i]
        i ≔ i + 1
        k ≔ k + 1
    fin tant que
    tant que j < m faire
        C[k] ≔ B[j]
        j ≔ j + 1
        k ≔ k + 1
    fin tant que
    retourner C
fin fonction
```

encore d'éléments dans A, mais B épuise

encore d'éléments dans B, mais A épuise

A et B épuisés

```
tant que i < n faire
    C[k] ≔ A[i]
    i ≔ i + 1
    k ≔ k + 1
fin tant que
tant que j < m faire
    C[k] ≔ B[j]
    j ≔ j + 1
    k ≔ k + 1
fin tant que
    retourner C
fin fonction
```

encore d'éléments dans A, mais B épuise

encore d'éléments dans A, mais B épuise

A et B épuisés

Fusion

```
fonction fusionner(A, B)
    n ≔ longueur(A)
    m ≔ longueur(B)
    i ≔ j ≔ k ≔ 0
    tant que i < n et j < m faire
        si A[i] < B[j] alors
            C[k] ≔ A[i]
            i ≔ i + 1
        sinon
            C[k] ≔ B[j]
            j ≔ j + 1
        fin si
        k ≔ k + 1
    fin tant que
    tant que i < n faire
        C[k] ≔ A[i]
        i ≔ i + 1
        k ≔ k + 1
    fin tant que
    tant que j < m faire
        C[k] ≔ B[j]
        j ≔ j + 1
        k ≔ k + 1
    fin tant que
    retourner C
fin fonction
```

encore d'éléments dans A et B

encore d'éléments dans A, mais B épuise

encore d'éléments dans B, mais A épuisé

A et B épuisés

Tri fusion

```
fonction tri-fusion(A)
    n ≔ longueur(A)
    si n > 1 alors
        m ≔ ⌊n ÷ 2⌋
        B ≔ A[0, …, m–1]
        C ≔ A[m, …, n–1]
        B’ ≔ tri-fusion(B)
        C’ ≔ tri-fusion(C)
        retourner fusionner(B’, C’)
    sinon
        retourner A
    fin si
fin fonction
```