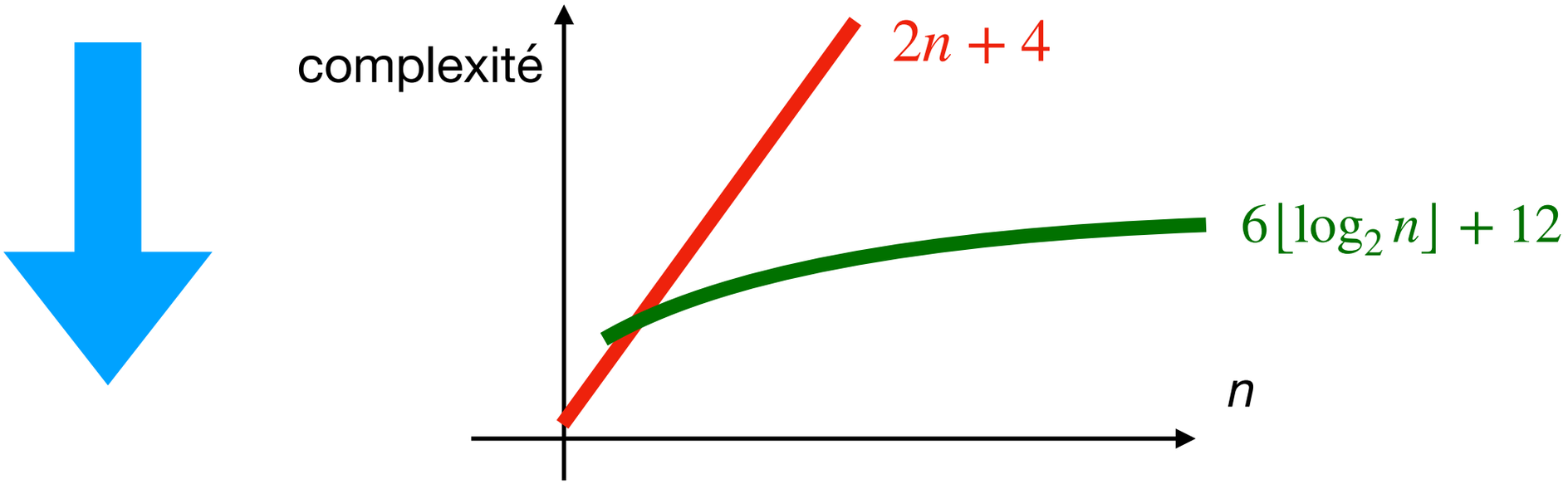
Introduction à la science informatique

Semaine 5
(2 séances de 2 heures)

Recherche séquentielle dans un tableau quelconque

$2n + 4$ opérations dans le pire des cas



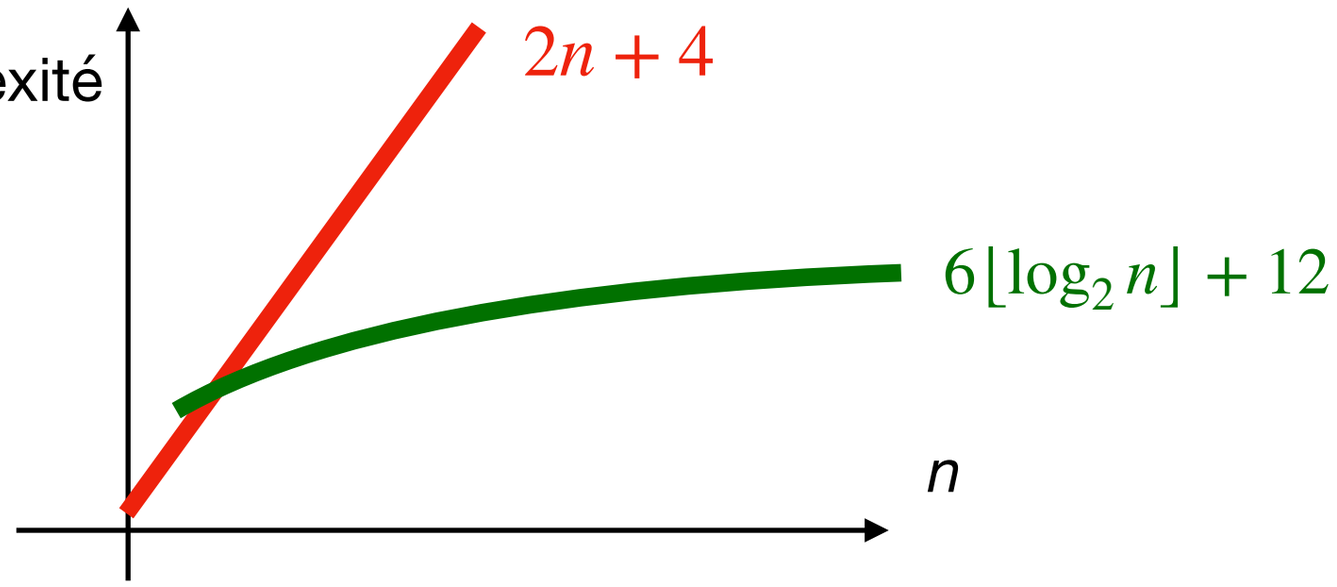
Recherche dichotomique dans un tableau trié

$6\lfloor \log_2 n \rfloor + 12$ opérations dans le pire des cas

Recherche séquentielle dans un tableau quelconque

$2n + 4$ opérations dans le pire des cas

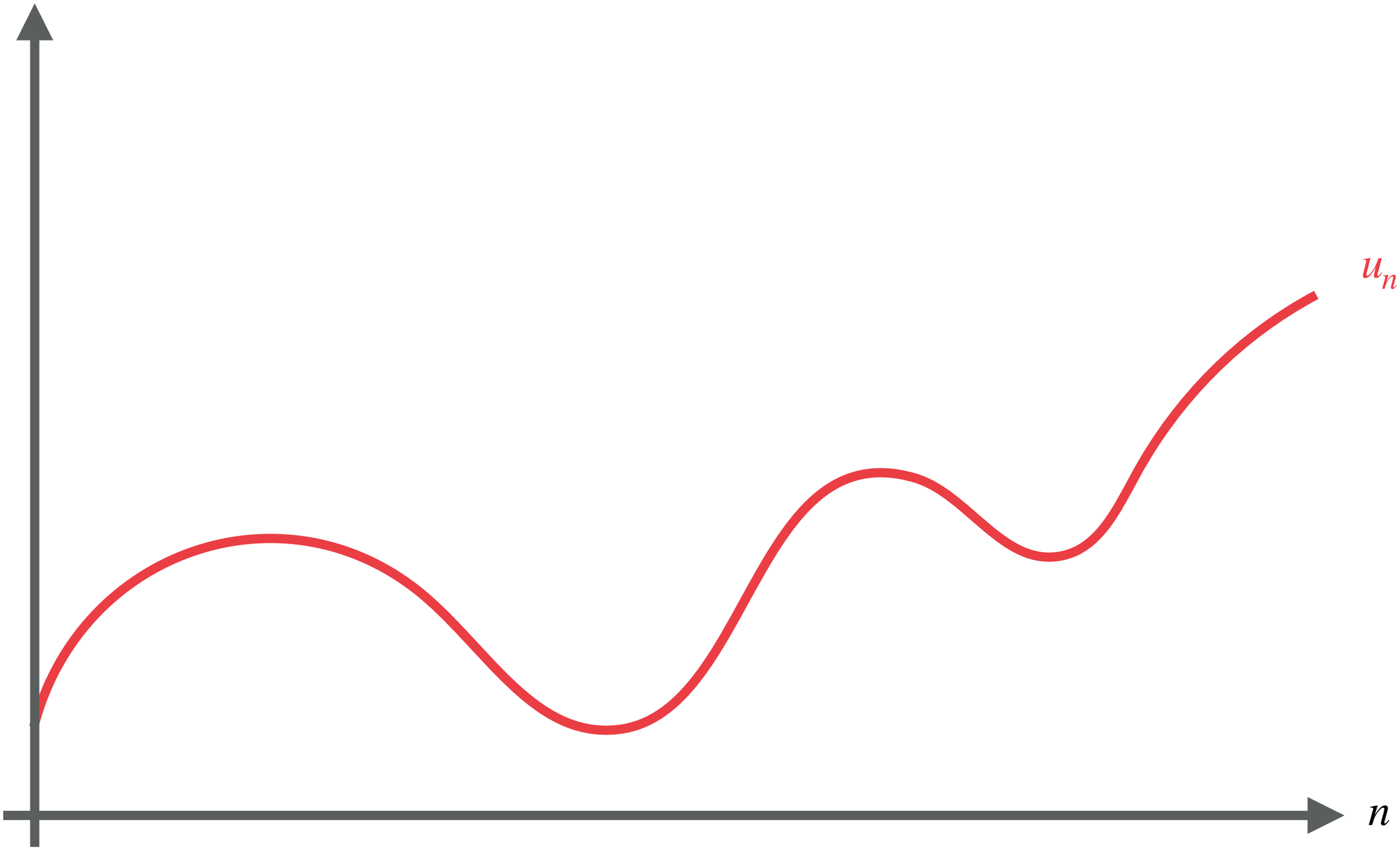
C'est pénible ces nombres si détaillés !!



Recherche dichotomique dans un tableau trié

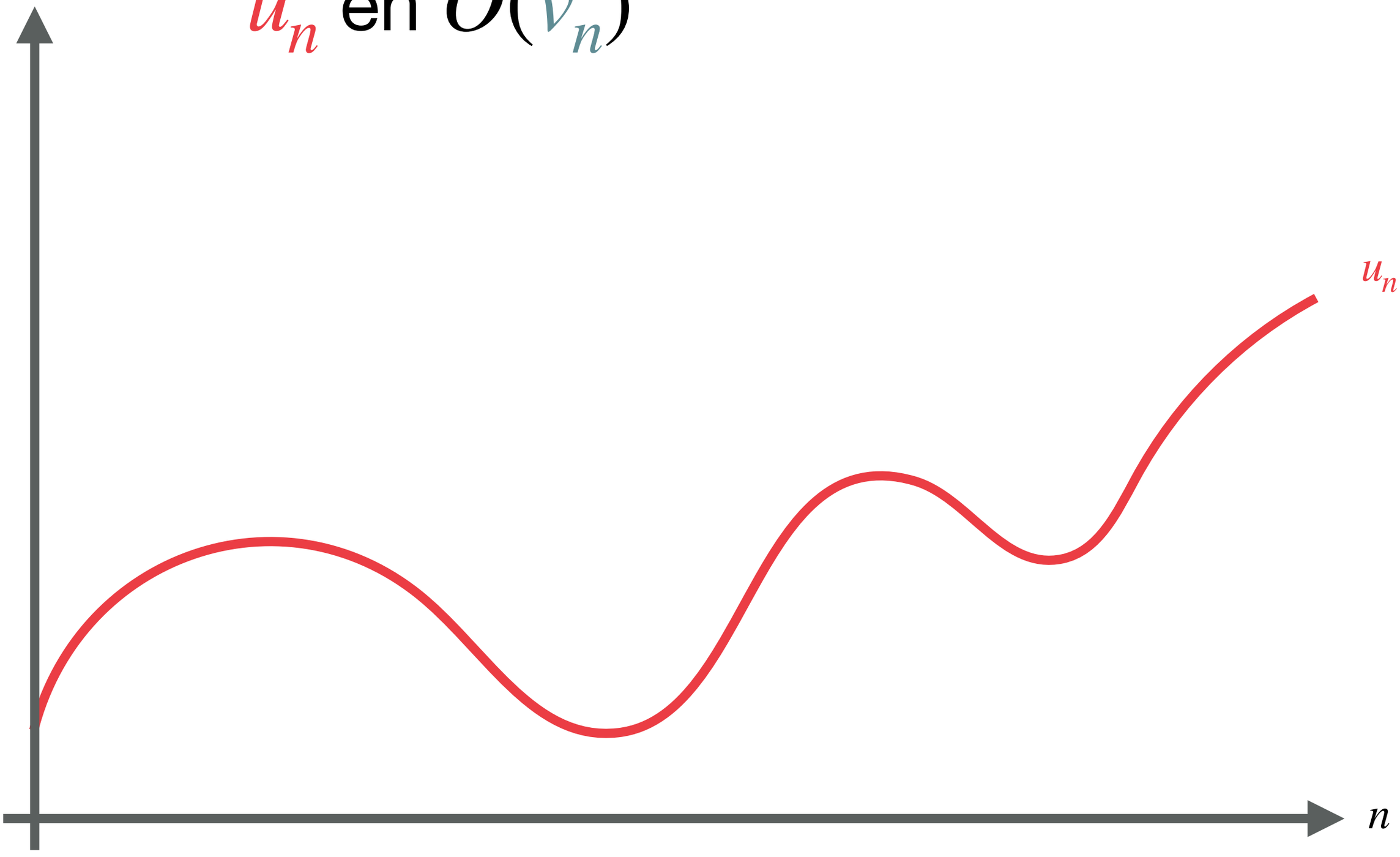
$6\lfloor \log_2 n \rfloor + 12$ opérations dans le pire des cas

Notation « grand O »



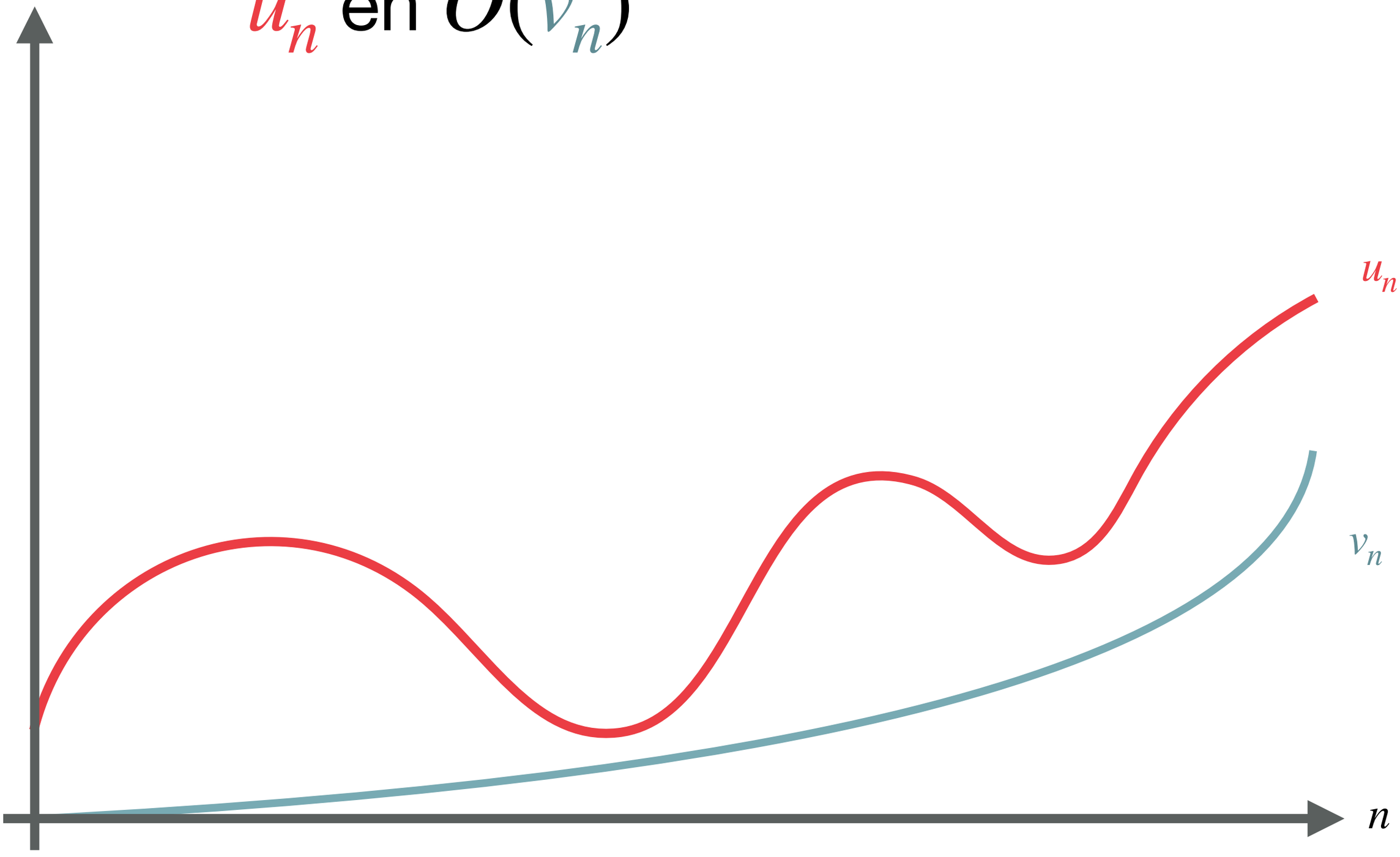
Notation « grand O »

u_n en $O(v_n)$



Notation « grand O »

u_n en $O(v_n)$

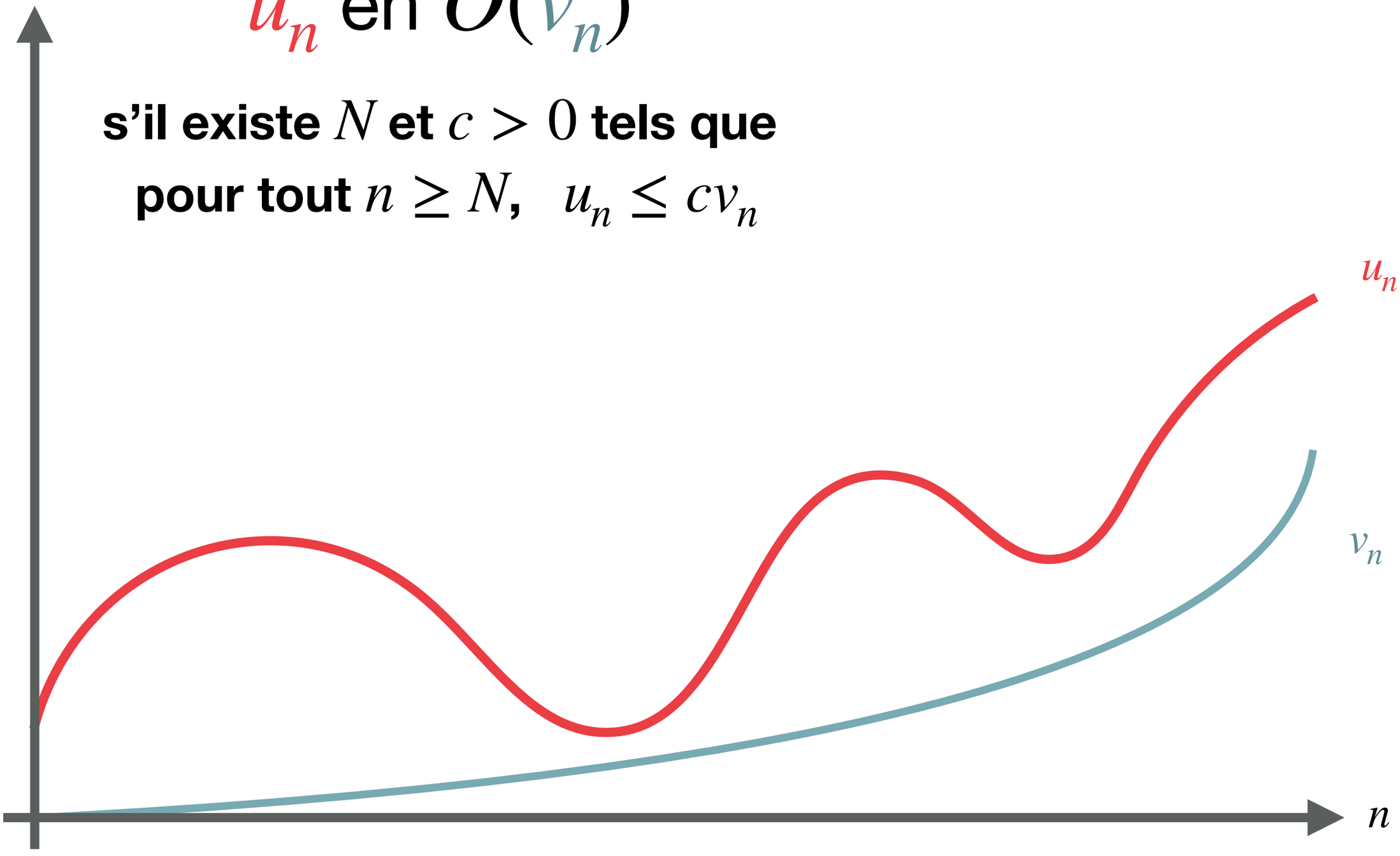


Notation « grand O »

u_n en $O(v_n)$

s'il existe N et $c > 0$ tels que

pour tout $n \geq N$, $u_n \leq cv_n$

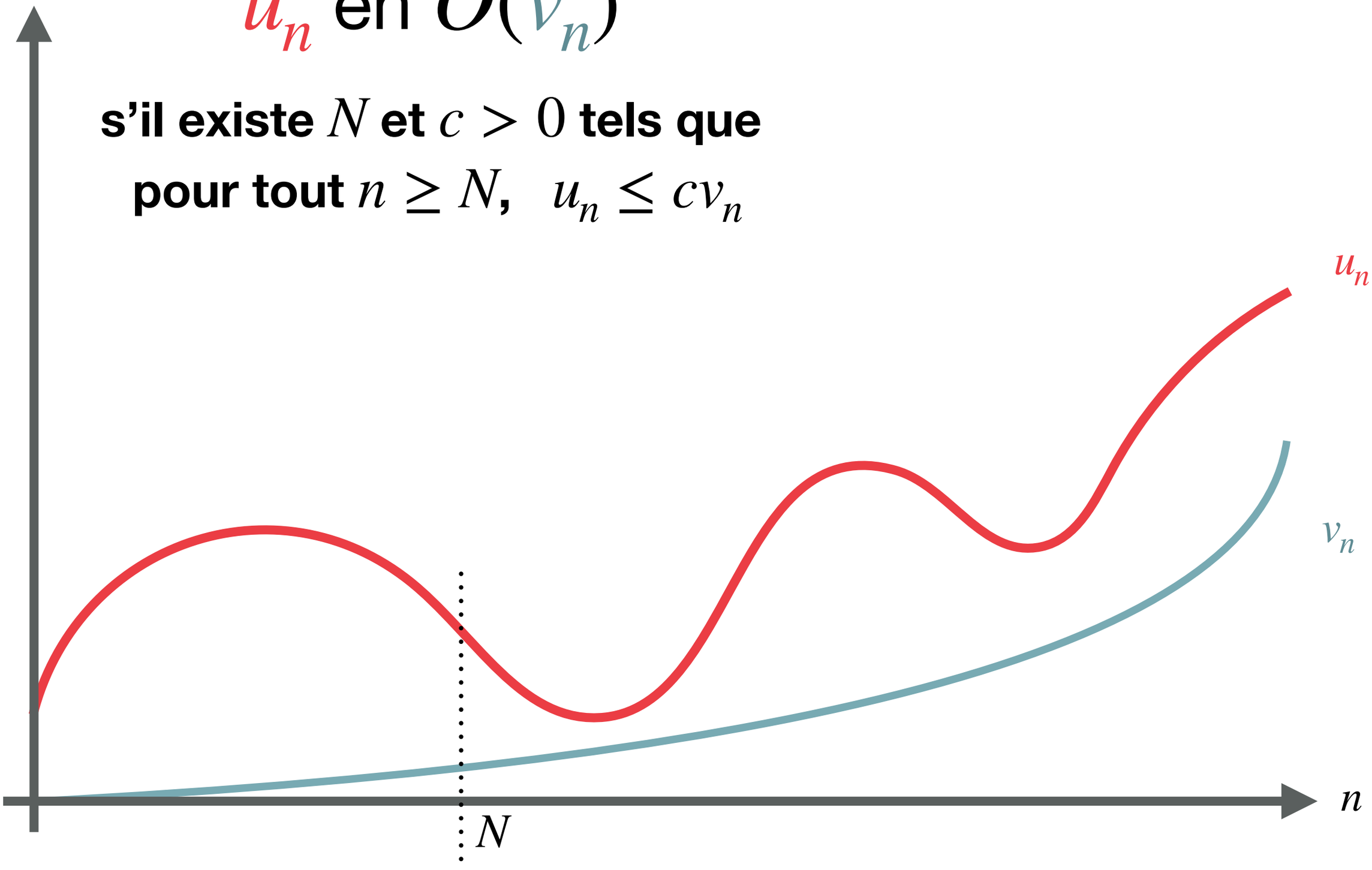


Notation « grand O »

u_n en $O(v_n)$

s'il existe N et $c > 0$ tels que

pour tout $n \geq N$, $u_n \leq cv_n$

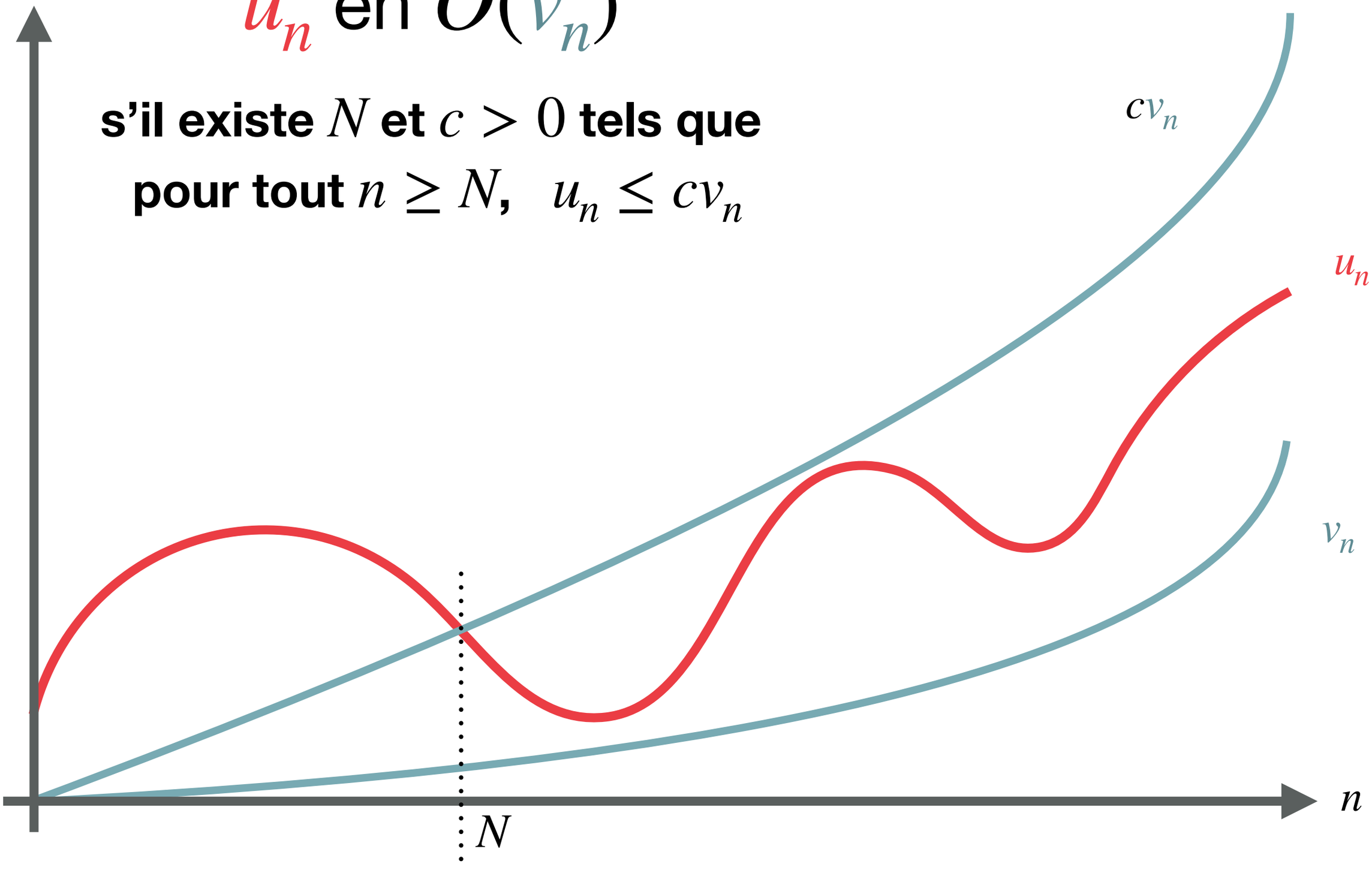


Notation « grand O »

u_n en $O(v_n)$

s'il existe N et $c > 0$ tels que

pour tout $n \geq N$, $u_n \leq cv_n$



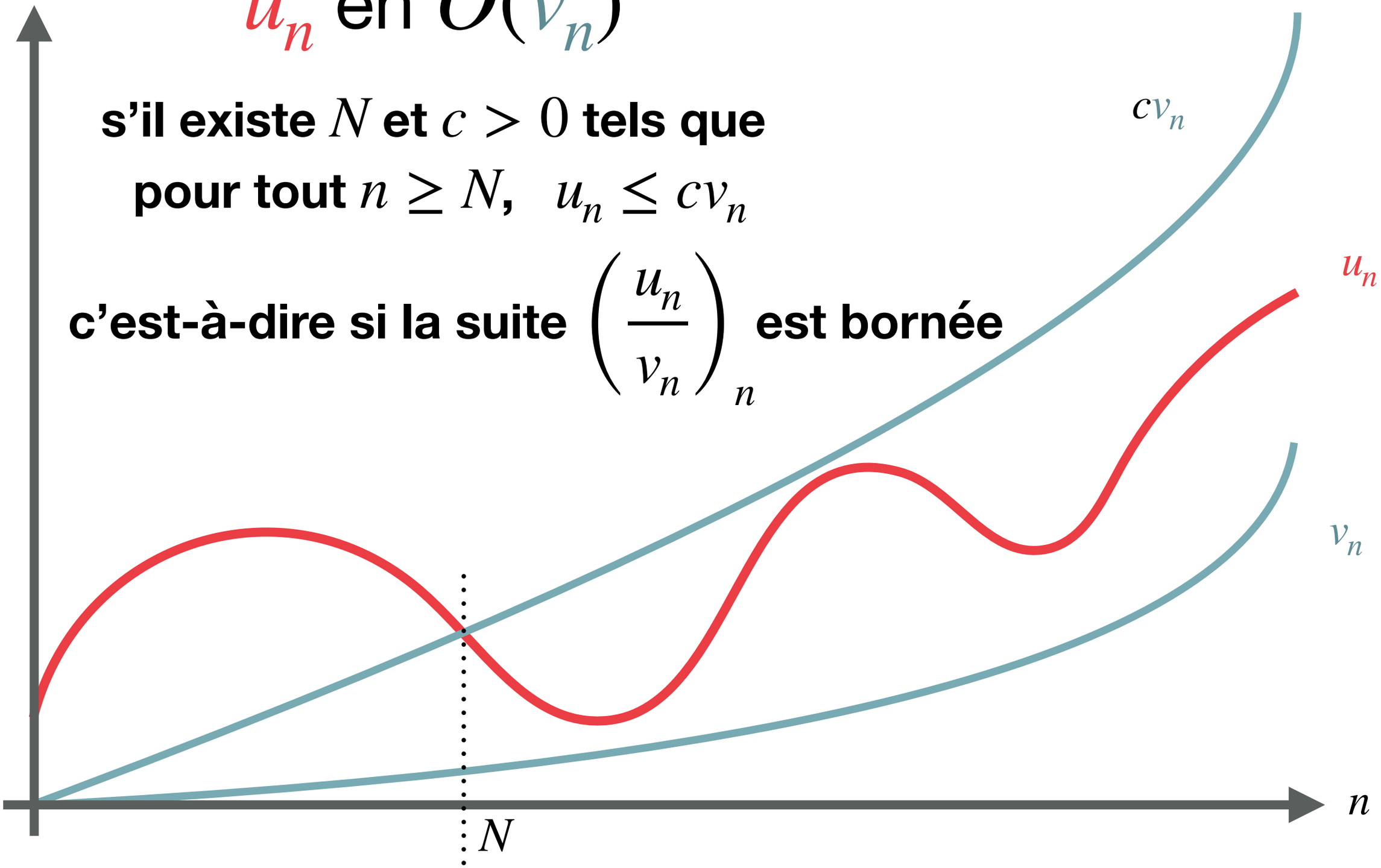
Notation « grand O »

u_n en $O(v_n)$

s'il existe N et $c > 0$ tels que

pour tout $n \geq N$, $u_n \leq cv_n$

c'est-à-dire si la suite $\left(\frac{u_n}{v_n}\right)_n$ est bornée



Exercice 1

Ordres de grandeur

Ordres de grandeur

- $n + 3$ est en $O(n)$

Ordres de grandeur

- $n + 3$ est en $O(n)$
- $2n + 5$ est en $O(n)$

Ordres de grandeur

- $n + 3$ est en $O(n)$
- $2n + 5$ est en $O(n)$
- $n^2 + 2n - 3$ est en $O(n^2)$

Ordres de grandeur

- $n + 3$ est en $O(n)$
- $2n + 5$ est en $O(n)$
- $n^2 + 2n - 3$ est en $O(n^2)$
- $n \log_2 n + 3n - 4$ est en $O(n \log_2 n)$

Ordres de grandeur

- $n + 3$ est en $O(n)$
- $2n + 5$ est en $O(n)$
- $n^2 + 2n - 3$ est en $O(n^2)$
- $n \log_2 n + 3n - 4$ est en $O(n \log_2 n)$
- $\lfloor \log_2 n \rfloor$ est en $O(\log_2 n)$

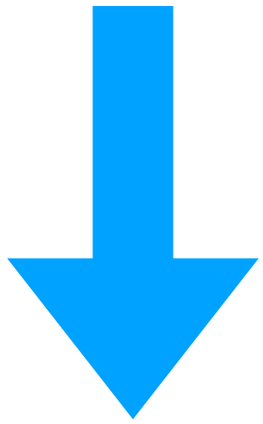
Ordres de grandeur

- $n + 3$ est en $O(n)$
- $2n + 5$ est en $O(n)$
- $n^2 + 2n - 3$ est en $O(n^2)$
- $n \log_2 n + 3n - 4$ est en $O(n \log_2 n)$
- $\lfloor \log_2 n \rfloor$ est en $O(\log_2 n)$

Dans une somme, on ne garde que le terme qui « grandit le plus vite », on oublie les constantes et les parties entières…

Recherche séquentielle dans un tableau quelconque

$2n + 4$ opérations dans le pire des cas

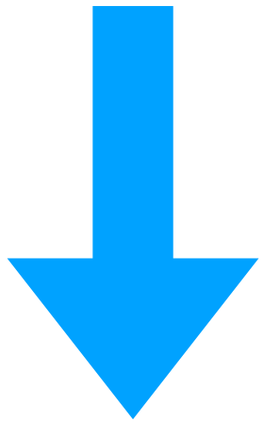


Recherche dichotomique dans un tableau trié

$6\lfloor \log_2 n \rfloor + 12$ opérations dans le pire des cas

Recherche séquentielle dans un tableau quelconque

$O(n)$ opérations dans le pire des cas



Recherche dichotomique dans un tableau trié

$O(\log_2 n)$ opérations dans le pire des cas

Exercice 2

Chaînes de caractères

… des tableaux un peu particuliers !

Chaînes de caractères

… des tableaux un peu particuliers !

```
>>> s = 'Bonjour'
```

Chaînes de caractères

… des tableaux un peu particuliers !

```
>>> s = 'Bonjour'
>>> s
'Bonjour'
```

Chaînes de caractères

… des tableaux un peu particuliers !

```
>>> s = 'Bonjour'
>>> s
'Bonjour'
>>> len(s)
7
```

Chaînes de caractères

… des tableaux un peu particuliers !

```
>>> s = 'Bonjour'
>>> s
'Bonjour'
>>> len(s)
7
>>> s[0]
'B'
```

Chaînes de caractères

… des tableaux un peu particuliers !

```
>>> s = 'Bonjour'
>>> s
'Bonjour'
>>> len(s)
7
>>> s[0]
'B'
>>> s + ' !'
'Bonjour !'
```

Chaînes de caractères

… des tableaux un peu particuliers !

```
>>> s = 'Bonjour'
>>> s
'Bonjour'
>>> len(s)
7
>>> s[0]
'B'
>>> s + ' !'
'Bonjour !'
```

+ pour *concaténer* deux chaînes de caractères, c'est-à-dire les mettre bout à bout

Chaînes de caractères

… des tableaux un peu particuliers !

```
>>> s = 'Bonjour'
>>> s
'Bonjour'
>>> len(s)
7
>>> s[0]
'B'
>>> s + ' !'
'Bonjour !'
>>> s[4] = 'O'
Traceback (most recent call last):
```

+ pour *concaténer* deux chaînes de caractères, c'est-à-dire les mettre bout à bout

Chaînes de caractères

… des tableaux un peu particuliers !

```
>>> s = 'Bonjour'
>>> s
'Bonjour'
>>> len(s)
7
>>> s[0]
'B'
>>> s + ' !'
'Bonjour !'
>>> s[4] = 'O'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

+ pour *concaténer* deux chaînes de caractères, c'est-à-dire les mettre bout à bout

Chaînes de caractères

… des tableaux un peu particuliers !

```
>>> s = 'Bonjour'
>>> s
'Bonjour'
>>> len(s)
7
>>> s[0]
'B'
>>> s + ' !'
'Bonjour !'
>>> s[4] = 'O'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

+ pour *concaténer* deux chaînes de caractères, c'est-à-dire les mettre bout à bout

Chaînes de caractères

… des tableaux un peu particuliers !

```
>>> s = 'Bonjour'
>>> s
'Bonjour'
>>> len(s)
7
>>> s[0]
'B'
>>> s + ' !'
'Bonjour !'
>>> s[4] = 'O'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

+ pour *concaténer* deux chaînes de caractères, c'est-à-dire les mettre bout à bout

les chaînes de caractères sont *immuables*, c'est-à-dire qu'on ne peut plus les modifier une fois créées

Exercice 3

Représenter des caractères : code ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Représenter des caractères : code ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
2	20	[SPACE]	64	40	@	96	60	`
3	21	!	65	41	A	97	61	a
4	22	"	66	42	B	98	62	b
5	23	#	67	43	C	99	63	c
6	24	$	68	44	D	100	64	d
7	25	%	69	45	E	101	65	e
8	26	&	70	46	F	102	66	f
9	27	'	71	47	G	103	67	g
0	28	(	72	48	H	104	68	h
1	29	)	73	49	I	105	69	i
2	2A	*	74	4A	J	106	6A	j
3	2B	+	75	4B	K	107	6B	k
4	2C	,	76	4C	L	108	6C	l
5	2D	-	77	4D	M	109	6D	m
6	2E	.	78	4E	N	110	6E	n
7	2F	/	79	4F	O	111	6F	o
8	30	0	80	50	P	112	70	p
9	31	1	81	51	Q	113	71	q
0	32	2	82	52	R	114	72	r
1	33	3	83	53	S	115	73	s

Représenter des caractères : code ASCII

- 128 codes possibles (de 0 à 127)
- 7 bits par code (nécessite 2 chiffres en hexadécimal)
- Pas uniquement des caractères lisibles

Code ASCII en Python

Code ASCII en Python

```
>>> ord('a')
```

Code ASCII en Python

```
>>> ord('a')
97
```

Code ASCII en Python

```
>>> ord('a')
97
>>> ord('Z')
```

Code ASCII en Python

```
>>> ord('a')
97
>>> ord('Z')
90
```

Code ASCII en Python

```
>>> ord('a')
97
>>> ord('Z')
90
>>> chr(98)
```

Code ASCII en Python

```
>>> ord('a')
97
>>> ord('Z')
90
>>> chr(98)
'b'
```

Code ASCII en Python

```
>>> ord('a')
97
>>> ord('Z')
90
>>> chr(98)
'b'
>>> chr(89)
```

Code ASCII en Python

```
>>> ord('a')
97
>>> ord('Z')
90
>>> chr(98)
'b'
>>> chr(89)
'Y'
```

Code ASCII en Python

```
>>> ord('a')
97
>>> ord('Z')
90
>>> chr(98)
'b'
>>> chr(89)
'Y'
```

`ord` pour accéder au code entier du caractère

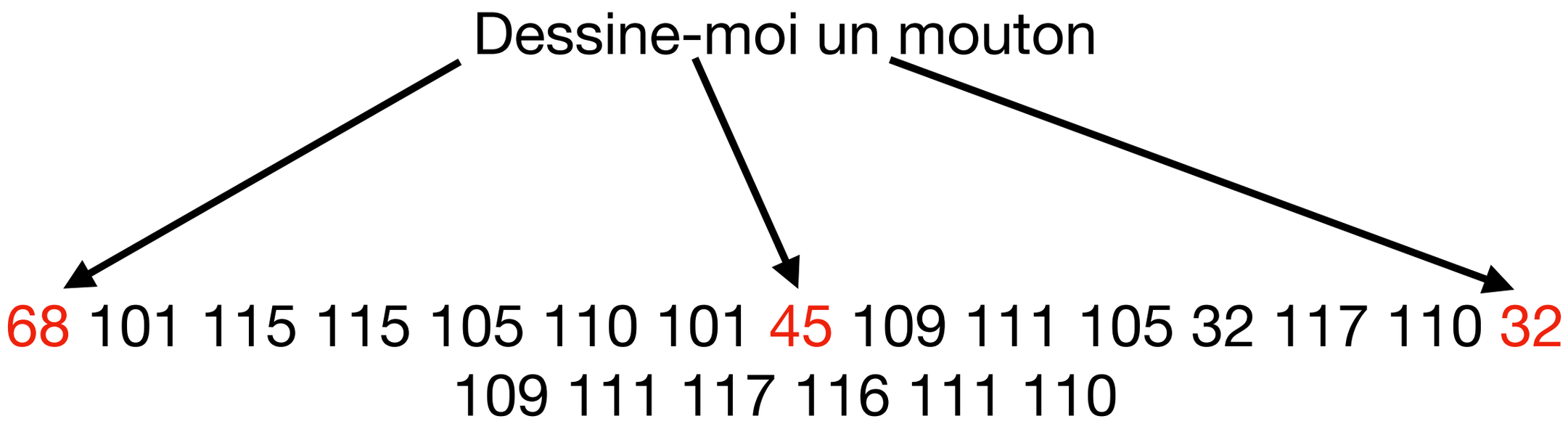
Code ASCII en Python

```
>>> ord('a')
97
>>> ord('Z')
90
>>> chr(98)
'b'
>>> chr(89)
'Y'
```

`ord` pour accéder au code entier du caractère

`chr` pour obtenir le caractère dont le code entier est donné

Représenter du texte



Exercice 4

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

- 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

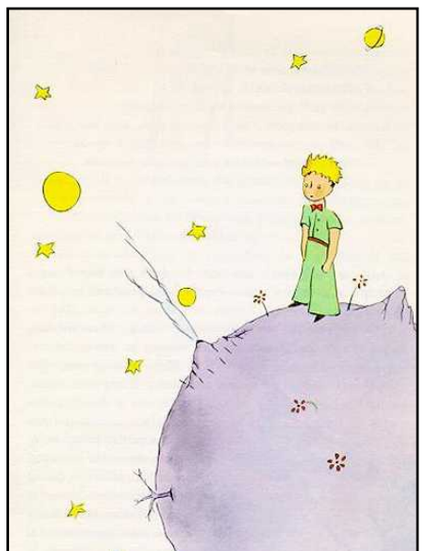
amazonie

Chercher : Le Petit Prince

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

Trier par :

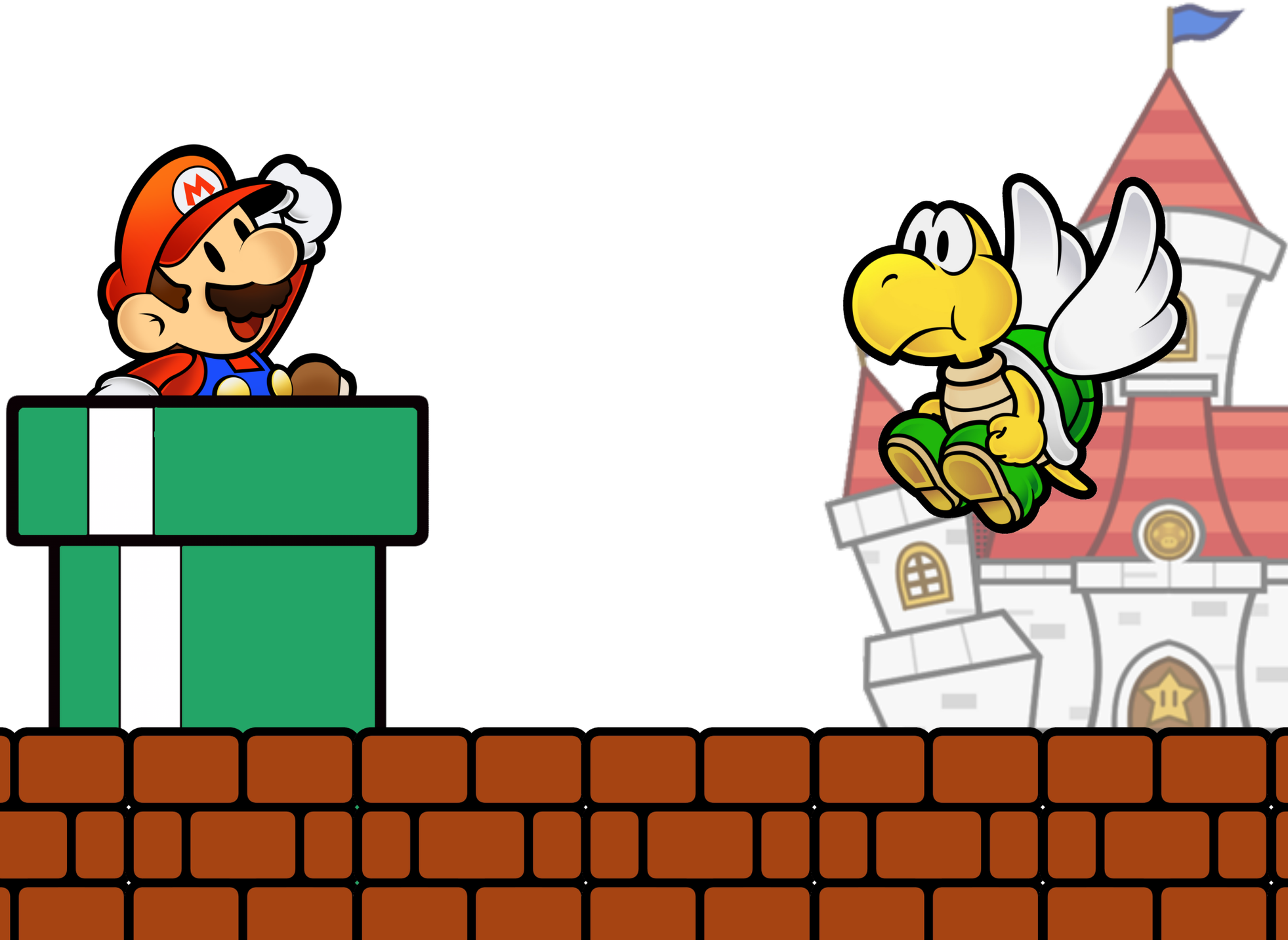
- prix croissant
- prix décroissant
- note moyenne
- nouveauté



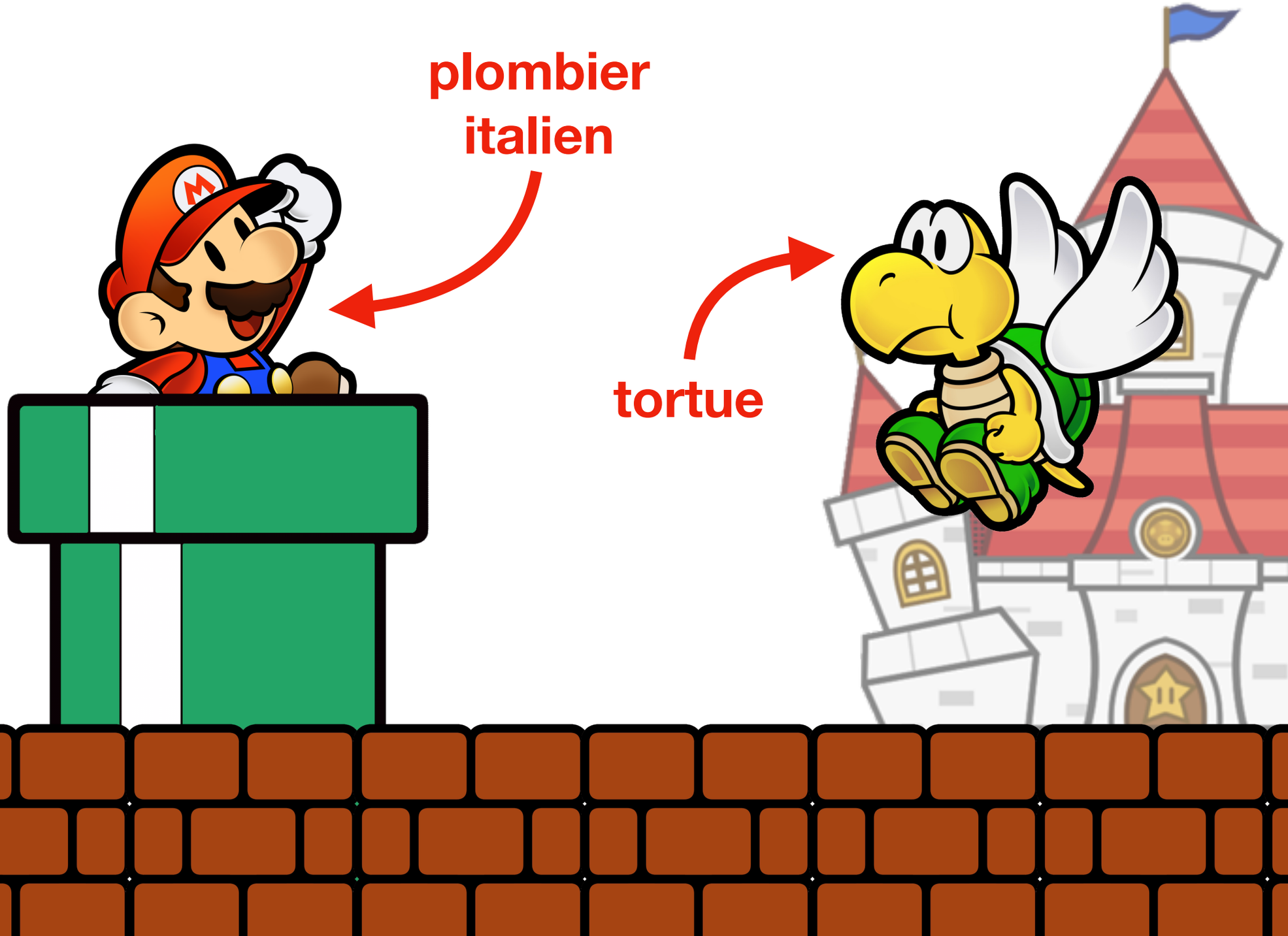
Le Petit Prince
de Antoine de Saint-Exu

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

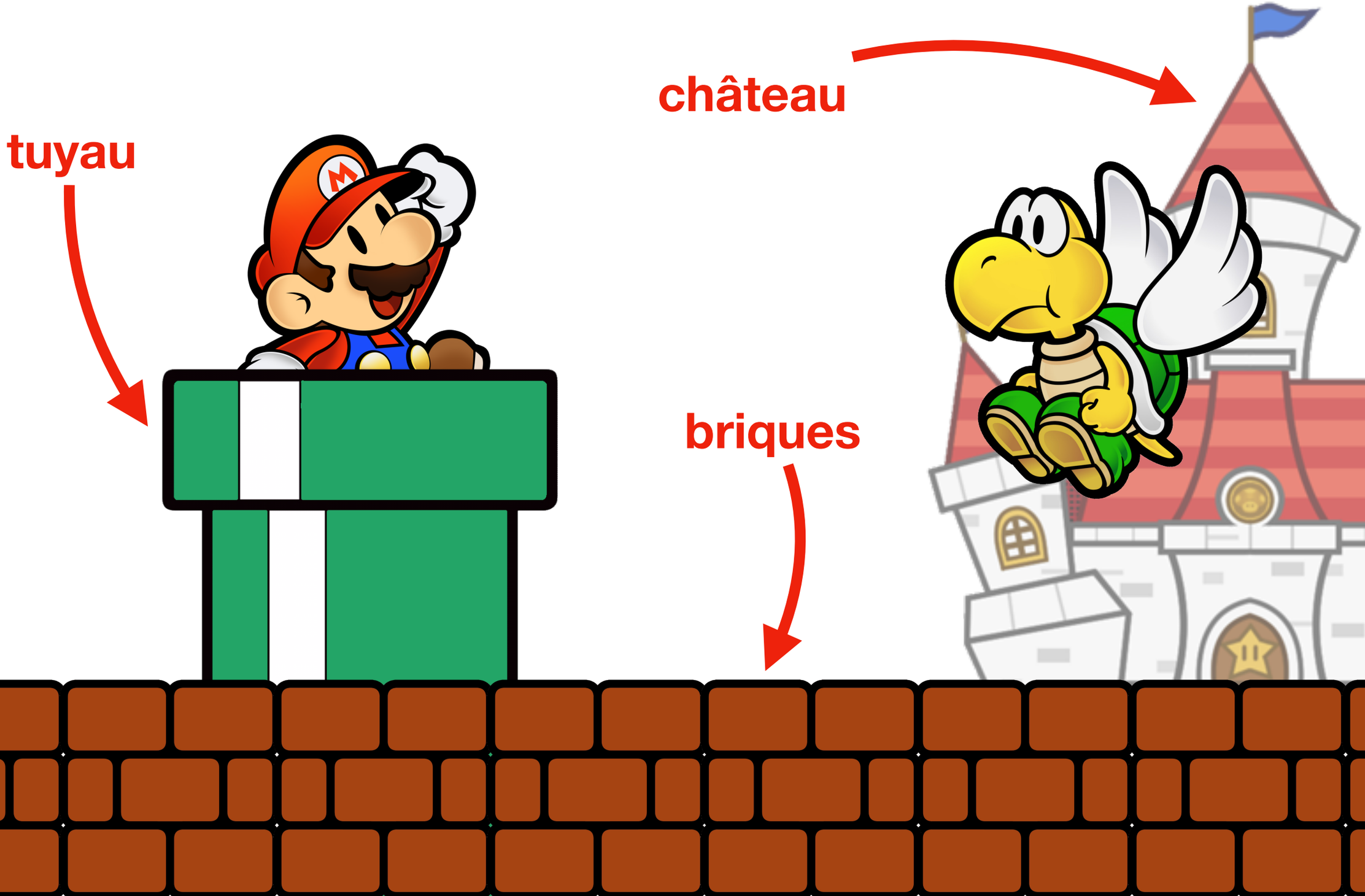
Algos de tri dans le jeux vidéo « Super Plombiers Italiens »



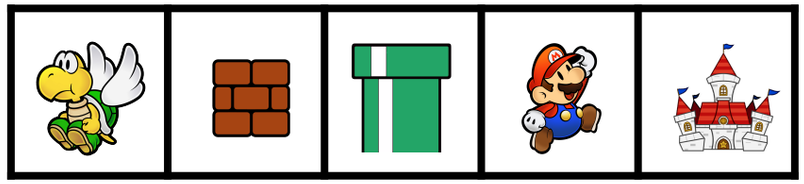
Algos de tri dans le jeux vidéo « Super Plombiers Italiens »



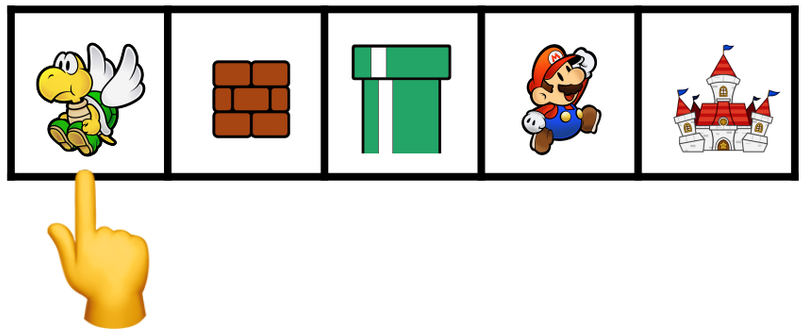
Algos de tri dans le jeux vidéo « Super Plombiers Italiens »



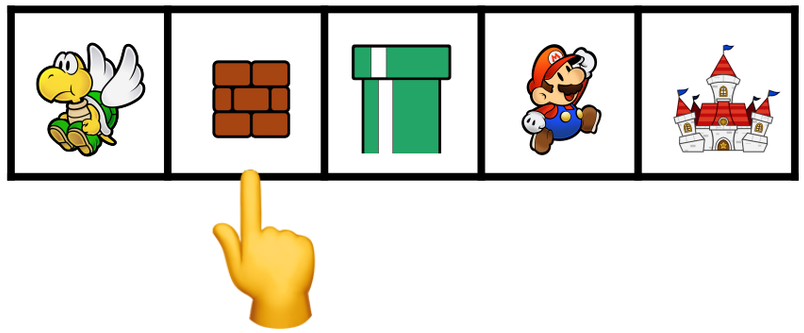
Affichage des objets dans le mauvais ordre



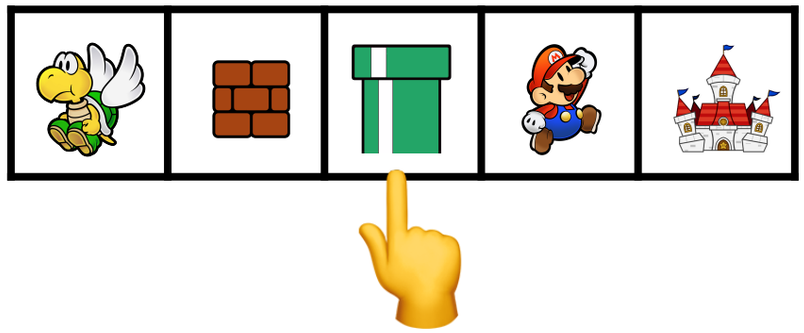
Affichage des objets dans le mauvais ordre

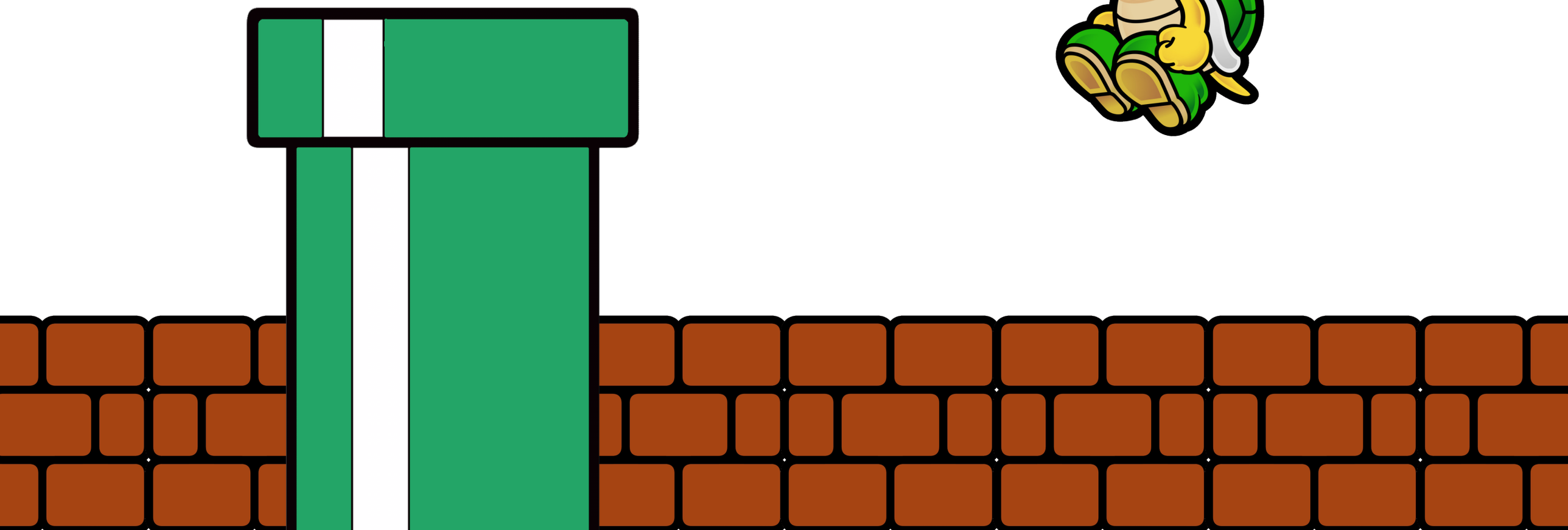


Affichage des objets dans le mauvais ordre

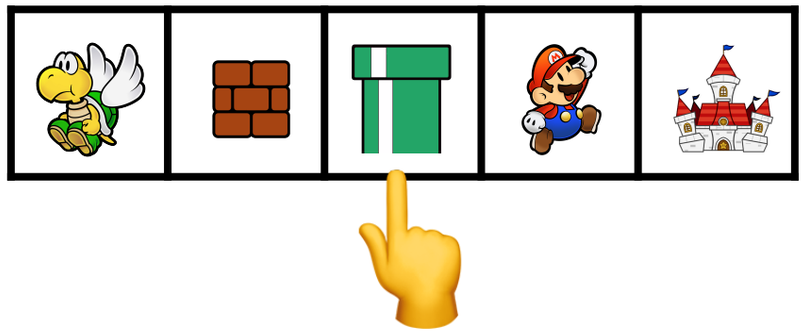


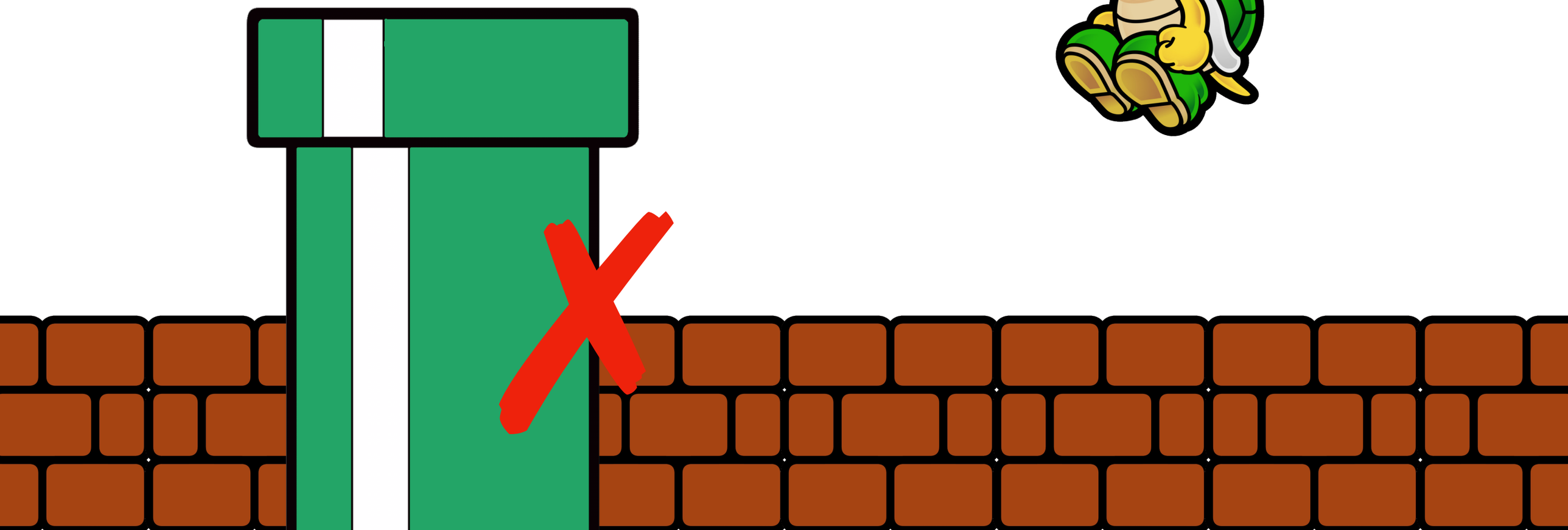
Affichage des objets dans le mauvais ordre



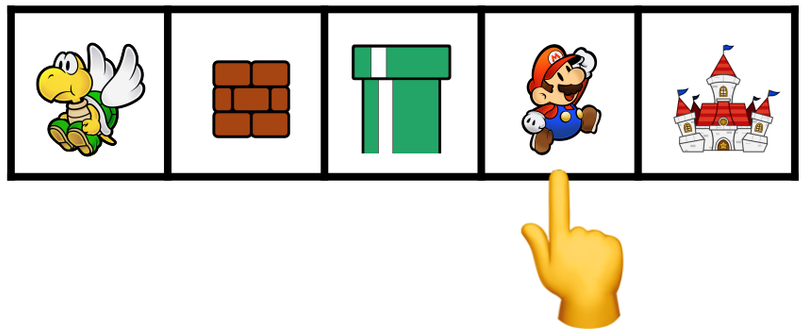


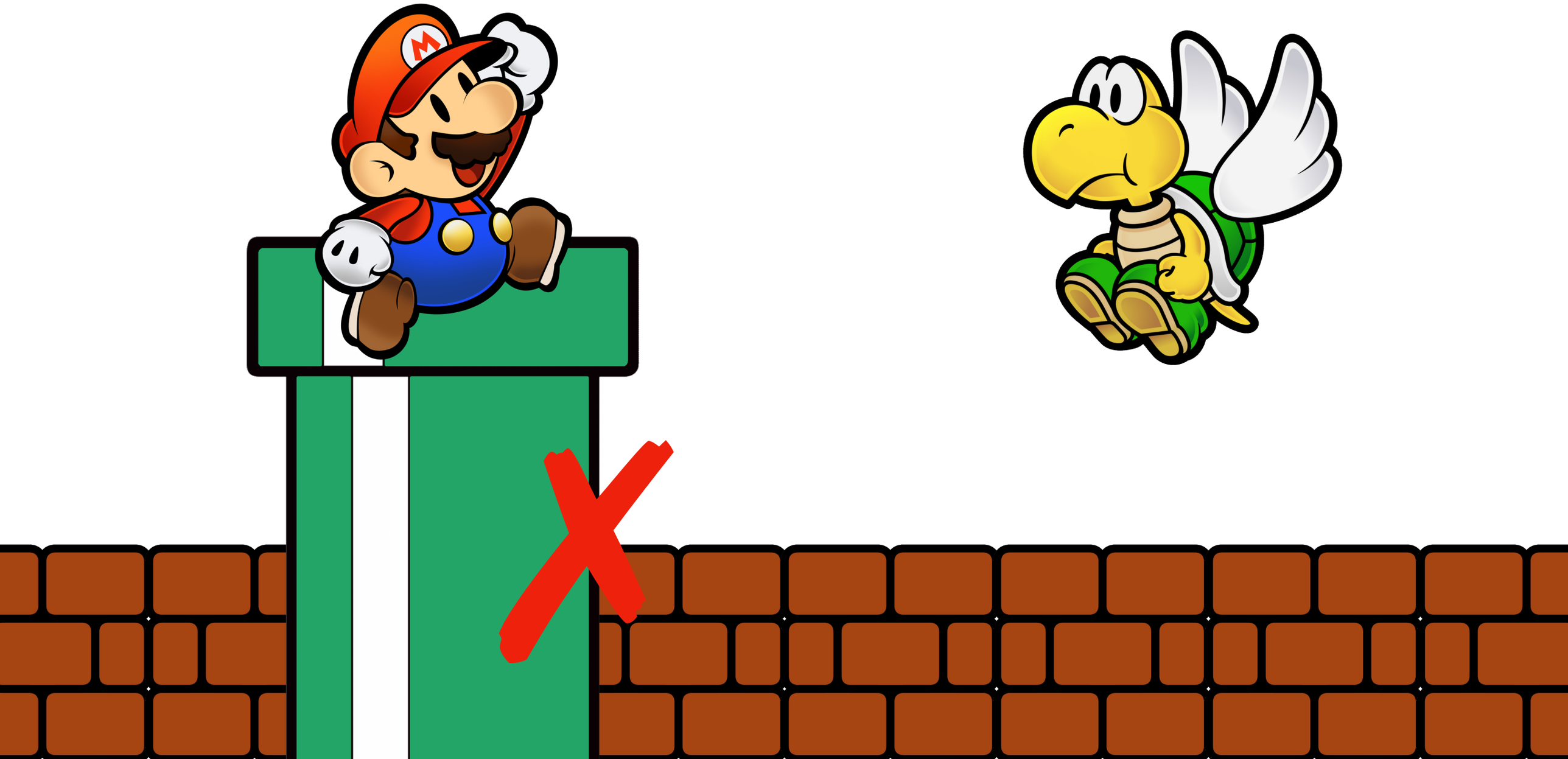
Affichage des objets dans le mauvais ordre



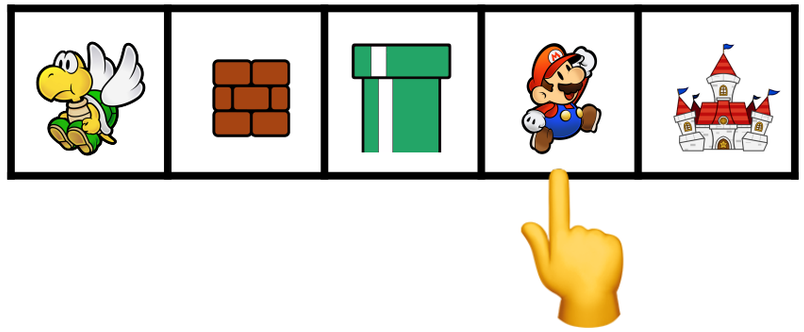


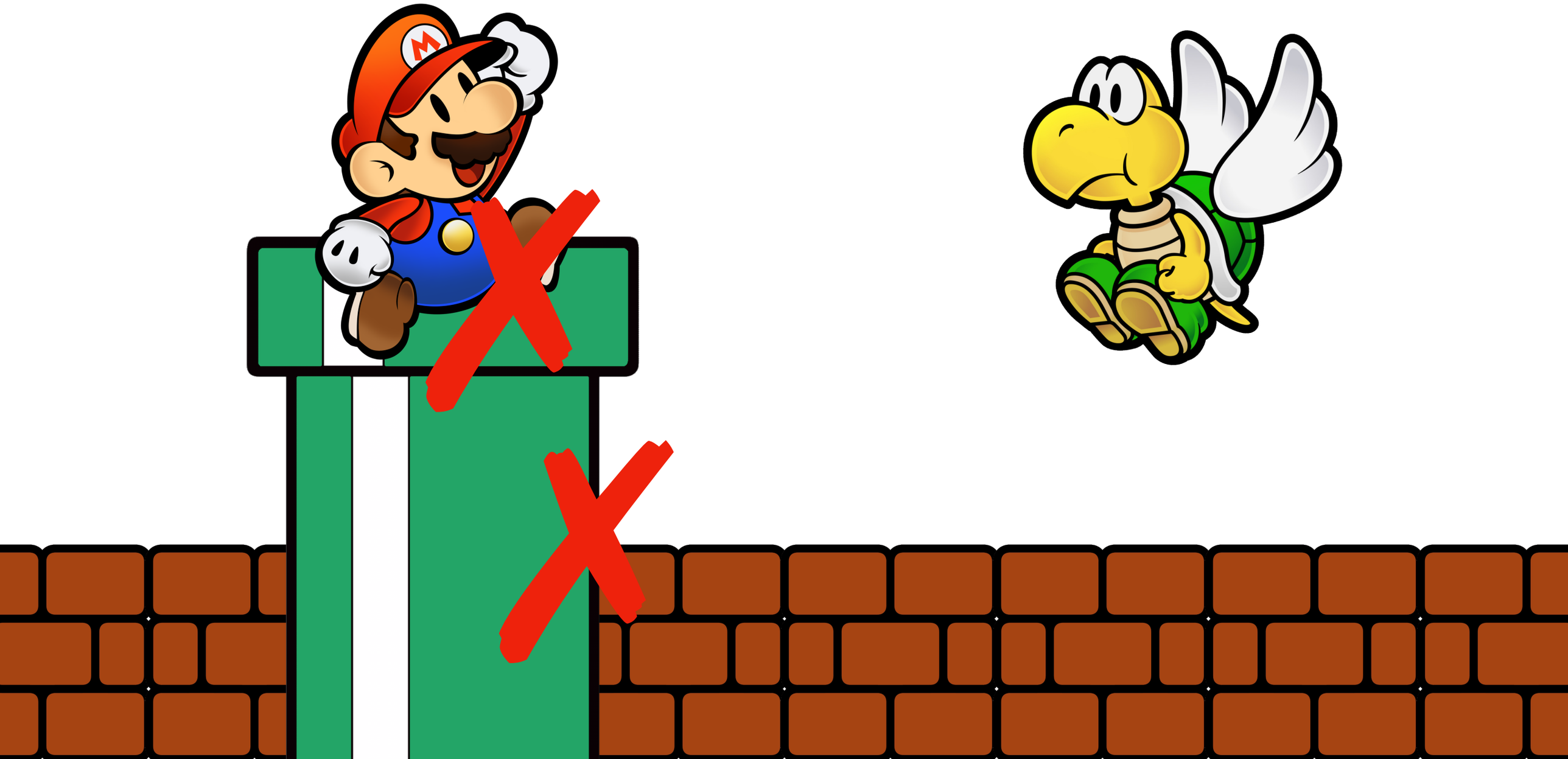
Affichage des objets dans le mauvais ordre



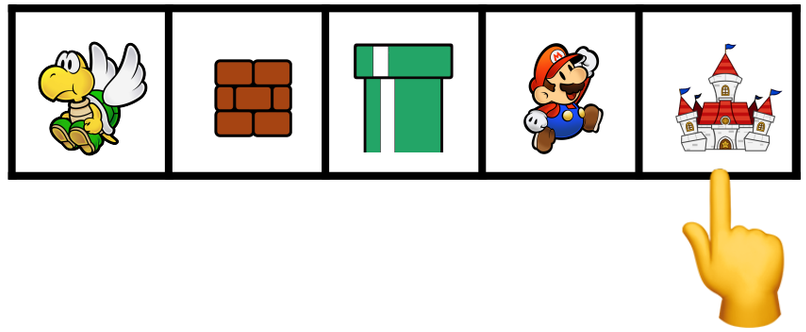


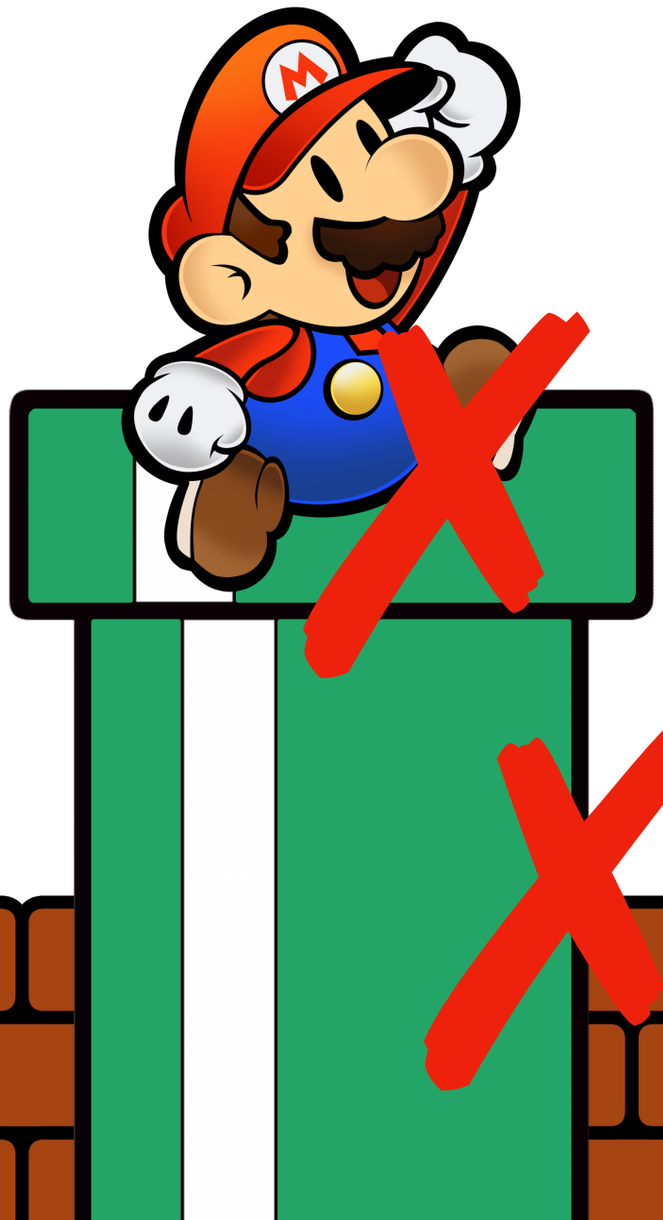
Affichage des objets dans le mauvais ordre



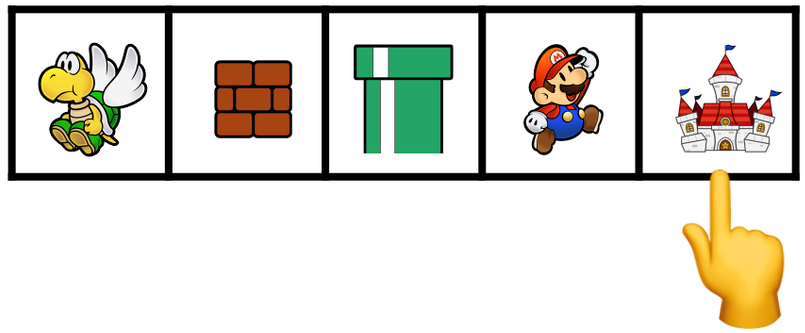


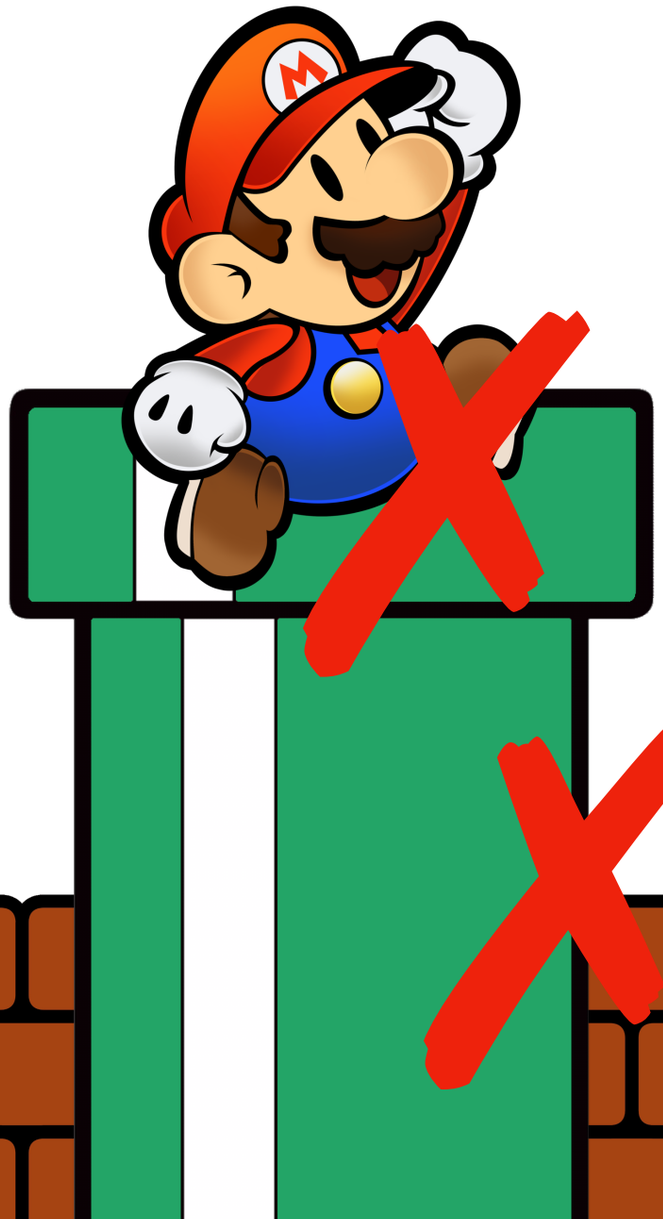
Affichage des objets dans le mauvais ordre



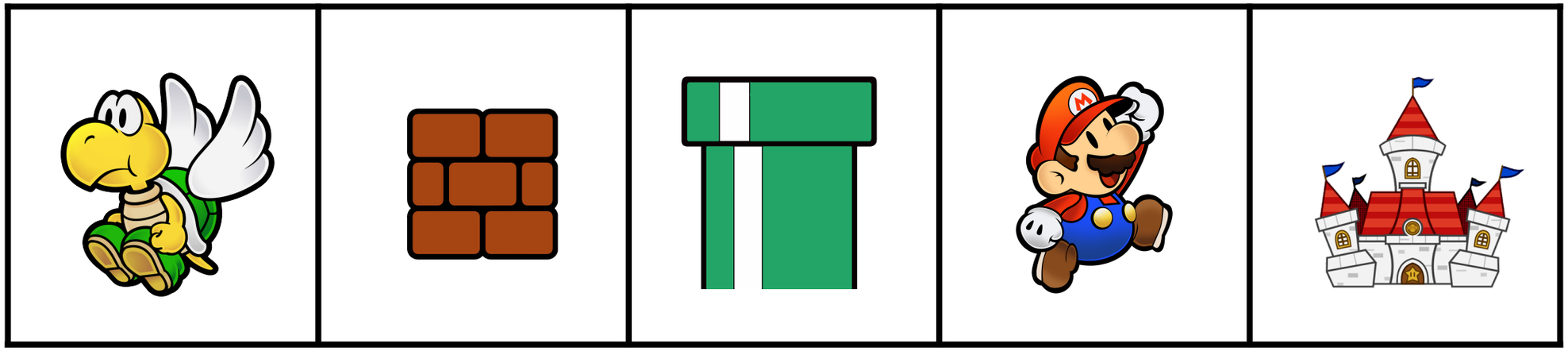


Affichage des objets dans le mauvais ordre



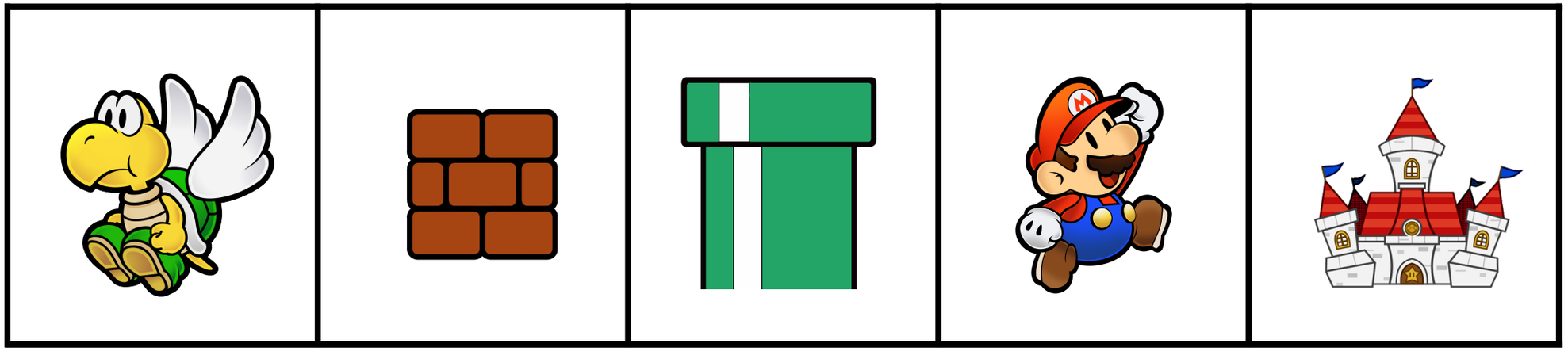


Affichage des objets dans le mauvais ordre



0 1 2 3 4

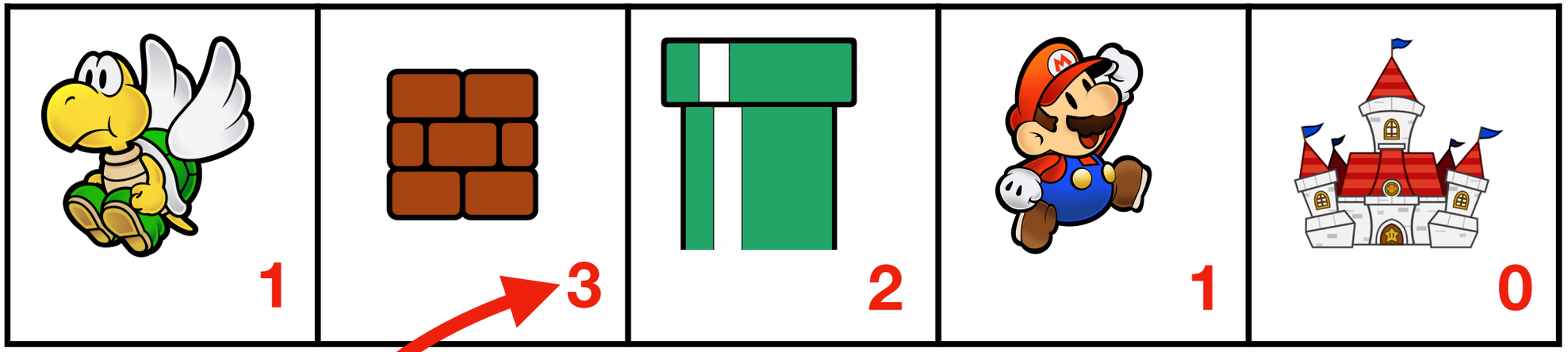
Affichage des objets dans le mauvais ordre



0 1 2 3 4

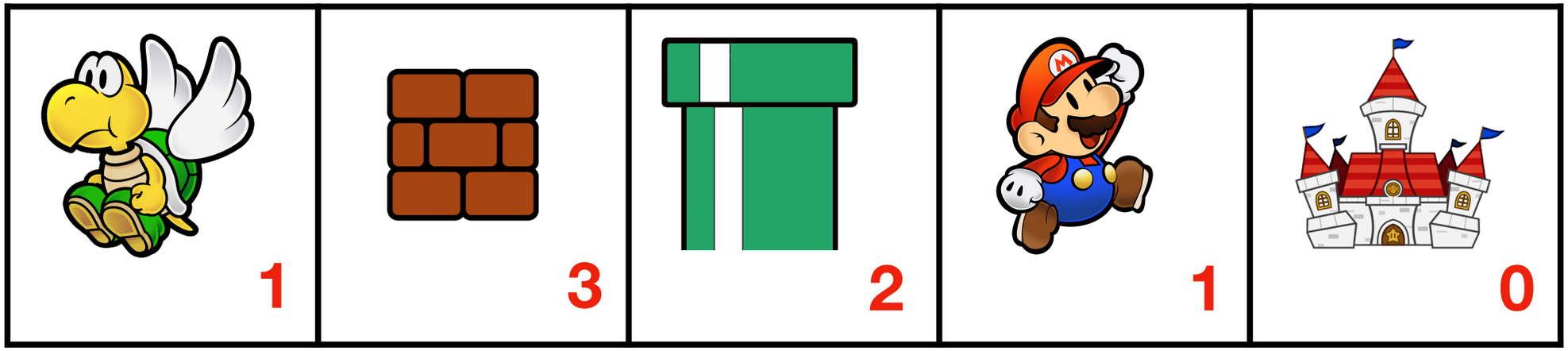
ordre d’affichage

Affichage des objets dans le mauvais ordre



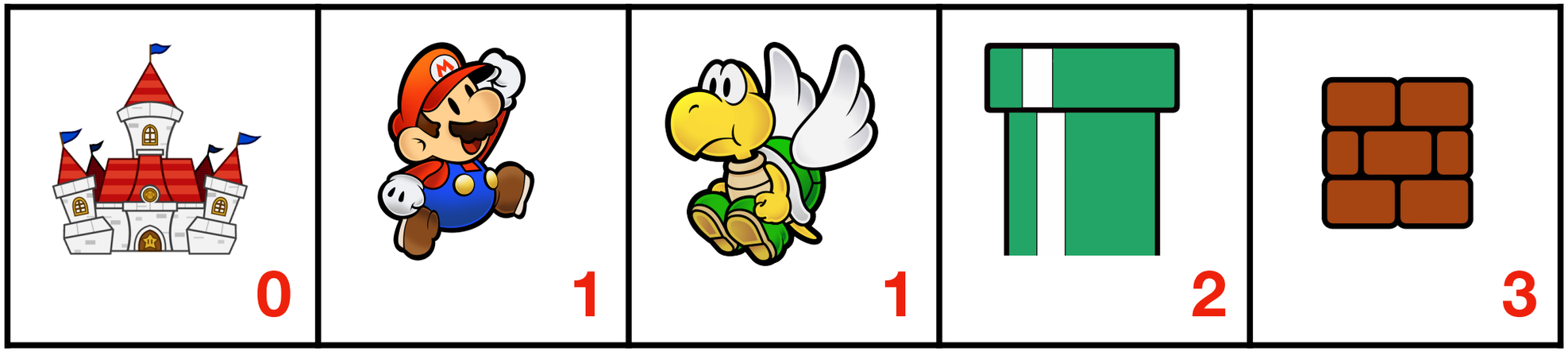
Tri ! 

Affichage des objets dans le mauvais ordre



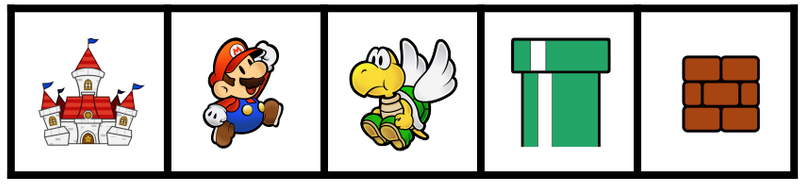
0 1 2 3 4

Affichage des objets dans le bon ordre

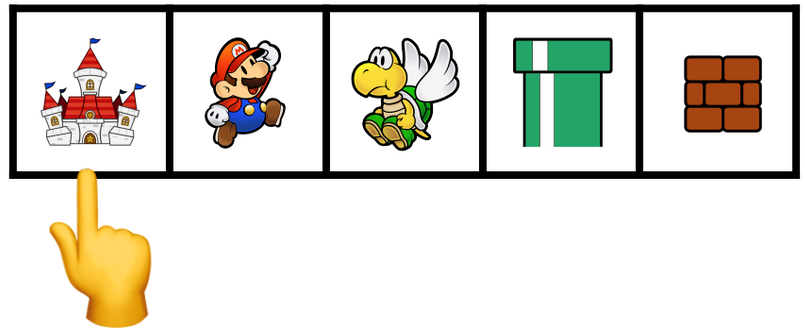


0 1 2 3 4

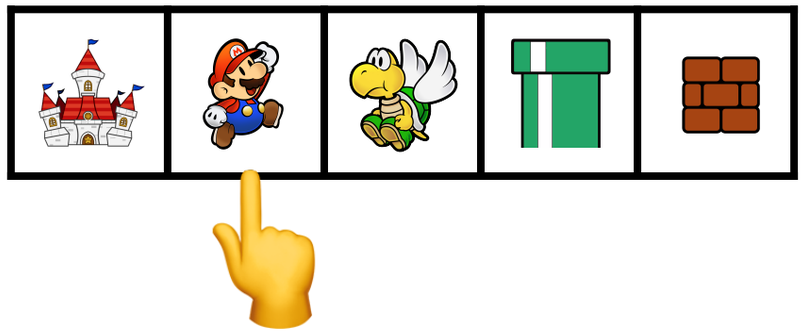
Affichage des objets dans le bon ordre



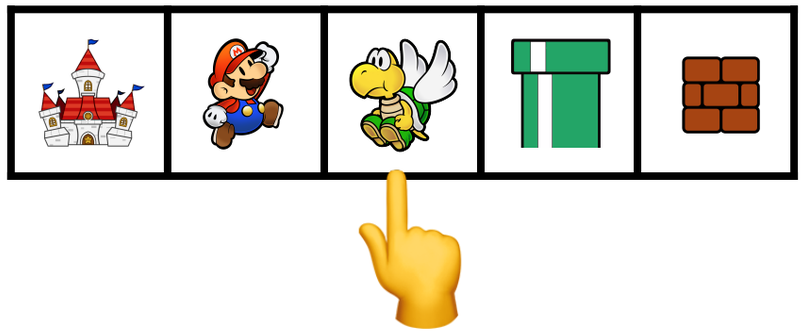
Affichage des objets dans le bon ordre

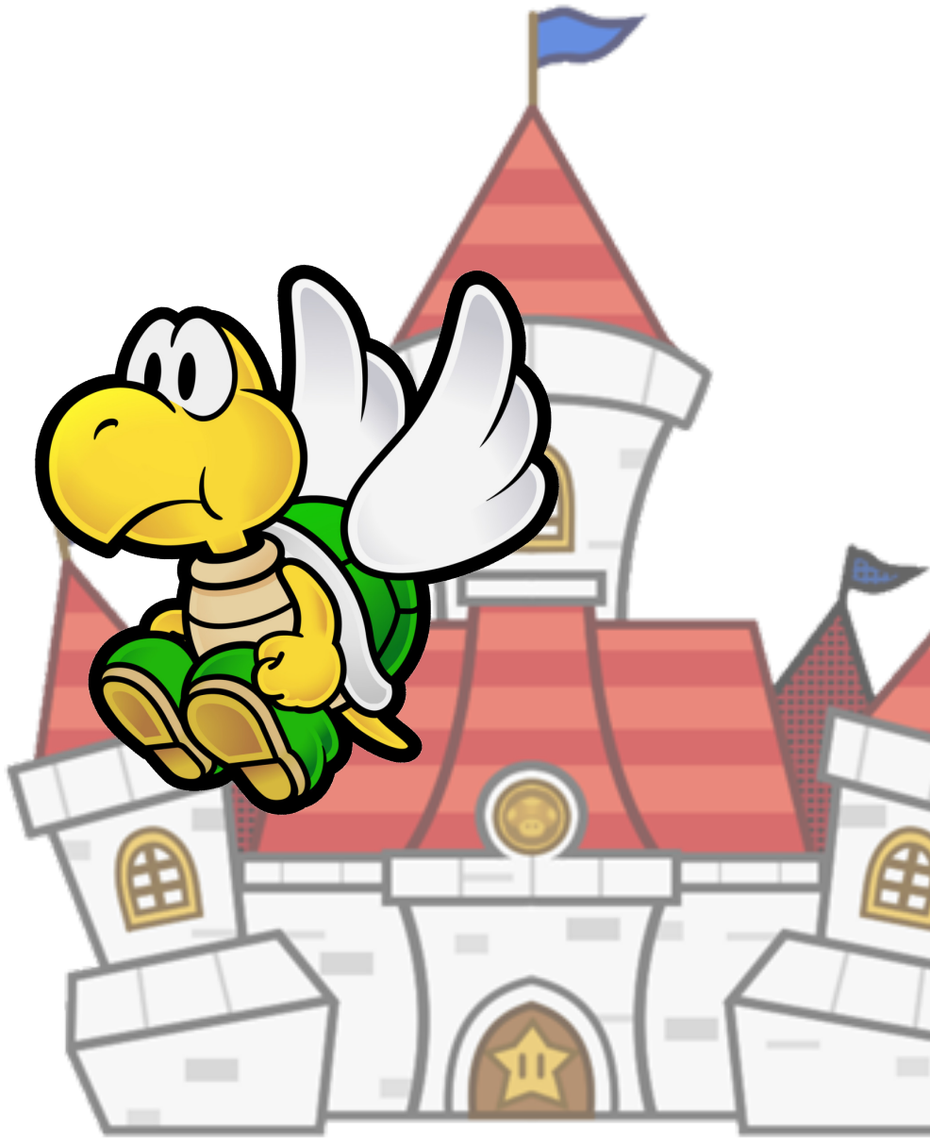


Affichage des objets dans le bon ordre

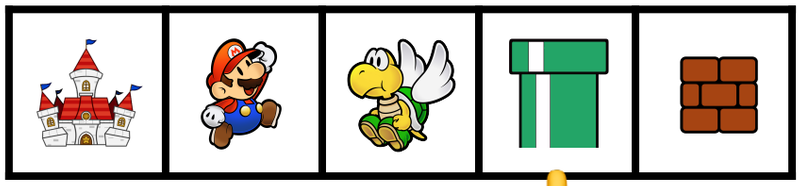


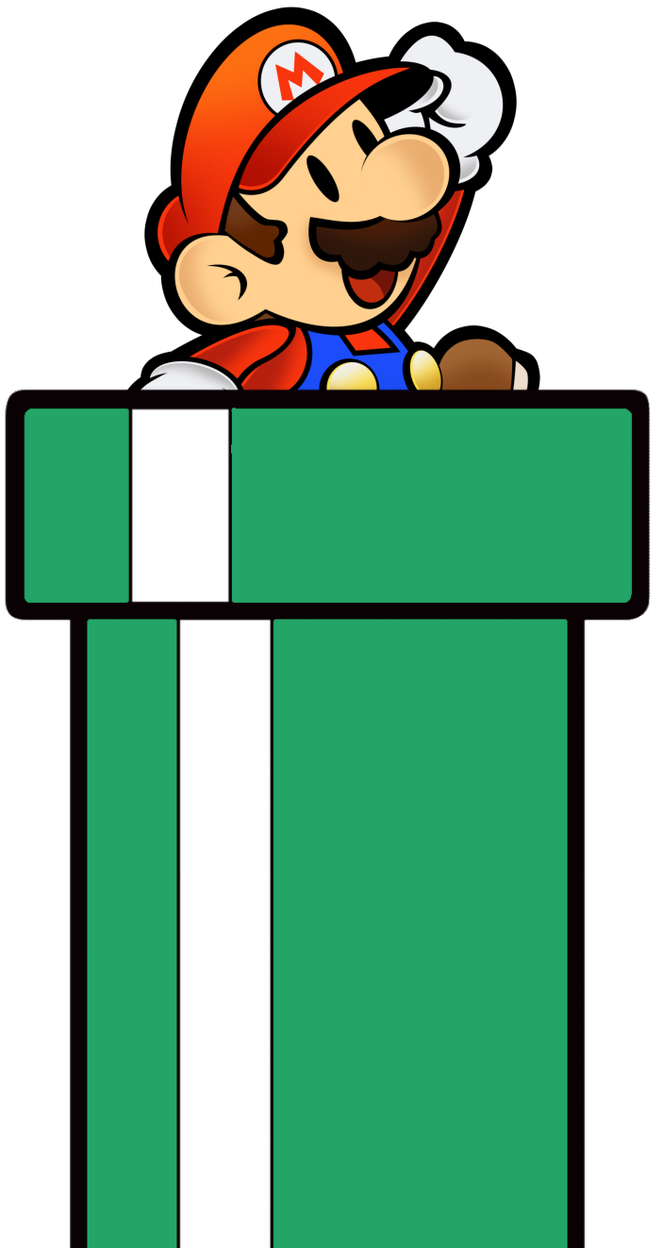
Affichage des objets dans le bon ordre



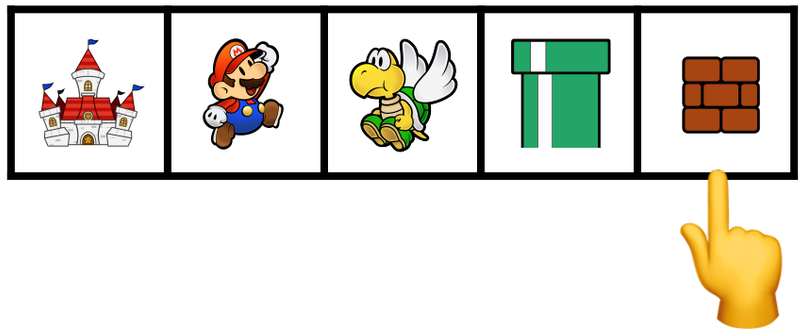


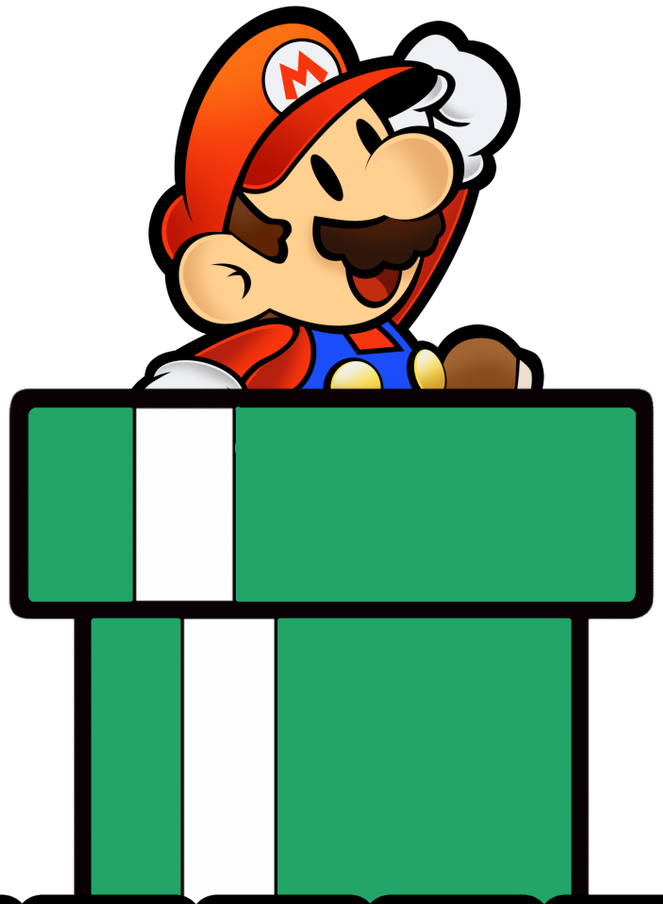
Affichage des objets dans le bon ordre

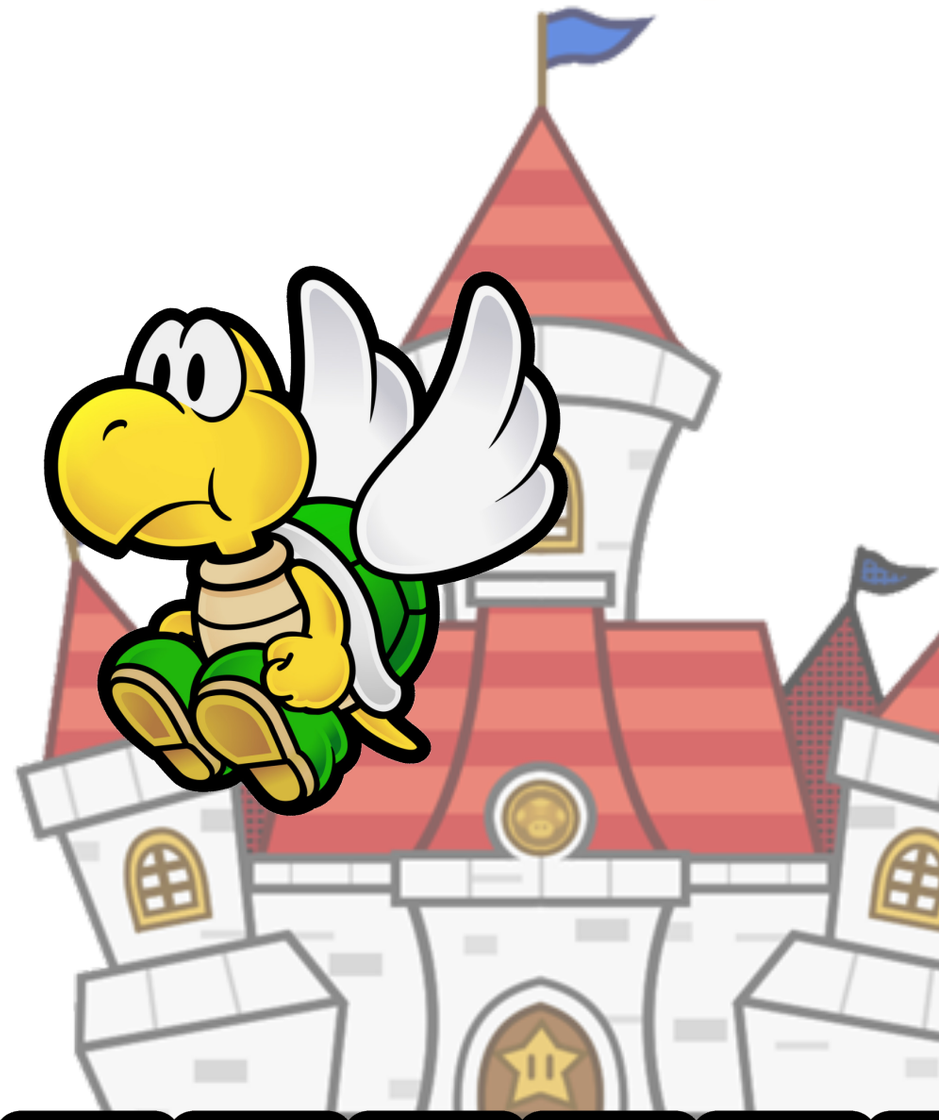




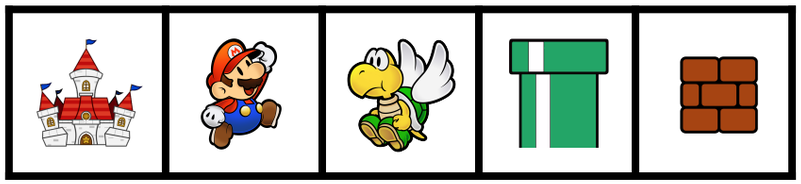
Affichage des objets dans le bon ordre

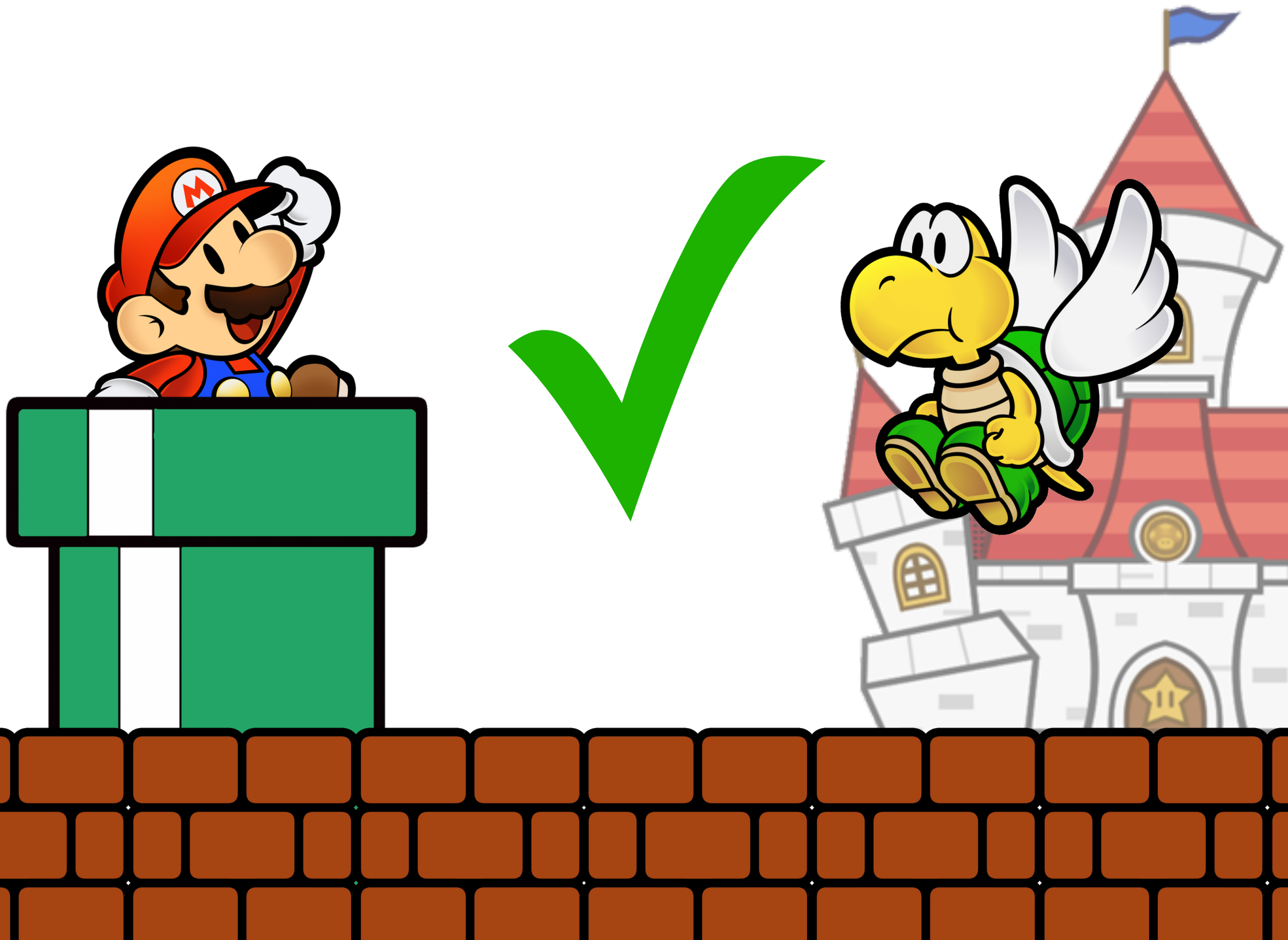




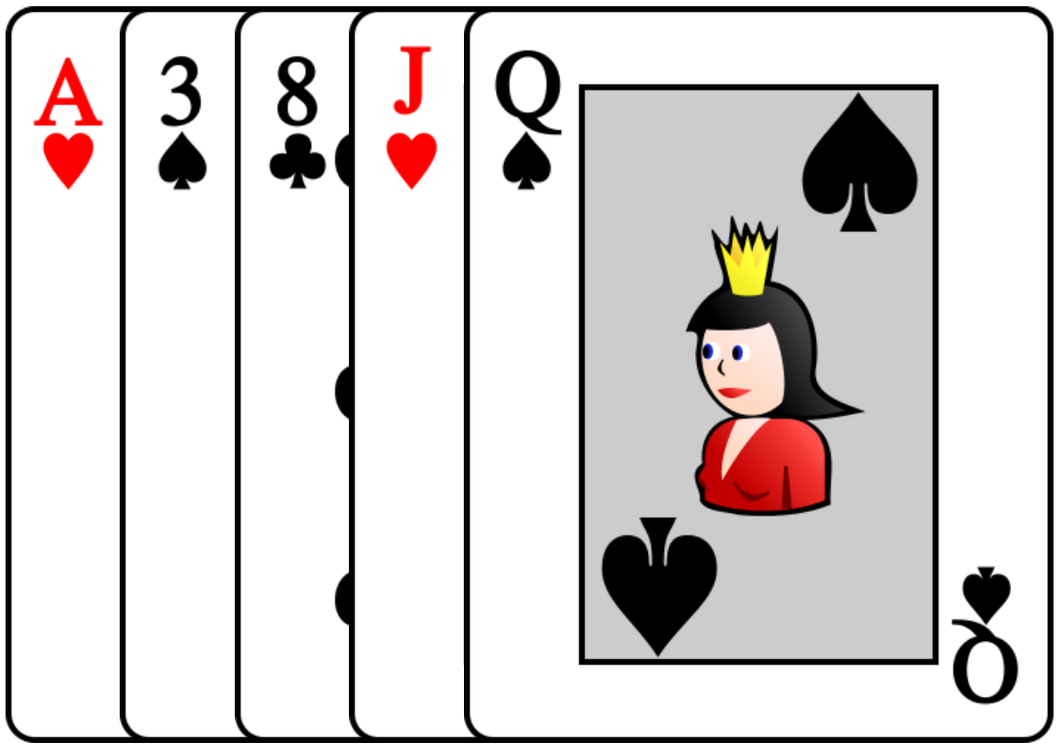


Affichage des objets dans le bon ordre



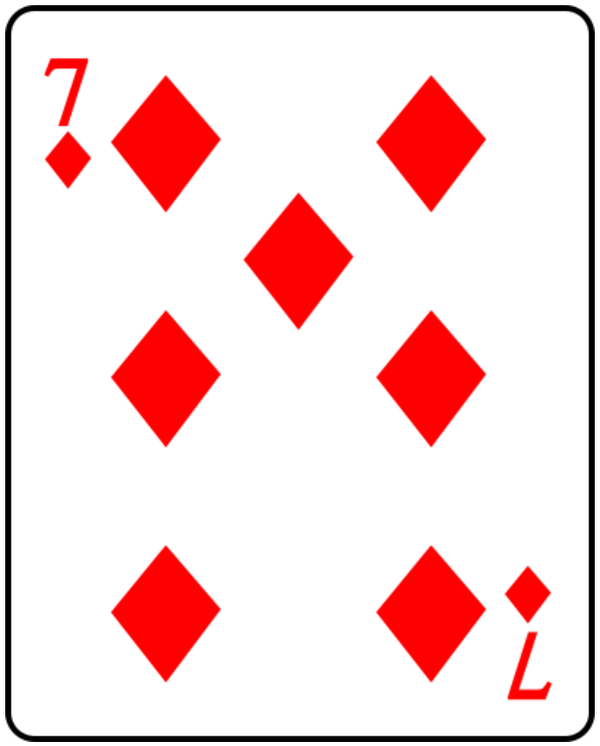


Comment trier un jeu de cartes ?

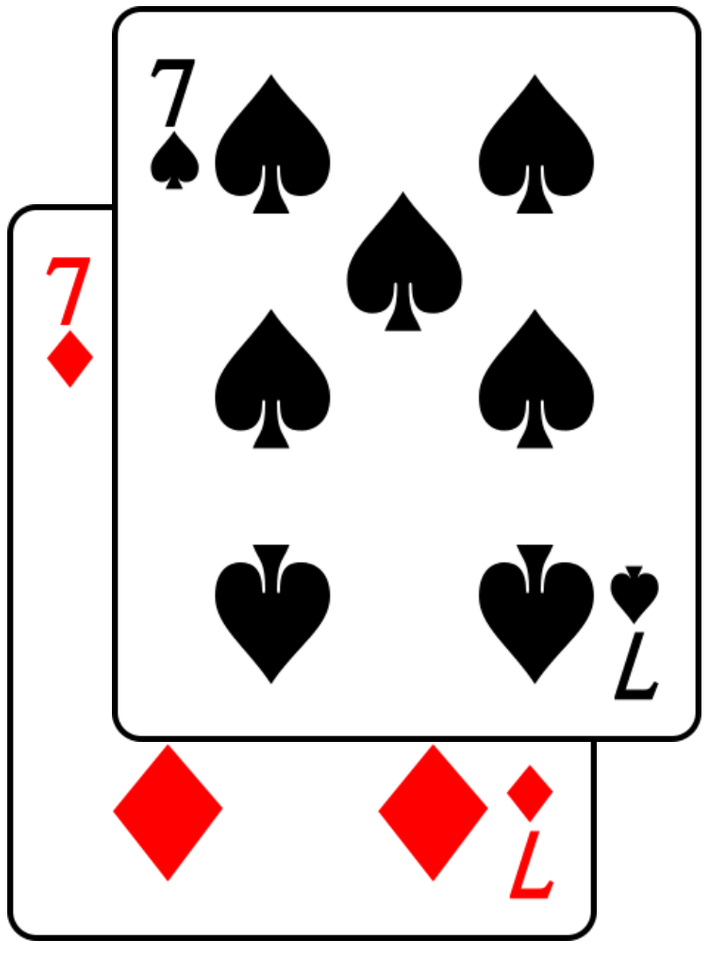


Tri par insertion

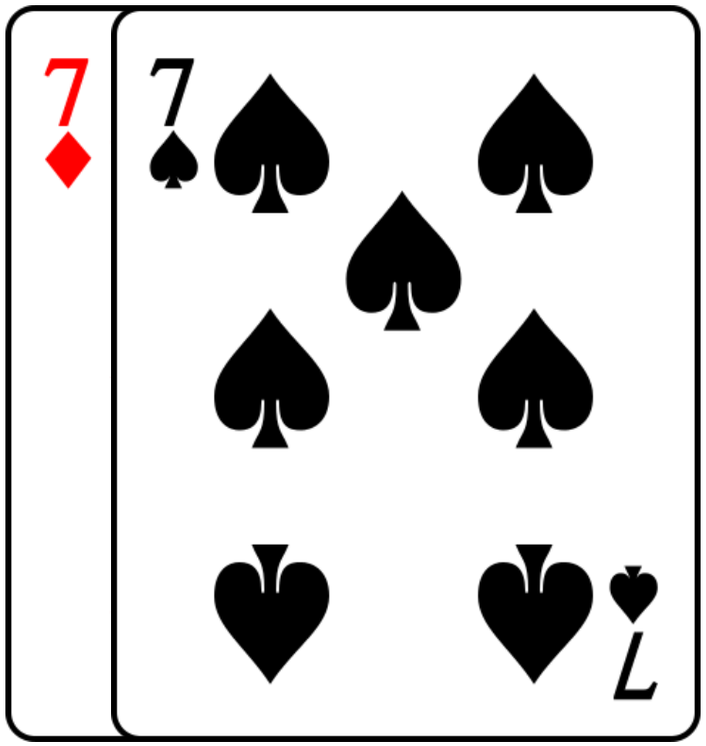
Tri par insertion



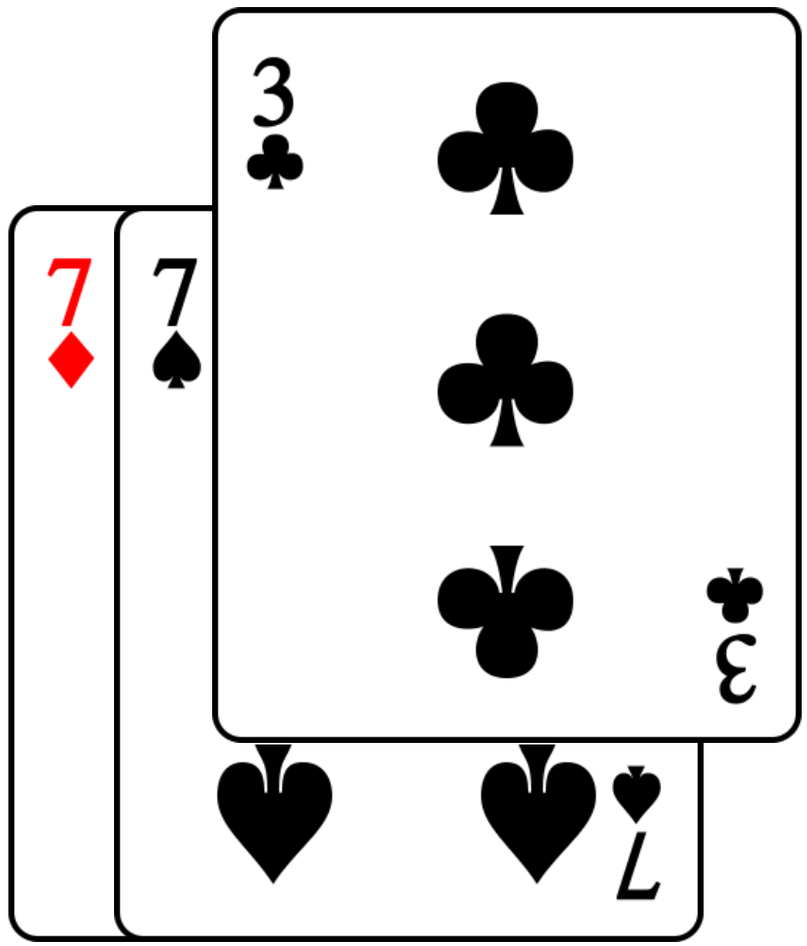
Tri par insertion



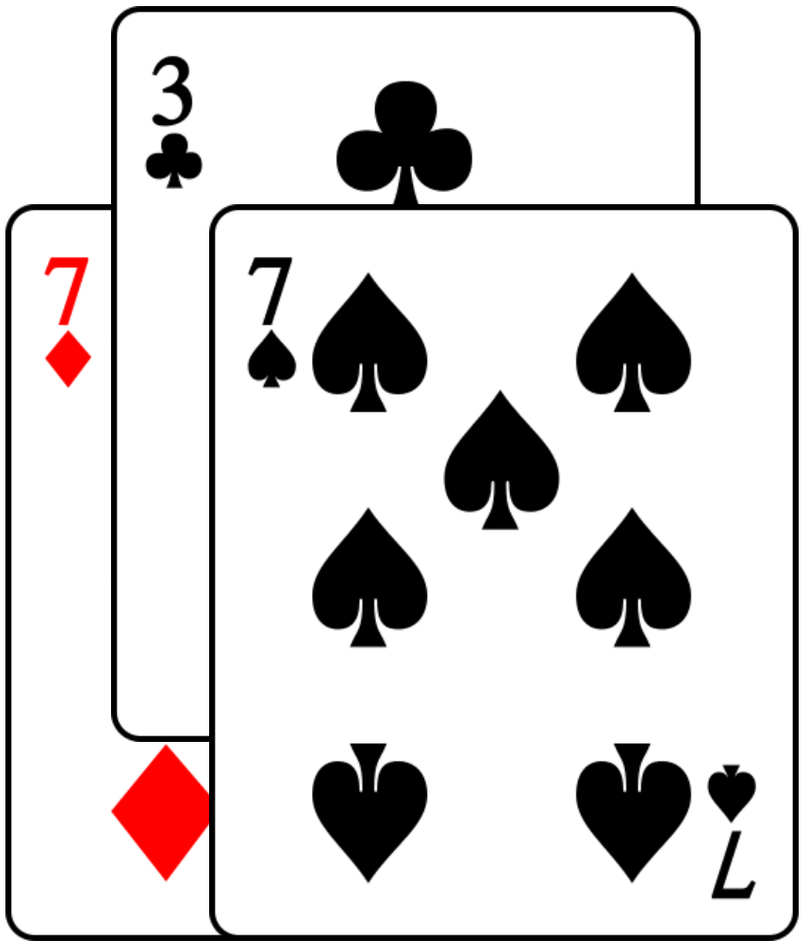
Tri par insertion



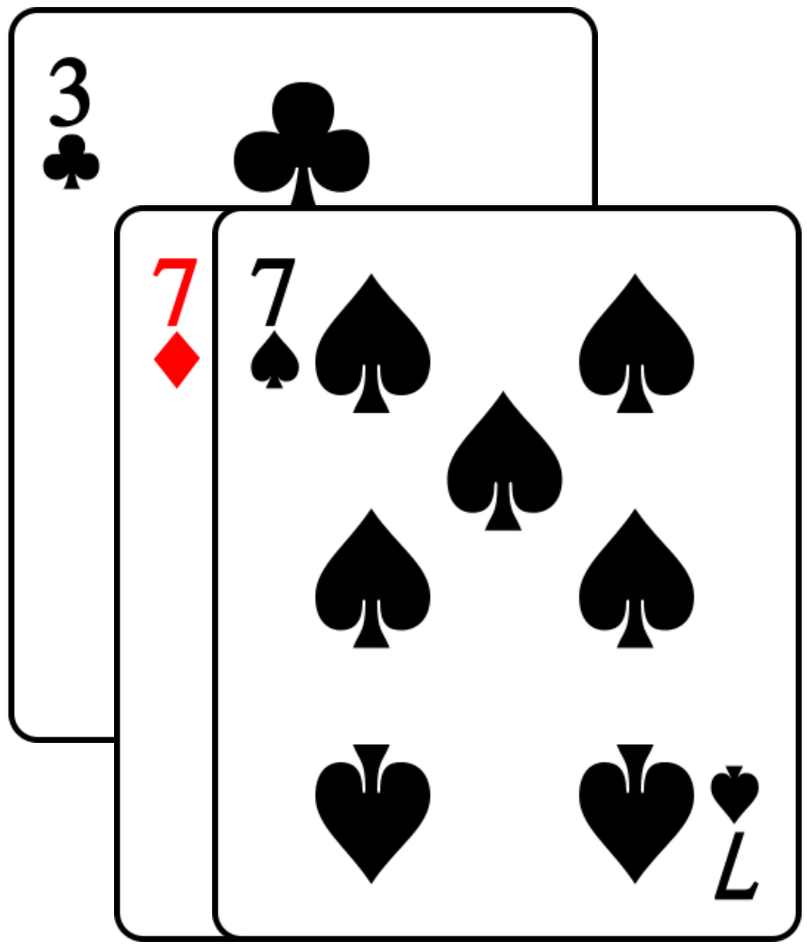
Tri par insertion



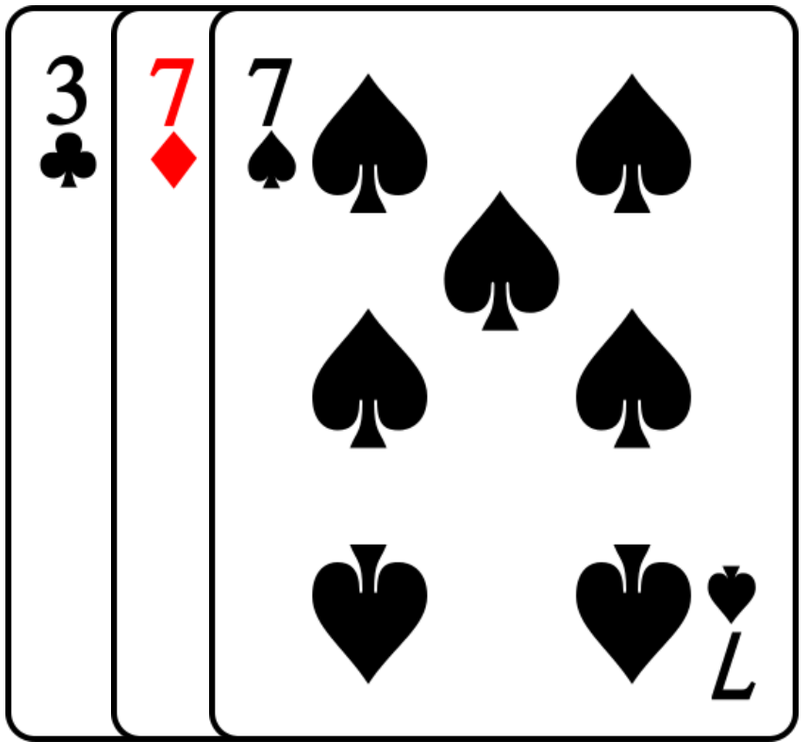
Tri par insertion



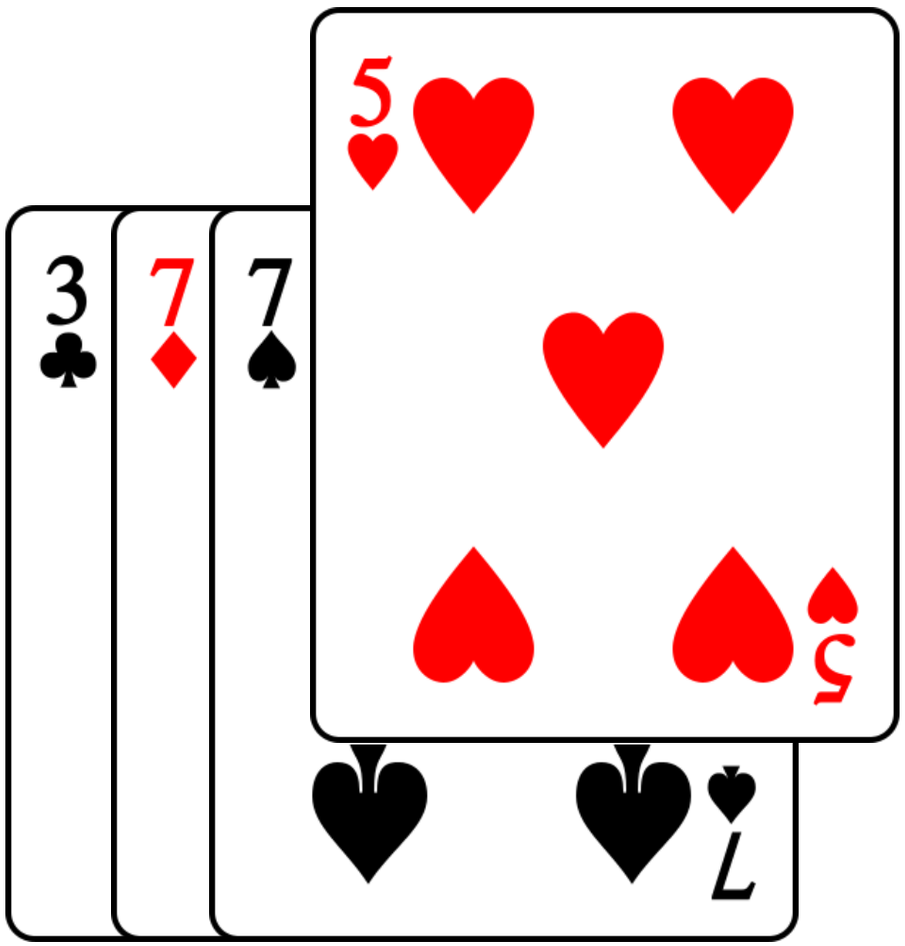
Tri par insertion



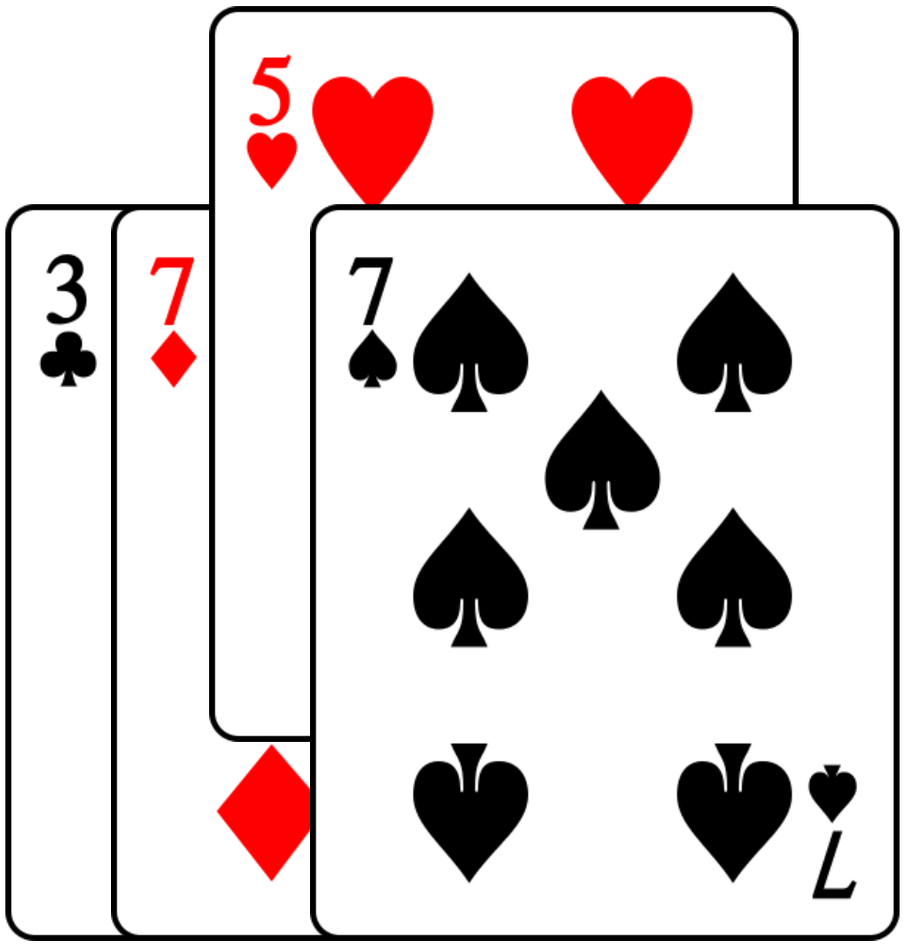
Tri par insertion



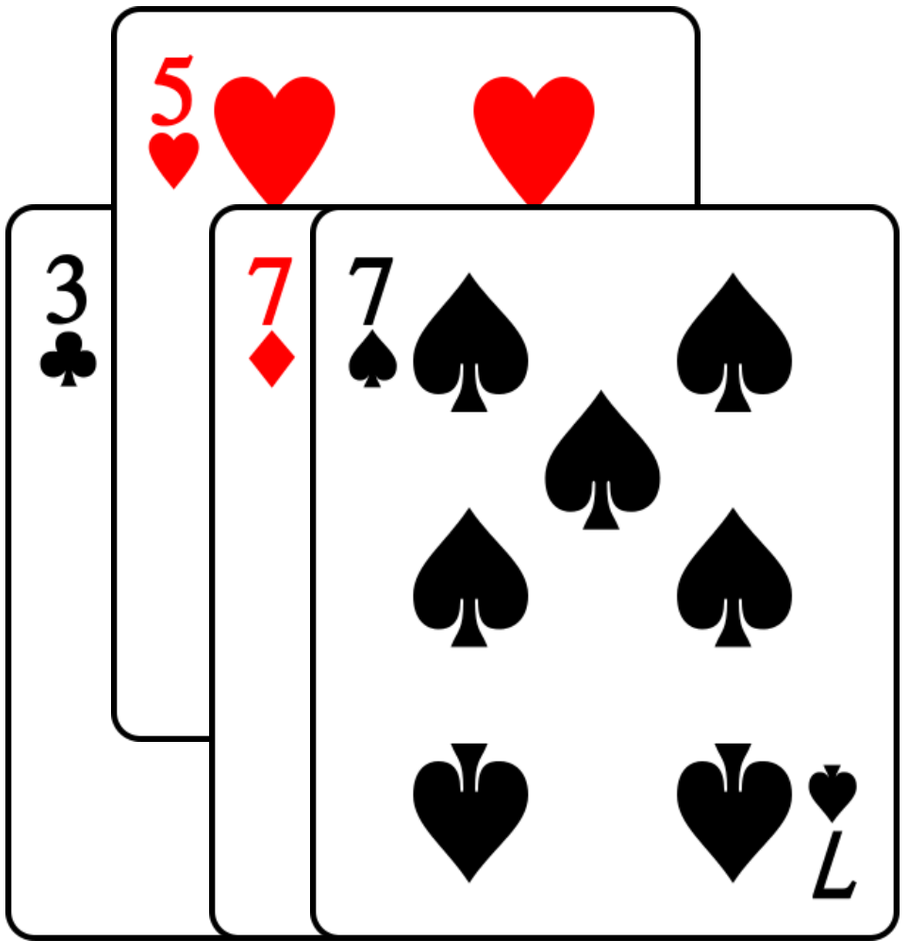
Tri par insertion



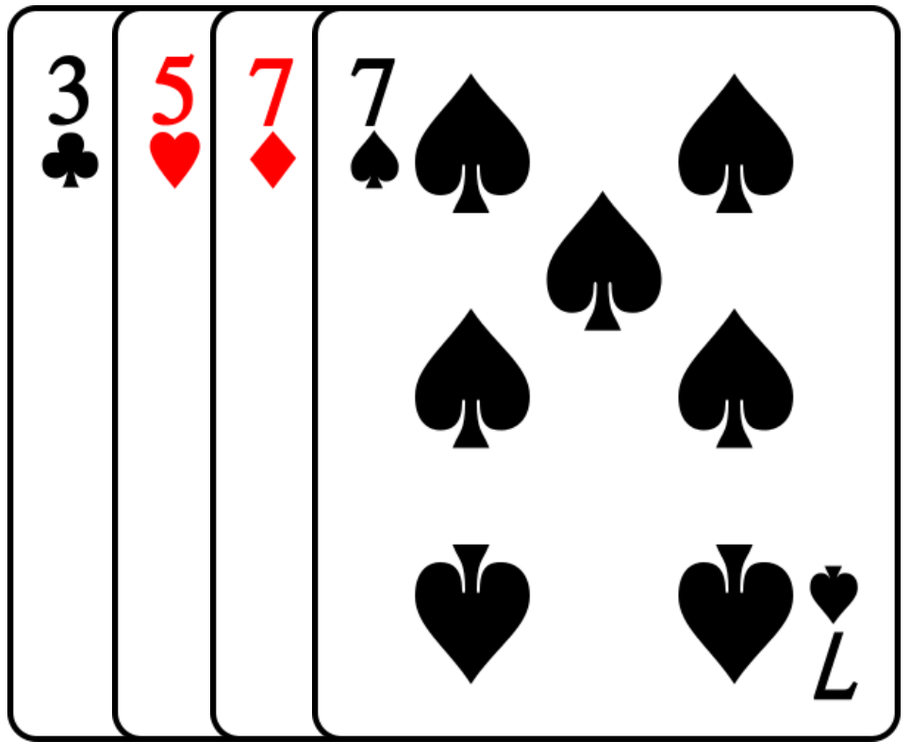
Tri par insertion



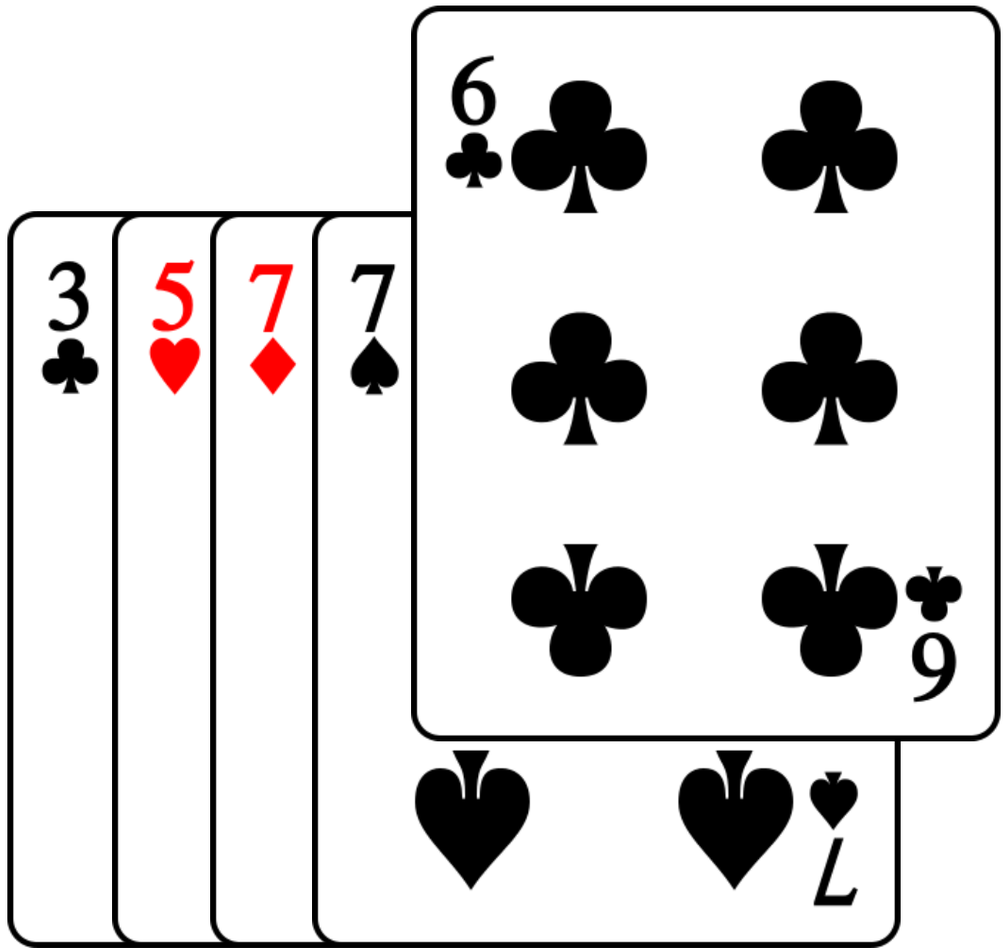
Tri par insertion



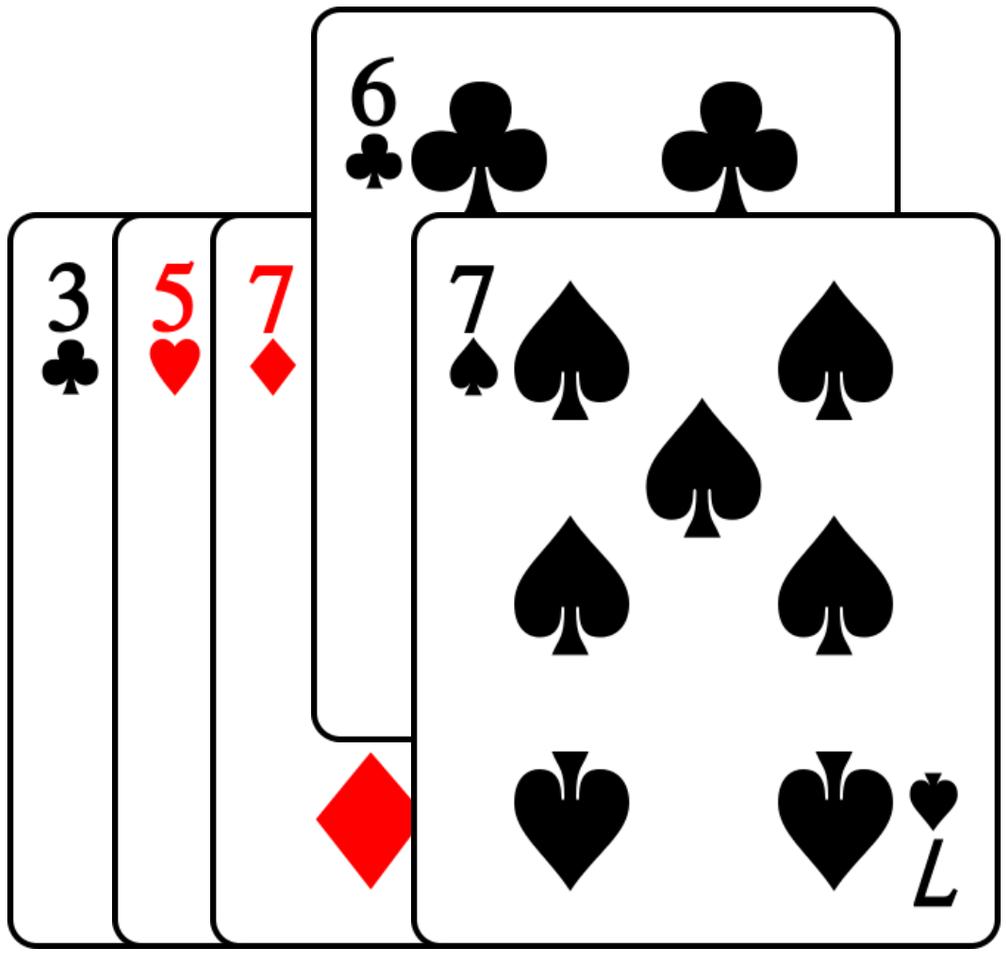
Tri par insertion



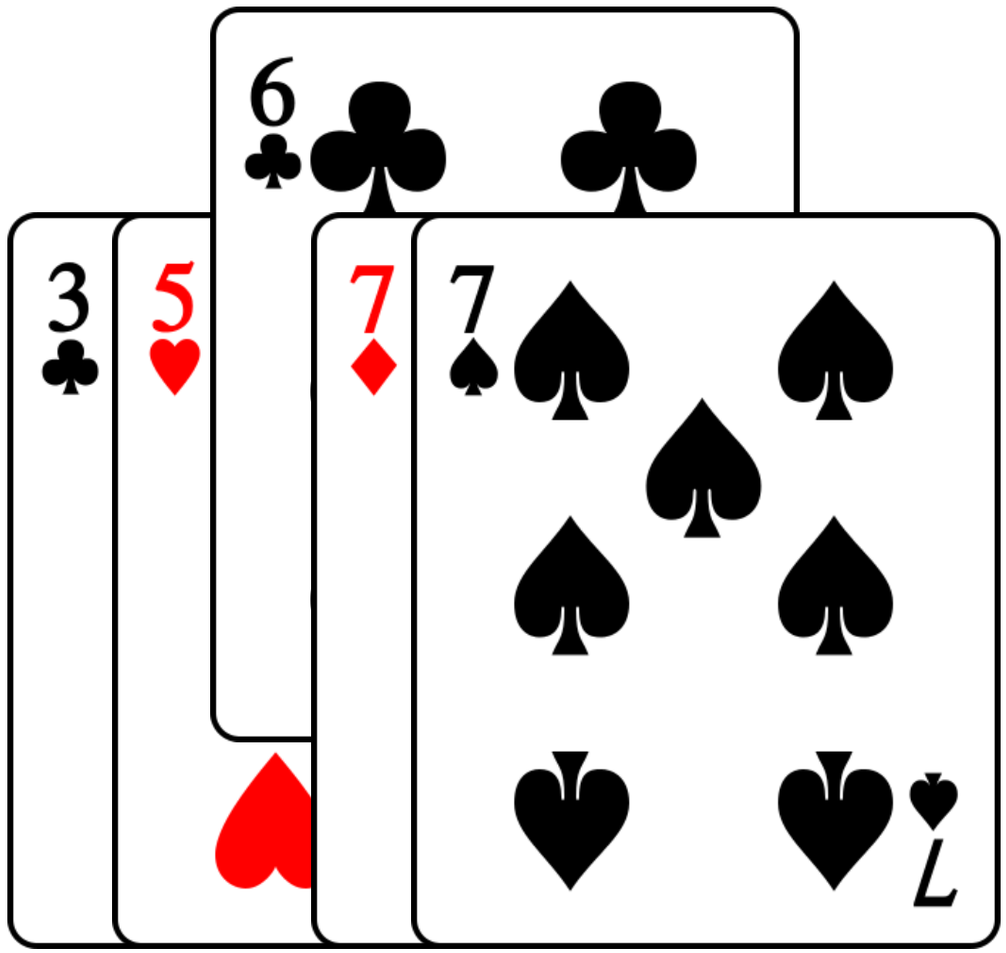
Tri par insertion



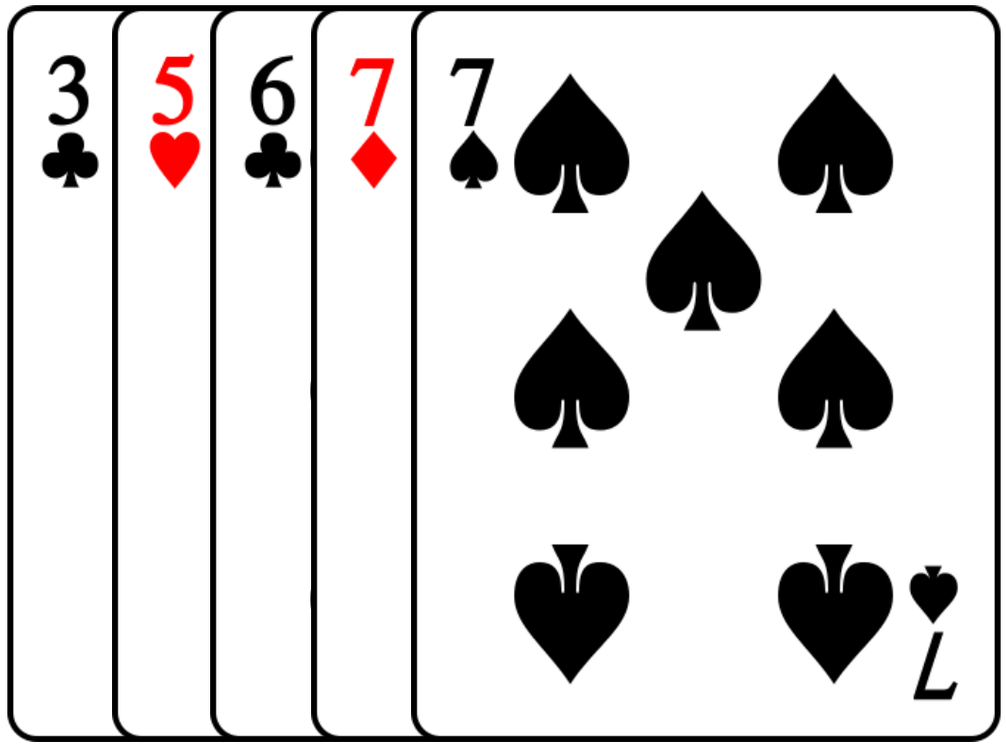
Tri par insertion



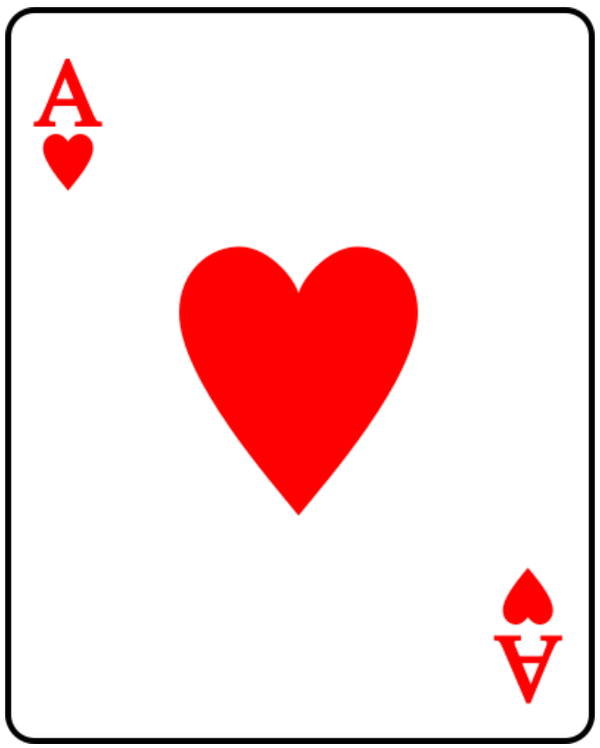
Tri par insertion



Tri par insertion

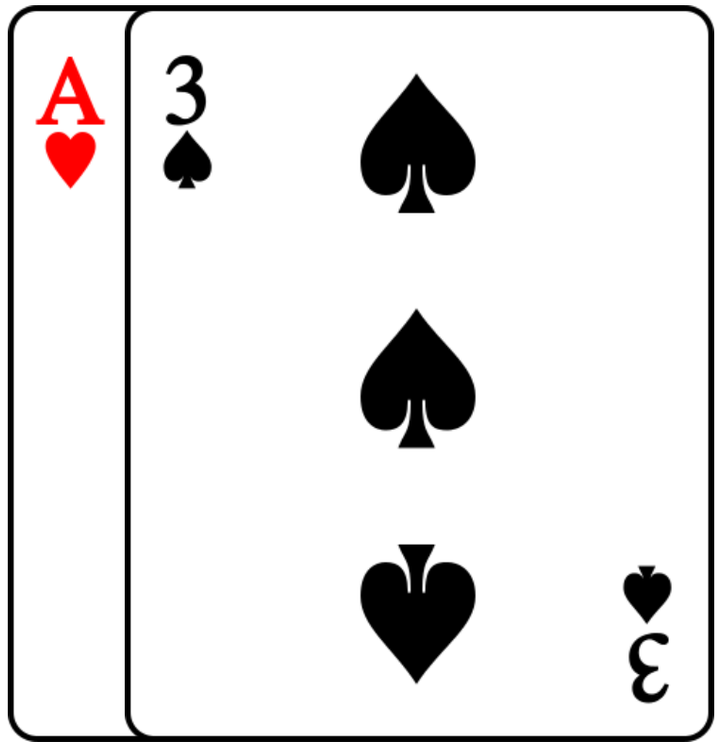


Le meilleur des cas



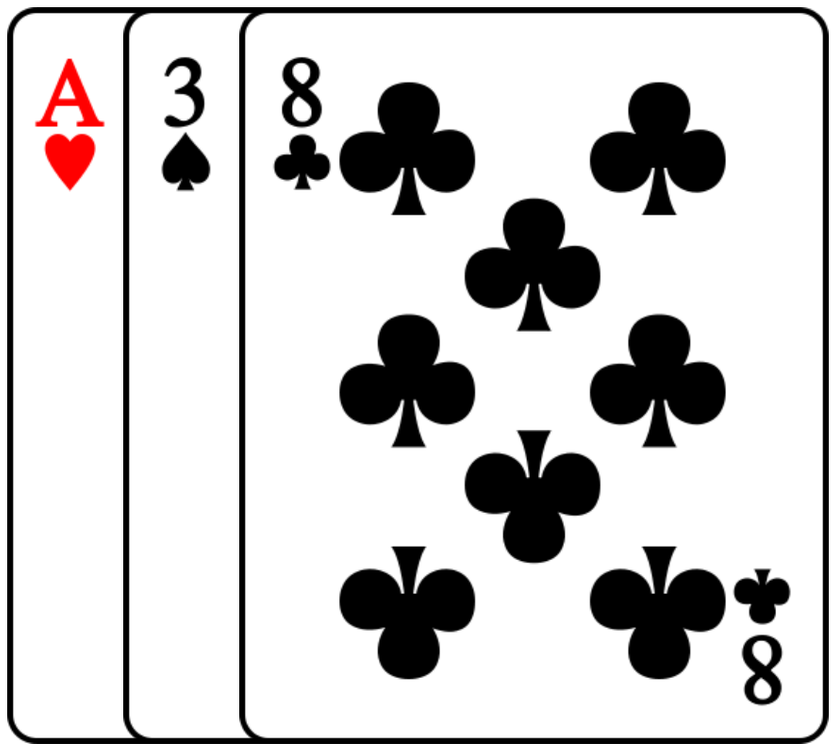
№ operations = 1

Le meilleur des cas



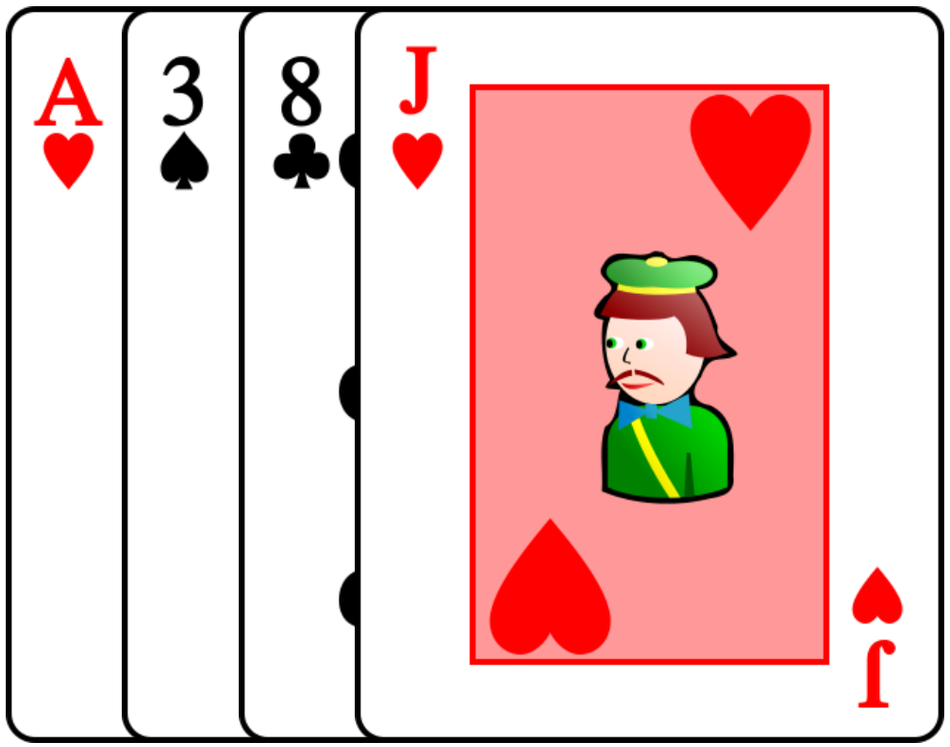
№ operations = 2

Le meilleur des cas



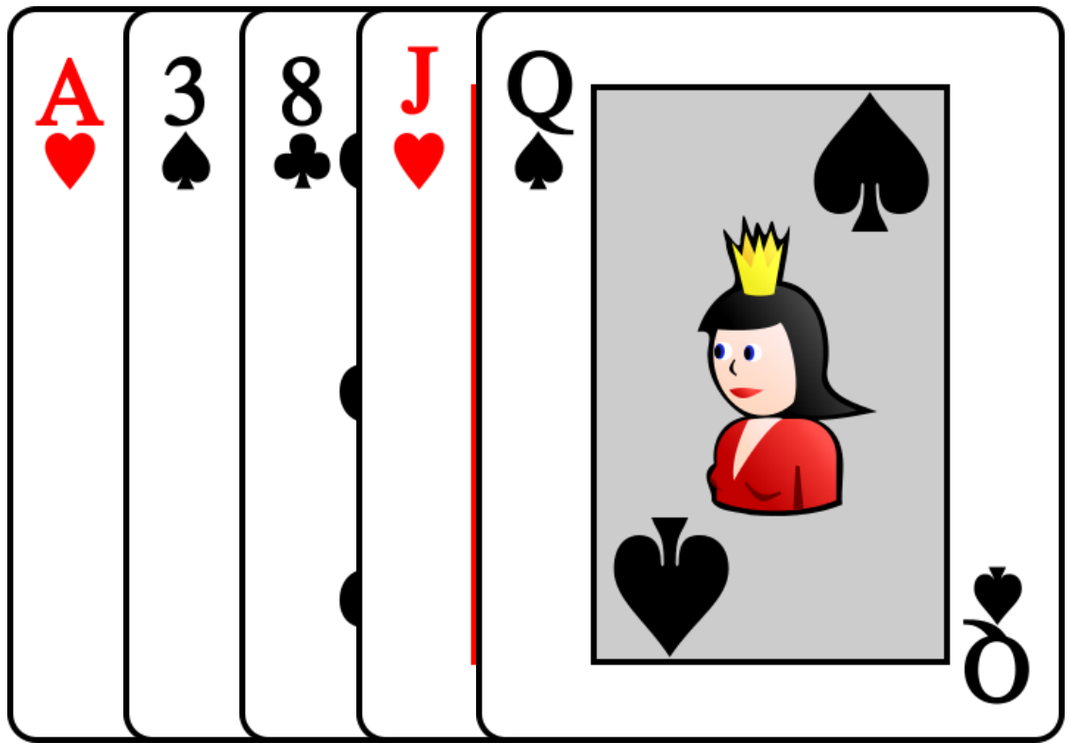
№ operations = 3

Le meilleur des cas



№ operations = 4

Le meilleur des cas



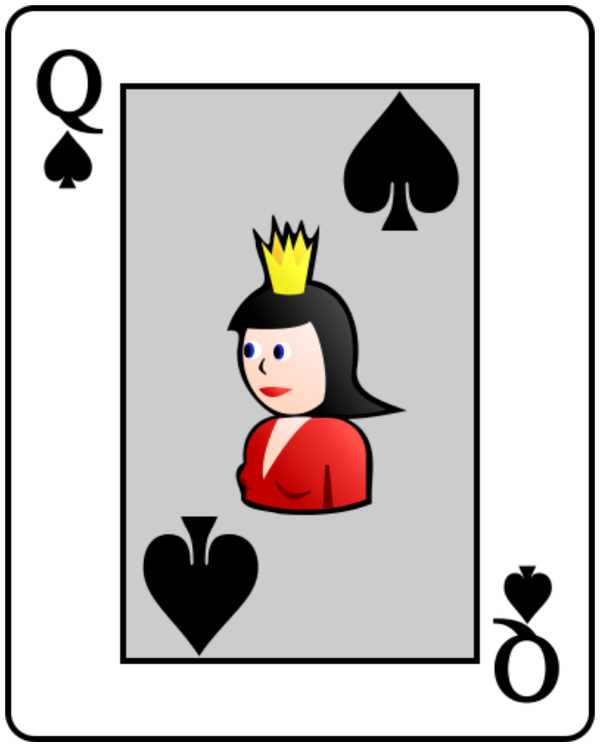
№ operations = 5

Le meilleur des cas

- Les cartes arrivent déjà triées

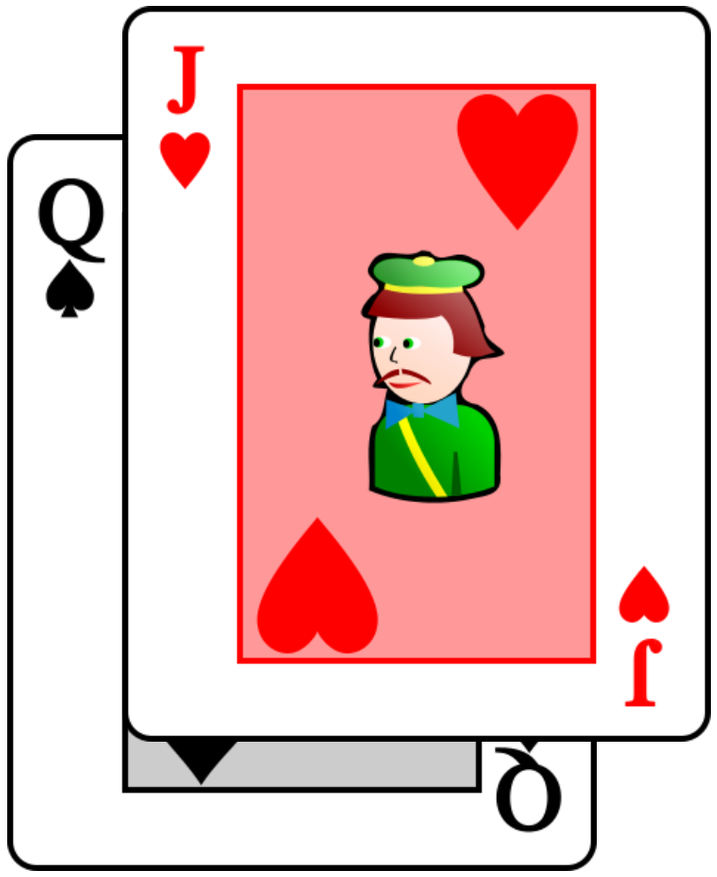
- On fait *n* opérations (déplacements de cartes)

Le pire des cas



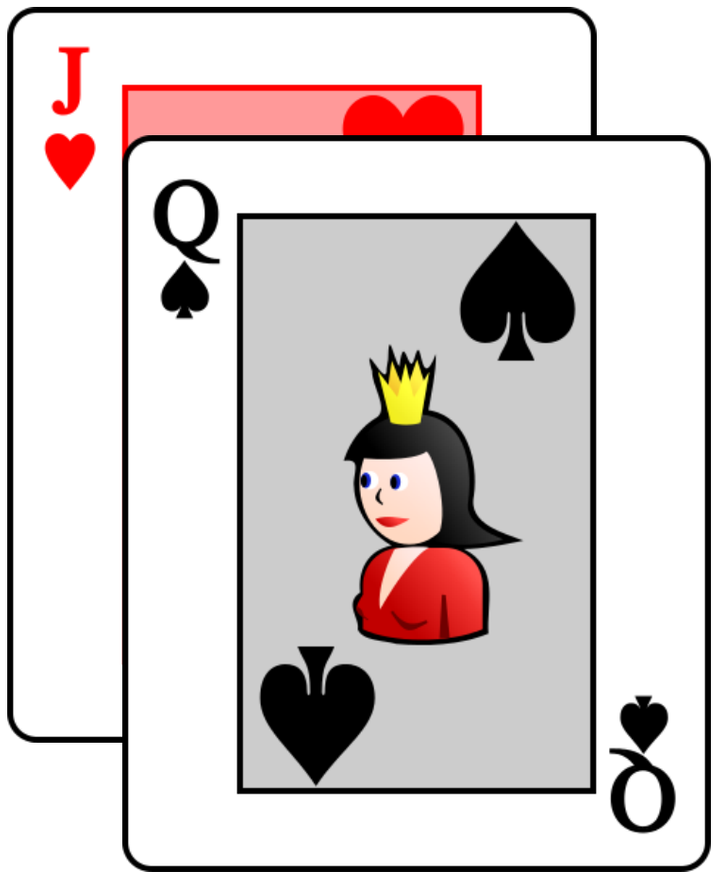
№ operations = 1

Le pire des cas



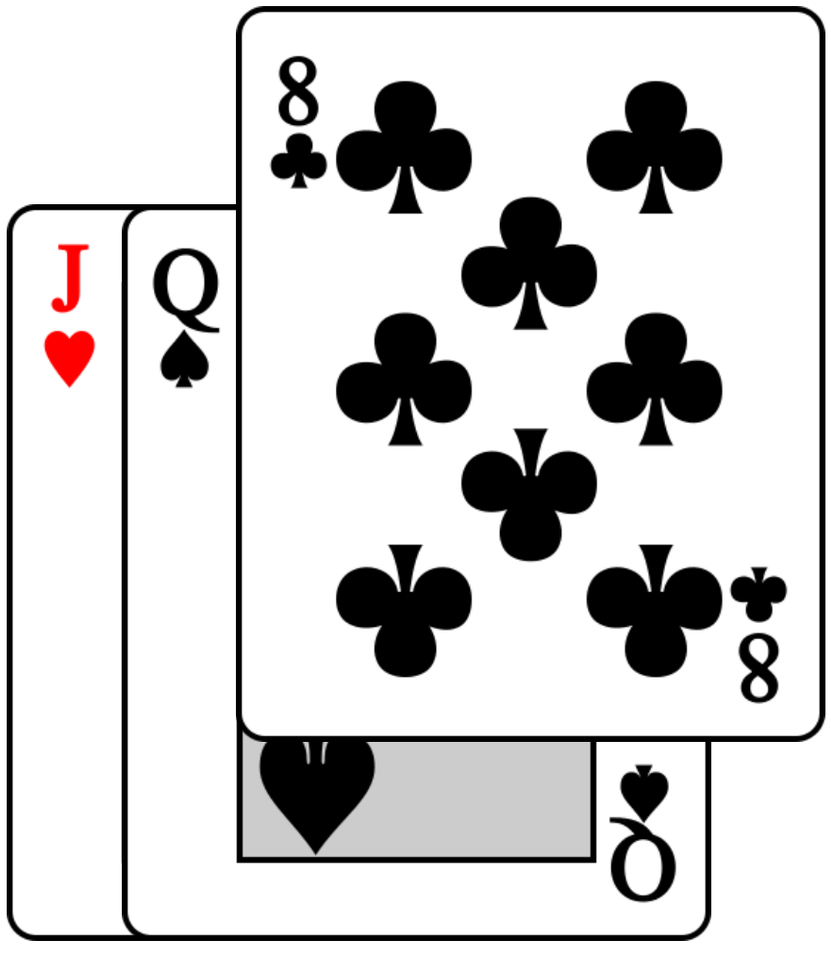
№ operations = 1 + 1

Le pire des cas



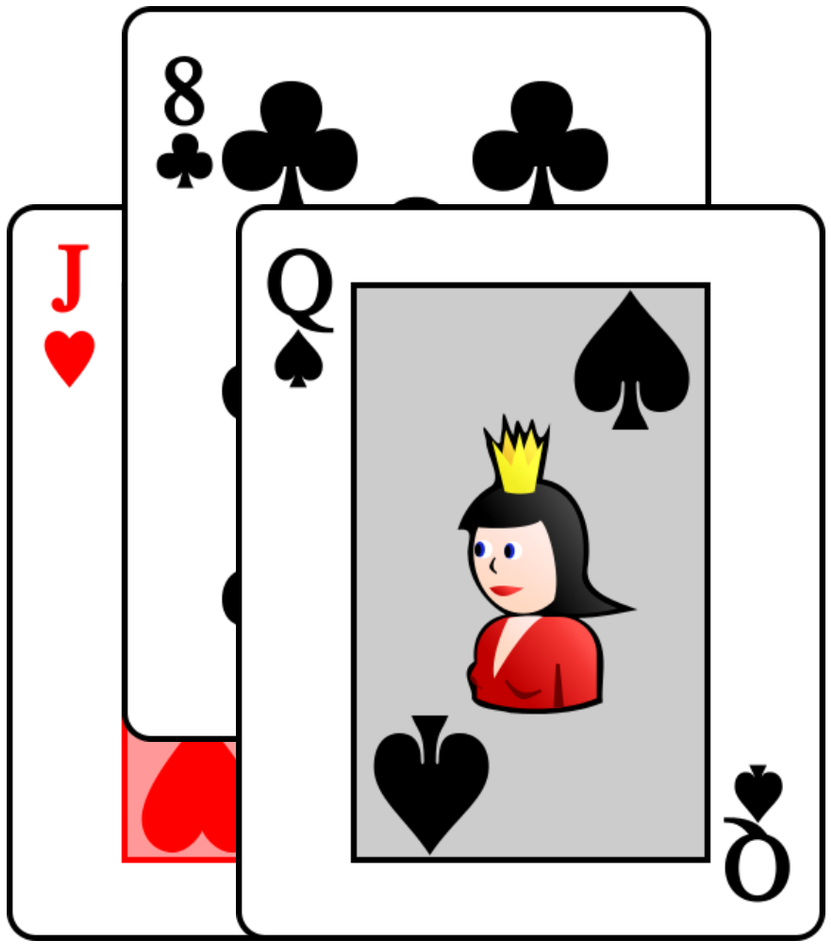
№ operations = 1 + 2

Le pire des cas



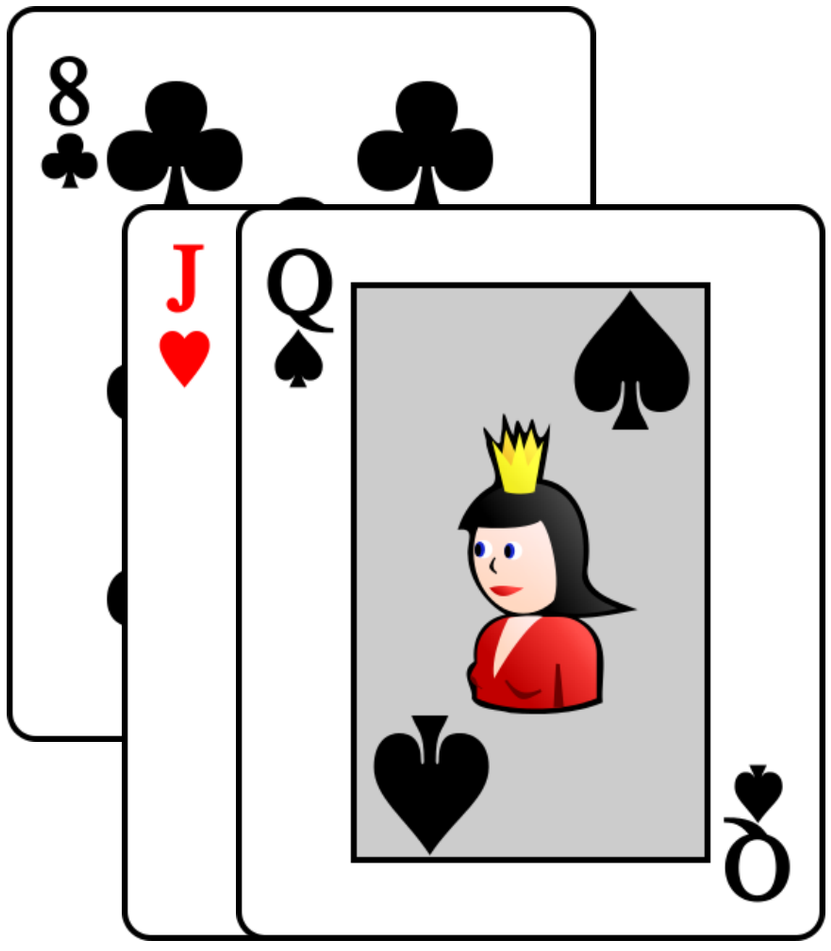
№ operations = 1 + 2 + 1

Le pire des cas



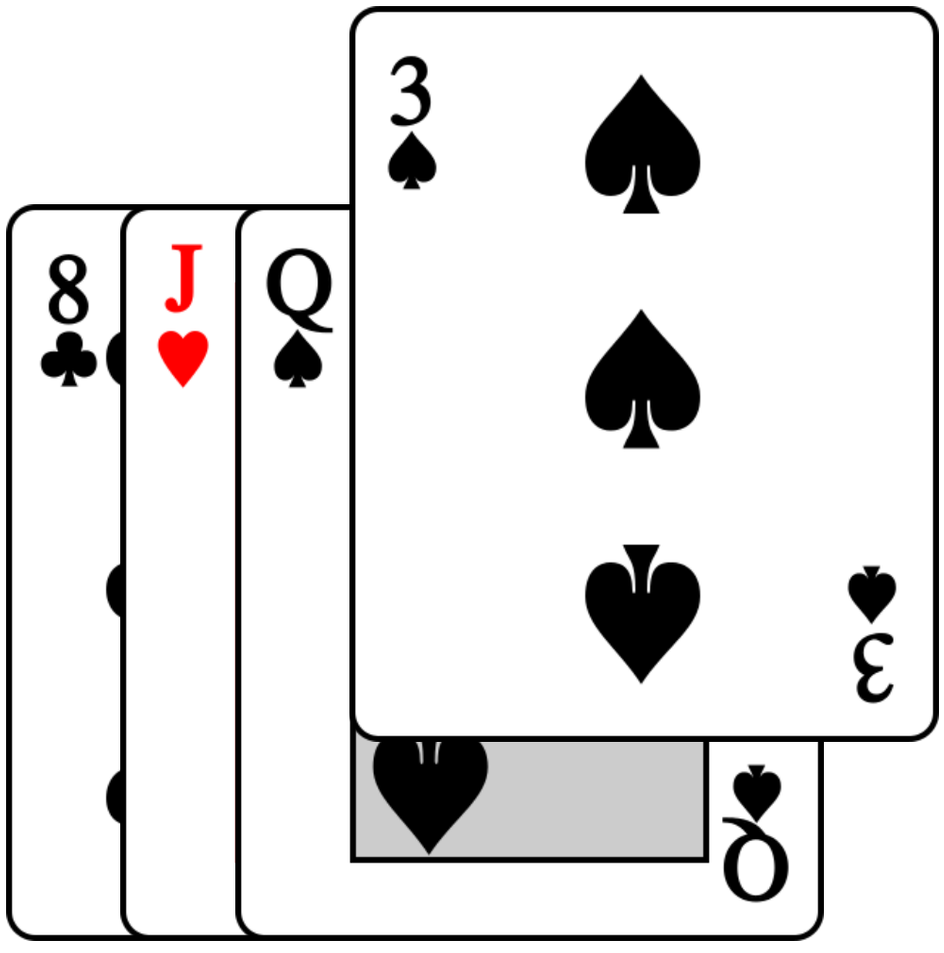
№ operations = 1 + 2 + 2

Le pire des cas



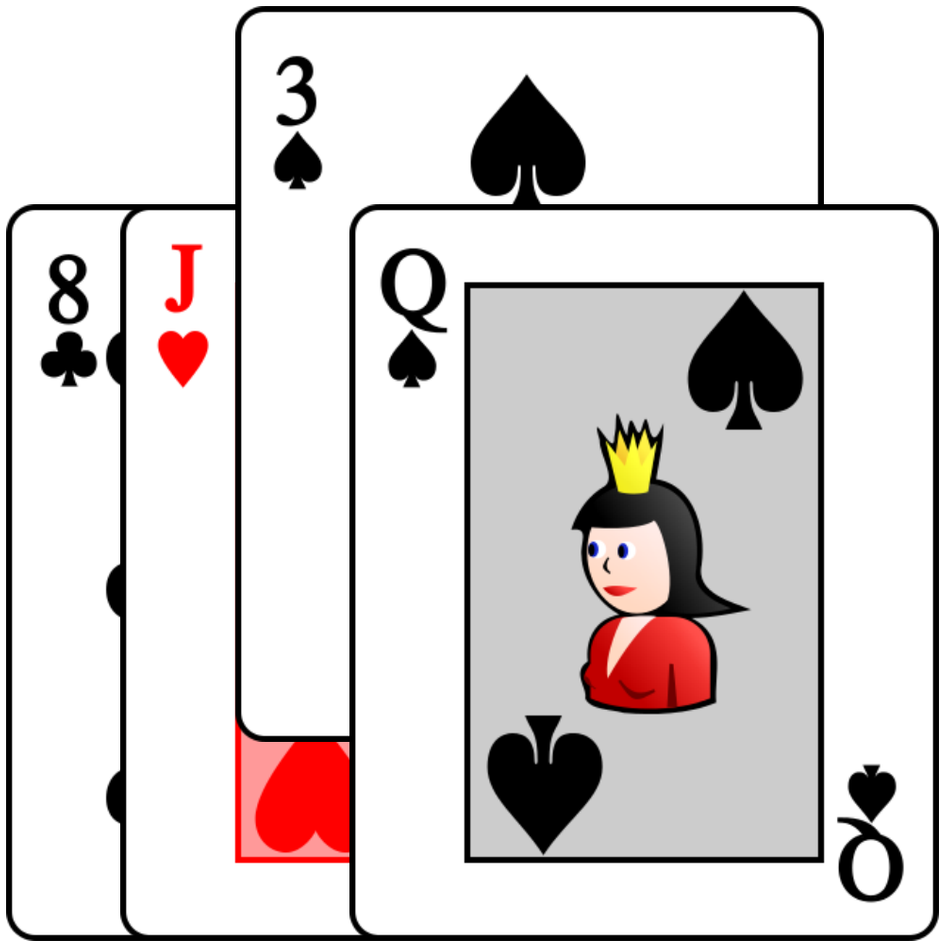
№ operations = 1 + 2 + 3

Le pire des cas



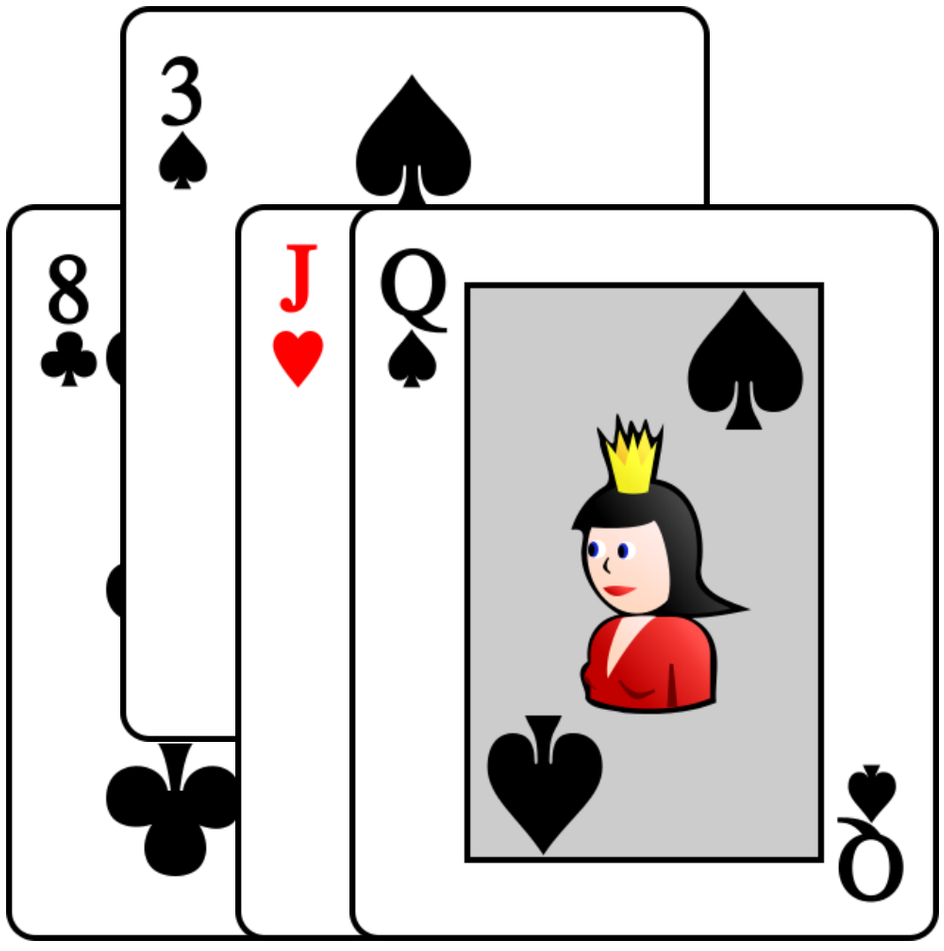
№ operations = 1 + 2 + 3 + 1

Le pire des cas



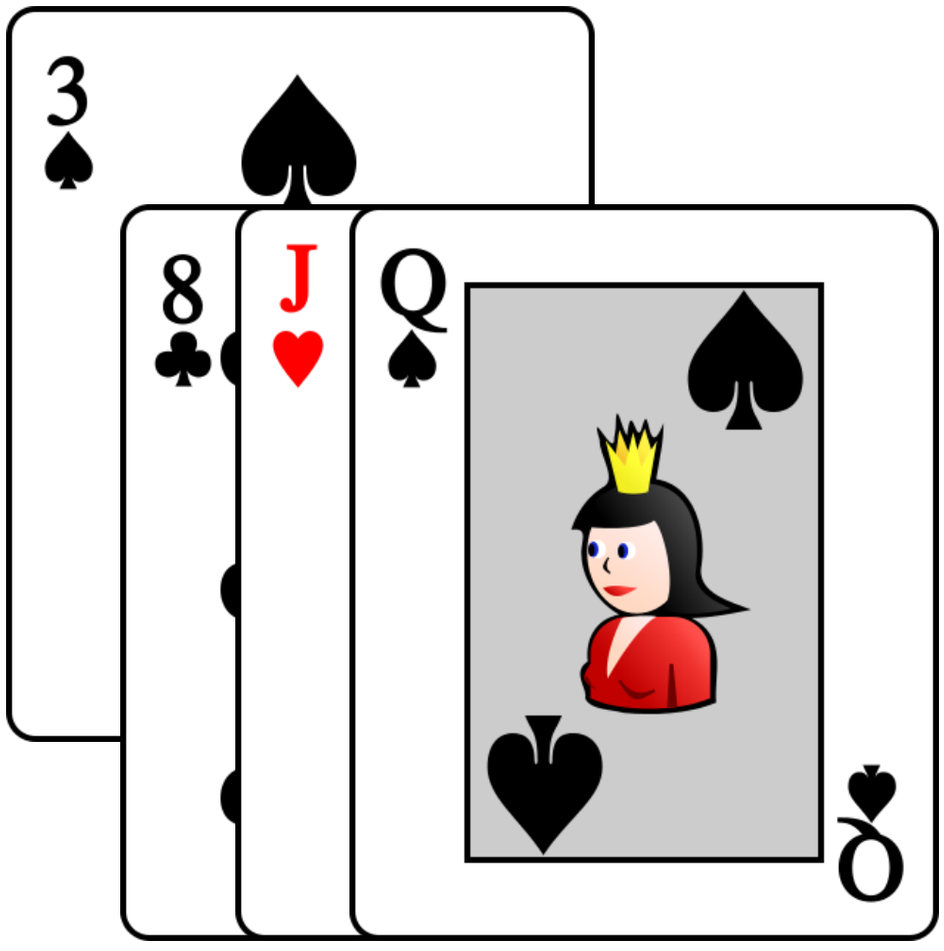
№ operations = 1 + 2 + 3 + 2

Le pire des cas



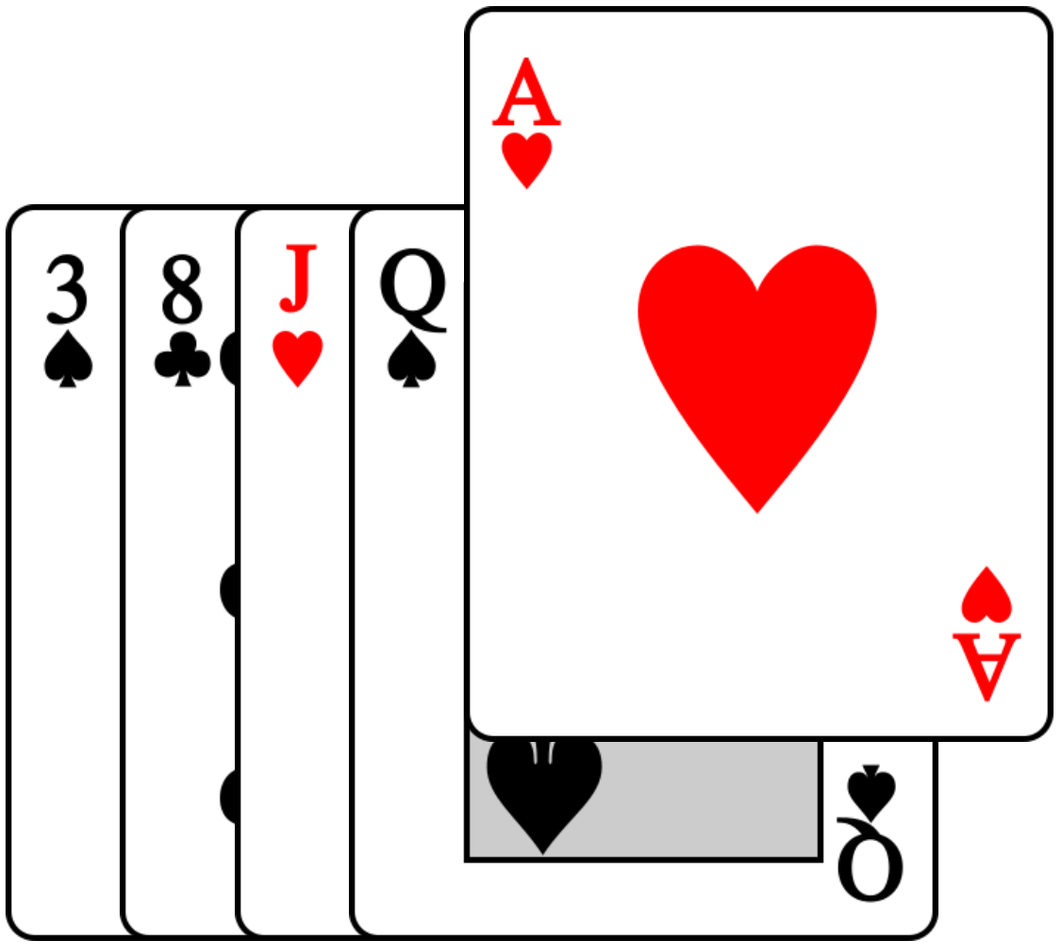
№ operations = 1 + 2 + 3 + 3

Le pire des cas



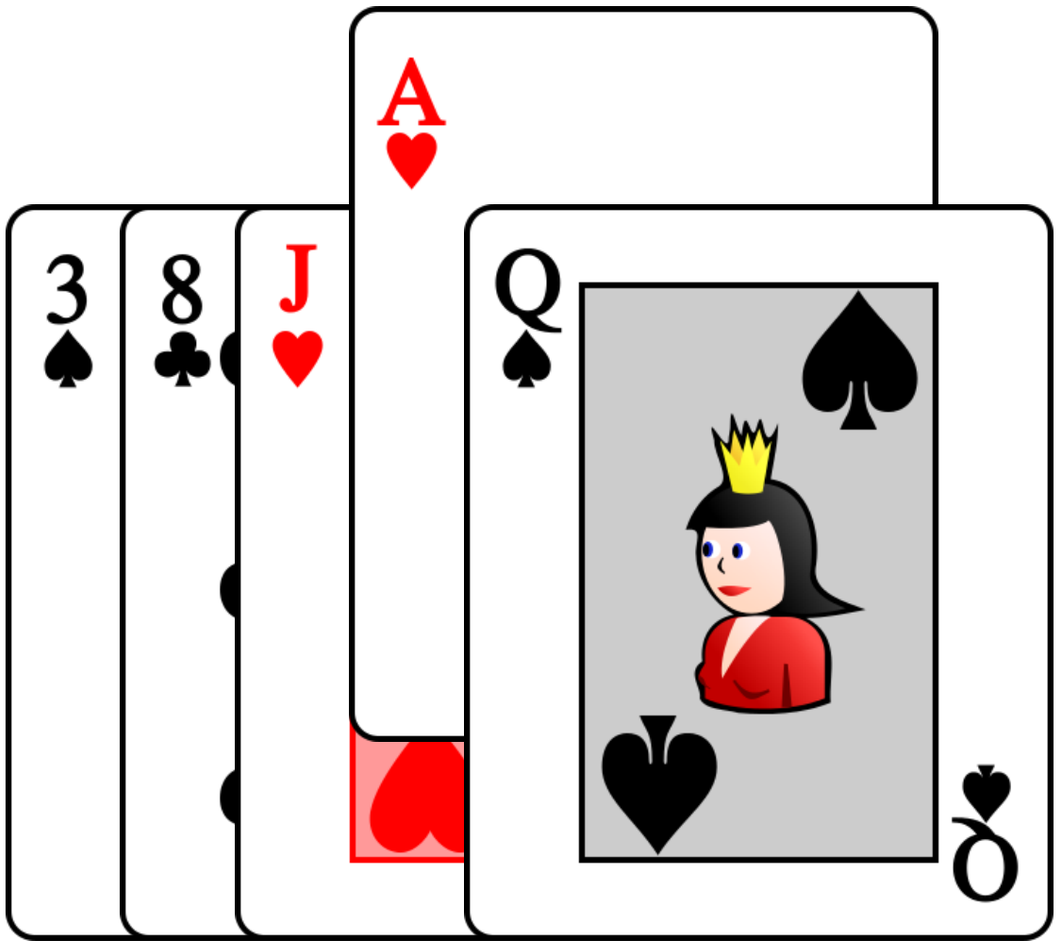
№ operations = 1 + 2 + 3 + 4

Le pire des cas



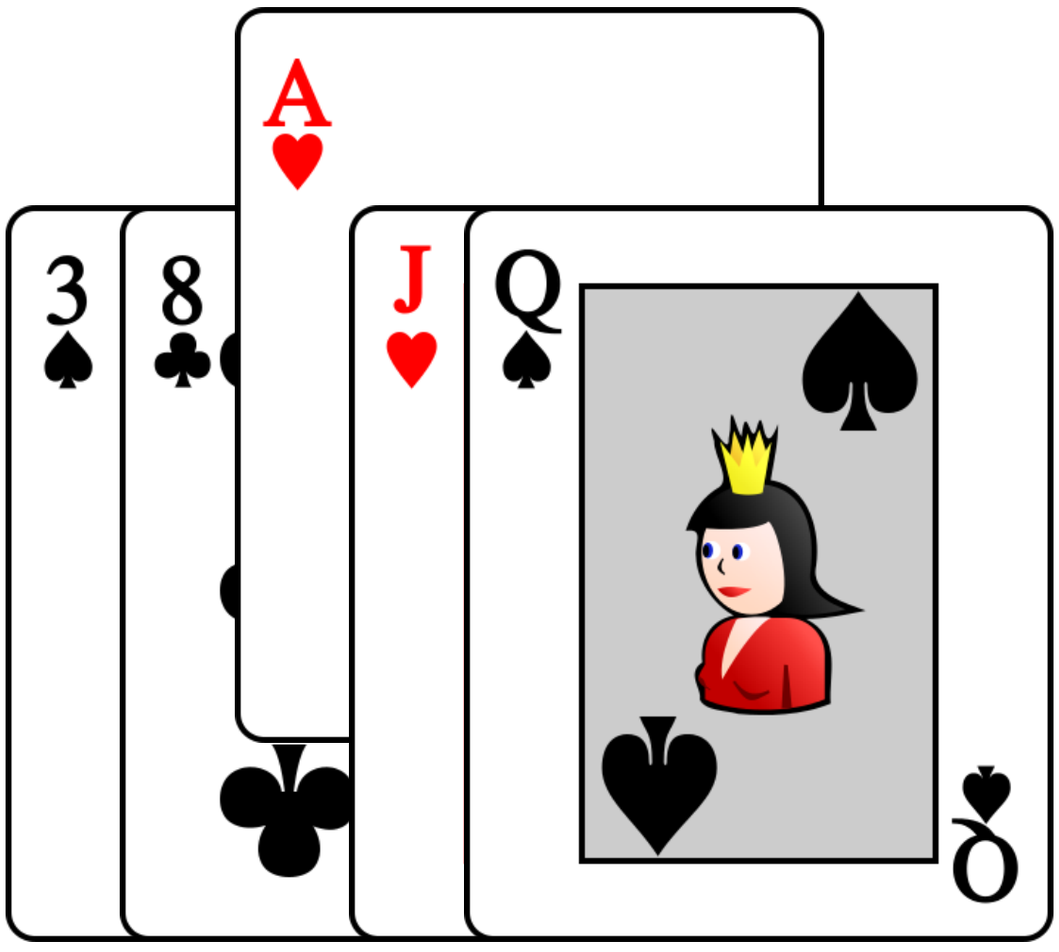
№ operations = 1 + 2 + 3 + 4 + 1

Le pire des cas



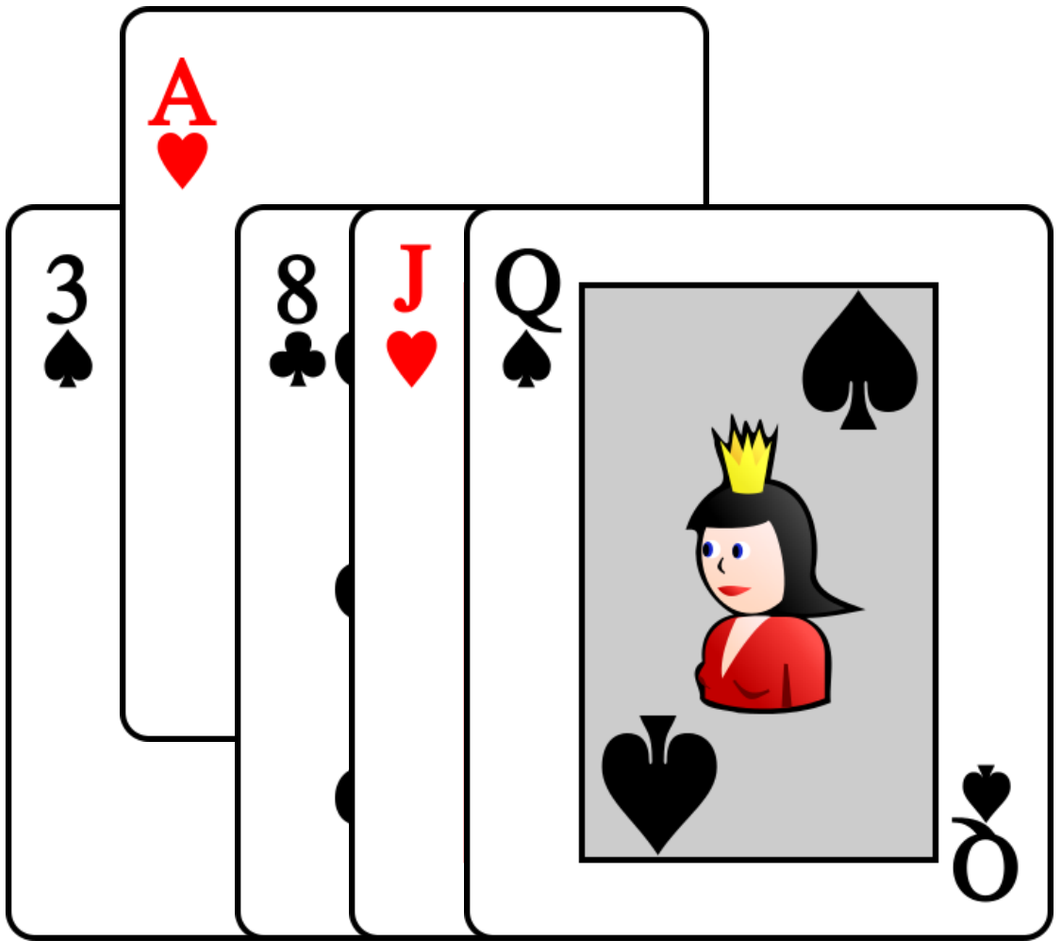
№ operations = 1 + 2 + 3 + 4 + 2

Le pire des cas



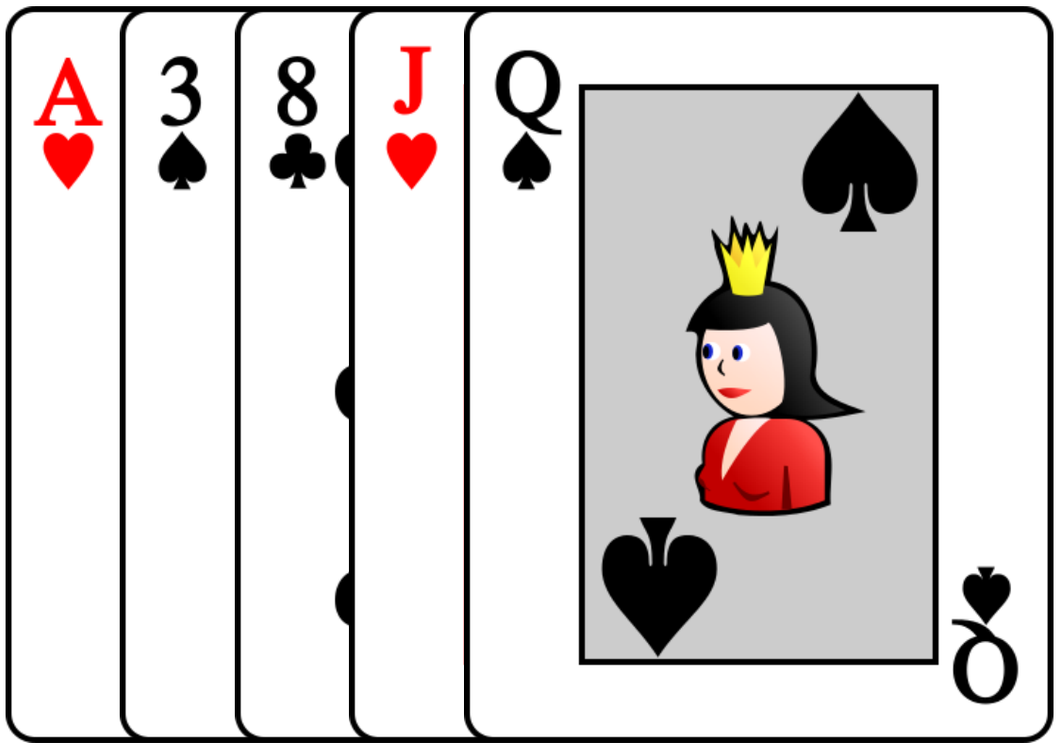
№ operations = 1 + 2 + 3 + 4 + 3

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*-ième carte
- Le nombre total est $1 + 2 + 3 + \cdots + n$

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

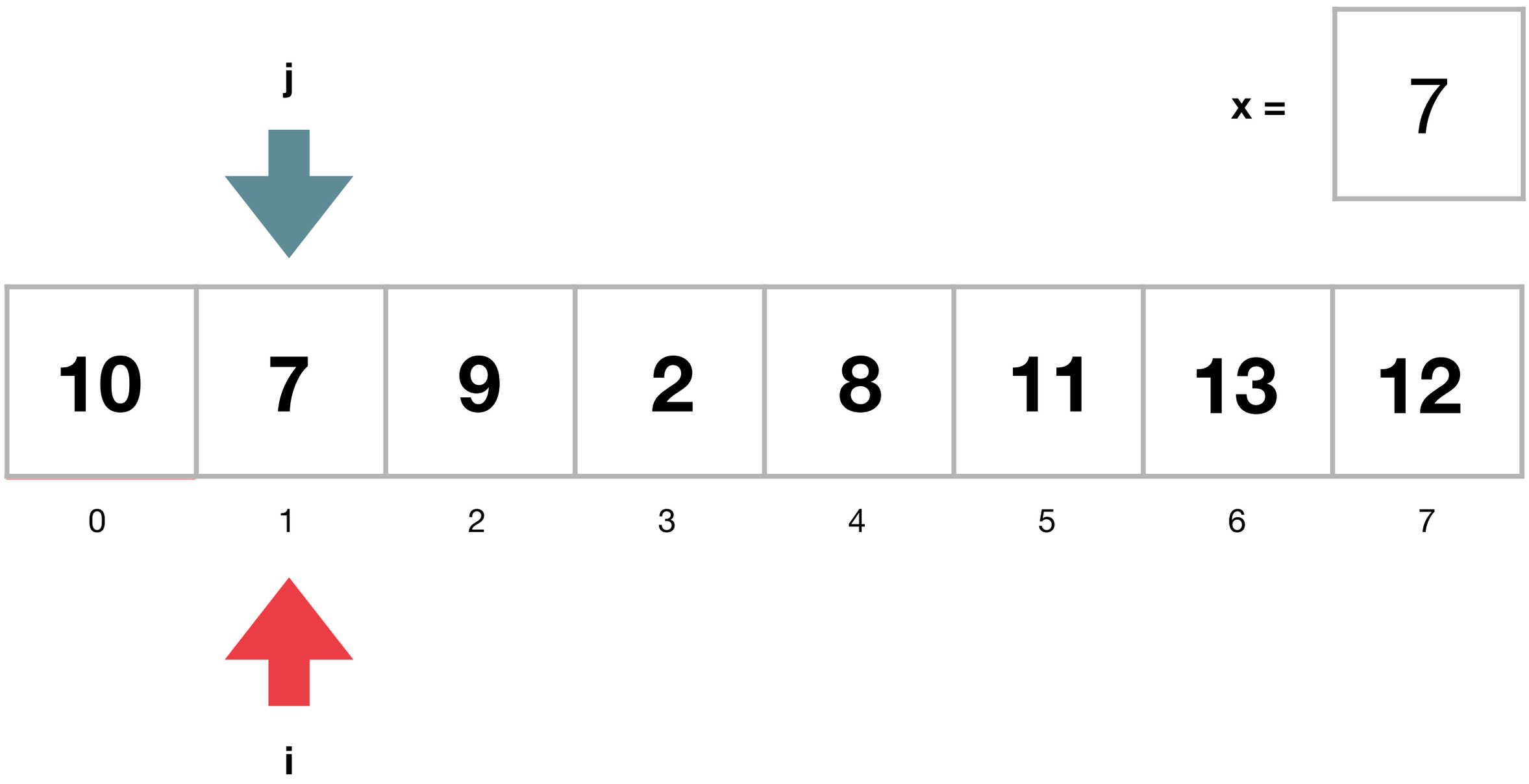
Tri par insertion

```
def insertionsort(t):
    n = len(t)
    for i in range(1, n):
        x = t[i]
        # insérer x parmi les i premiers éléments
        j = i
        while j > 0 and x < t[j - 1]:
            # décaler d'un élément
            t[j] = t[j - 1]
            j = j - 1
        # ici, x >= t[j - 1] ou bien j == 0
        t[j] = x
# le tableau est trié !
return t
```

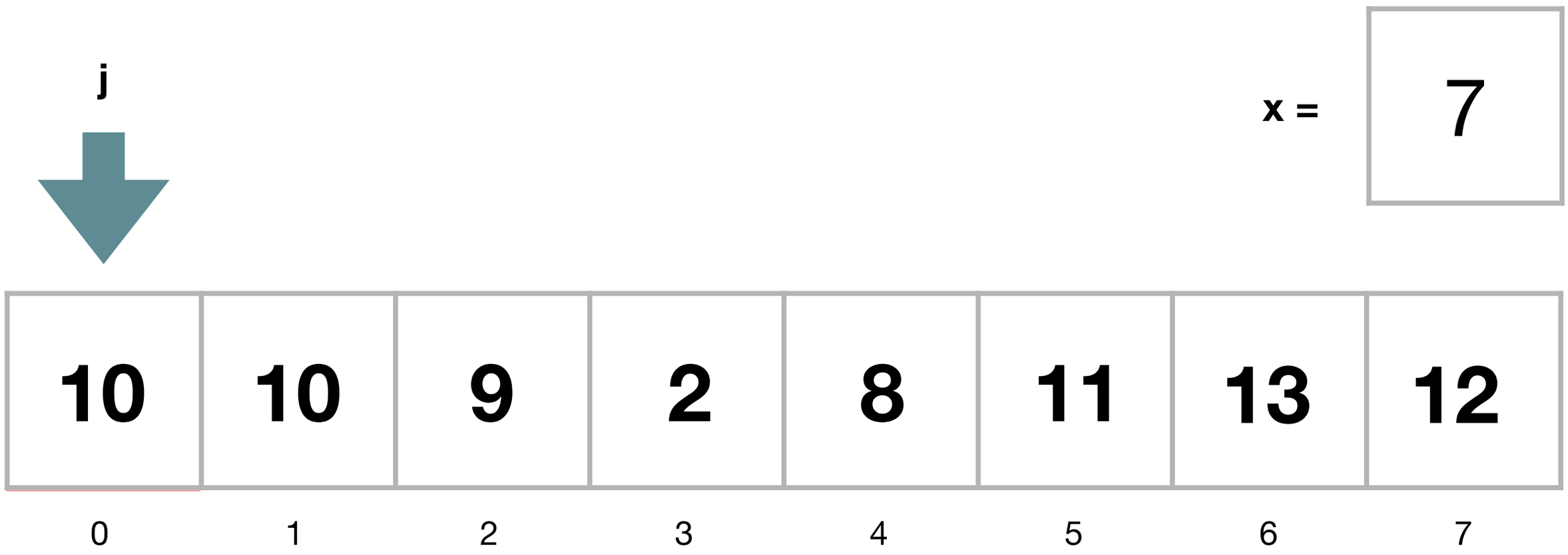
Tri par insertion

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

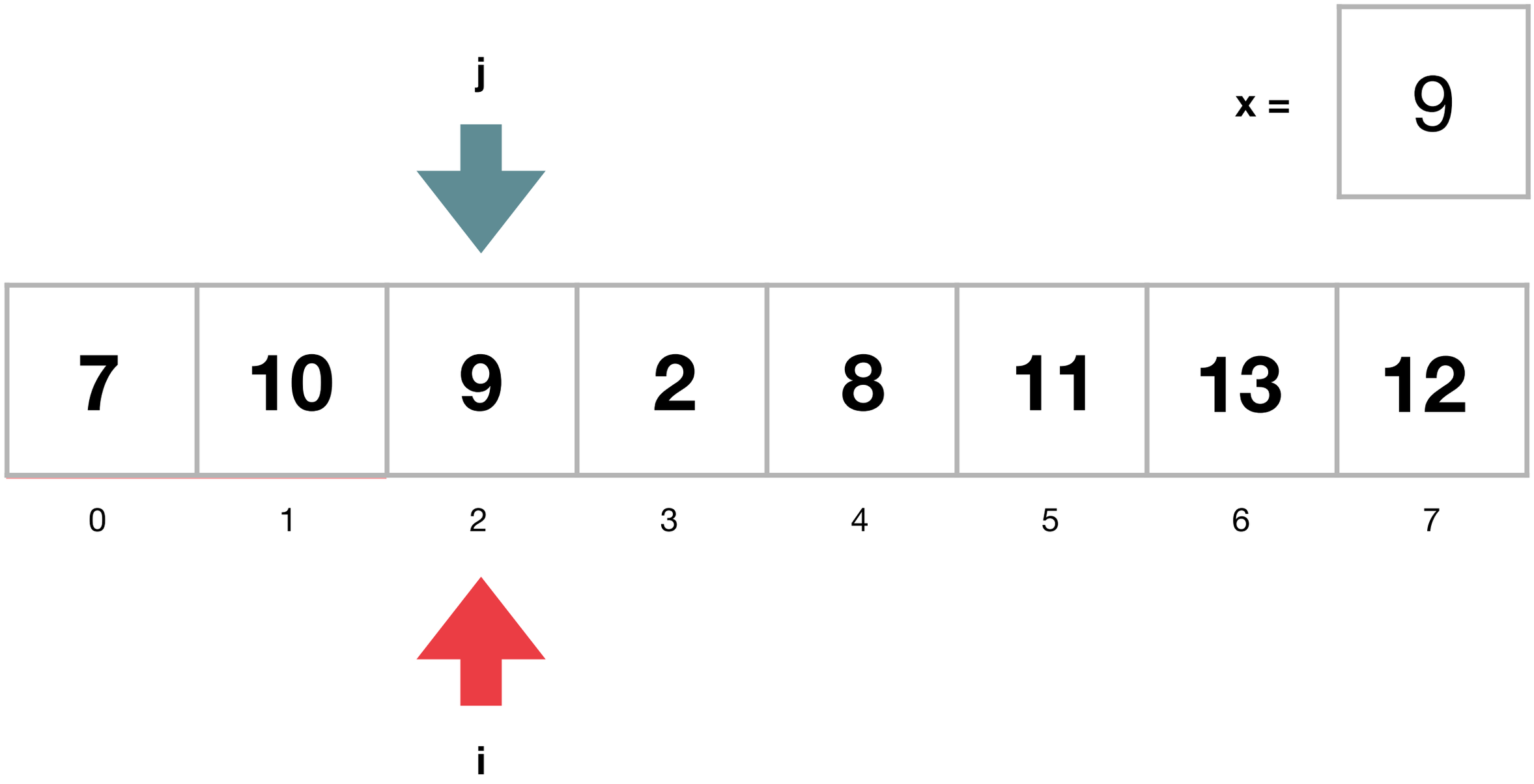
j

x = 7

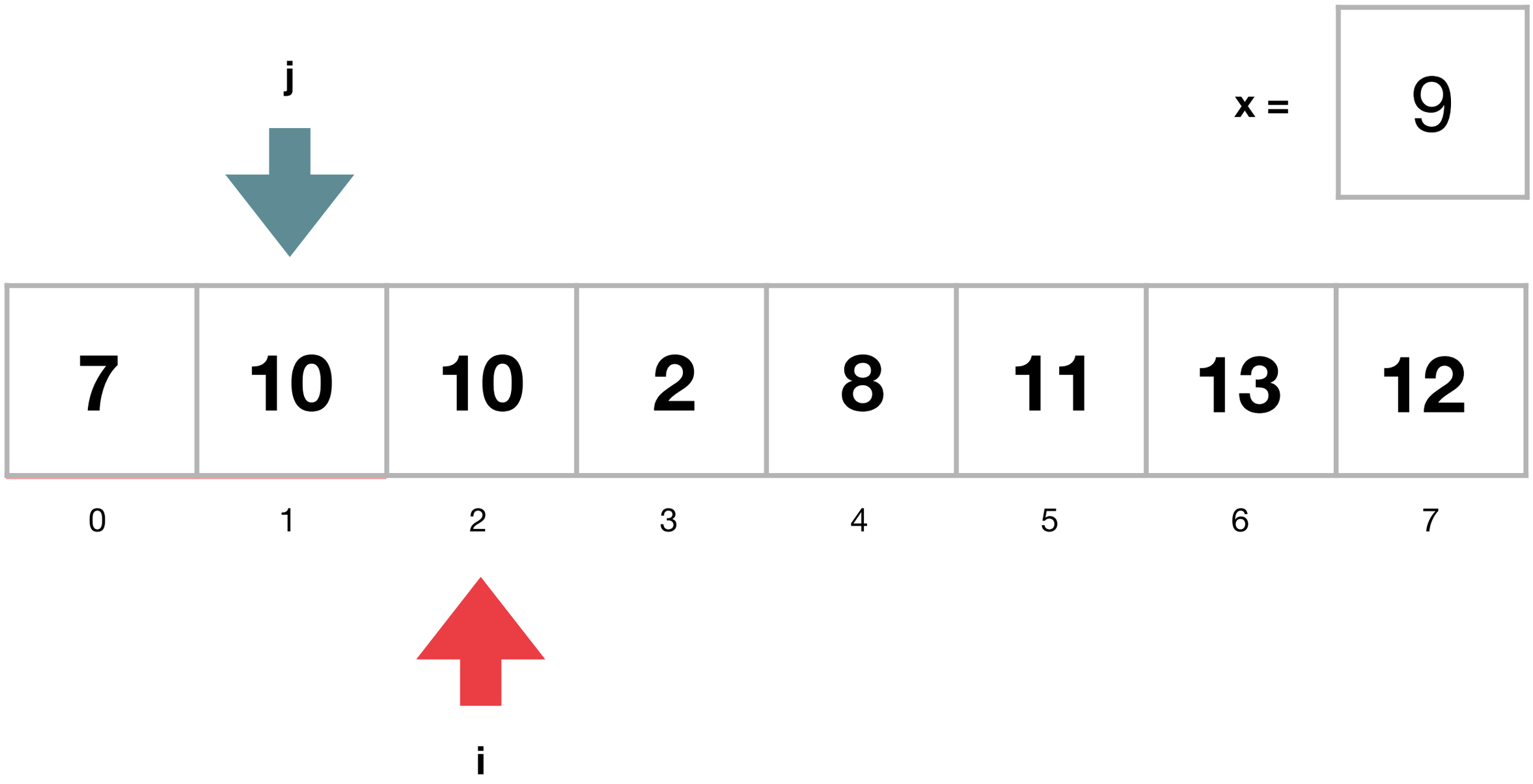
7	10	9	2	8	11	13	12
0	1	2	3	4	5	6	7

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

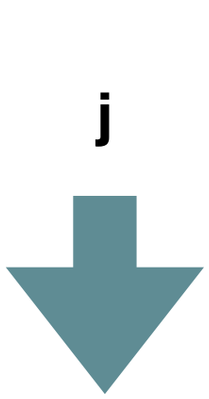
j

x = 2

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

i

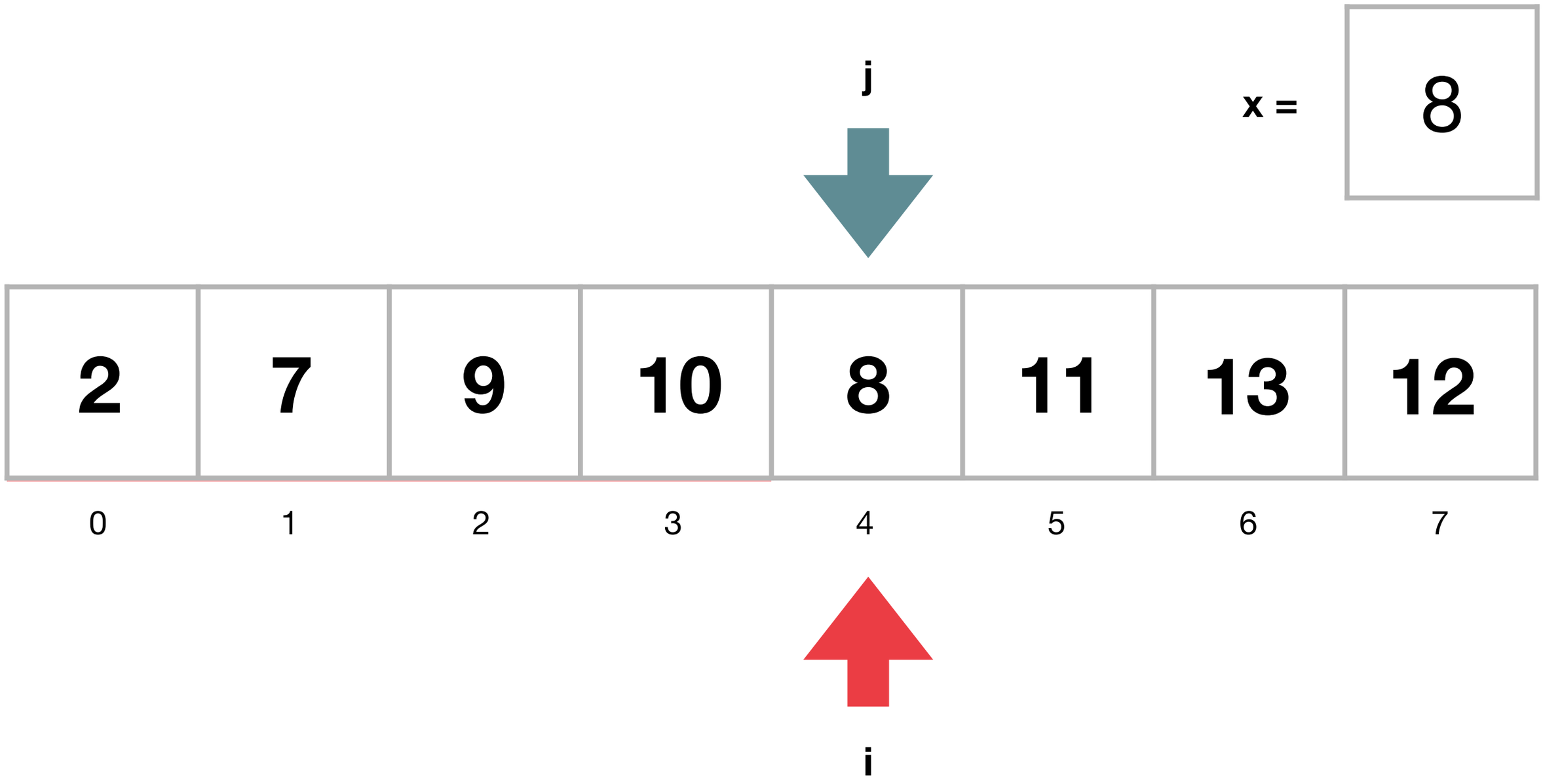
Tri par insertion



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

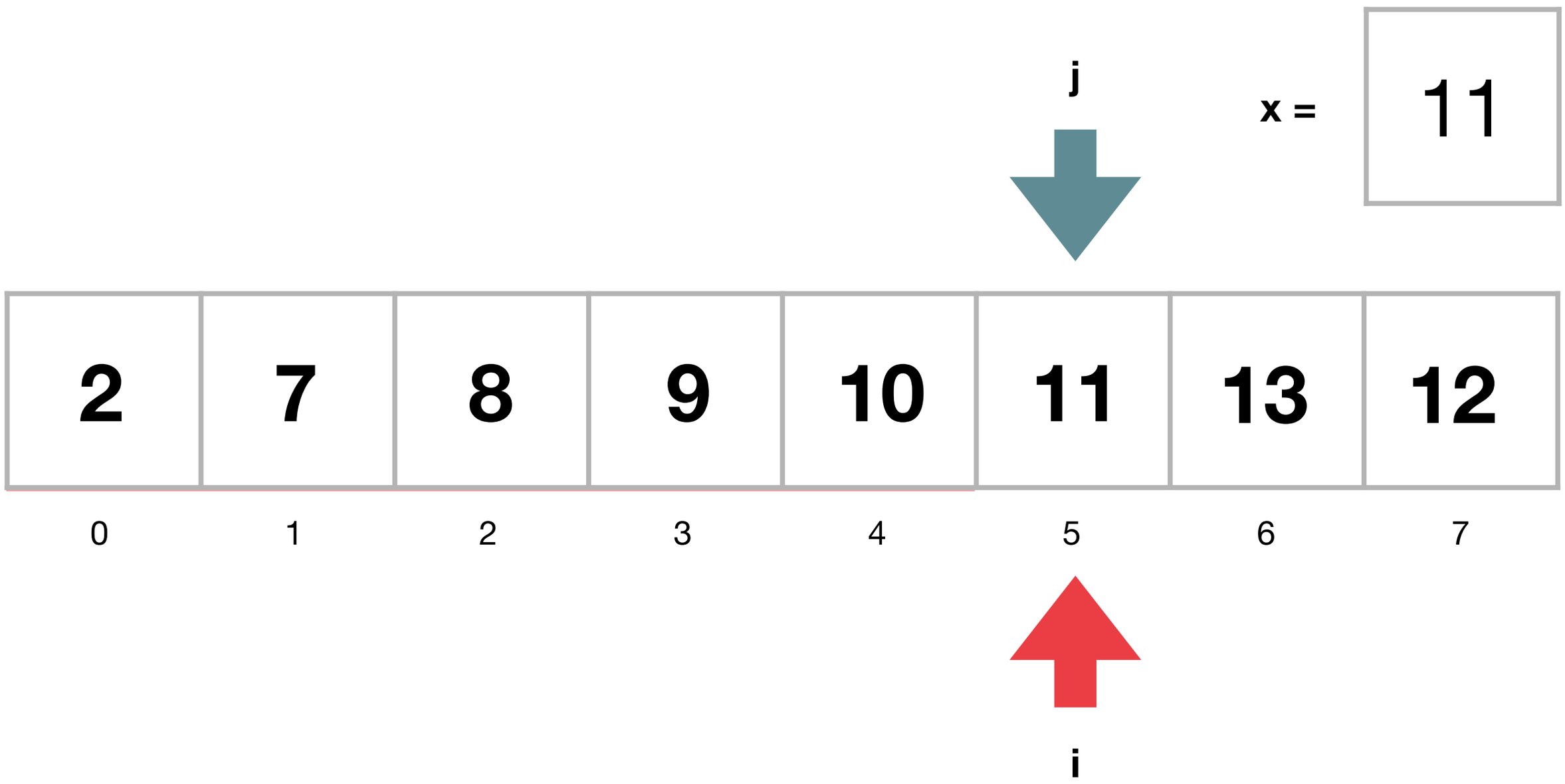
j

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

Exercice 5

Complexité

```
def insertionsort(t):
    n = len(t)
    for i in range(1, n):
        x = t[i]
        j = i
        while j > 0 and x < t[j - 1]:
            t[j] = t[j - 1]
            j = j - 1
        t[j] = x
return t
```

Complexité

```
def insertionsort(t):
    n = len(t)
    for i in range(1, n):
        x = t[i]
        j = i
        while j > 0 and x < t[j - 1]:
            t[j] = t[j - 1]
            j = j - 1
        t[j] = x
return t
```

Dans le pire des cas, la boucle while nécessite i itérations, et i prend les valeurs 1 à $n - 1$ donc $O(n^2)$ opérations

Complexité

```
def insertionsort(t):
    n = len(t)
    for i in range(1, n):
        x = t[i]
        j = i
        while j > 0 and x < t[j - 1]:
            t[j] = t[j - 1]
            j = j - 1
        t[j] = x
return t
```

Dans le pire des cas, la boucle while nécessite i itérations, et i prend les valeurs 1 à $n - 1$ donc $O(n^2)$ opérations

Dans le meilleur des cas (le tableau est trié), la boucle while ne nécessite aucune itération, donc $O(n)$ opérations

Exercice 6