Science informatique

CM2

Antonio E. Porreca
aeporreca.org/scienceinfo

C’est quoi un algorithme ?

La description non ambiguë d’une séquence finie d’instructions permettant de résoudre un problème

محمد بن موسى الخوارزمي

Muḥammad ibn Mūsā al-Khwārizmī

Un algorithme récursif

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

Exemple d’exécution

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

Exemple d’exécution

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

```
factoriel(5) =
```

Exemple d’exécution

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

```
factoriel(5) =
5 * factoriel(4) =
```

Exemple d’exécution

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

```
factoriel(5) =
5 * factoriel(4) =
5 * 4 * factoriel(3) =
```

Exemple d’exécution

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

```
factoriel(5) =
5 * factoriel(4) =
5 * 4 * factoriel(3) =
5 * 4 * 3 * factoriel(2) =
```

Exemple d’exécution

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

```
factoriel(5) =
5 * factoriel(4) =
5 * 4 * factoriel(3) =
5 * 4 * 3 * factoriel(2) =
5 * 4 * 3 * 2 * factoriel(1) =
```

Exemple d’exécution

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

```
factoriel(5) =
5 * factoriel(4) =
5 * 4 * factoriel(3) =
5 * 4 * 3 * factoriel(2) =
5 * 4 * 3 * 2 * factoriel(1) =
5 * 4 * 3 * 2 * 1 * factoriel(0) =
```

Exemple d’exécution

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

```
factoriel(5) =
5 * factoriel(4) =
5 * 4 * factoriel(3) =
5 * 4 * 3 * factoriel(2) =
5 * 4 * 3 * 2 * factoriel(1) =
5 * 4 * 3 * 2 * 1 * factoriel(0) =
5 * 4 * 3 * 2 * 1 * 1
```

Exemple d’exécution

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

```
factoriel(5) =
5 * factoriel(4) =
5 * 4 * factoriel(3) =
5 * 4 * 3 * factoriel(2) =
5 * 4 * 3 * 2 * factoriel(1) =
5 * 4 * 3 * 2 * 1 * factoriel(0) =
5 * 4 * 3 * 2 * 1 * 1
120
```

Exemple d'exécution

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

```
factoriel(5) =
5 * factoriel(4) =
5 * 4 * factoriel(3) =
5 * 4 * 3 * factoriel(2) =
5 * 4 * 3 * 2 * factoriel(1) =
5 * 4 * 3 * 2 * 1 * factoriel(0) =
5 * 4 * 3 * 2 * 1 * 1
120
```

chaque appel récursif :
condition `if n == 0` et soit
`return 1`, soit
`return n * factoriel(n - 1)`
total $6 \times 2 = 12$ instructions

Nombre total d’opérations pour un algorithme récursif

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

Nombre total d'opérations pour un algorithme récursif

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

$$T(0) = 2$$

Nombre total d'opérations pour un algorithme récursif

```python
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

$$T(0) = 2$$
$$T(n) = 2 + T(n-1) \textbf{ pour } n > 0$$

Nombre total d'opérations pour un algorithme récursif

```python
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

$$T(0) = 2$$
$$T(n) = 2 + T(n-1) \text{ pour } n > 0$$

$$\Rightarrow T(n) = 2(n+1) \in O(n)$$

Exercice 1 du TD2

Montrer la terminaison d'un algorithme

- Il pourrait y avoir des boucles `while` infinies, ou des appels récursifs qui n'atteignent jamais un cas de base

- Idée principale : il faut trouver un variant de boucle, une expression dont la valeur change à chaque itération et permet l'invalidation de la condition de la boucle en temps fini

- Pour les algorithmes récursifs, montrer que les paramètres des appels récursives nous permettent d'atteindre l'un des cas de base

Variants de boucle pour prouver la terminaison

```
def puissance(x, n):
    i = 0
    p = 1
    while i < n:
        p = x * p
        i = i + 1
    return p
```

Variants de boucle pour prouver la terminaison

```
def puissance(x, n):
    i = 0
    p = 1
    while i < n:
        p = x * p
        i = i + 1
    return p
```

variant de boucle : l'expression i, initialement ayant la valeur 0

Variants de boucle pour prouver la terminaison

```
def puissance(x, n):
    i = 0
    p = 1
    while i < n:
        p = x * p
        i = i + 1
    return p
```

variant de boucle : l'expression i, initialement ayant la valeur 0

augmente de 1 à chaque tour de boucle et donc, tot ou tard, atteint la valeur n

Variants de boucle pour prouver la terminaison

```
def puissance(x, n):
    i = 0
    p = 1
    while i < n:
        p = x * p
        i = i + 1
    return p
```

variant de boucle : l'expression i, initialement ayant la valeur 0

augmente de 1 à chaque tour de boucle et donc, tot ou tard, atteint la valeur n

ce qui cause la terminaison de l'algorithme

Un algorithme récursif

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

Un algorithme récursif

ici le paramètre n, initialement ayant une valeur entière naturelle

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

Un algorithme récursif

ici le paramètre n, initialement ayant une valeur entière naturelle

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

est décrémenté à chaque appel récursif

Un algorithme récursif

ici le paramètre n, initialement ayant une valeur entière naturelle

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n - 1)
```

ce qui cause la terminaison de l'algorithme en un temps fini

est décrémenté à chaque appel récursif

Exercice 2.1 du TD2

Montrer la correction d'un algorithme

- Encore une fois, normalement la partie difficile est due aux boucles

- Idée principale : il faut trouver un invariant de boucle, une proposition qui reste vraie tout au long l'execution de la boucle

- …dont on peut déduire que la sortie est celle qu'on souhaite

Invariants de boucle pour prouver la correction

```
def puissance(x, n):
    i = 0
    p = 1
    while i < n:
        p = x * p
        i = i + 1
    return p
```

Invariants de boucle pour prouver la correction

invariant de boucle :

$p = x^i$

```
def puissance(x, n):
    i = 0
    p = 1
    while i < n:
        p = x * p
        i = i + 1
    return p
```

Invariants de boucle pour prouver la correction

invariant de boucle :

$p = x^i$

c'est vrai juste avant de commencer la boucle : $1 = x^0$

```
def puissance(x, n):
    i = 0
    p = 1
    while i < n:
        p = x * p
        i = i + 1
    return p
```

Invariants de boucle pour prouver la correction

invariant de boucle :

$$p = x^i$$

```
def puissance(x, n):
    i = 0
    p = 1
    while i < n:
        p = x * p
        i = i + 1
    return p
```

c'est vrai juste avant de commencer la boucle : $1 = x^0$

ici p est multiplié par x, ce qui donne $p = x \cdot x^i = x^{i+1}$

Invariants de boucle pour prouver la correction

invariant de boucle :

$p = x^i$

```
def puissance(x, n):
    i = 0
    p = 1
    while i < n:
        p = x * p
        i = i + 1
    return p
```

c'est vrai juste avant de commencer la boucle : $1 = x^0$

ici p est multiplié par x, ce qui donne $p = x \cdot x^i = x^{i+1}$

ici i est incrémenté, ce qui donne rétablit donc la validité de l'invariant $p = x^i$

Invariants de boucle pour prouver la correction

invariant de boucle : $p = x^i$

```
def puissance(x, n):
    i = 0
    p = 1
    while i < n:
        p = x * p
        i = i + 1
    return p
```

c'est vrai juste avant de commencer la boucle : $1 = x^0$

ici p est multiplié par x, ce qui donne $p = x \cdot x^i = x^{i+1}$

en sortant de la boucle on a $i = n$ et donc $p = x^n$, le bon résultat !

ici i est incrémenté, ce qui donne rétablit donc la validité de l'invariant $p = x^i$

Exercice 2.2 du TD2

Peut-on faire mieux que $O(n)$ pour la recherche dans un tableau ?

Recherche dichotomique dans un tableau trié

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

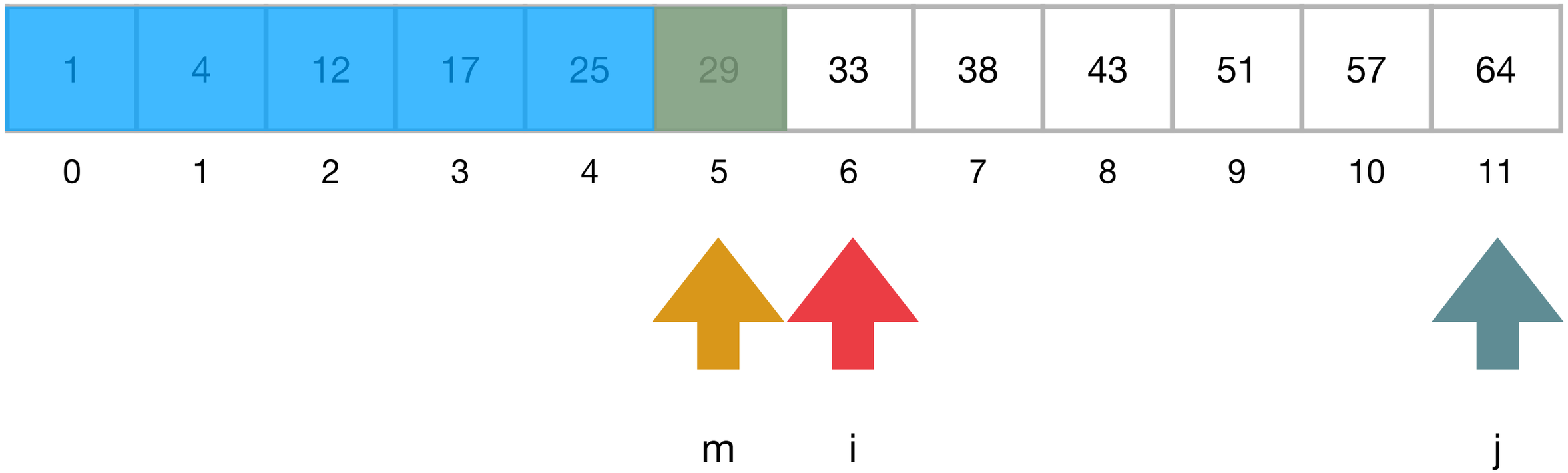
Recherche dichotomique

Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

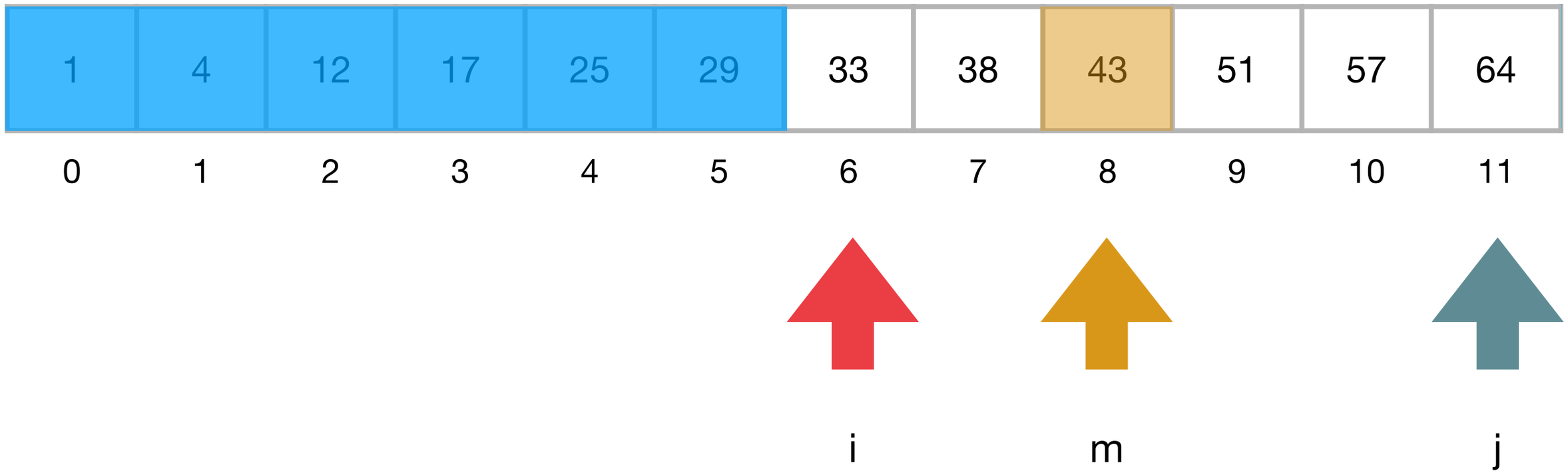
Recherche dichotomique

Recherche de 33



Recherche dichotomique

Recherche de 33



Recherche dichotomique

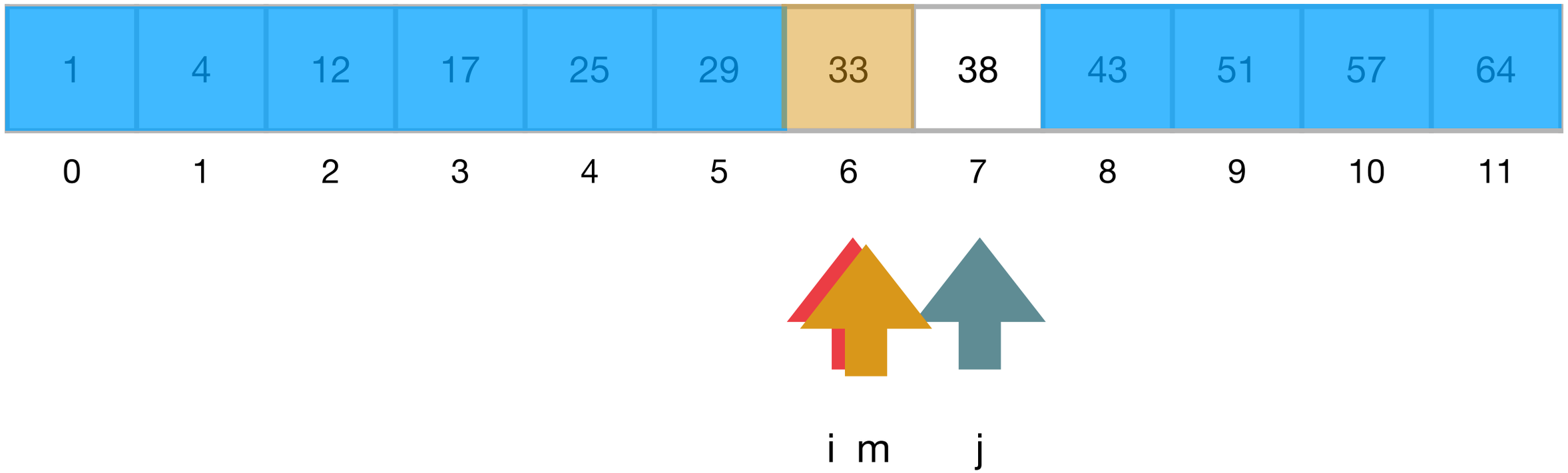
Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

i j m

Recherche dichotomique

Recherche de 33



Recherche dichotomique

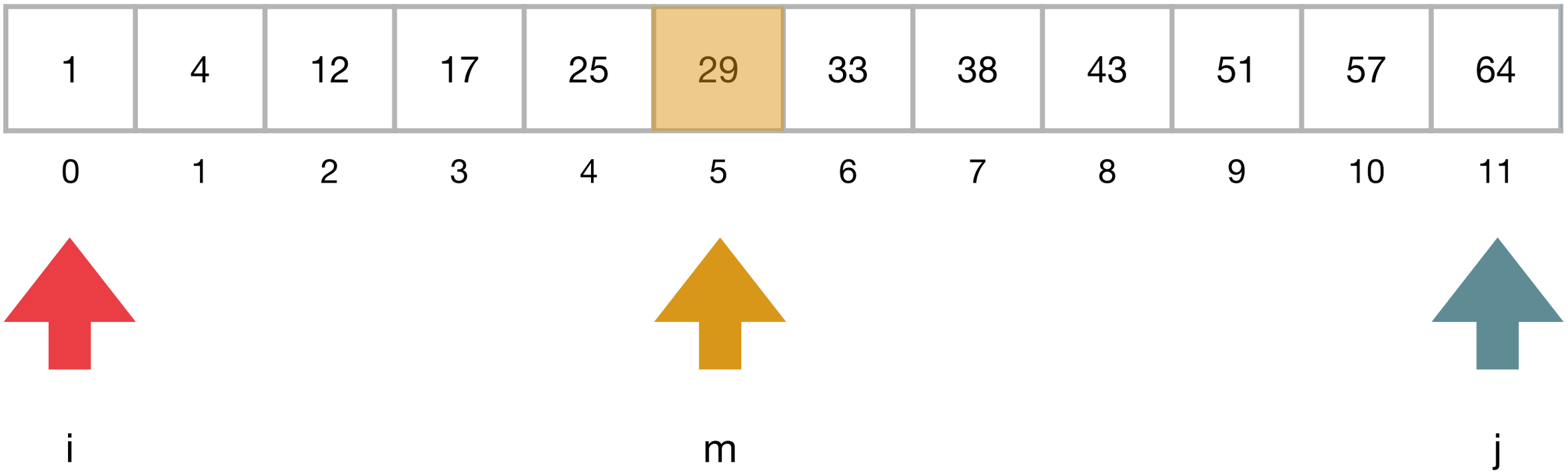
Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

i m j

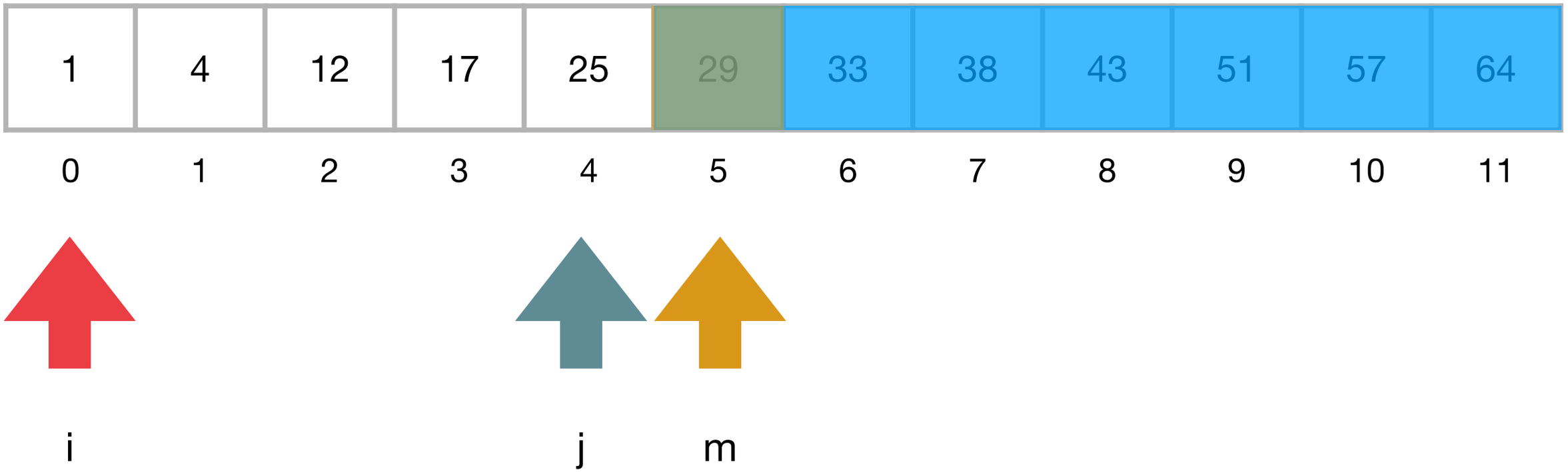
Recherche dichotomique

Recherche de 16



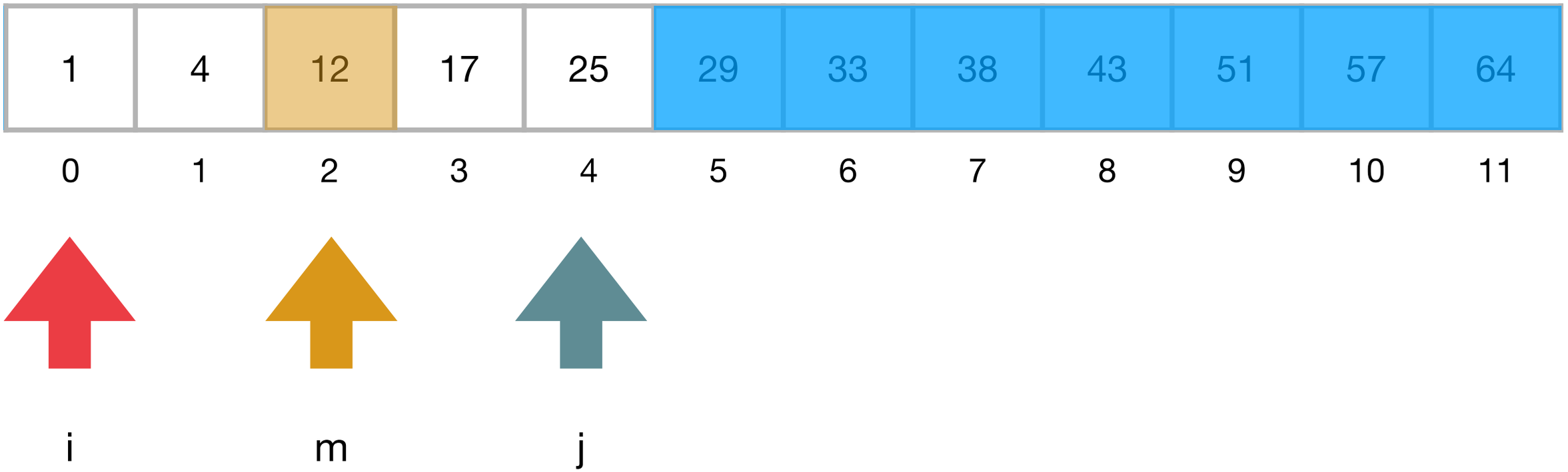
Recherche dichotomique

Recherche de 16



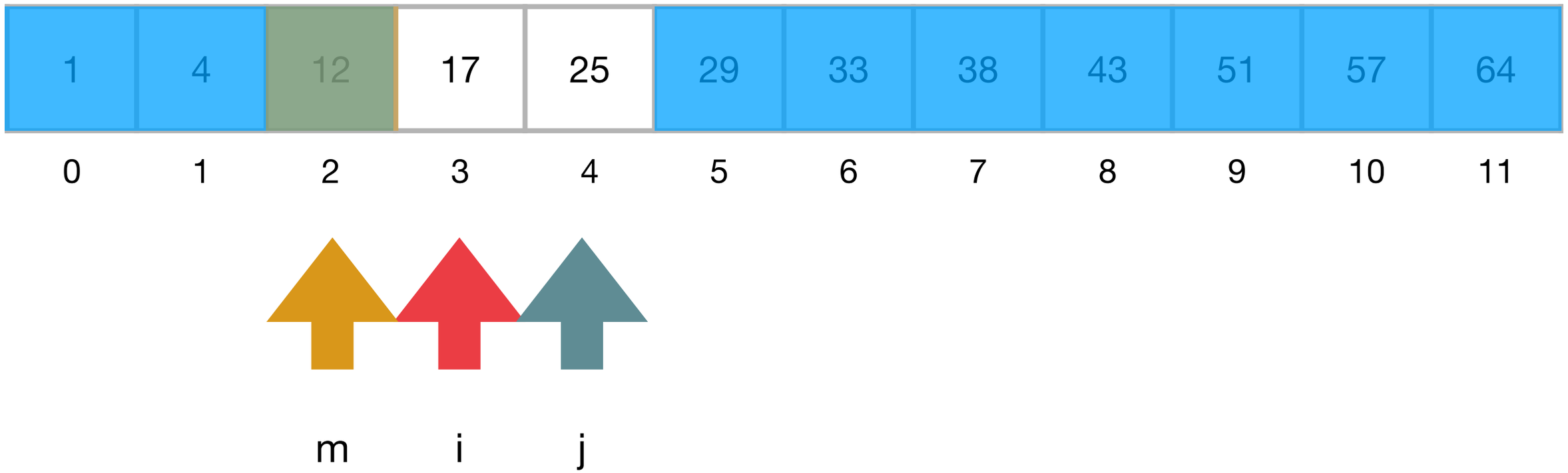
Recherche dichotomique

Recherche de 16



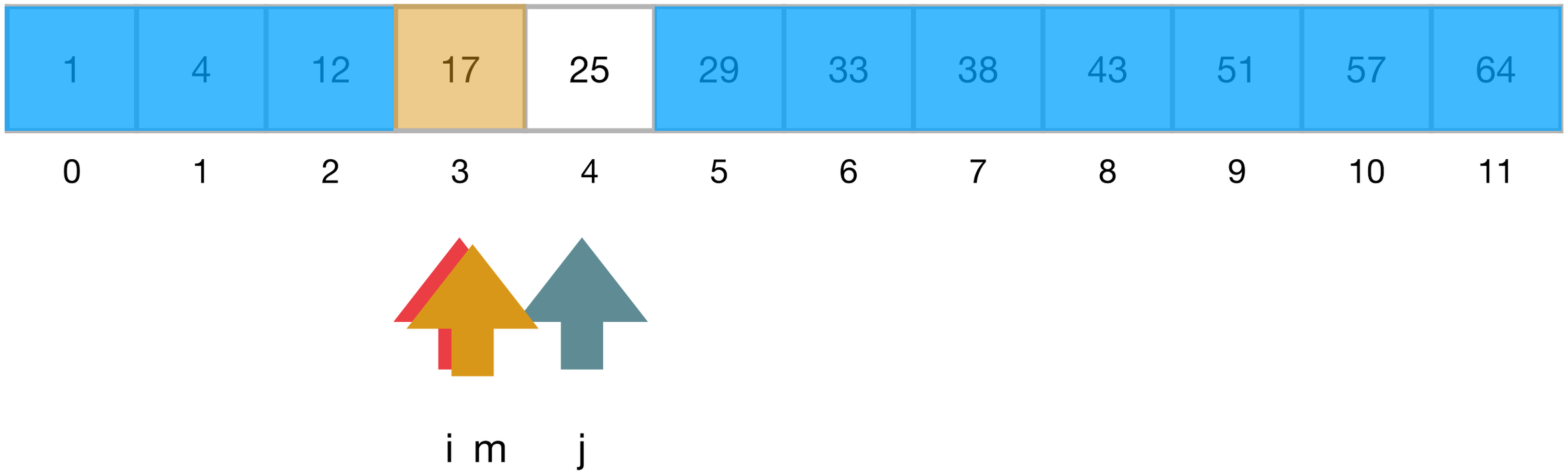
Recherche dichotomique

Recherche de 16



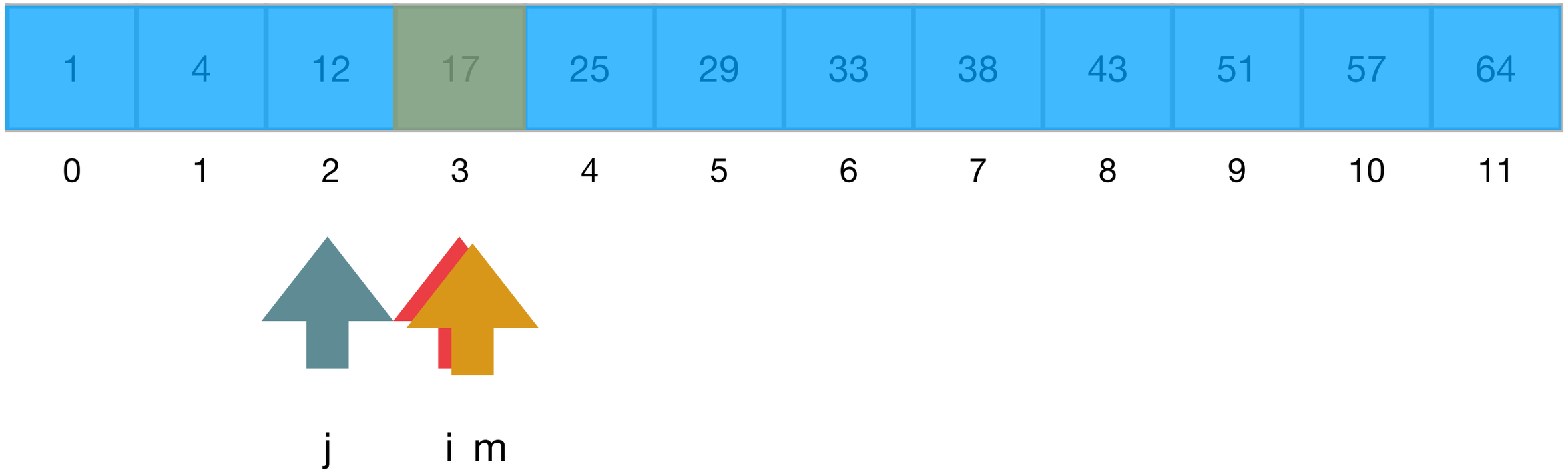
Recherche dichotomique

Recherche de 16



Recherche dichotomique

Recherche de 16



Recherche dichotomique

Recherche de 16

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

j i m

Recherche dichotomique dans un tableau d'entiers trié

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
return -1
```

Terminaison ?

Correction ?

Efficacité ?

Terminaison

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
return -1
```

- On termine quand $i > j$, c-à-d quand $j - i < 0$
- À chaque itération, soit i est incrémentée, soit j est décrémentée strictement
- Soit on trouve x, et on s'arrête immédiatement, soit il n'est pas là, et donc tôt ou tard $i > j$

Correction

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
return -1
```

- Invariant de boucle : si A est trié et que x est dans A, alors il se trouve dans le sous-tableau $A[i, \ldots, j]$
  - C'est vrai au début de l'algorithme, puisque $A[i, \ldots, j] = A$
  - Ça reste vrai à chaque itération de la boucle, parce qu'on vérifie toujours si $x = A[m]$ ou $x < A[m]$ ou $x > A[m]$
- Si on sort de la boucle avec $i > j$, alors si x est dans le tableau, il est dans le sous-tableau vide $A[i, j]$, c'est à dire qu'il n'est pas là

Efficacité

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

- Dans le pire des cas, x n'est pas là
- Comme on élimine à chaque itération la moitié du tableau, on exécute la boucle $\log_2 n$ fois au maximum
- Ça fait $O(\log_2 n)$ opérations

Donc la recherche prend $O(n)$ pour un tableau quelconque, $O(\log_2 n)$ pour un tableau trié

Algorithmes de tri

Algorithmes de tri pour accélérer la recherche dans un tableau

- La recherche dans un tableau non trié prend temps O(n) avec la recherche séquentielle (ou linéaire)

- Par contre, on peut faire une recherche dichotomique dans un tableau trié en temps $O(\log_2 n)$

- Donc ça vaut la peine de trier le tableau si on a beaucoup de recherches à faire

Algorithmes de tri dans le commerce électronique

amazonie.fr

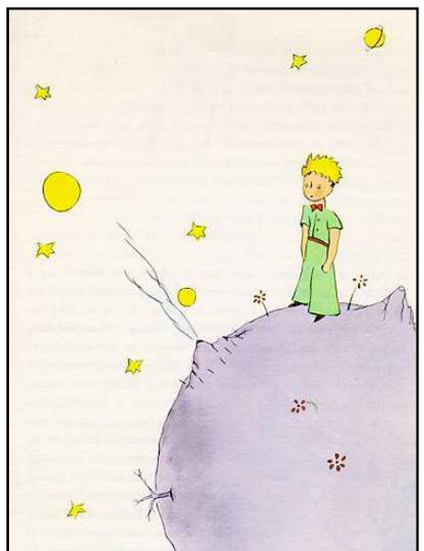
amazonie

Chercher : Le Petit Prince

Résultats 1–20 sur 928572785 pour « Le Petit Prince »

Trier par :

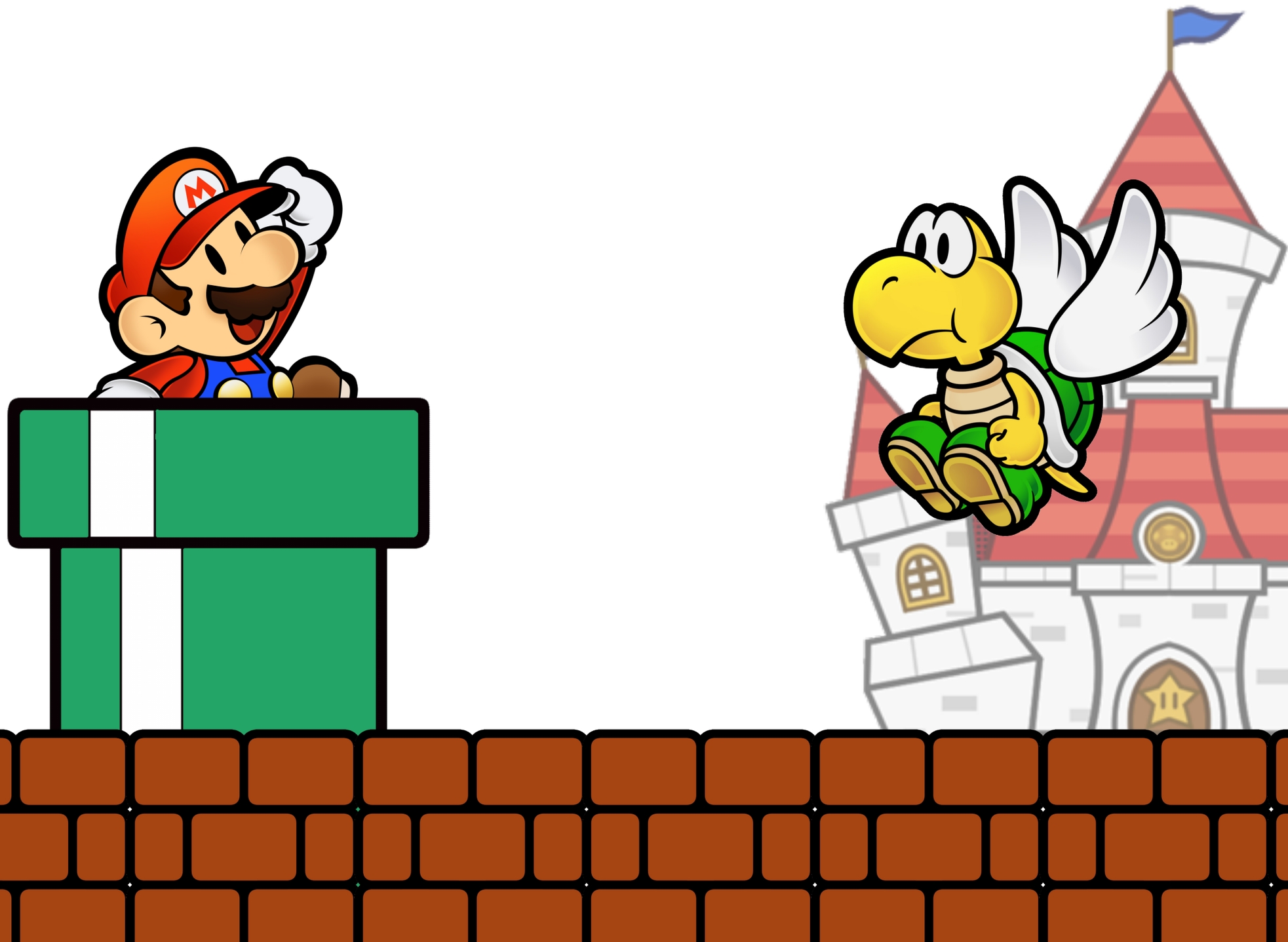
- prix croissant
- prix décroissant
- note moyenne
- nouveauté



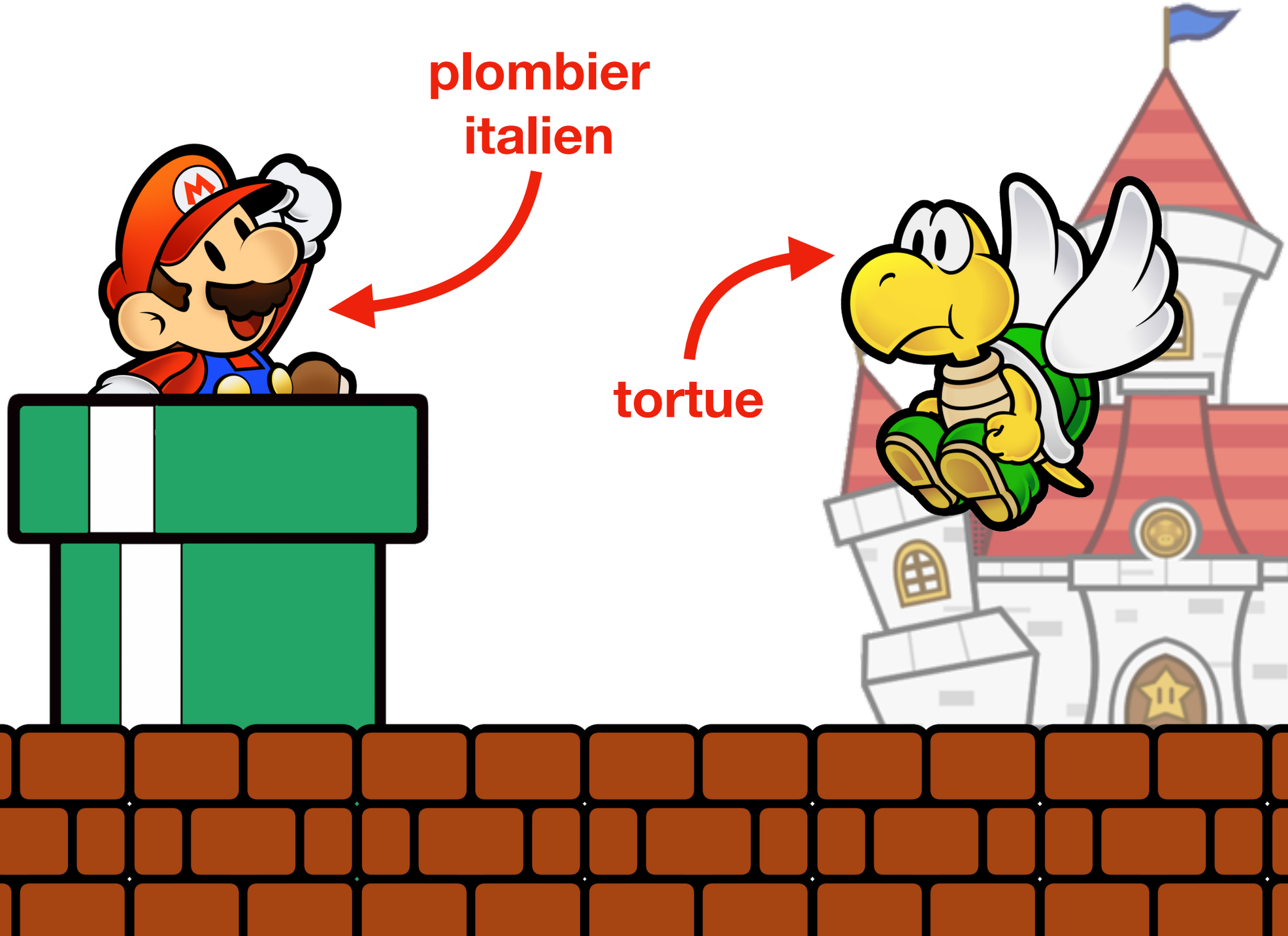
Le Petit Prince
de Antoine de Saint-Exu

Format poche 6,90 €
Format Kinder 6,49 €

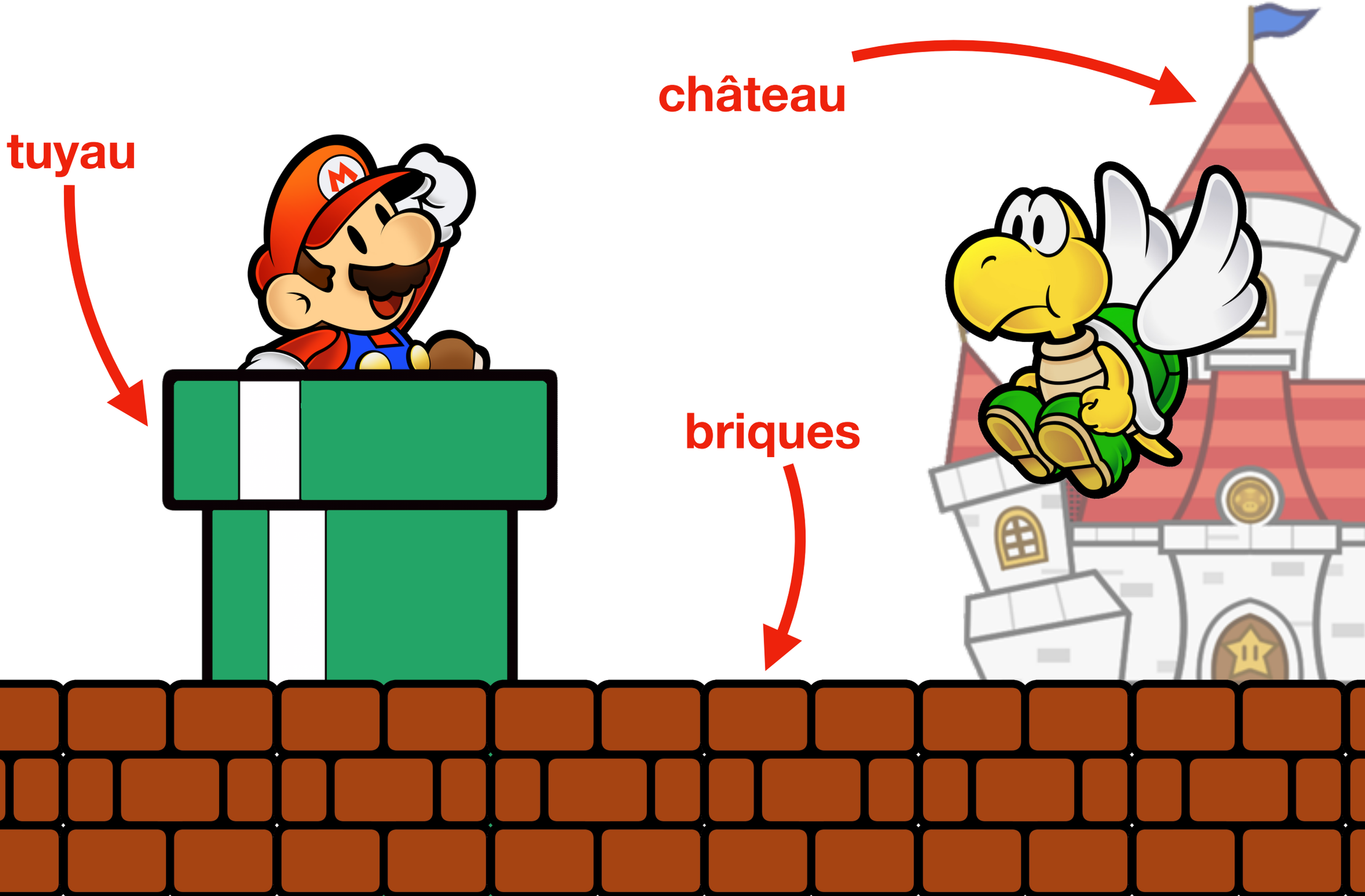
Algos de tri dans le jeux vidéo « Super Plombiers Italiens »



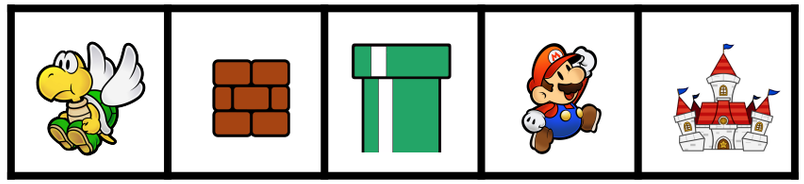
Algos de tri dans le jeux vidéo « Super Plombiers Italiens »



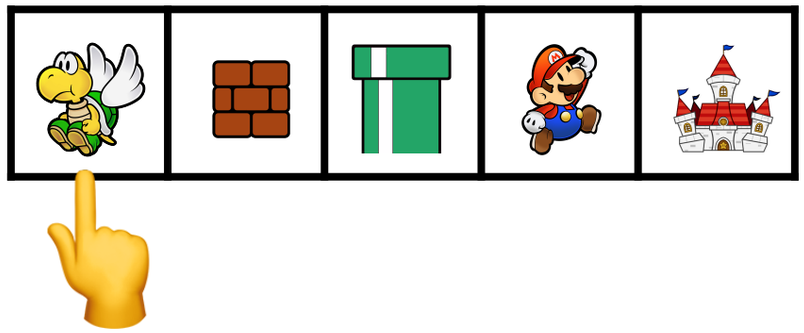
Algos de tri dans le jeux vidéo « Super Plombiers Italiens »



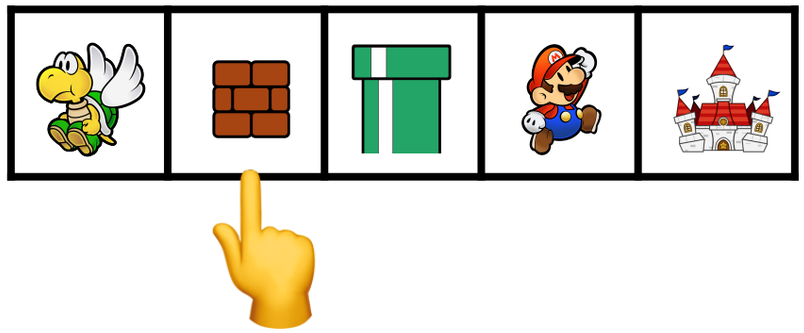
Affichage des objets dans le mauvais ordre



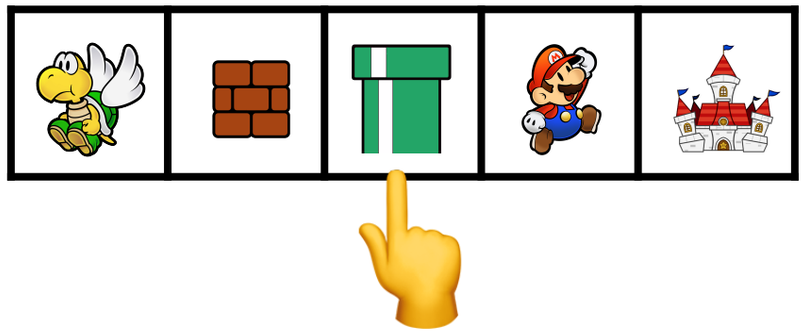
Affichage des objets dans le mauvais ordre

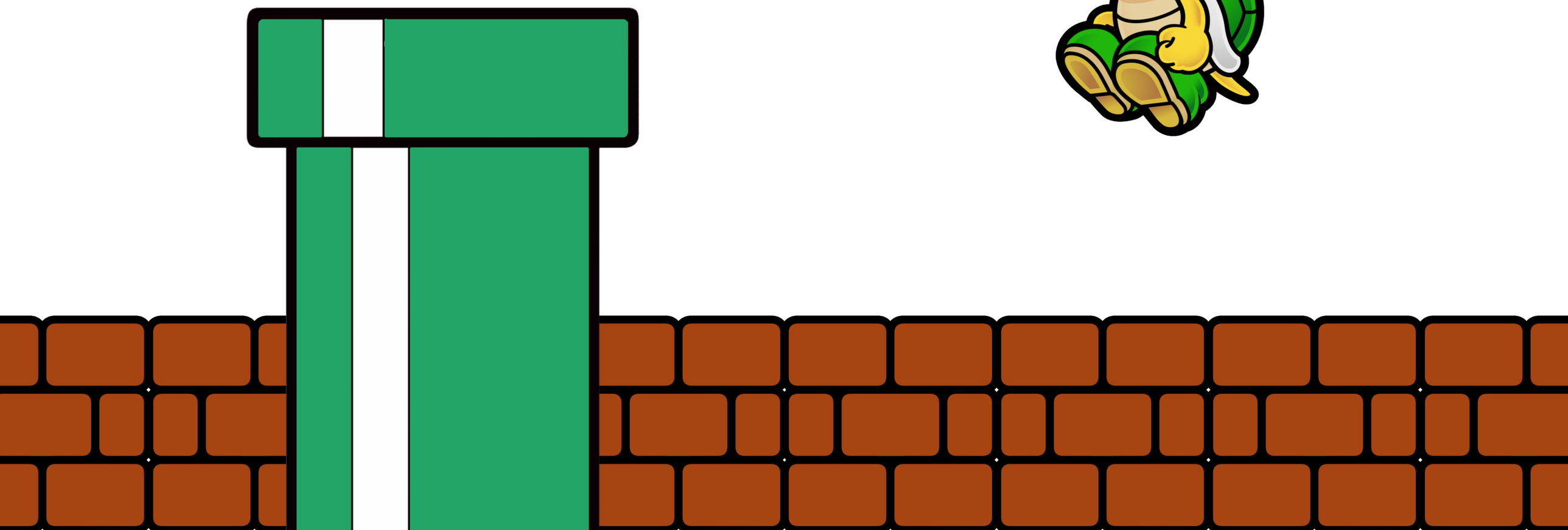


Affichage des objets dans le mauvais ordre

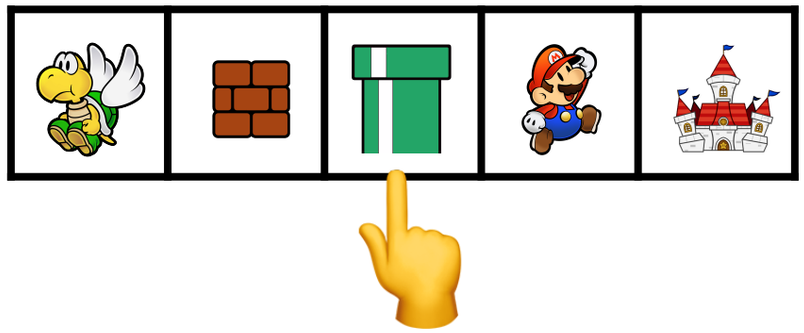


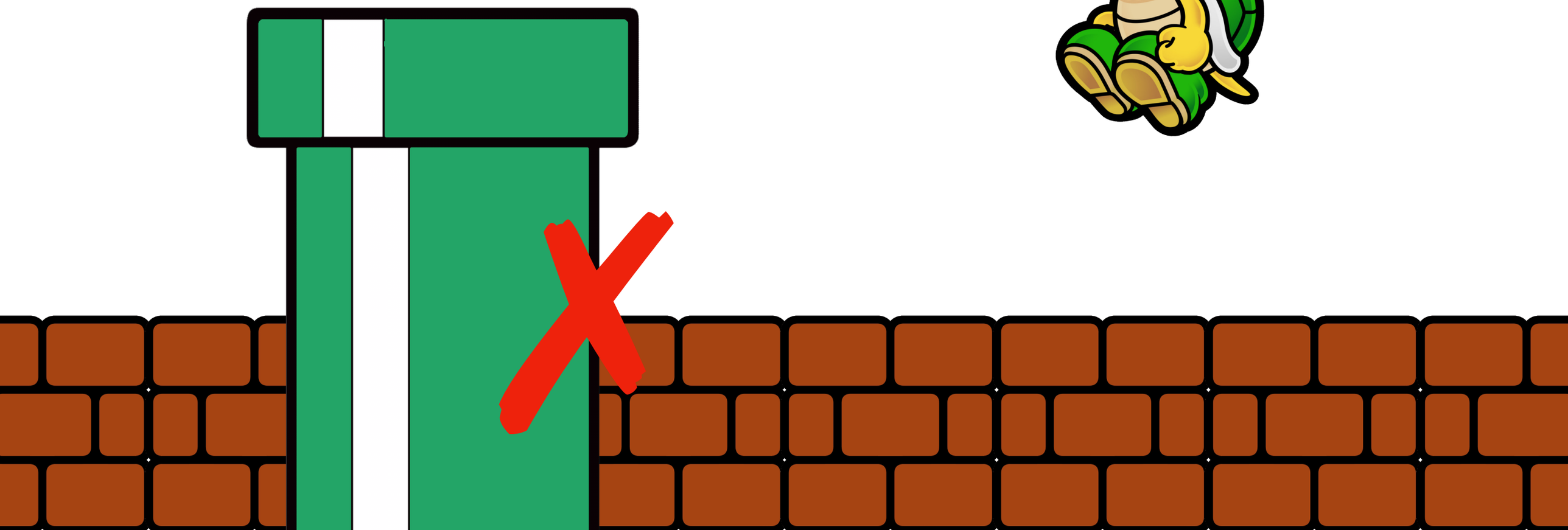
Affichage des objets dans le mauvais ordre



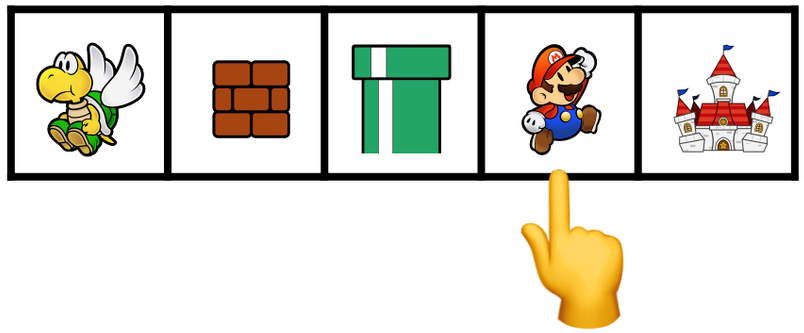


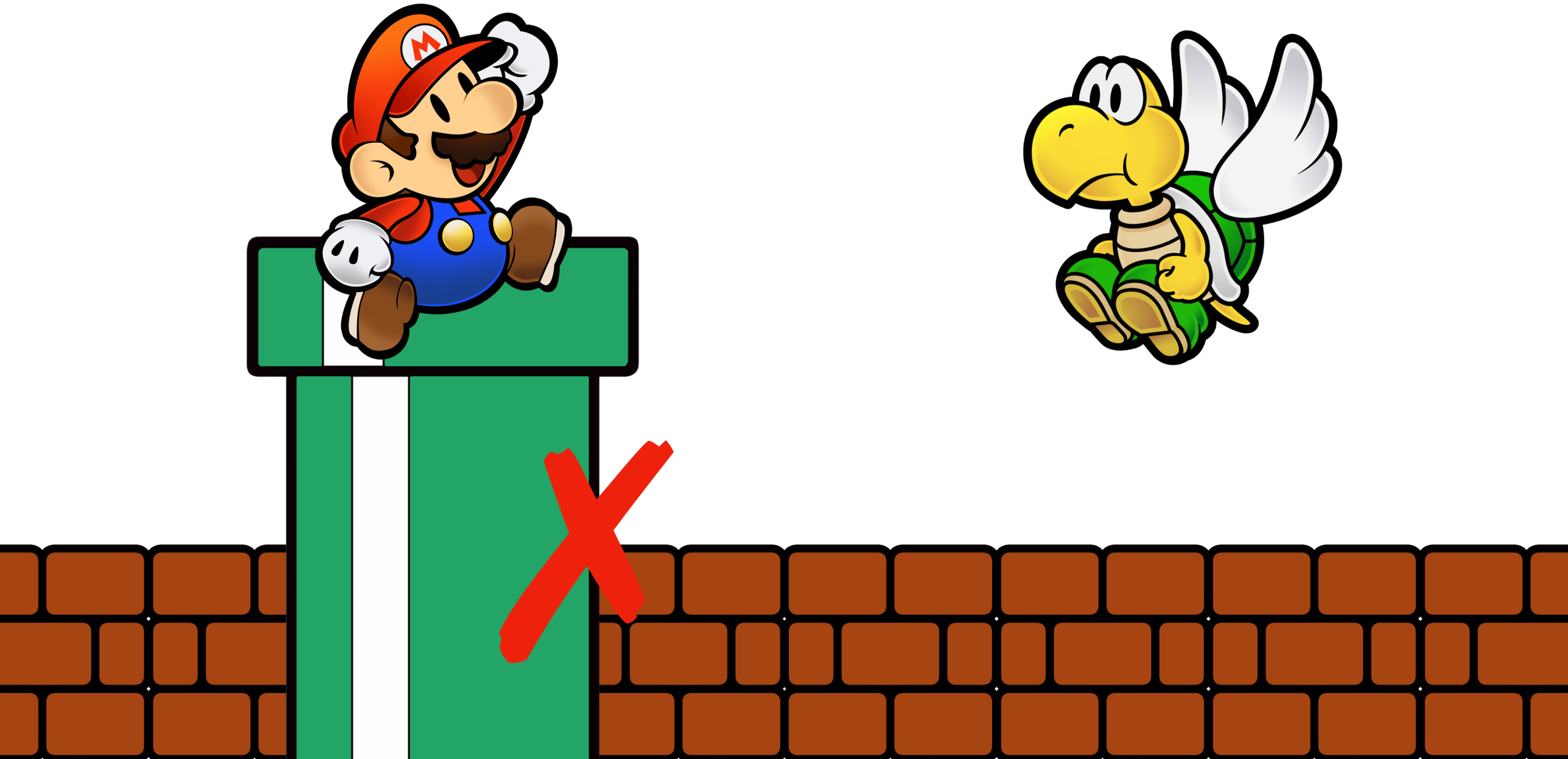
Affichage des objets dans le mauvais ordre



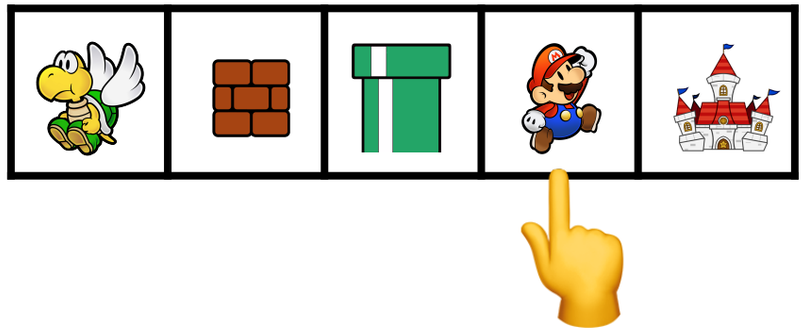


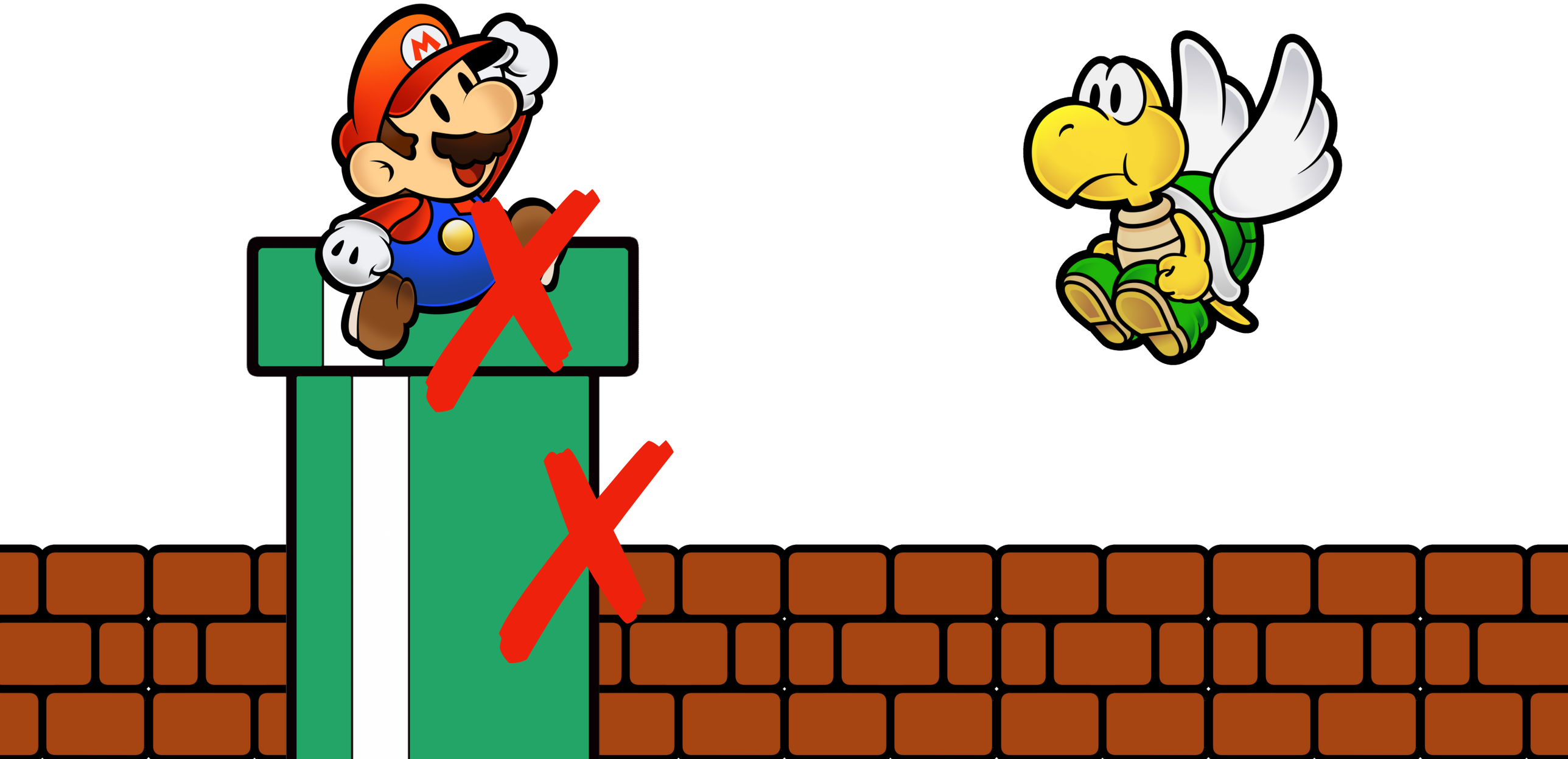
Affichage des objets dans le mauvais ordre



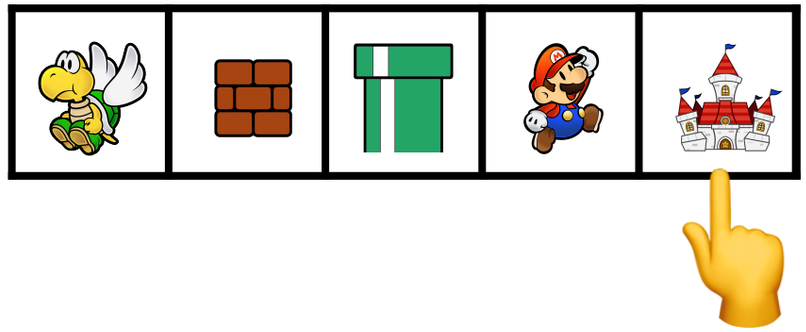


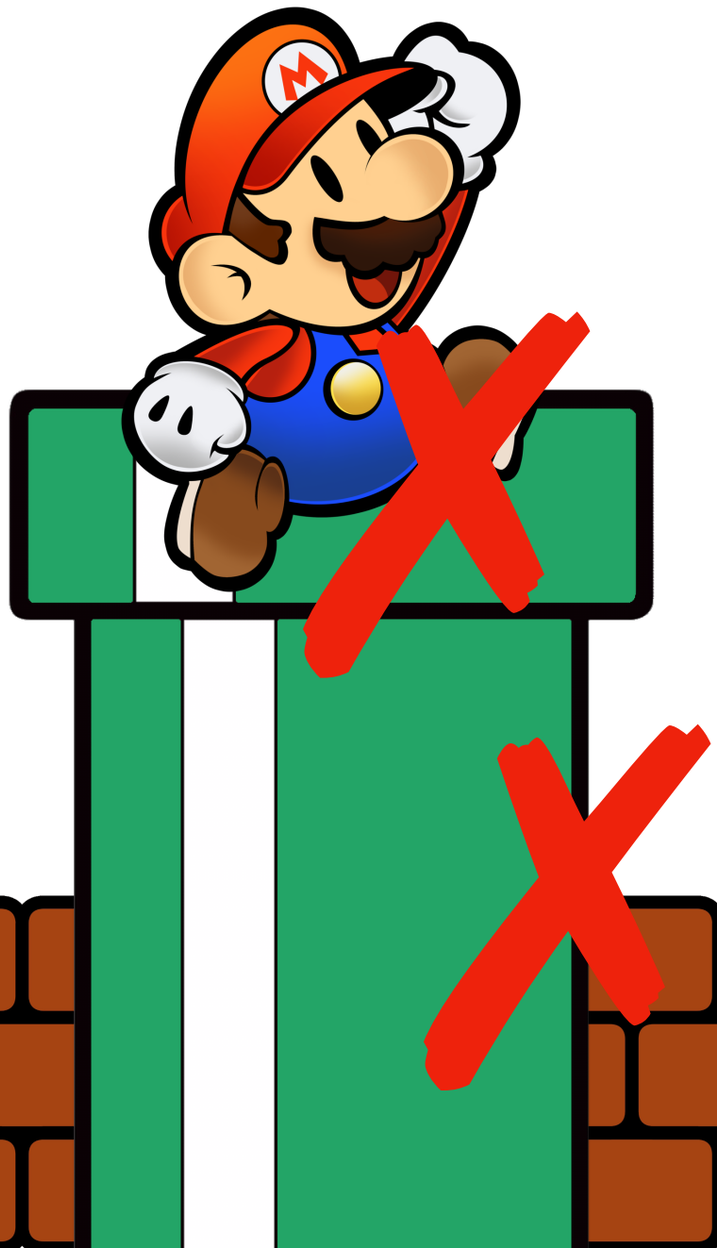
Affichage des objets dans le mauvais ordre



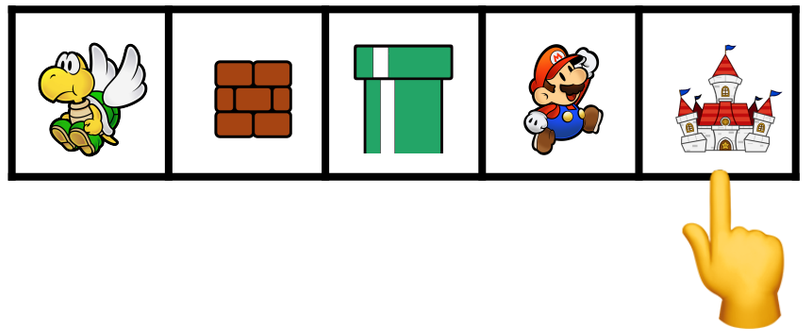


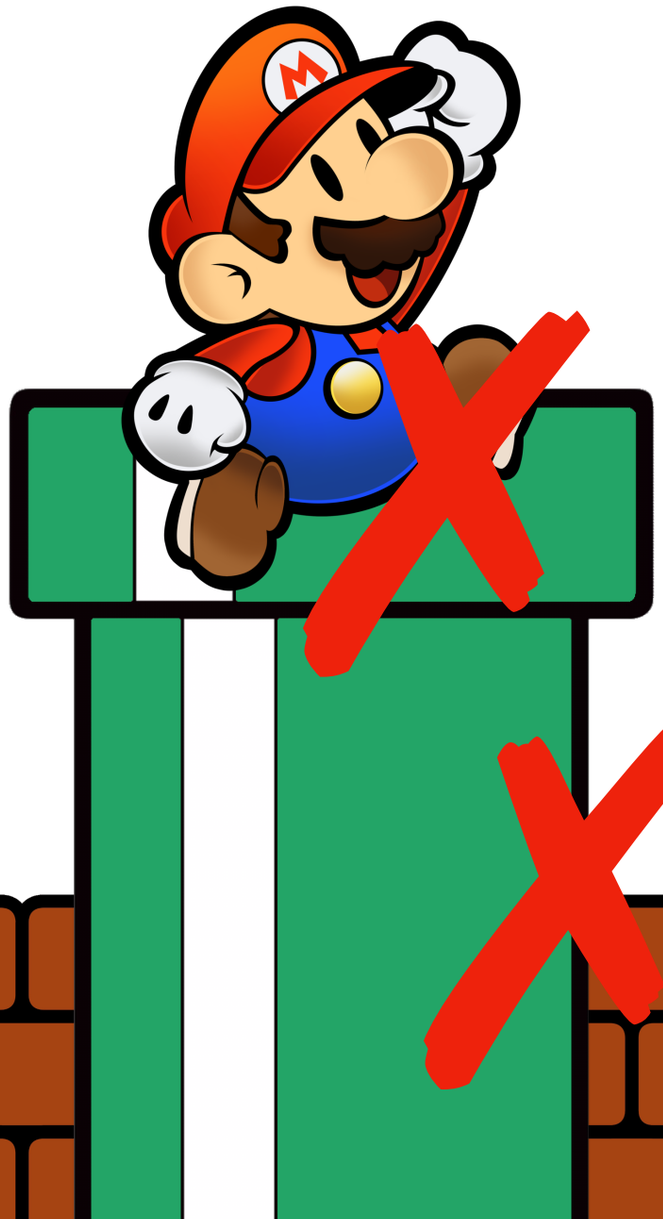
Affichage des objets dans le mauvais ordre



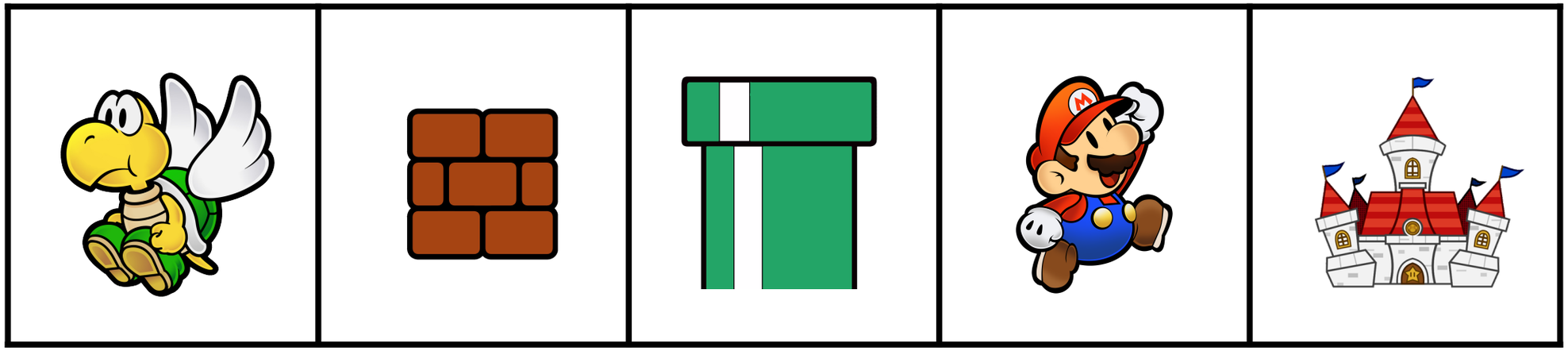


Affichage des objets dans le mauvais ordre



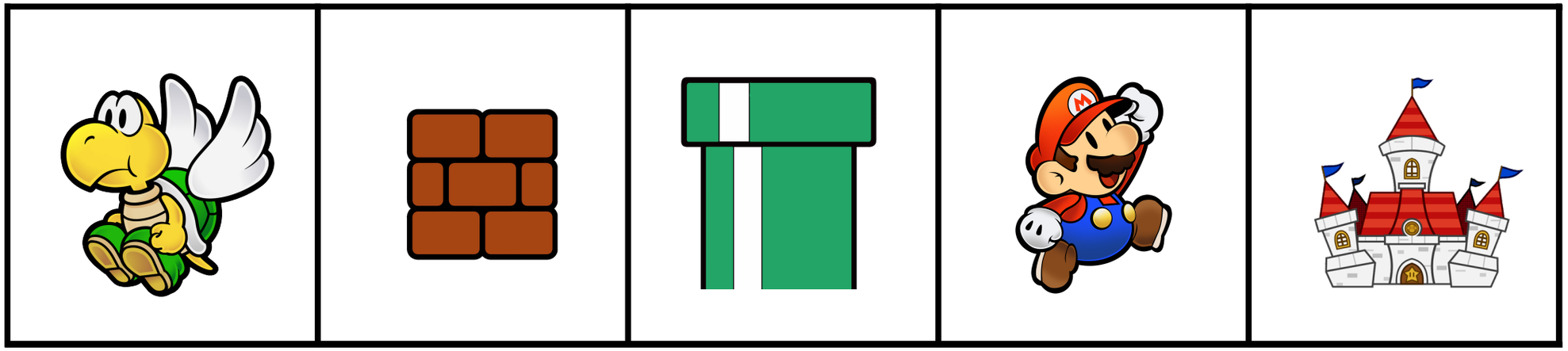


Affichage des objets dans le mauvais ordre



0 1 2 3 4

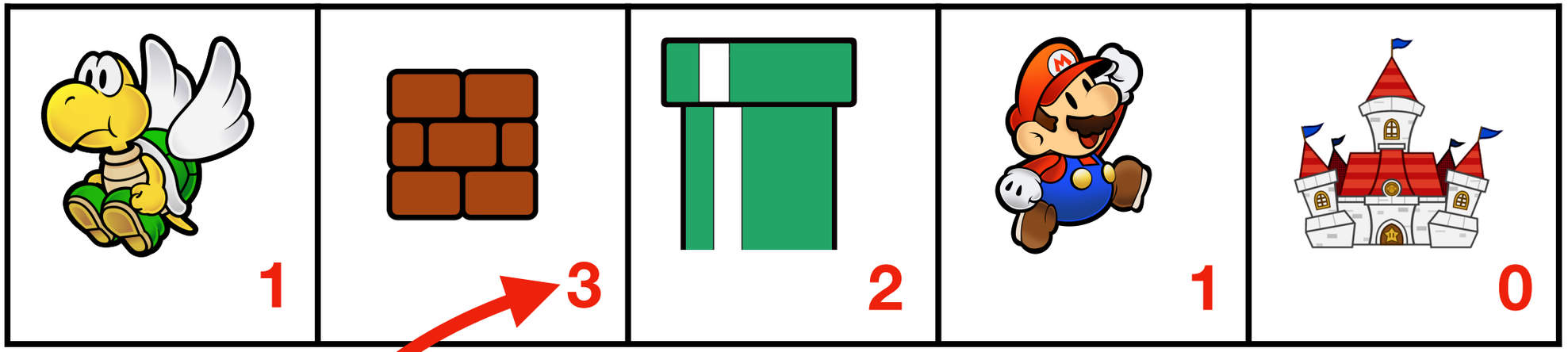
Affichage des objets dans le mauvais ordre



0 1 2 3 4

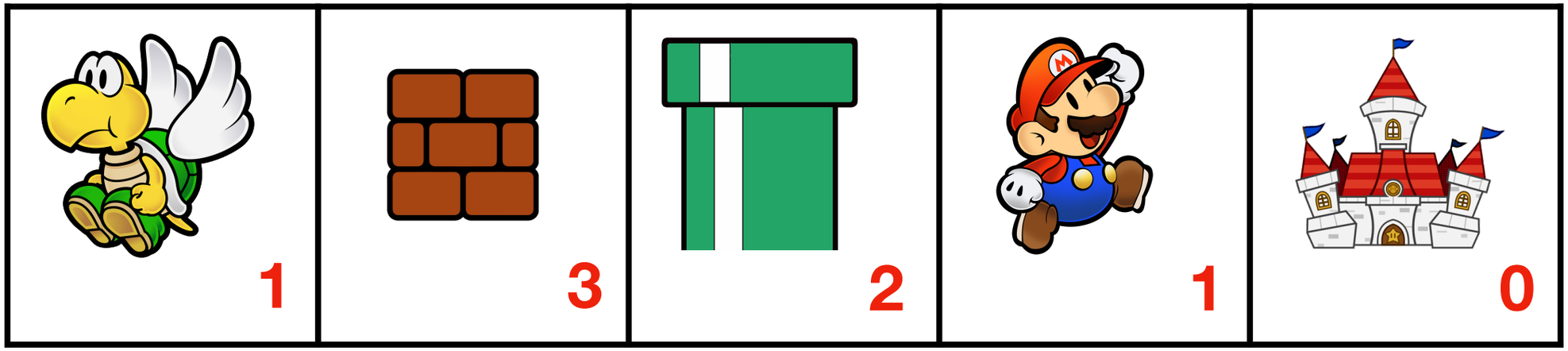
ordre d’affichage

Affichage des objets dans le mauvais ordre



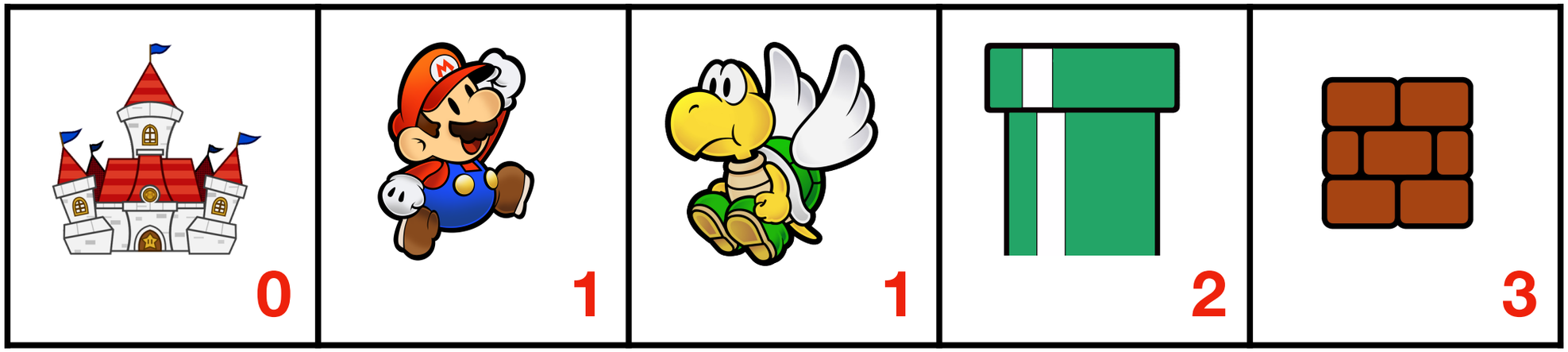
Tri ! 

Affichage des objets dans le mauvais ordre



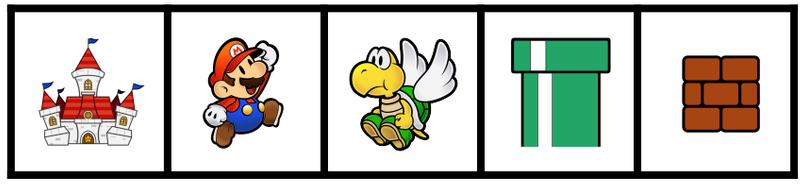
0 1 2 3 4

Affichage des objets dans le bon ordre

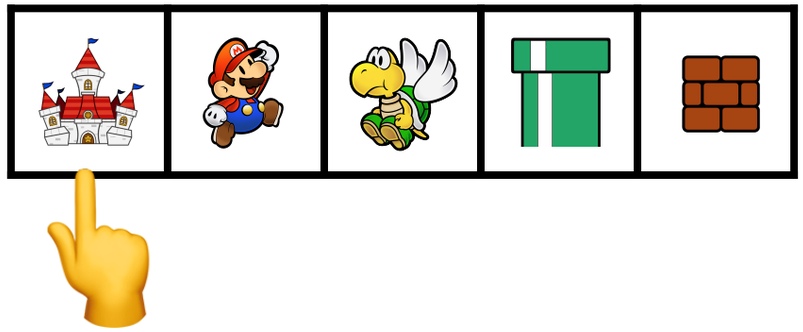


0 1 2 3 4

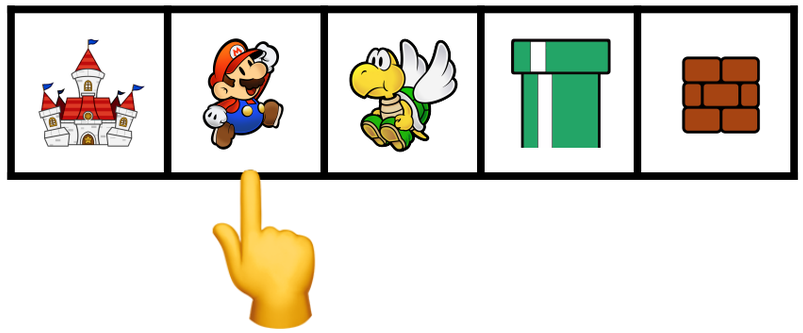
Affichage des objets dans le bon ordre



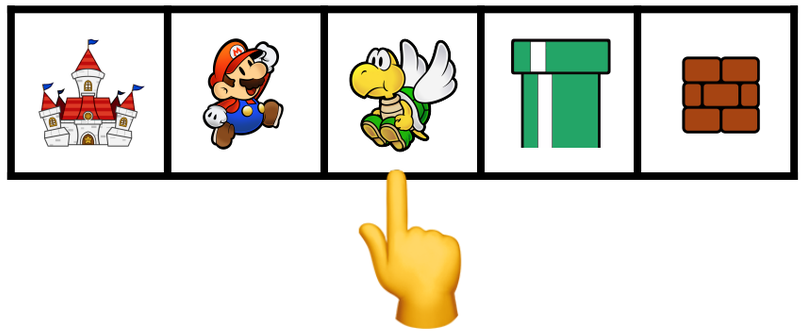
Affichage des objets dans le bon ordre

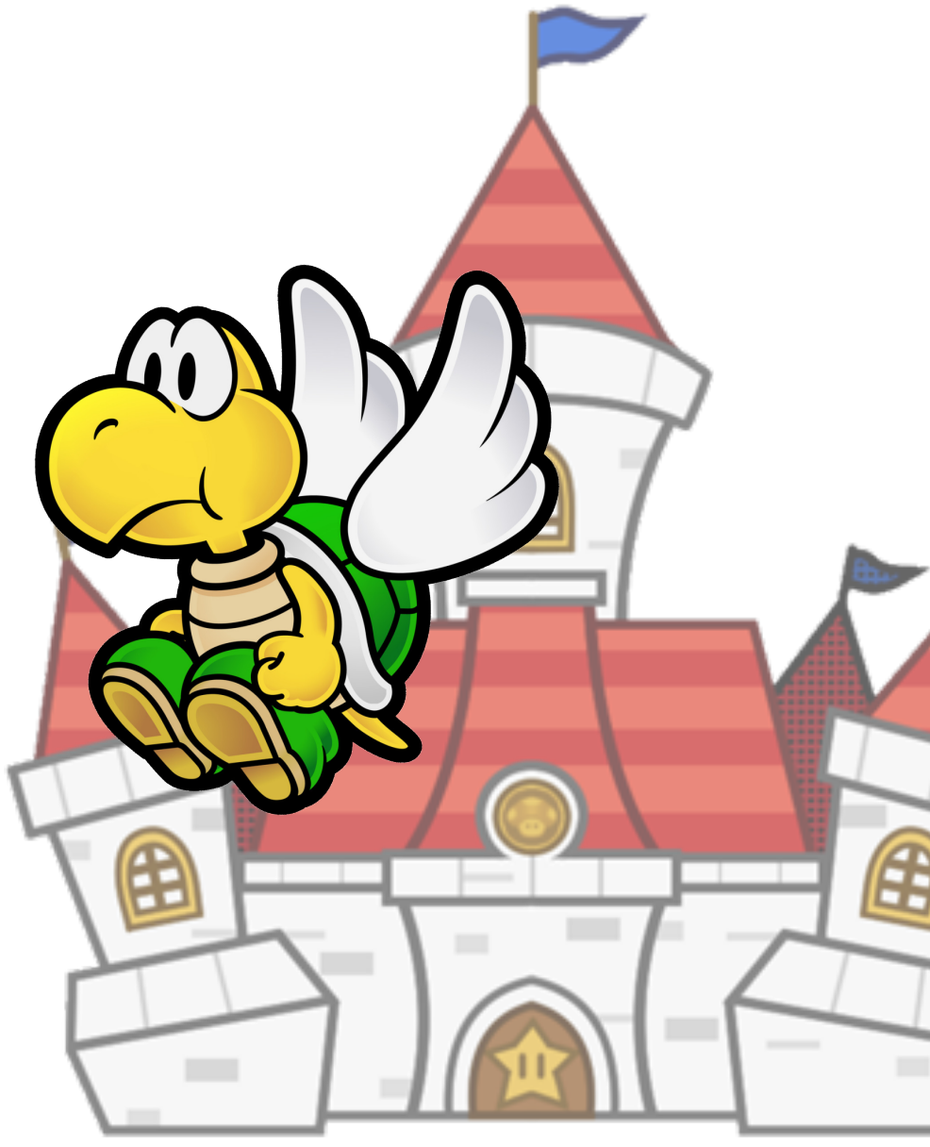


Affichage des objets dans le bon ordre

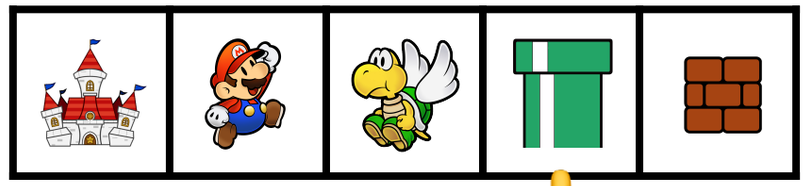


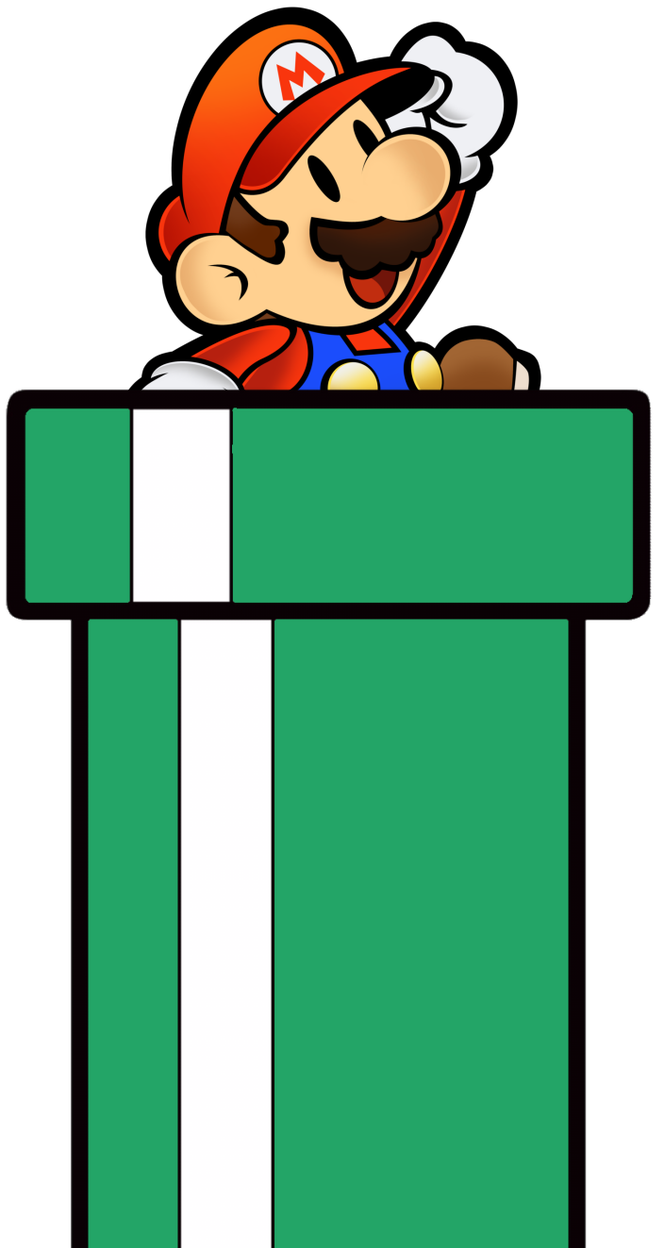
Affichage des objets dans le bon ordre



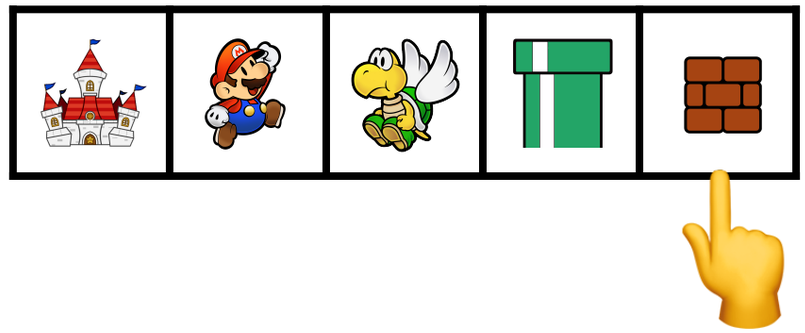


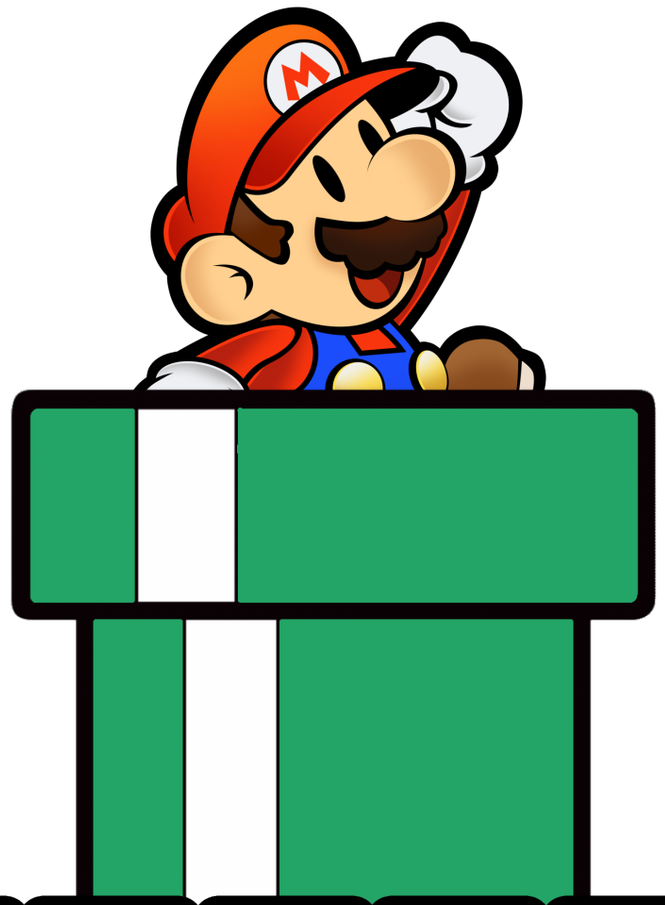
Affichage des objets dans le bon ordre

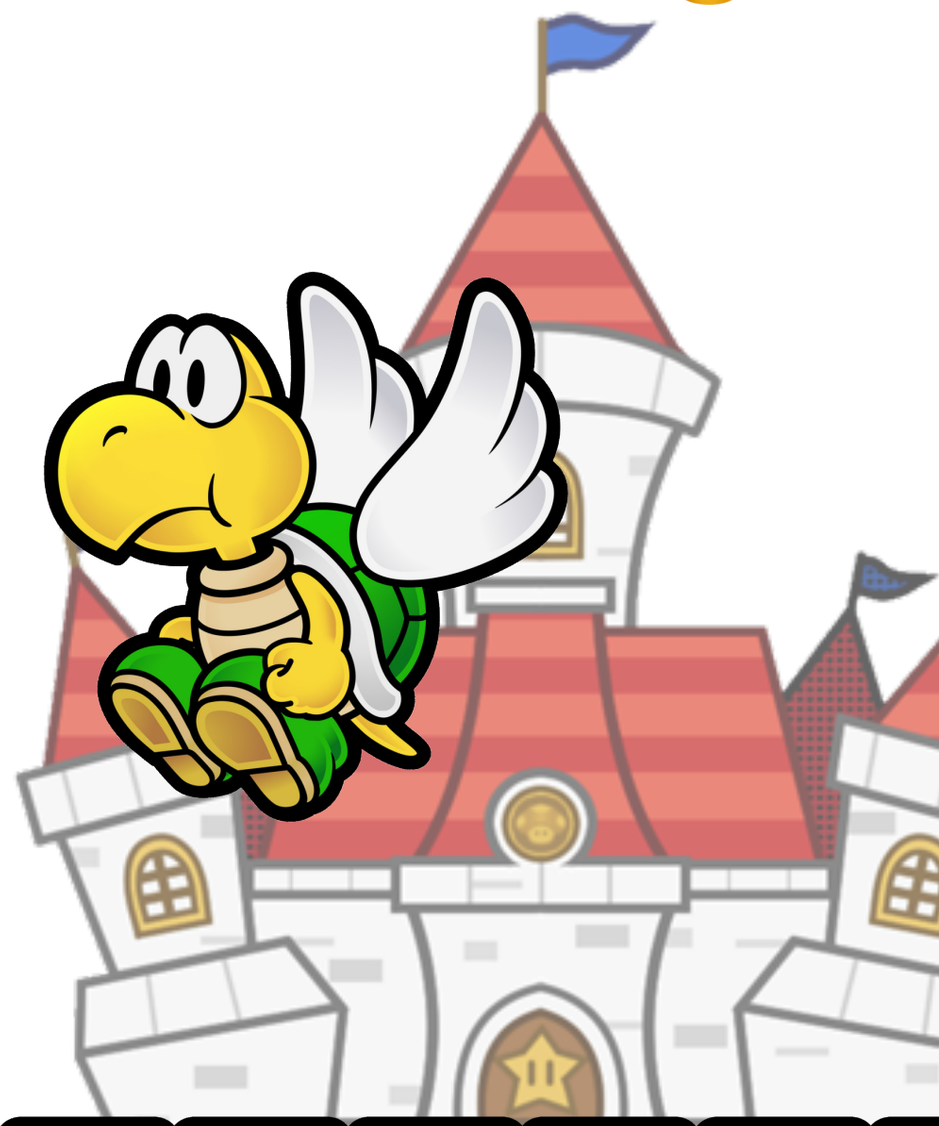




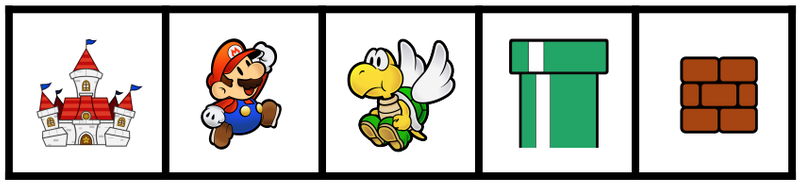
Affichage des objets dans le bon ordre

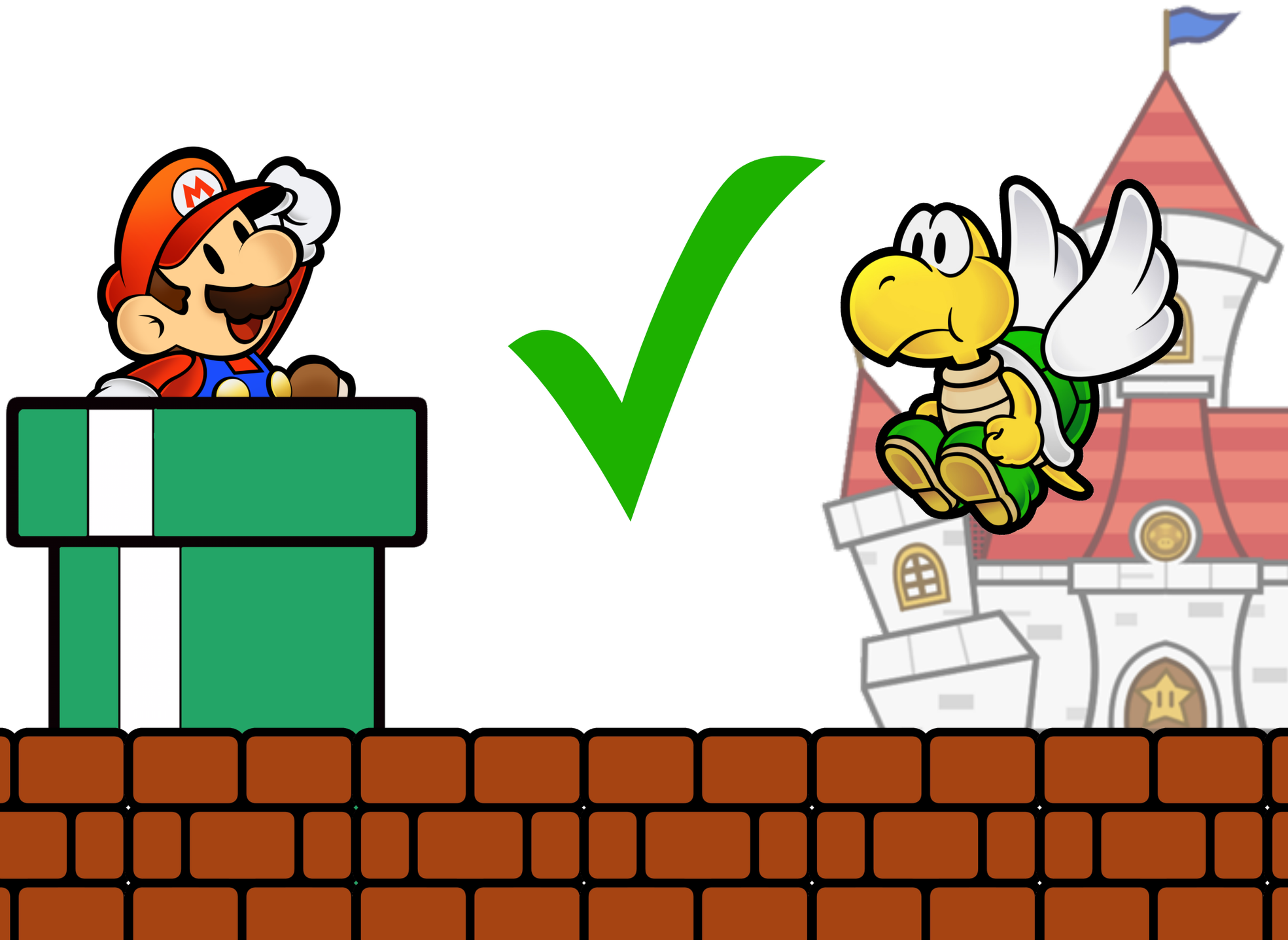




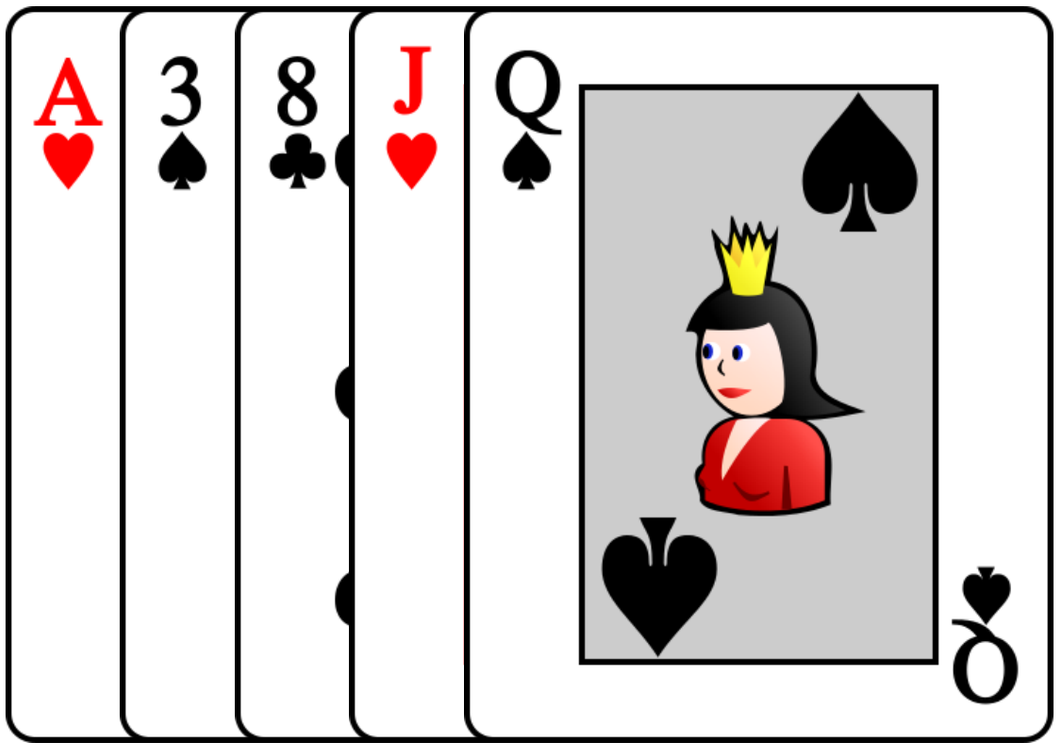


Affichage des objets dans le bon ordre



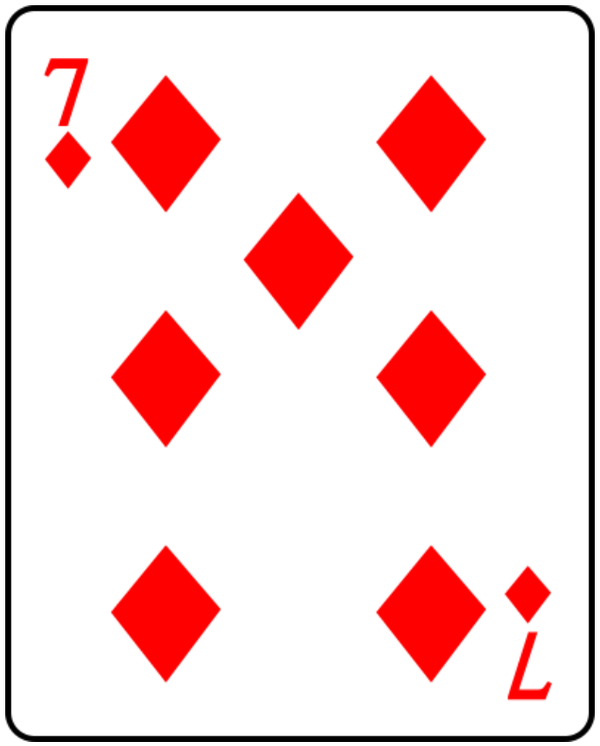


Comment trier un jeu de cartes ?

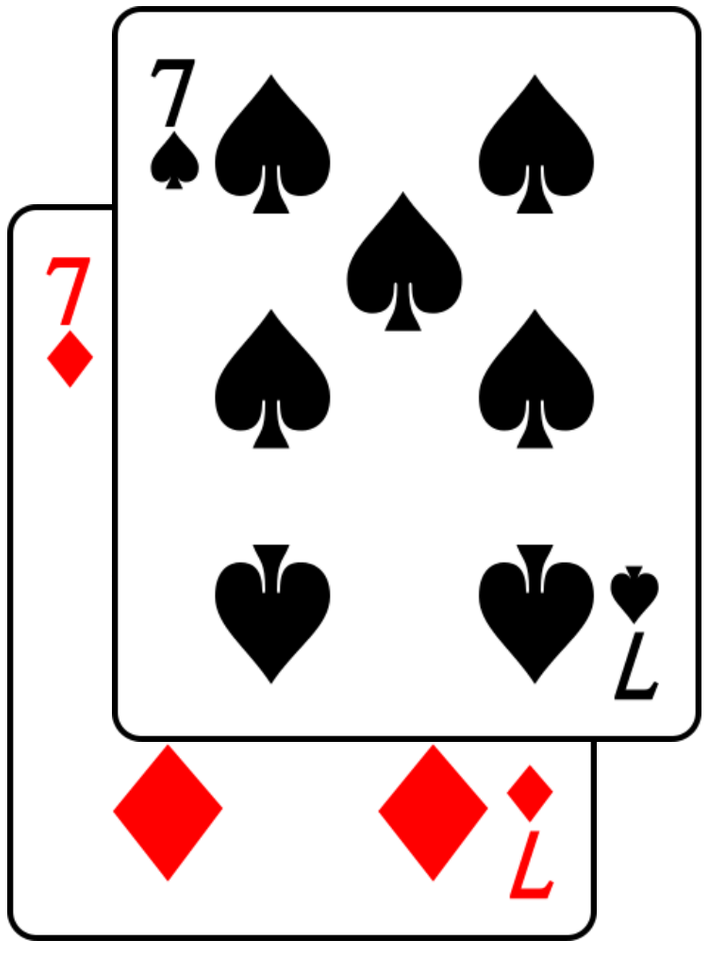


Tri par insertion

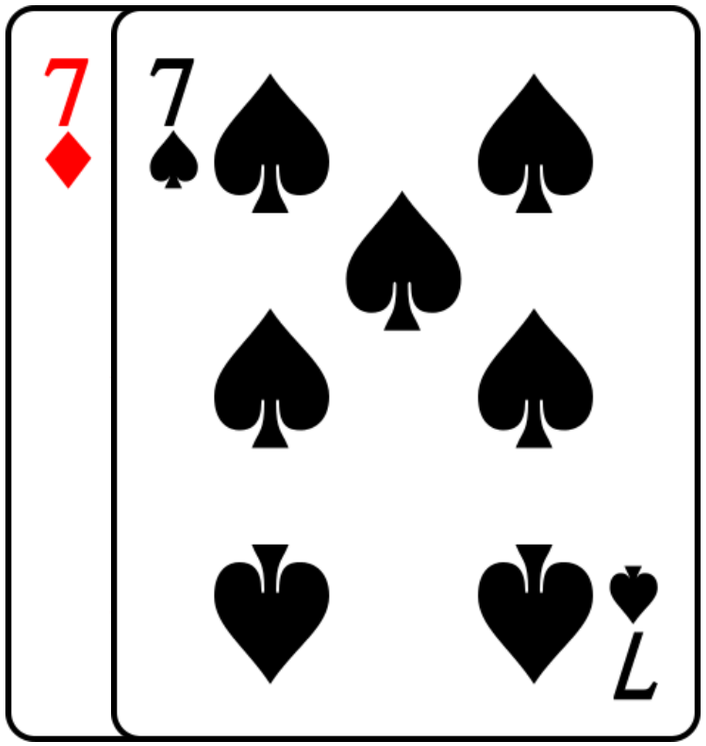
Tri par insertion



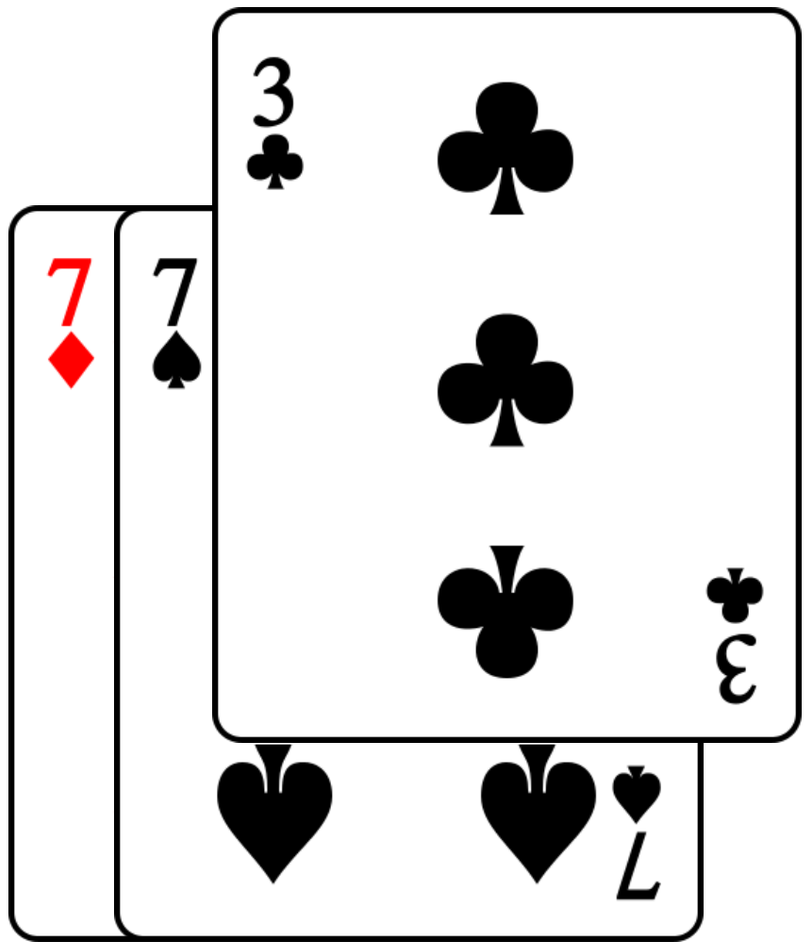
Tri par insertion



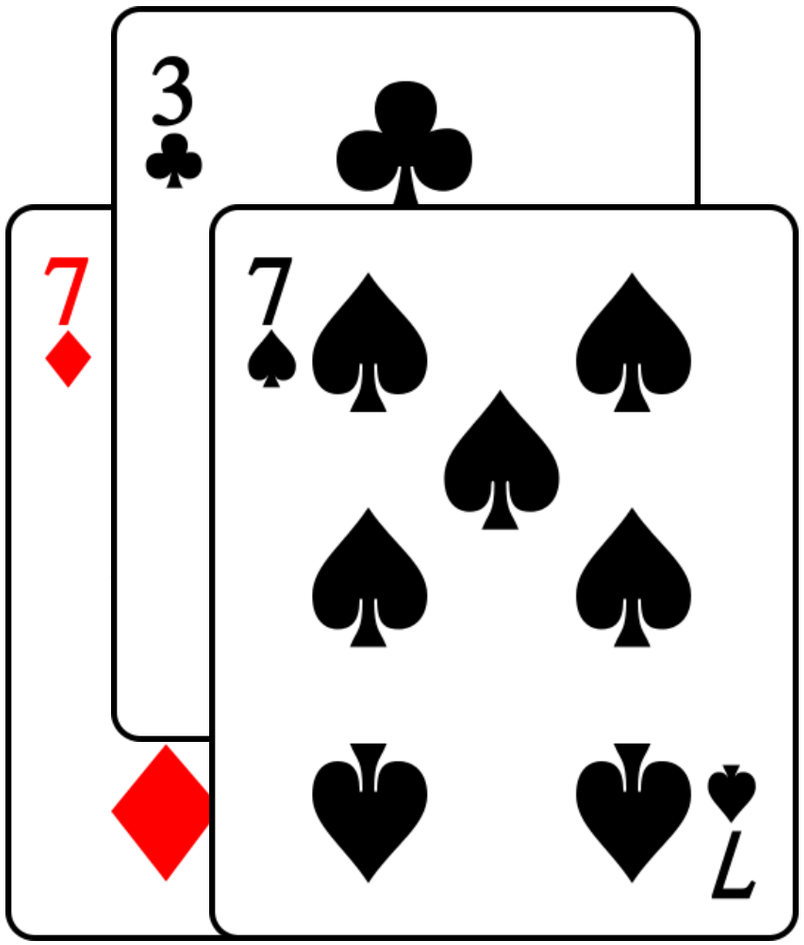
Tri par insertion



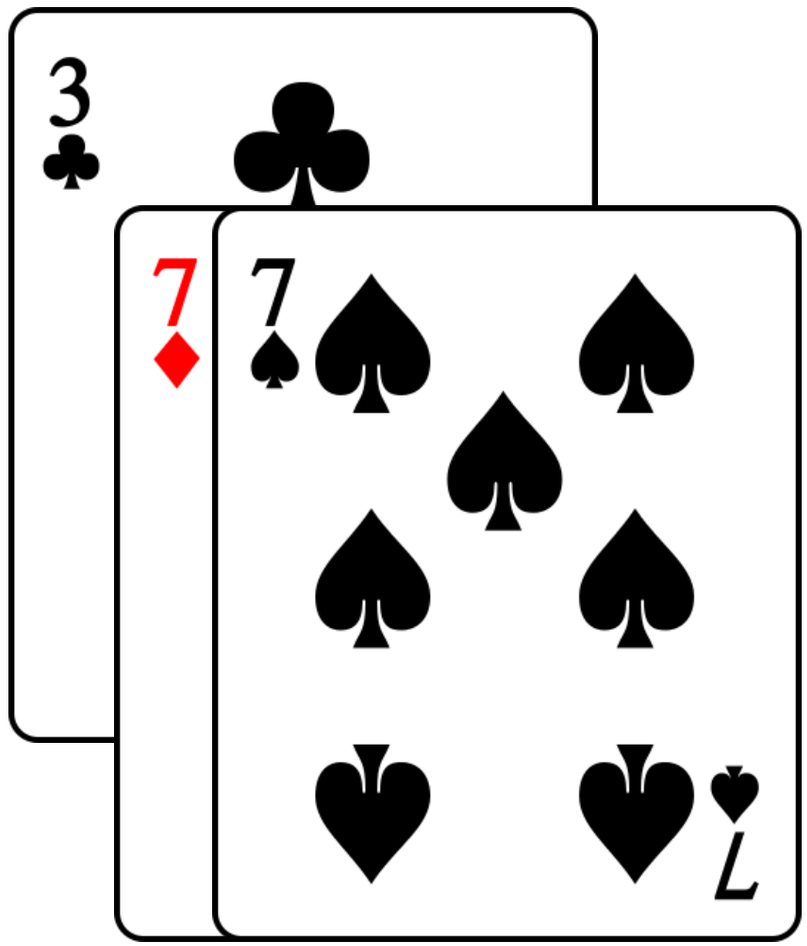
Tri par insertion



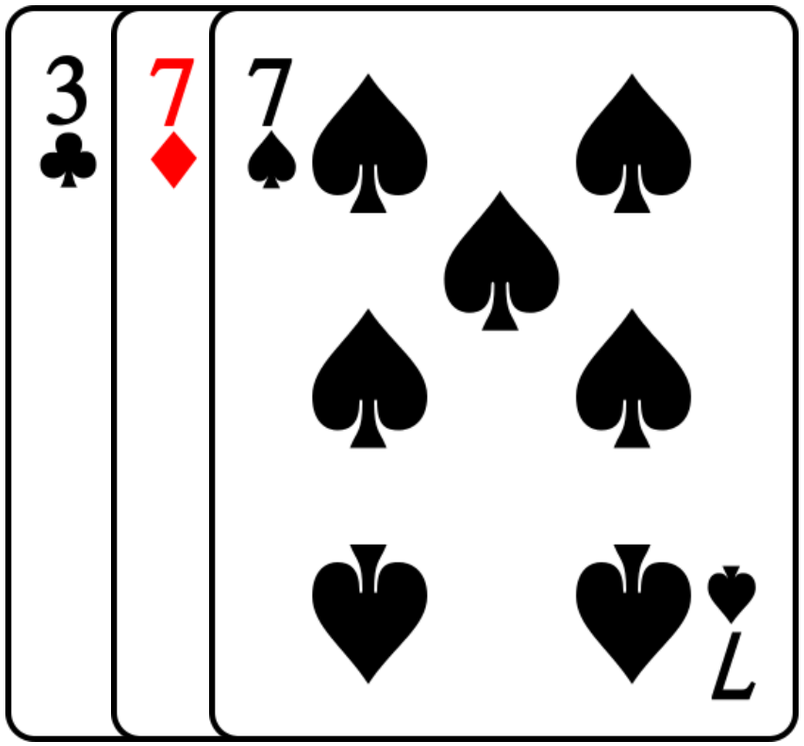
Tri par insertion



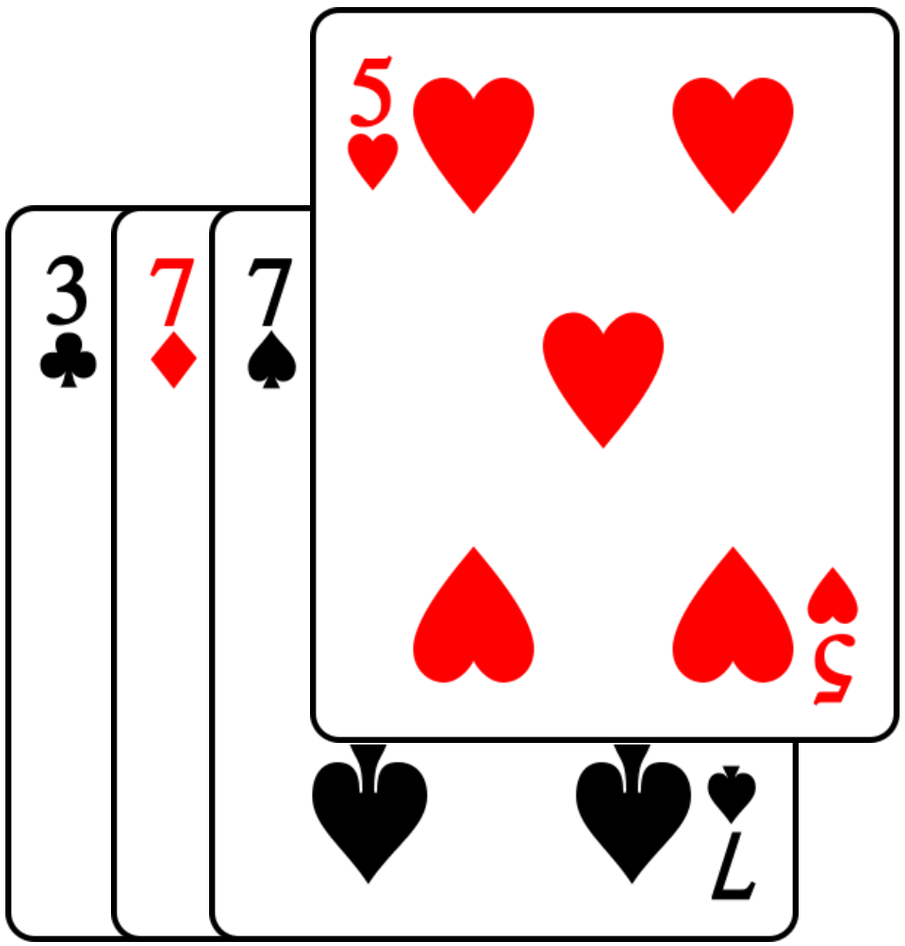
Tri par insertion



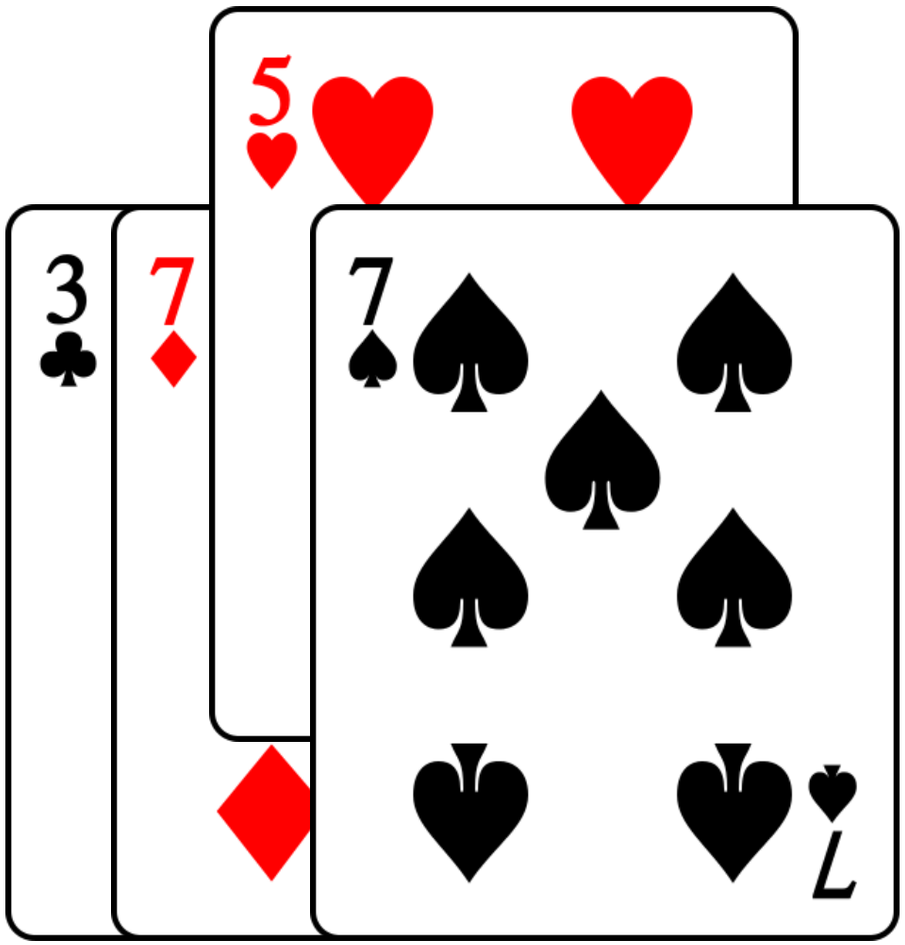
Tri par insertion



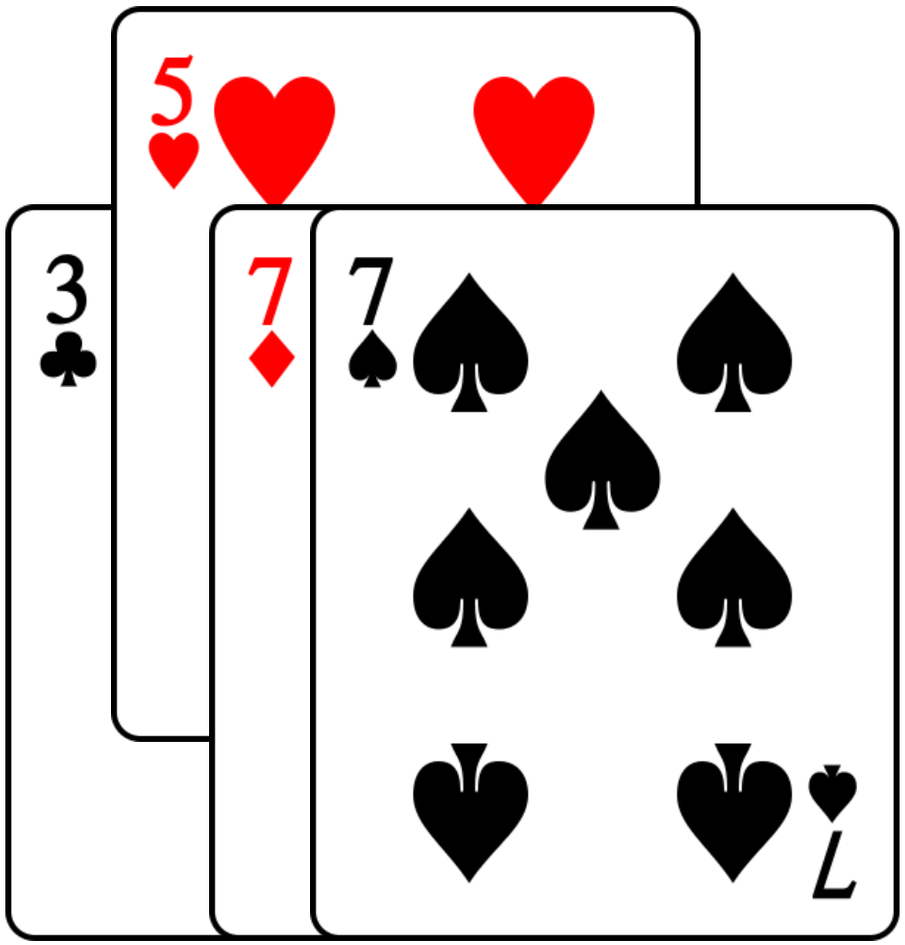
Tri par insertion



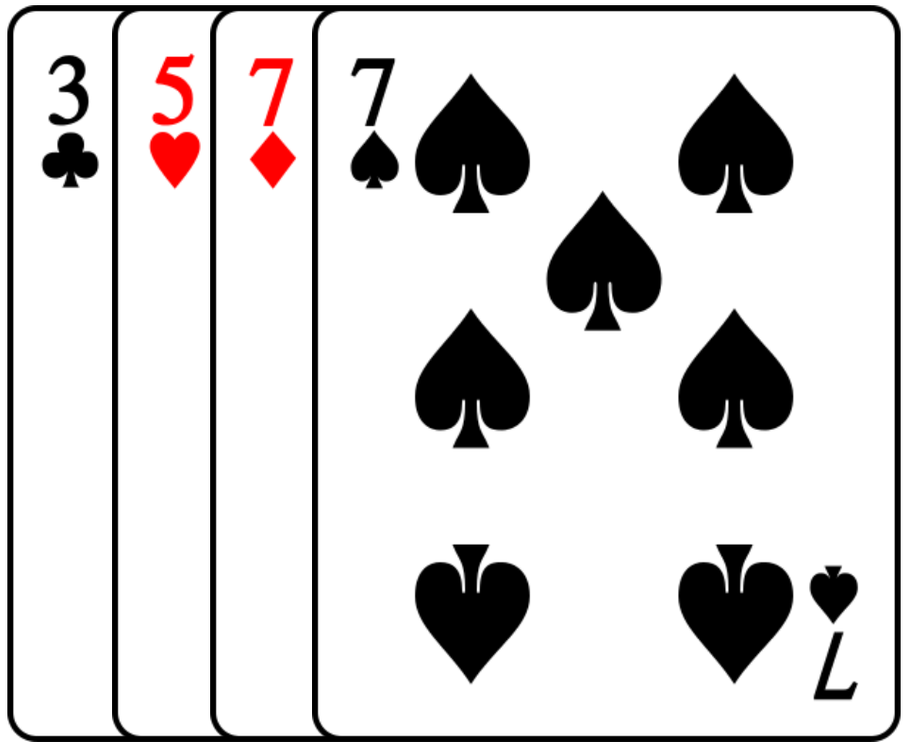
Tri par insertion



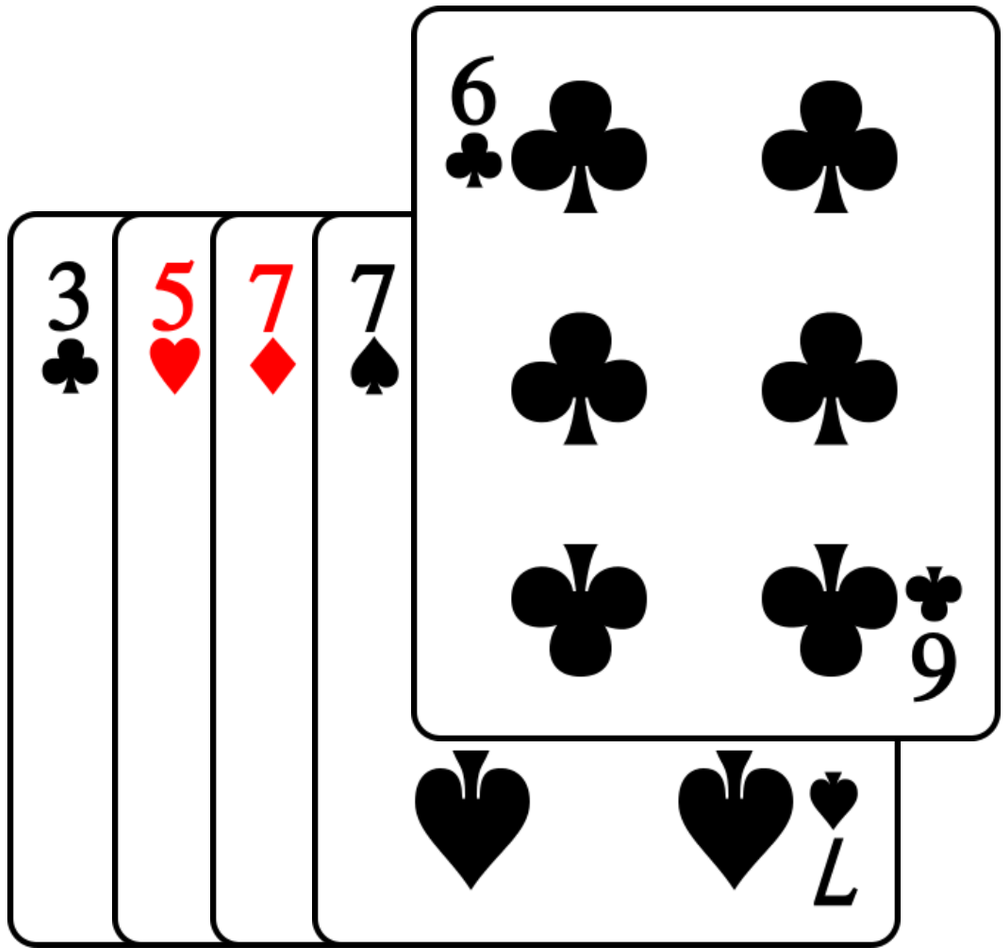
Tri par insertion



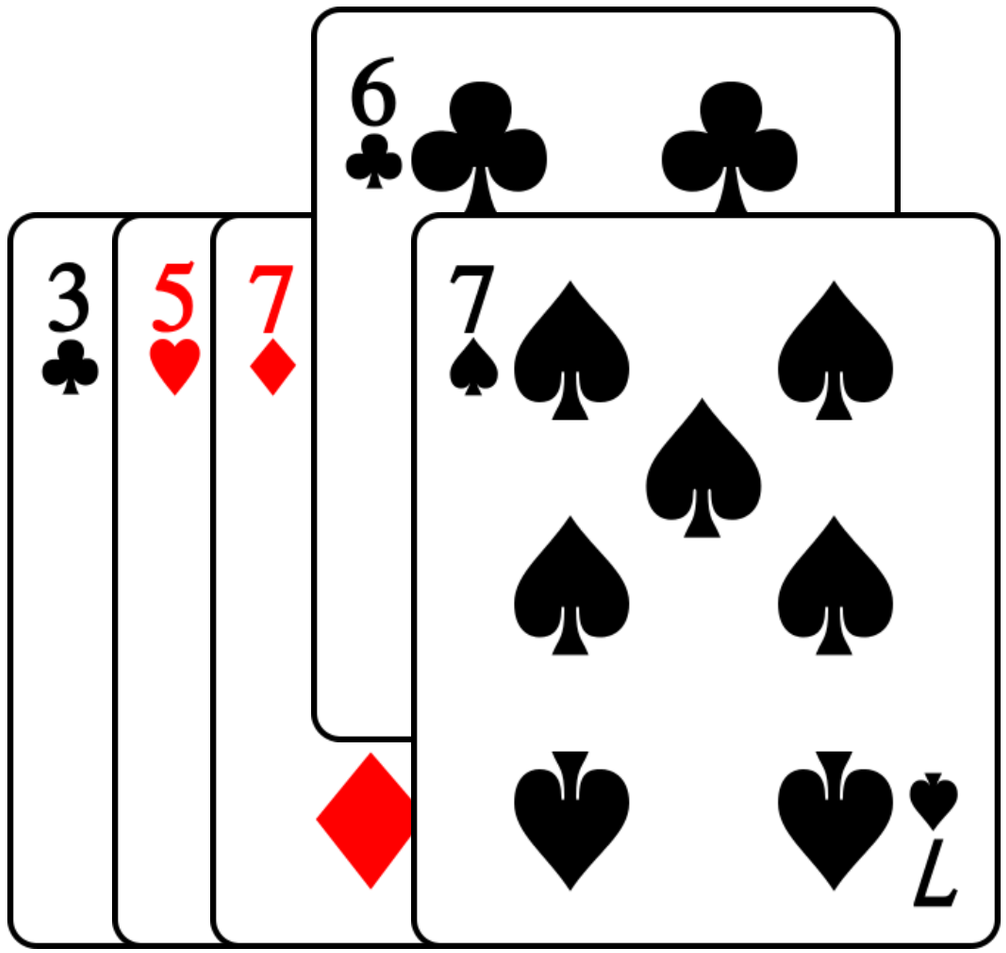
Tri par insertion



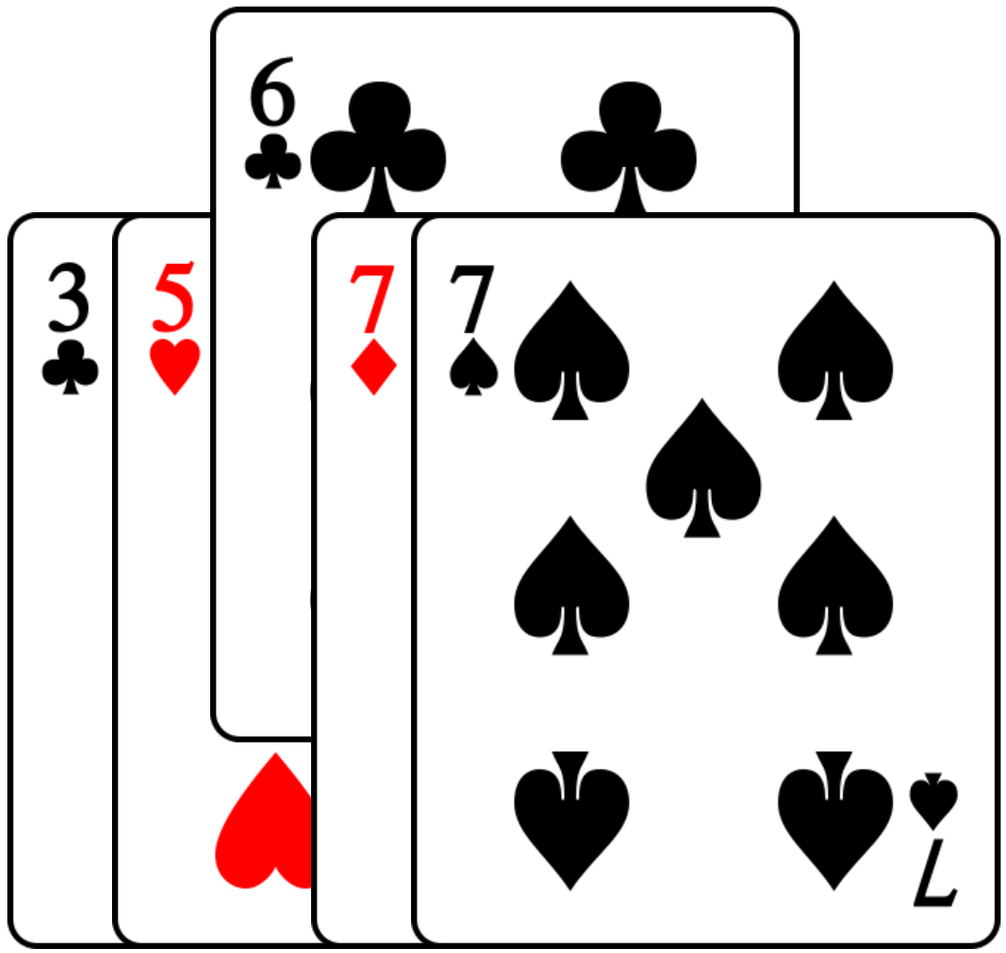
Tri par insertion



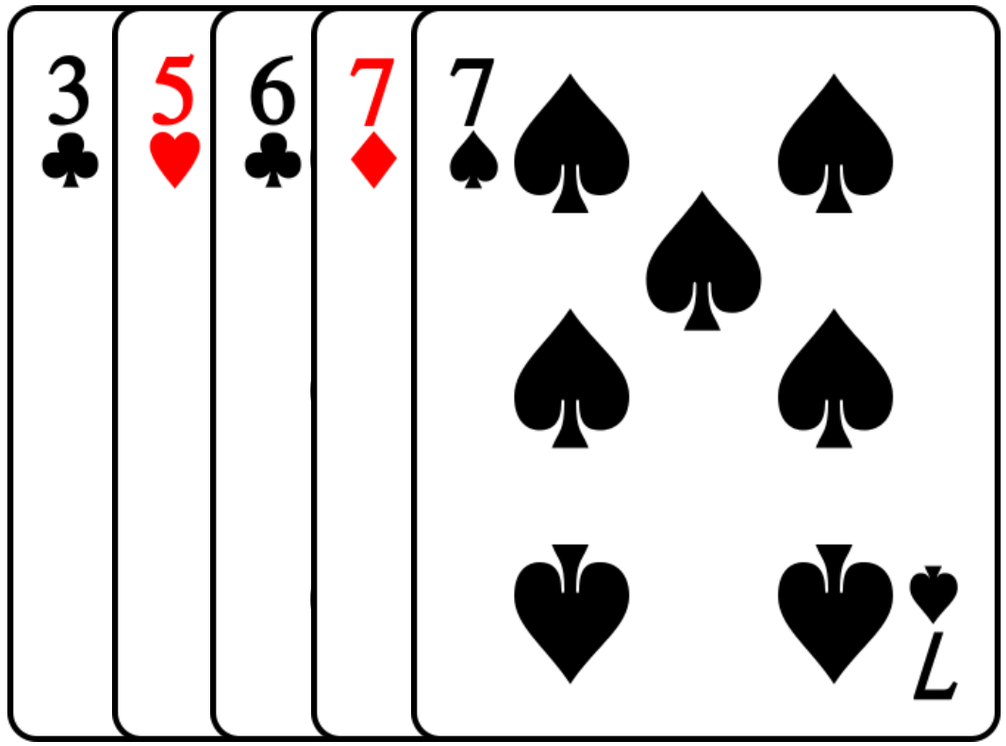
Tri par insertion



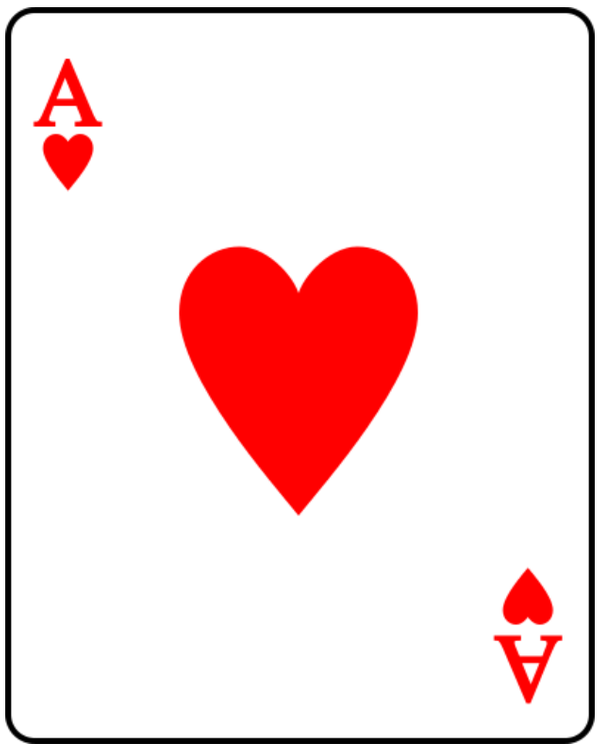
Tri par insertion



Tri par insertion

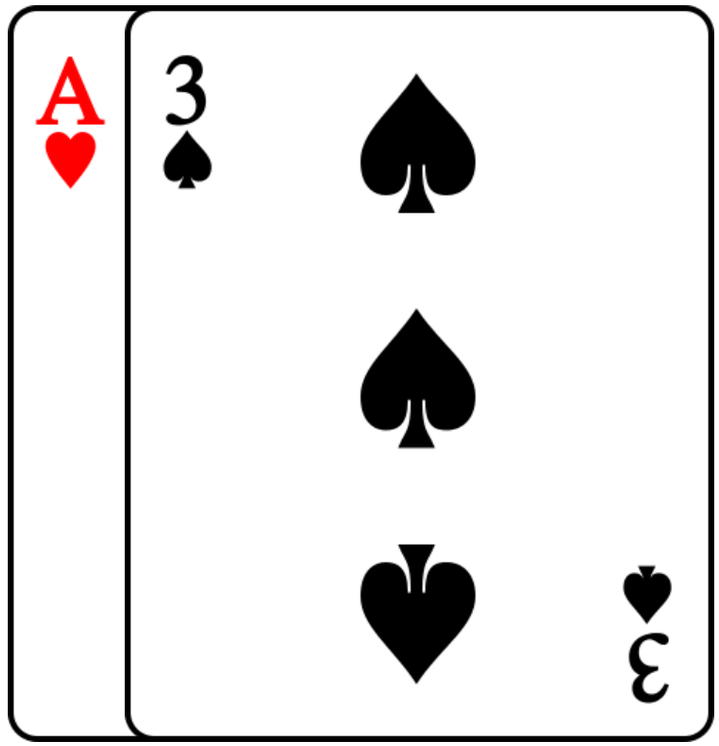


Le meilleur des cas



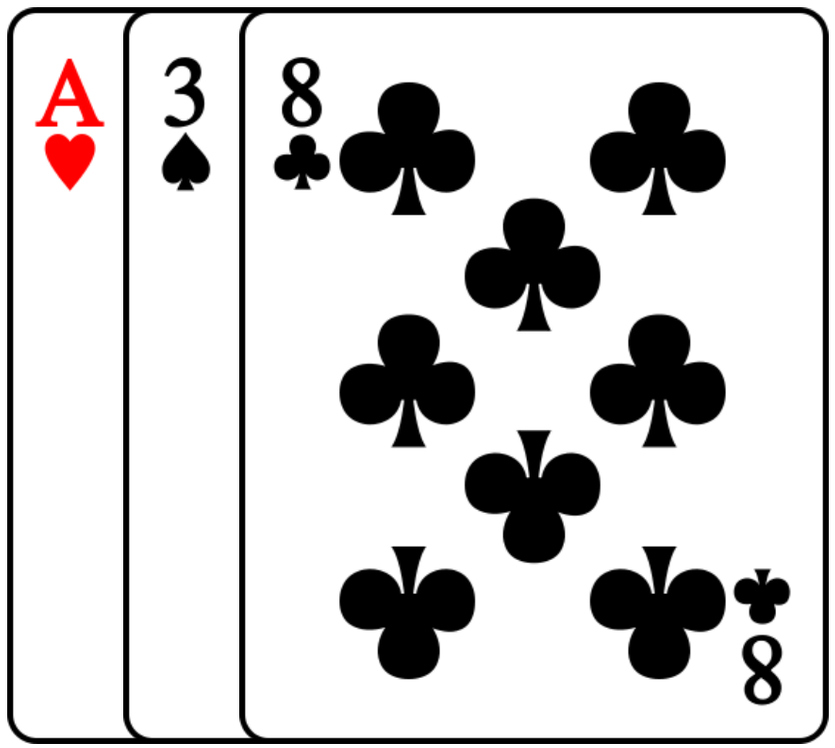
№ operations = 1

Le meilleur des cas



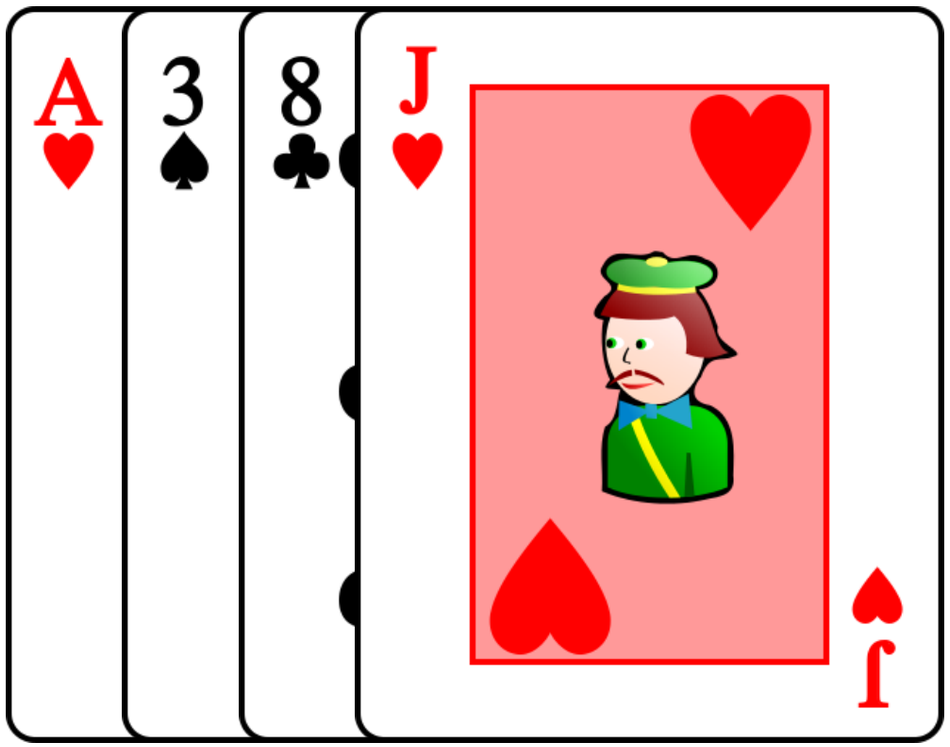
№ operations = 2

Le meilleur des cas



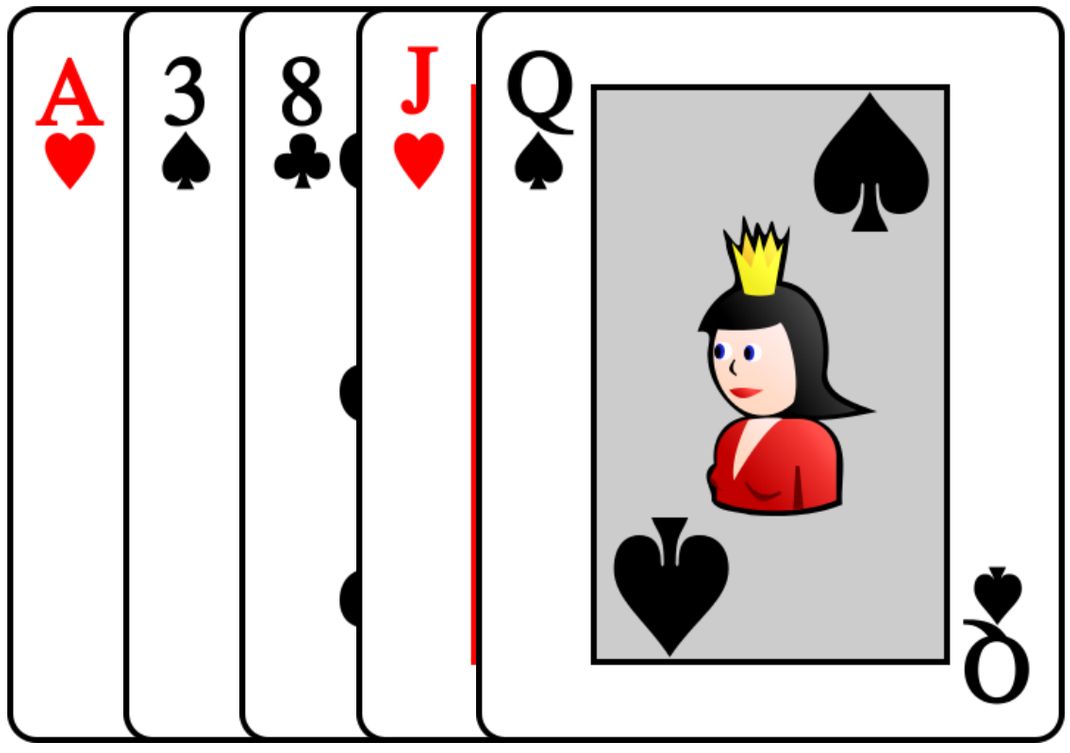
№ operations = 3

Le meilleur des cas



№ operations = 4

Le meilleur des cas



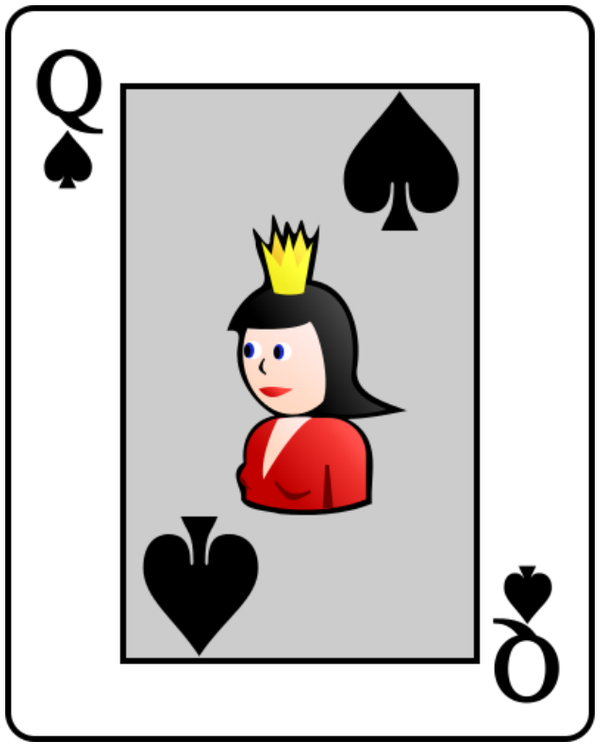
№ operations = 5

Le meilleur des cas

- Les cartes arrivent déjà triées

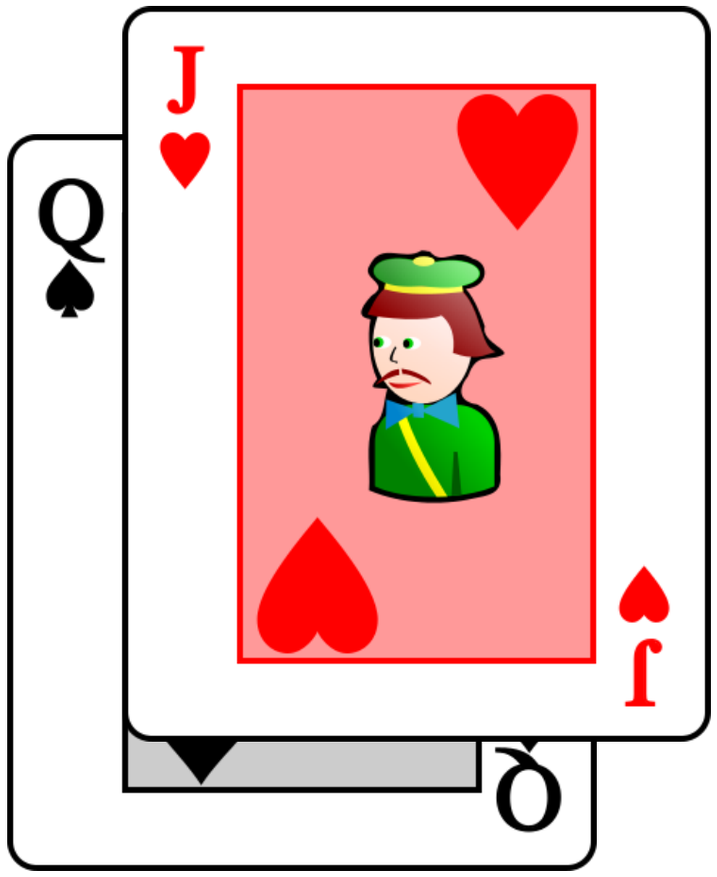
- On fait n opérations (déplacements de cartes)

Le pire des cas



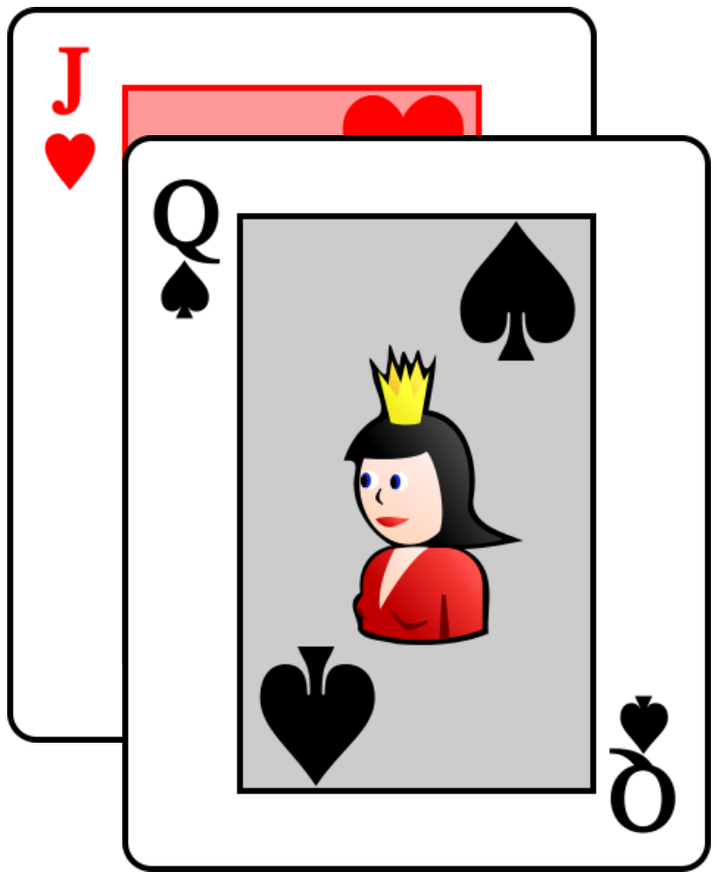
№ operations = 1

Le pire des cas



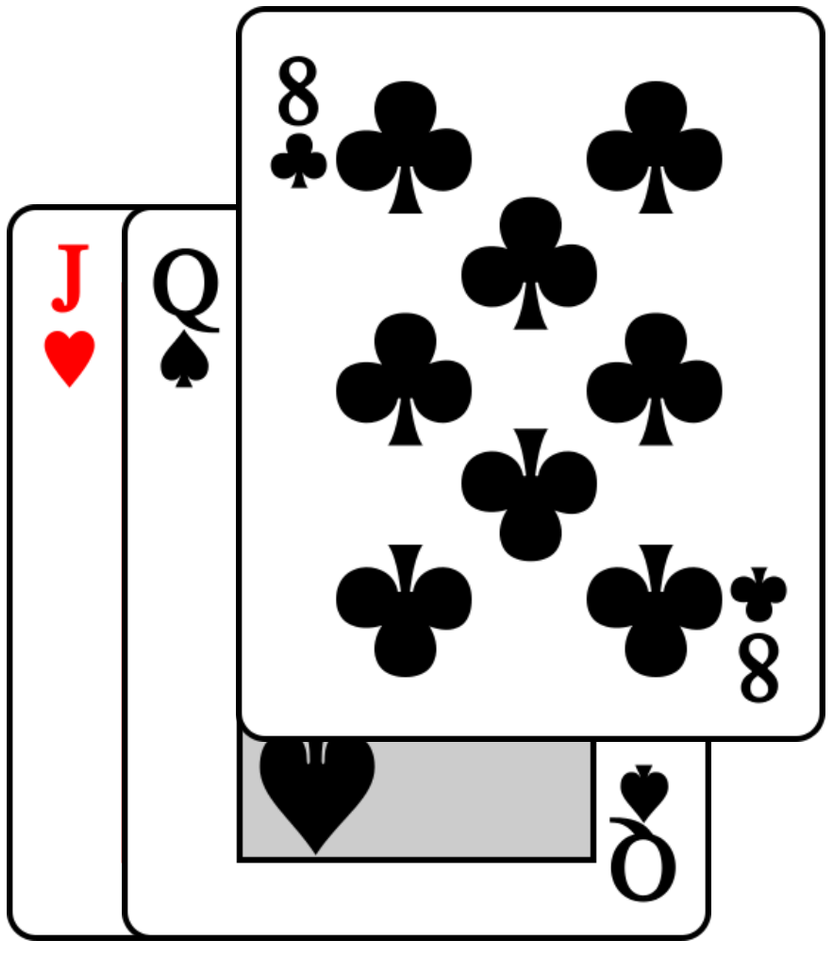
№ operations = 1 + 1

Le pire des cas



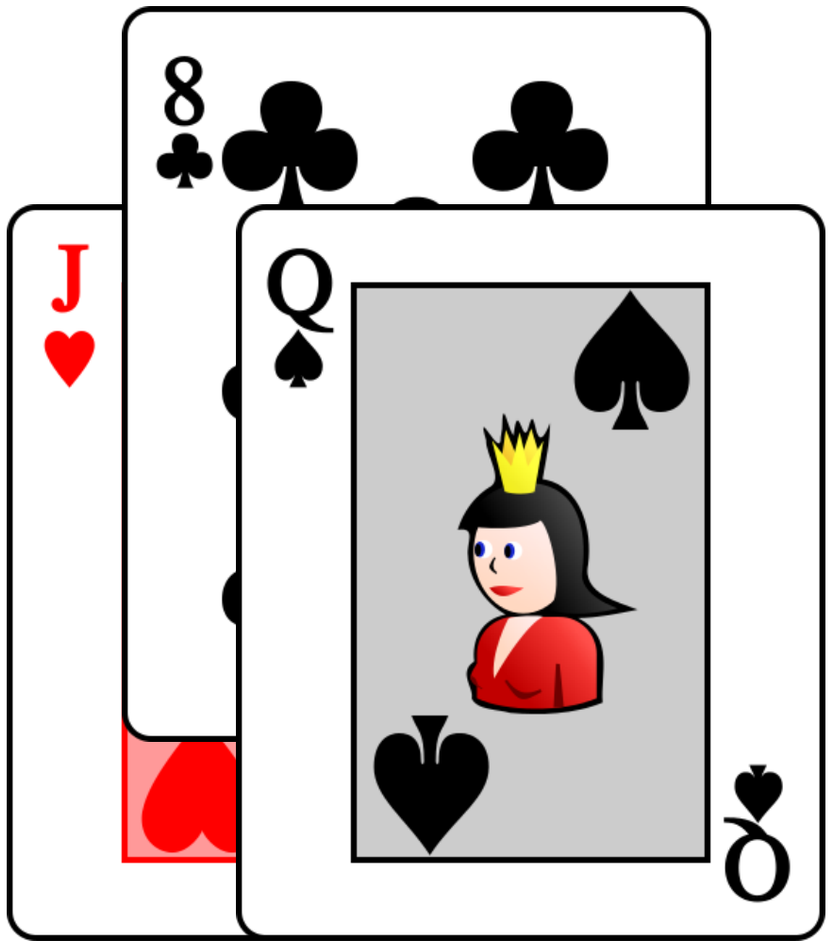
№ operations = 1 + 2

Le pire des cas



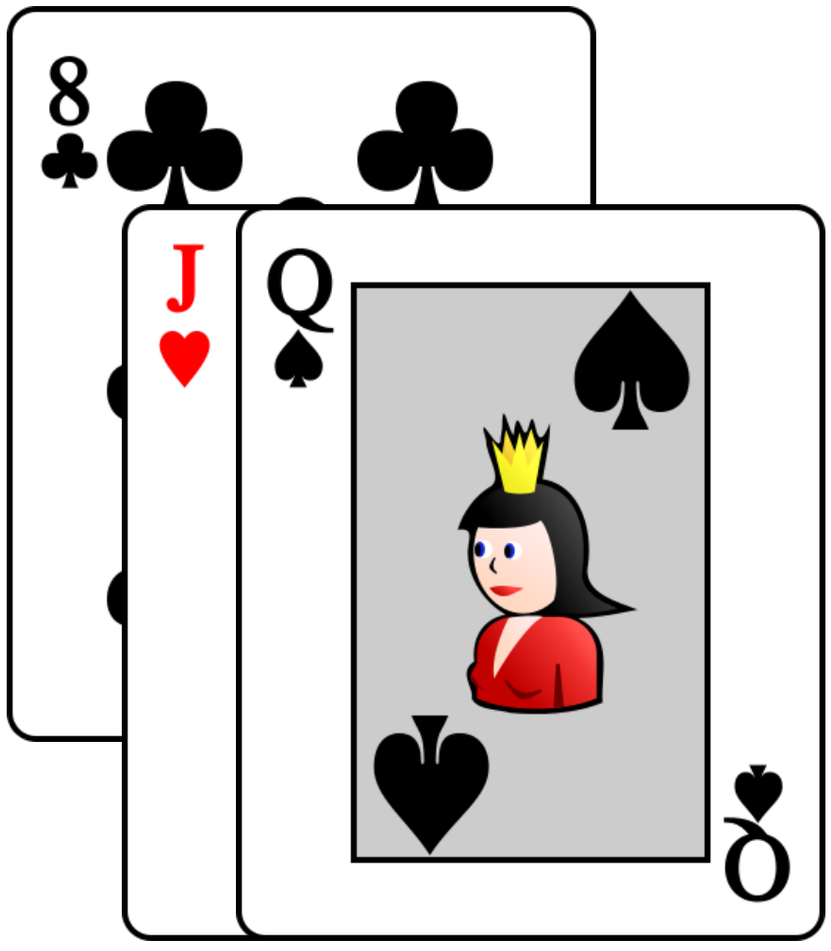
№ operations = 1 + 2 + 1

Le pire des cas



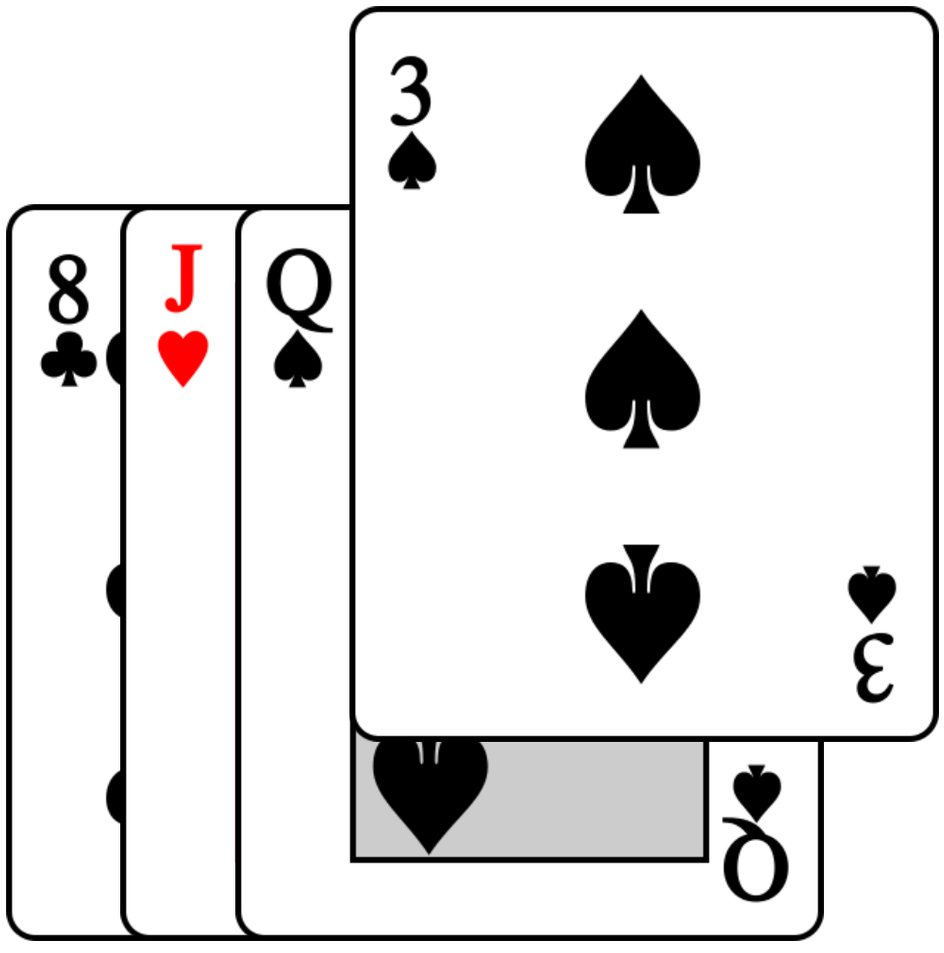
№ operations = 1 + 2 + 2

Le pire des cas



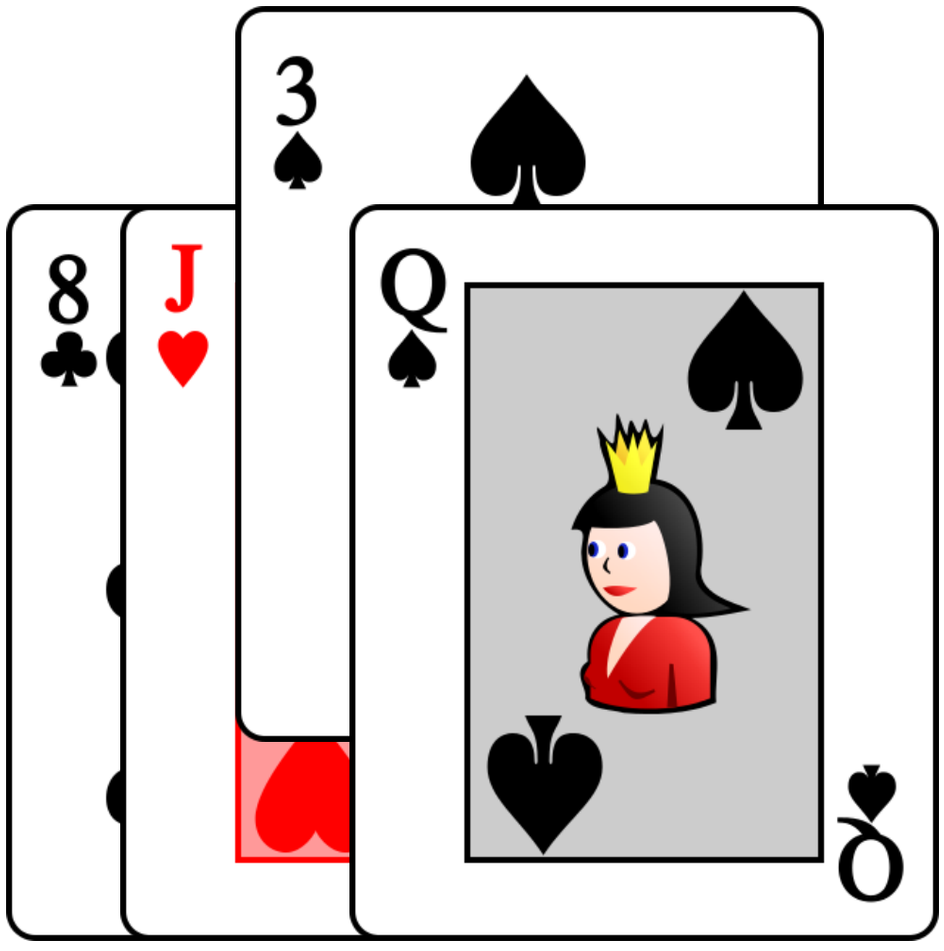
№ operations = 1 + 2 + 3

Le pire des cas



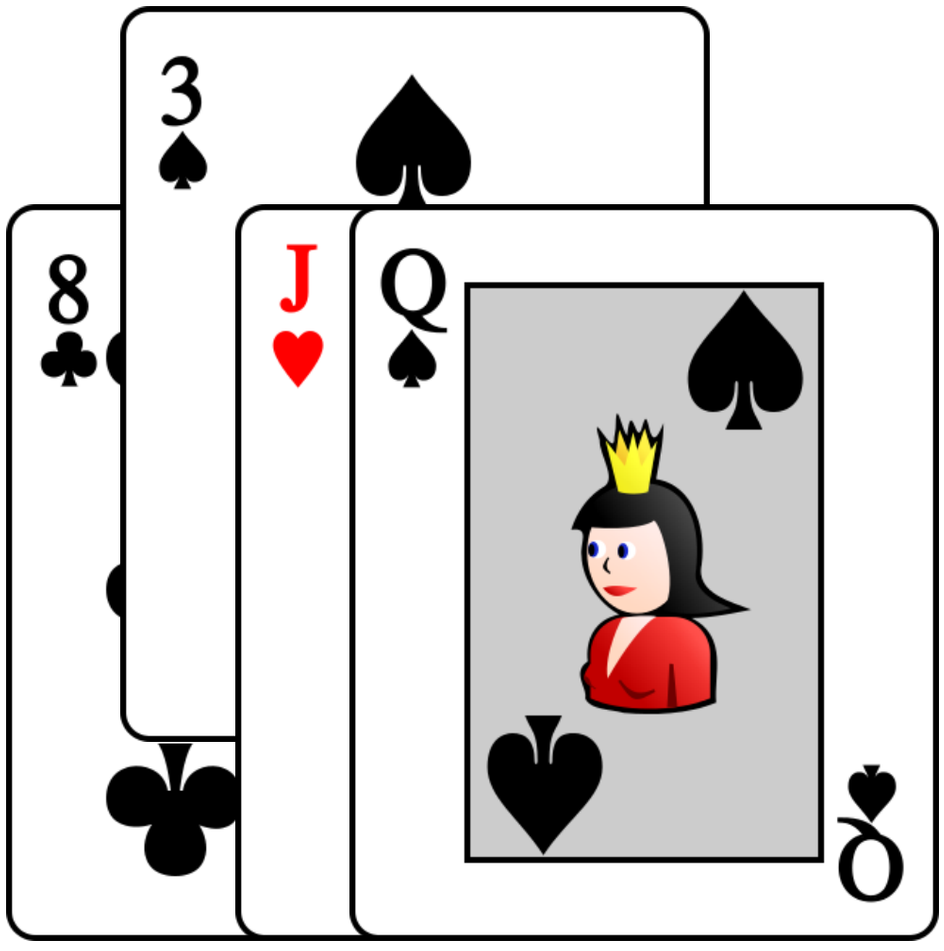
№ operations = 1 + 2 + 3 + 1

Le pire des cas



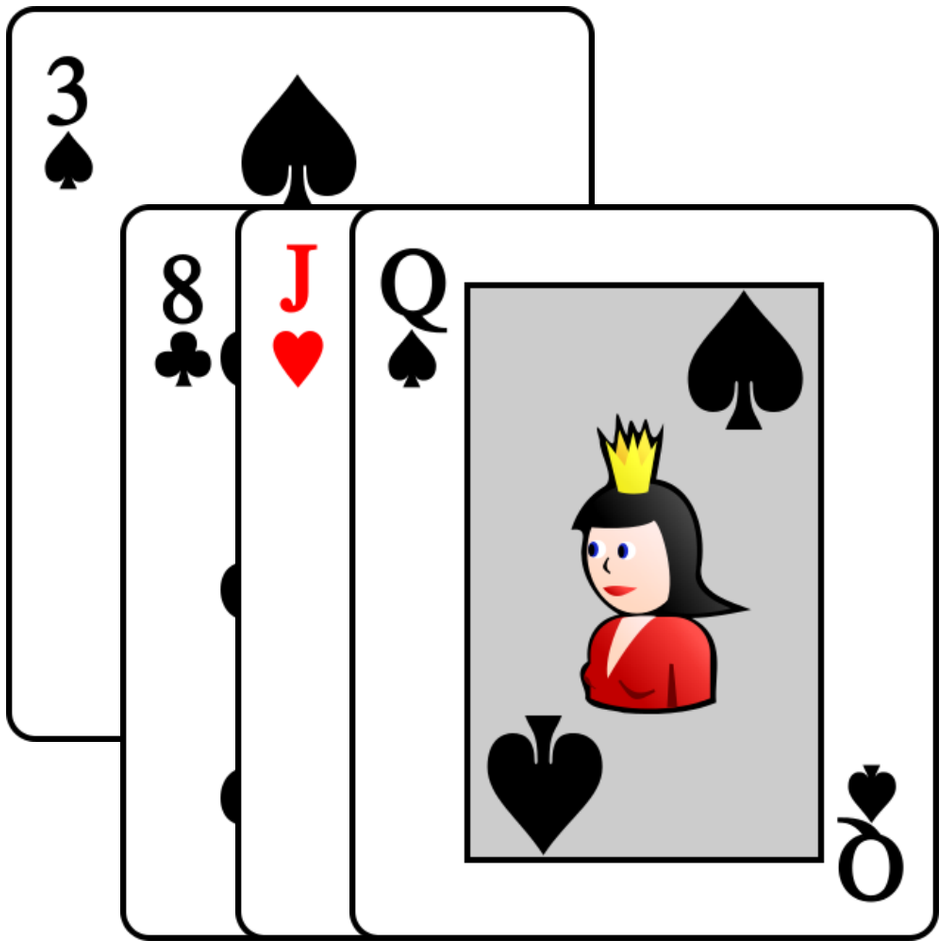
№ operations = 1 + 2 + 3 + 2

Le pire des cas



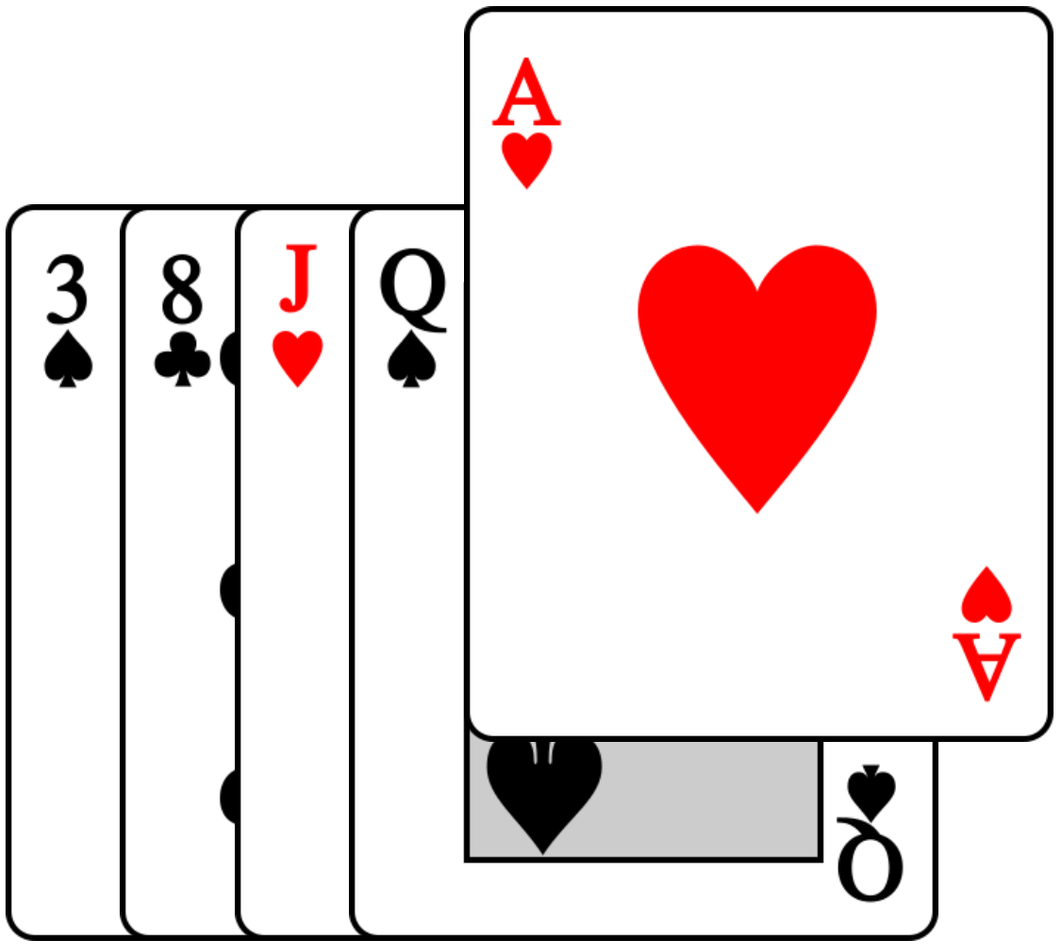
№ operations = 1 + 2 + 3 + 3

Le pire des cas



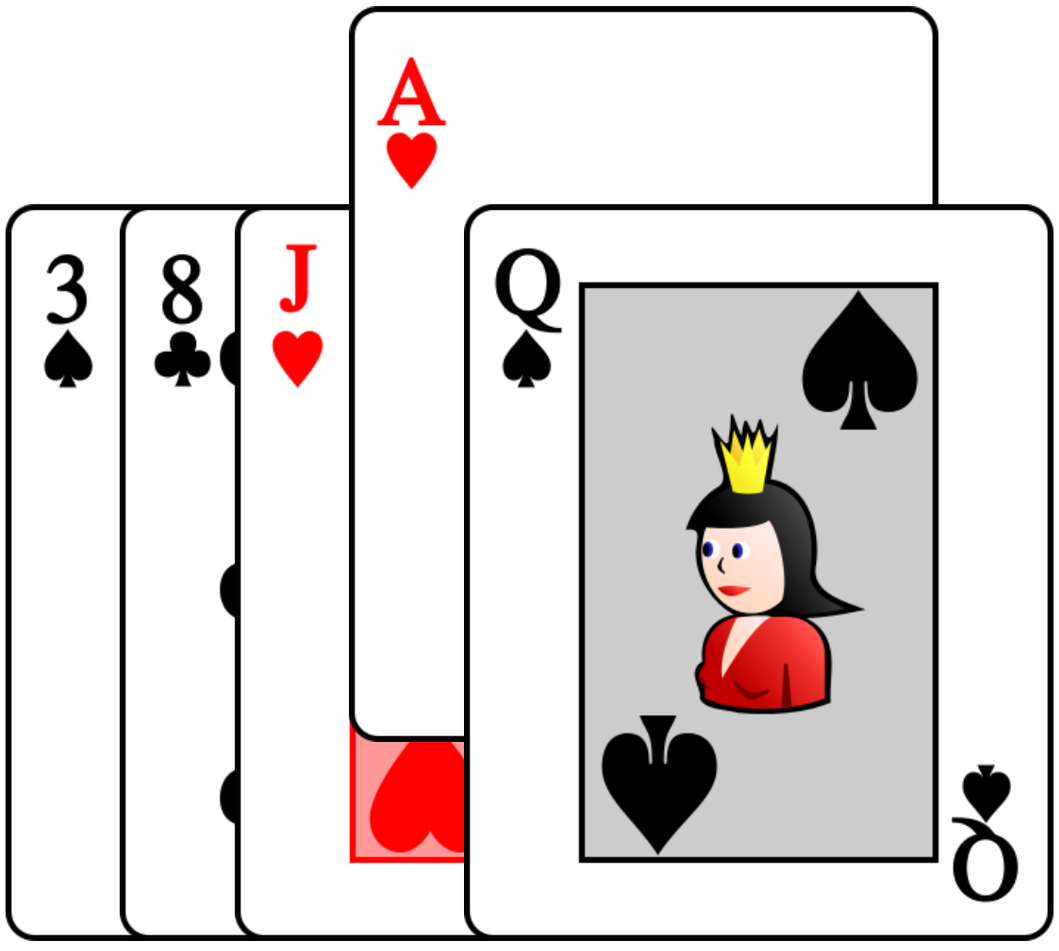
№ operations = 1 + 2 + 3 + 4

Le pire des cas



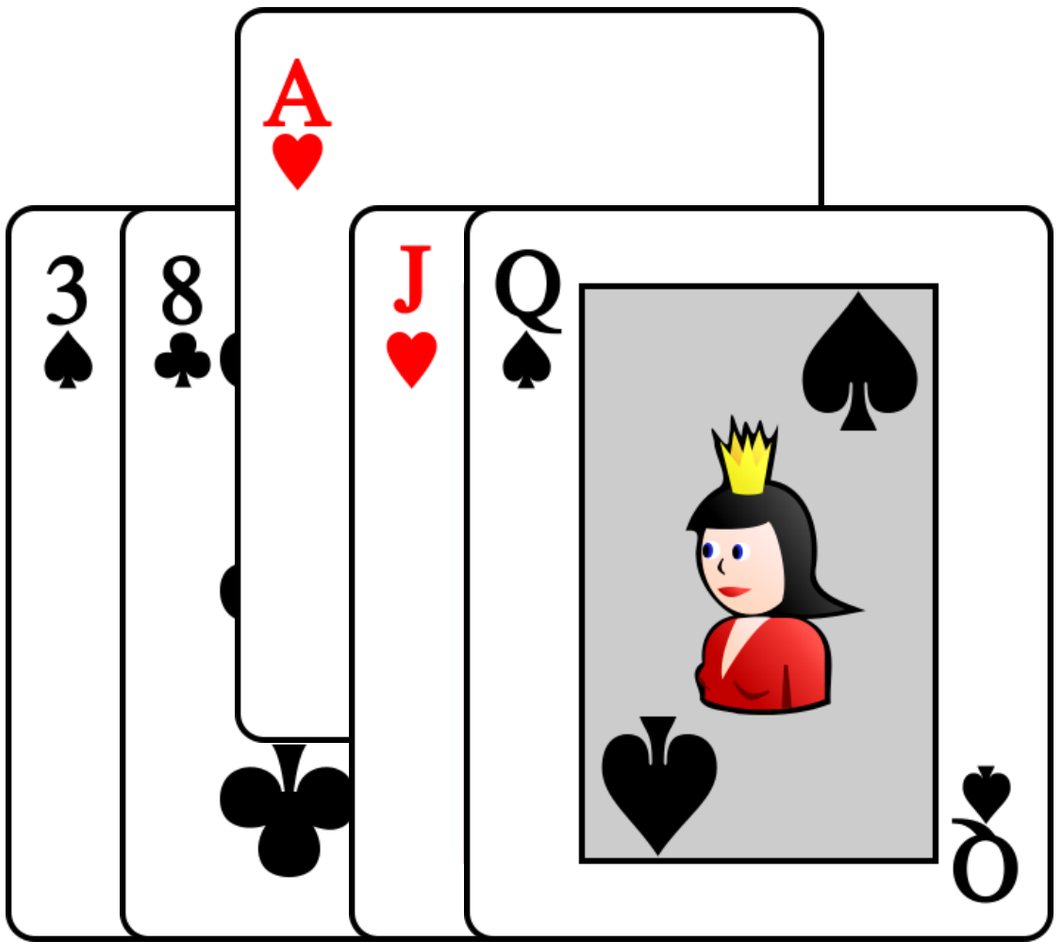
№ operations = 1 + 2 + 3 + 4 + 1

Le pire des cas



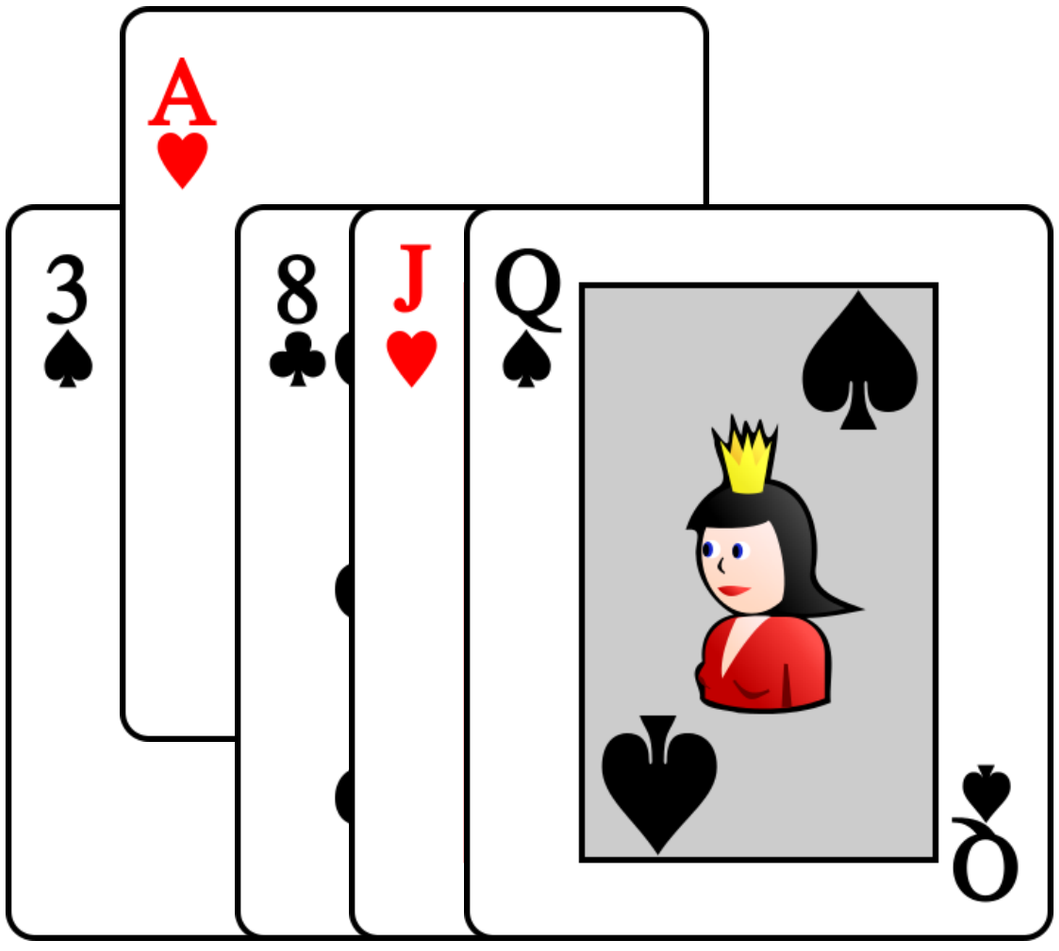
№ operations = 1 + 2 + 3 + 4 + 2

Le pire des cas



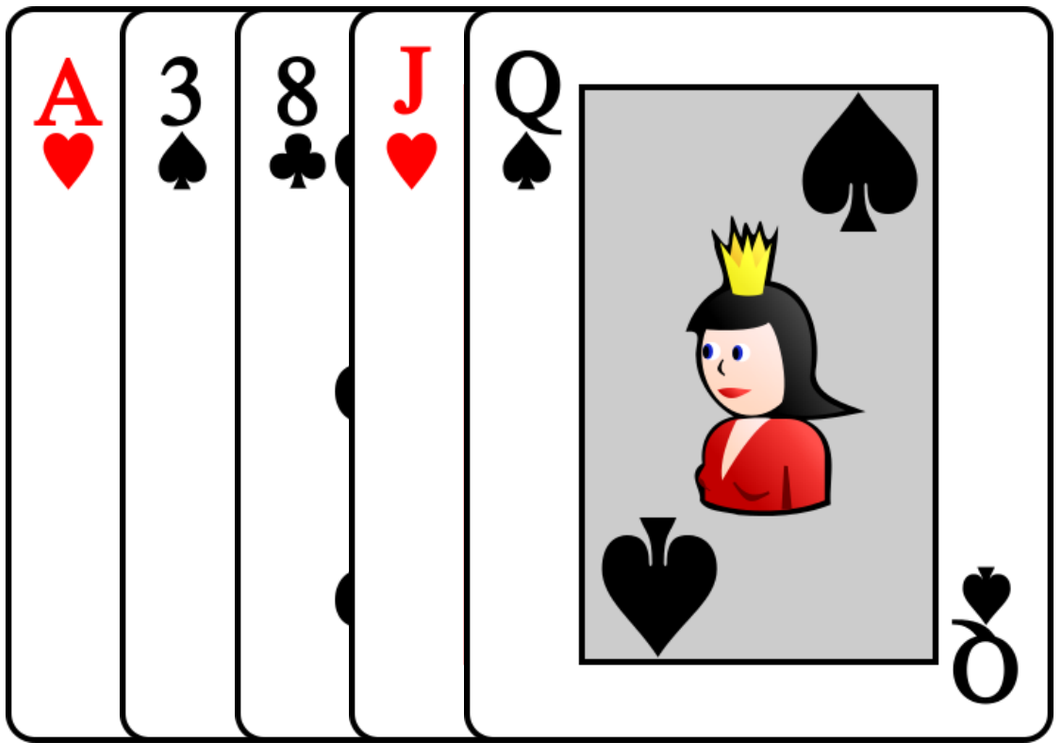
№ operations = 1 + 2 + 3 + 4 + 3

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 $1 + 2 + 3 + \cdots + n$

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