

# ALGO 1/5

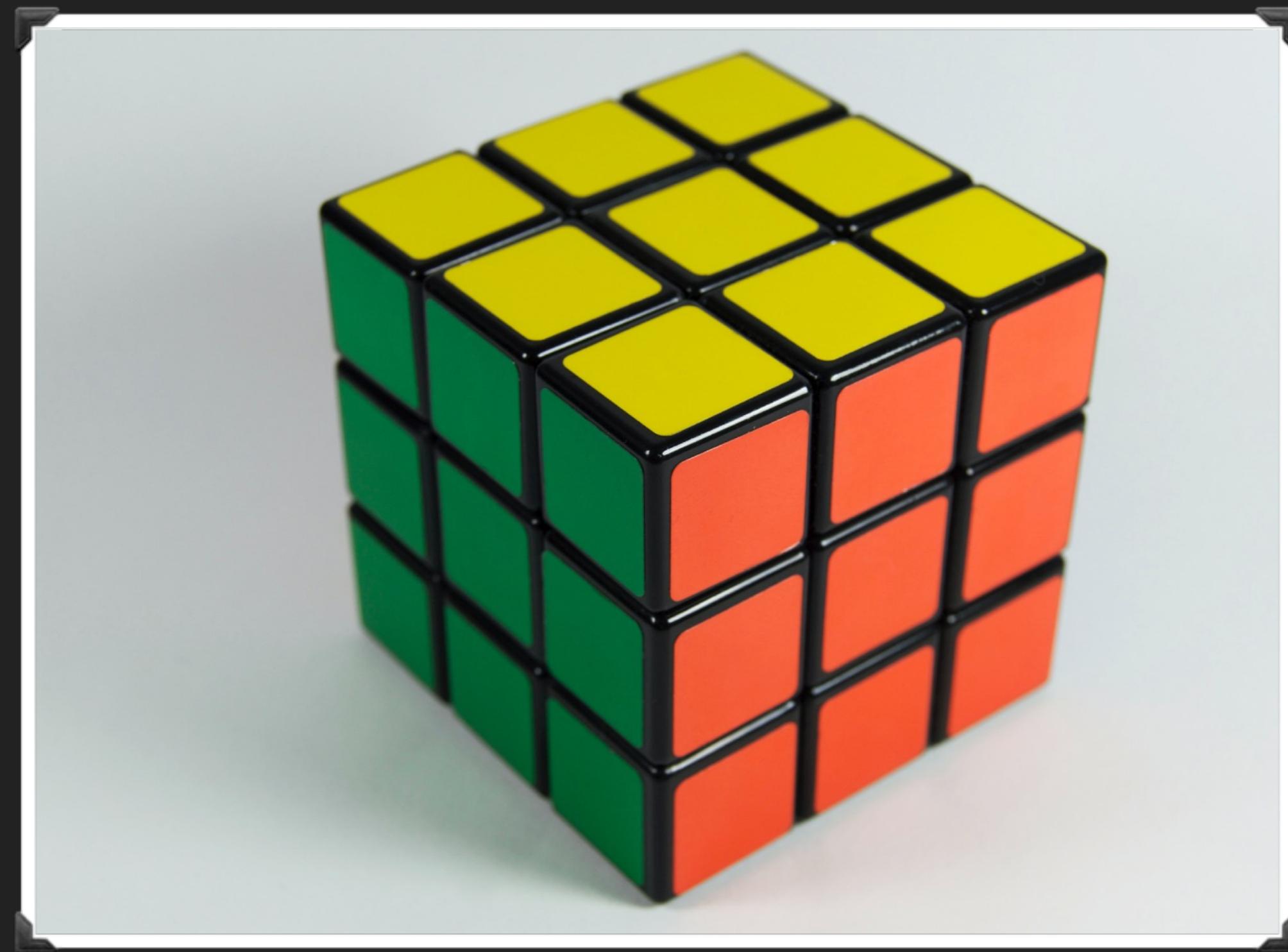
Auteur : Sébastien Inion

studi

<https://github.com/SebInfo/AlgoStudi01>



# RAPPELS DES CONCEPTS DE L'ALGO



# DEFINITION

- ▶ Un algorithme est une séquence d'instructions ou de règles qui permettent de résoudre un problème.
- ▶ Le terme algorithme fut inventé par le mathématicien **Mohammed Ibn Musa-Al Khwarizmi**, dans le courant du neuvième siècle avant Jésus Christ.
- ▶ C'est une procédure bien définie qui prend en entrée un ensemble de valeurs ou de données, les manipule selon des règles préétablies, et produit en sortie un résultat ou une solution.
- ▶ Les algorithmes existaient avant les ordinateurs. Ils sont encore plus présents avec l'informatique. Avec l'essor de l'intelligence artificielle, ce terme est de plus en plus utilisé et a rejoint la catégorie des **buzzwords**.

## DES EXEMPLES ANCIENS

$$\begin{array}{r} 812 \\ -6 \quad | \\ \hline 21 \\ -21 \quad | \\ \hline 02 \\ -0 \quad | \\ \hline 2 \end{array}$$

$$\begin{array}{r} 3 \\ \hline 270 \end{array}$$

dividende = diviseur × quotient + reste

$$812 = 3 \times 270 + 2$$

## CRIBLE D'ÉRATOSTHÈNE

	2	3	X	5	X	7	X	9	X	Prime numbers	
11	X	X	13	X	15	X	17	X	19	X	2
21	X	X	23	X	25	X	27	X	29	X	3
31	X	X	33	X	35	X	37	X	39	X	
41	X	X	43	X	45	X	47	X	49	X	
51	X	X	53	X	55	X	57	X	59	X	
61	X	X	63	X	65	X	67	X	69	X	
71	X	X	73	X	75	X	77	X	79	X	
81	X	X	83	X	85	X	87	X	89	X	
91	X	X	93	X	95	X	97	X	99	X	
101	X	X	103	X	105	X	107	X	109	X	
111	X	X	113	X	115	X	117	X	119	X	

Qui permet de trouver les nombres premiers.

## L'UN DES PREMIER ALGORITHME

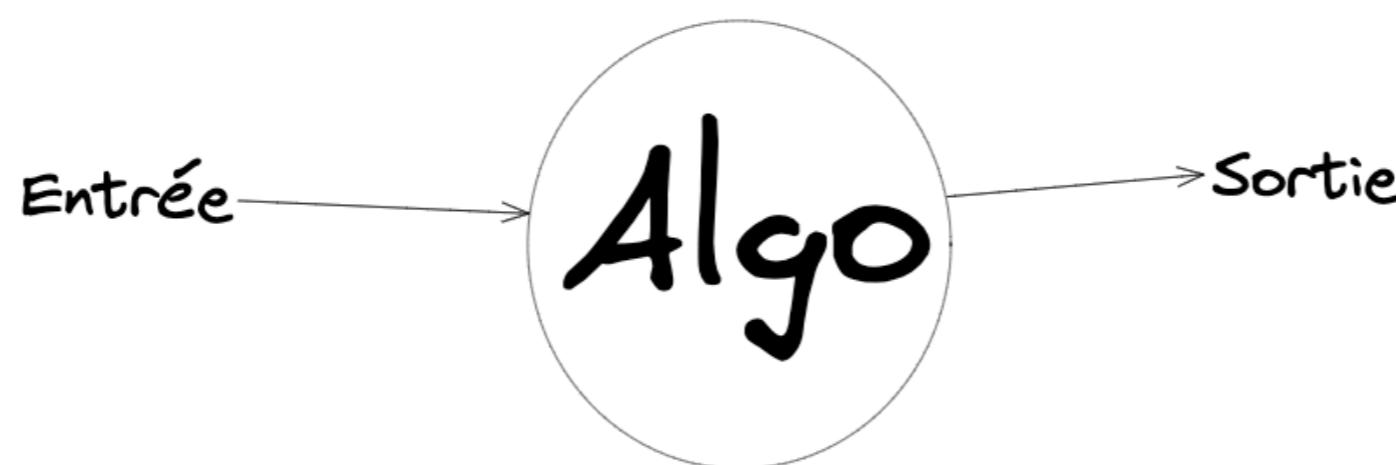
- ▶ L'un des premier connu est celui d'Euclide et date de **300 av J.C.** il permet de calculer le plus grand commun diviseur commun (PGCD) de deux nombres entiers.
- ▶ Voici la description de l'algorithme d'Euclide :
  1. Prendre deux nombres entiers naturels non nuls,  $a$  et  $b$ . On suppose  $a$  plus grand ou égal à  $b$ .
  2. Effectuer la division euclidienne de  $a$  par  $b$  et noter le reste.
  3. Si le reste est égal à zéro, alors  $b$  est le PGCD des deux nombres et l'algorithme s'arrête.
  4. Si le reste est différent de zéro, alors remplacer  $a$  par  $b$  et  $b$  par le reste, et retourner à l'étape 2.

## LE PREMIER ALGORITHME

```
1  # Calcul du PGCD algo Euclide 300 av J.C.
2  a=eval(input("Valeur de a ?"))
3  b=eval(input("Valeur de b ?"))
4  # On change les valeurs si besoin pour que a >= b
5  if ( b > a ):
6      a, b = b, a
7
8  reste = a%b
9  while(reste != 0):
10     a = b
11     b = reste
12     reste = a%b
13
14 print(b) # Le PGCD est trouvé
```

## RAPPELS

- ▶ Les algorithmes sont une technologie, au même titre que les matériels, les réseaux, etc.
- ▶ Si les ordinateurs étaient infiniment rapide et avec des ressources infinies. On n'aurait pas besoin d'optimiser les algos. Juste vérifier qu'ils s'arrêtent et donne la bonne réponse.
- ▶ Un algorithme transforme l'entrée en sortie.



## LE PREMIER ALGORITHME

- ▶ Pour calculer le PGCD Euclide il y a d'autres algorithmes comme celui par différences successives.
- ▶ On part à nouveau de deux entiers naturels non nuls tel que  $a > b$  :
  1. On calcul la différence  $a - b$  qu'on affecte à  $d$
  2. a prend le plus grand nombre entre  $d$  et  $b$
  3. b le nombre le plus petite entre  $d$  et  $b$
  4. Si  $a$  est différent de  $b$  on repart à l'étape 1 sinon on arrête l'algo.

# LE PREMIER ALGORITHME

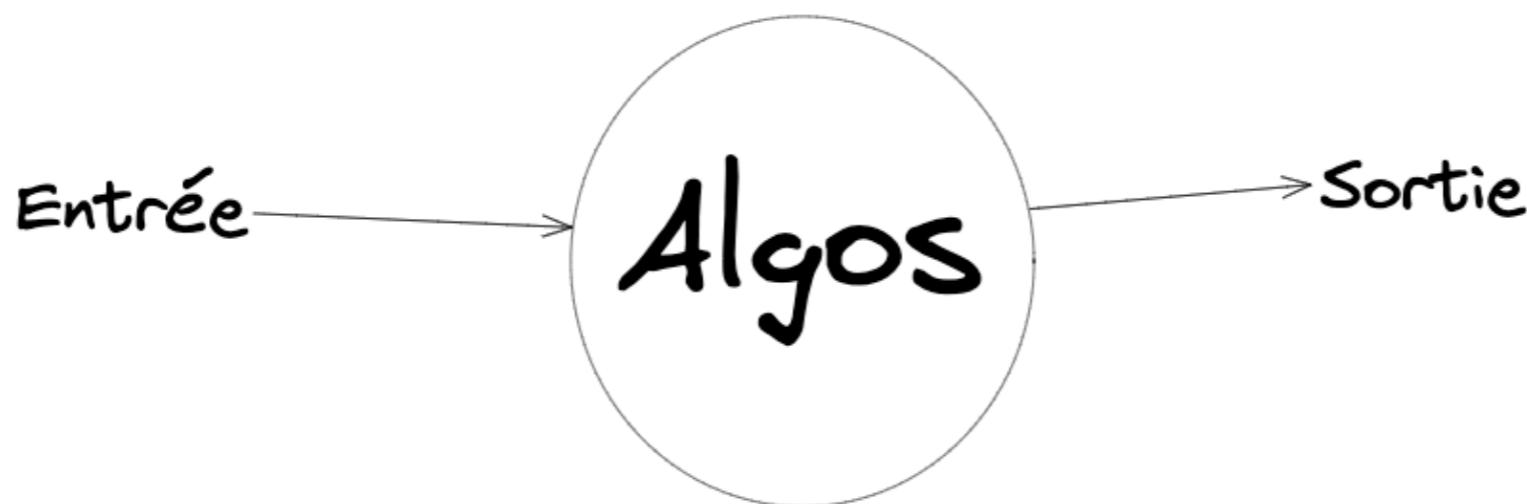
```
1 # Calcul d'un PGCD par différences successives
2 a=eval(input("Valeur de a ?"))
3 b=eval(input("Valeur de b ?"))
4 while (a!=b):
5     d=(a-b)
6     a=max(d,b)
7     b=min(d,b)
8     print("pgcd=",d)
9     if d==1:
10        print("Les deux entiers sont premiers entre eux.")
```

Exemple avec 360 et 252

$$\begin{aligned} 360 - 252 &= 108 \\ 252 - 108 &= 144 \\ 144 - 108 &= 36 \\ 108 - 36 &= 72 \\ 72 - 36 &= 36 \\ 36 - 36 &= 0 \quad \text{STOP} \end{aligned}$$

## RAPPELS

- ▶ On peut donc modifier notre schéma en ajoutant un 's'
- ▶ En effet on se rend compte que plusieurs algorithmes peuvent donner la même sortie
- ▶ On pourra se poser par la suite la question de leurs différences en ressources...



## RAPPELS

- ▶ Il est important de spécifier l'entrée et la sortie.
- ▶ Par exemple si l'on veut trier une suite de nombres dans un ordre croissant :
- ▶ **Entrée** : Une suite de n nombres  $\langle a_1, a_2, \dots, a_n \rangle$
- ▶ **Sortie** : Une permutation (réorganisation)  $\langle a'_1, a'_2, \dots, a'_n \rangle$  de la suite donnée en entrée, de façon que  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

## RAPPELS

- ▶ Exemple  $\langle 11, 21, 23, 18, 87, 2 \rangle$  est **une instance** du problème de tri.
- ▶ Une instance est une entrée qui respecte les contraintes imposées.
- ▶ Un algorithme est correct si **pour chaque instance** en entrée il **se termine** en donnant **la bonne sortie**.
- ▶ Ici  $\langle 2, 11, 18, 21, 23, 87 \rangle$  est la bonne sortie.

# RAPPELS

- ▶ A contrario un mauvais algorithme peut ne pas se terminer et/ou donner une mauvaise réponse.
- ▶ Contrairement à ce qu'on peut penser on utilise parfois des algorithmes non corrects si leur taux d'erreur peut être contrôlé.
- ▶ On devra donc vérifier la terminaison d'un algorithme pour cela l'algorithme doit converger vers la valeur d'arrêt et non diverger.
- ▶ Pour prouver la validité d'un algorithme et donc aussi de son arrêt on utilisera un **Invariant**
- ▶ Pour mesurer la performance d'un algo on aura recours au calcul de sa **complexité**.
- ▶ Ces notions seront abordées dans les séances suivantes...

# QUELQUES EXEMPLES DE PROBLÈMES QUE PEUT RÉSOUDRE UN ALGORITHME

- ▶ Génome humain avec l'analyse des 3 milliards de paire de bases chimiques qui constituent l'ADN.
- ▶ Recherche de routes optimales pour l'acheminement de données sur Internet.
- ▶ Recherche de la route optimale sur une carte routière.
- ▶ Le commerce électronique qui utilise la cryptographie et les signatures numérique (RSA)
- ▶ En mathématique avec la résolution d'équation différentielle.
- ▶ L'I.A qui a besoin de nombreux algorithmes.
- ▶ etc.

# INTRODUCTION AUX ALGOS DE TRI

- Le tri est une opération majeure en informatique.
- On retrouve le tri dans beaucoup de programmes comme une étape intermédiaire.
- Il est logique d'en avoir inventé un grand nombre !
- L'algorithme optimal d'une application donnée dépend par exemple du nombre d'élément à trier, si les éléments sont déjà plus ou moins triés initialement, des valeurs possibles des éléments, du type de stockage : mémoire vive, disques ou pourquoi pas bande ;)

**TRI PAR SÉLECTION . . .**

### PRINCIPE

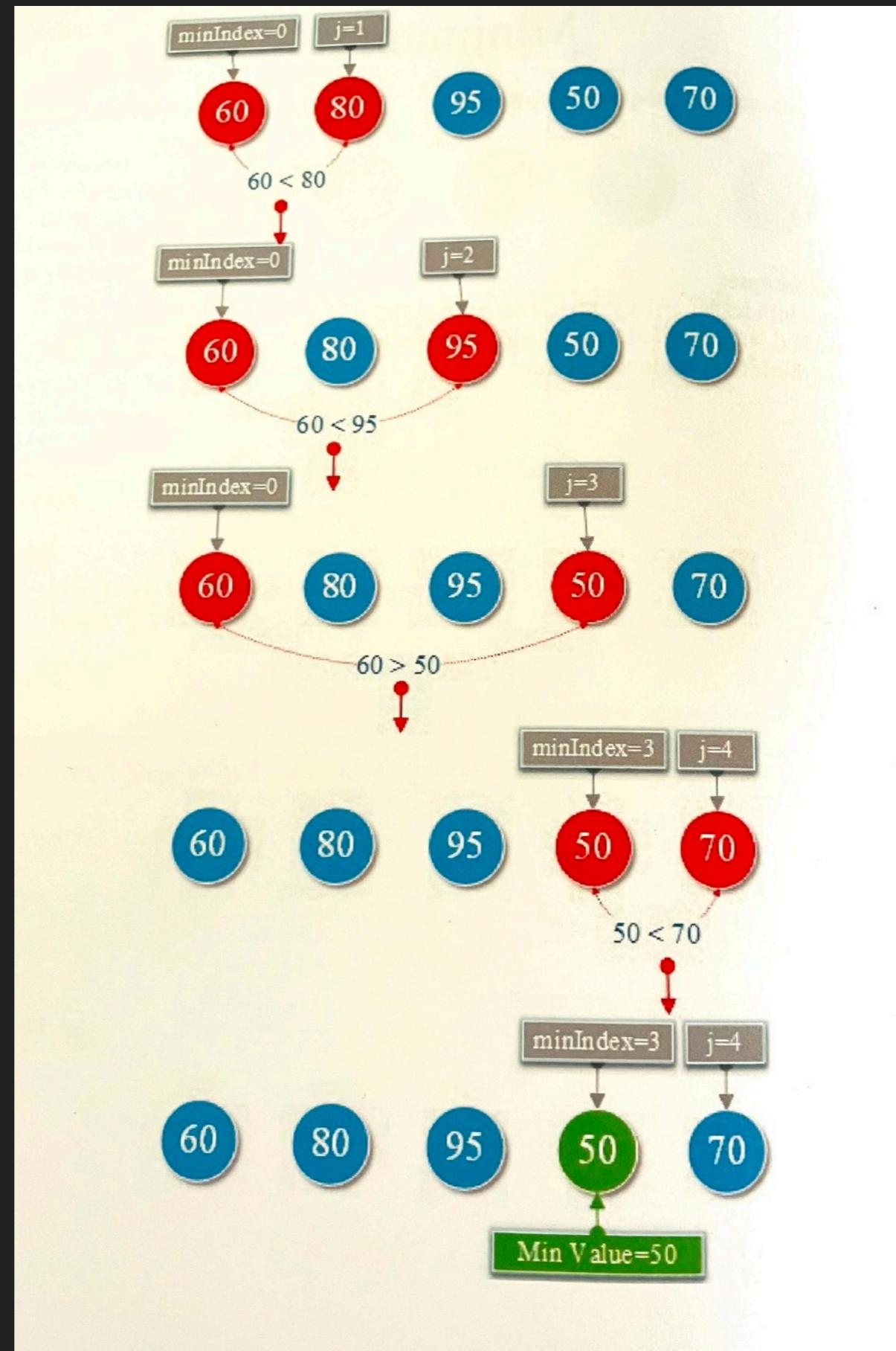
- ▶ Le principe du tri par sélection/échange (ou tri par extraction) est d'aller chercher le plus petit élément du tableau pour le mettre en premier, puis de repartir du second élément et d'aller chercher le plus petit élément du vecteur pour le mettre en second, etc.
- ▶ Voir demo : [http://lwh.free.fr/pages/algo/tri/  
tri\\_selection.html](http://lwh.free.fr/pages/algo/tri/tri_selection.html)
- ▶ On doit donc trouver le plus petit élément. Comment faire cela ?

### PRINCIPE

- ▶ Trouver le plus petit élément d'un tableau est déjà un algo en lui même : un problème à résoudre.
- ▶ Nous avons une vision ensembliste et rapidement nous détectons le plus petit si la liste n'est pas trop longue.
- ▶ Un ordinateur ne sait pas faire cela : il ne peut que comparer.
- ▶ On va donc partir du principe que le premier nombre est le plus petit et parcourir le reste du tableau en comparant à notre premier nombre. Si on trouve plus petit on mémorise son indice.

# ALGO RECHERCHE MIN DANS UN TABLEAU

## PRINCIPE



# PRINCIPE

```
def plus_petit(tableau):
    indice_petit = 0
    for i in range(1, len(tableau)):
        if tableau[i] < tableau[indice_petit]:
            indice_petit = i
    return indice_petit
```

### PRINCIPE

- ▶ Un fois qu'on a trouvé le plus petit on le met au début en permettant les valeurs...
- ▶ On est sur que le plus petit se trouve là mais l'autre valeur permuté pourra encore bouger.
- ▶ Et on poursuit avec le deuxième élément.
- ▶ On devra faire cela jusque  $n-1$  élément. En effet si on a trié  $n-1$  élément le  $n$  est forcément à sa place

# FONCTION EN PYTHON

```
def tri_selection(tableau):
    nb = len(tableau)
    for en_cours in range(0,nb-1):
        plus_petit = en_cours
        for j in range(en_cours+1,nb) :
            if tableau[j] < tableau[plus_petit] :
                plus_petit = j
        if plus_petit is not en_cours :
            temp = tableau=en_cours]
            tableau=en_cours] = tableau[plus_petit]
            tableau[plus_petit] = temp
```

## ALGO TRI PAR SÉLECTION

---

Le tri par sélection ne fonctionne efficacement que lorsque le petit ensemble d'éléments est impliqué ou que la liste est partiellement triée au préalable.

TRI PAR INSERTION . . .

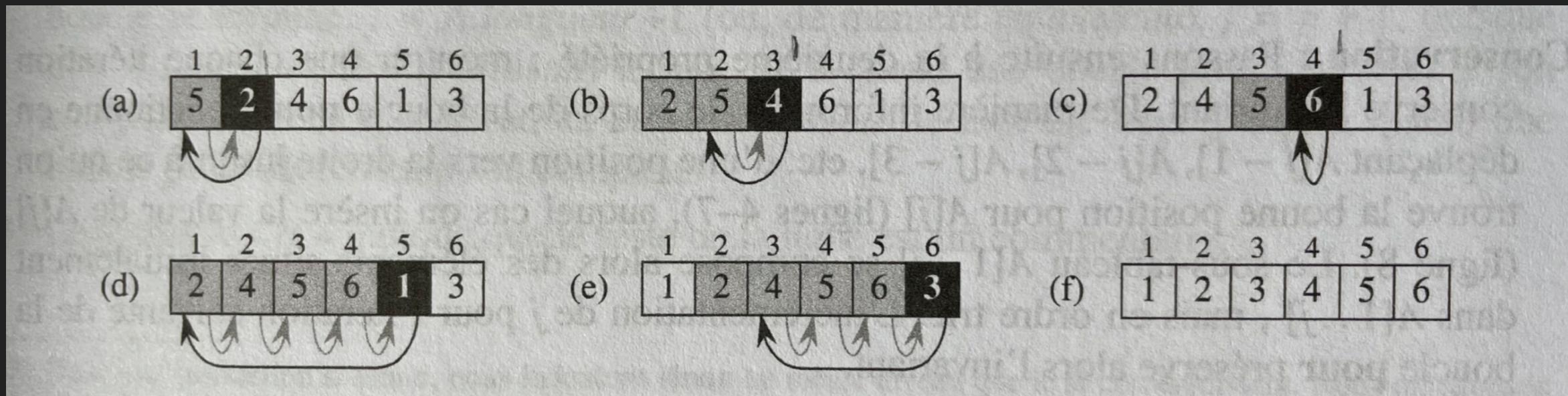
## PRINCIPE

- ▶ C'est le tri des joueurs de cartes.
- ▶ On a le jeu non trié dans la main droite et on constitue le jeu trié dans la main gauche
- ▶ Voir demo : [http://lwh.free.fr/pages/algo/tri/tri\\_insertion.html](http://lwh.free.fr/pages/algo/tri/tri_insertion.html)
- ▶ Ici pas de recherche de minimum mais juste placer la carte qui arrive de la main droite au bon endroit.
- ▶ Par contre ici pas d'échange mais un décalage pour placer la carte dans le jeu.



## PRINCIPE

- ▶ On aura un indice  $j$  qui indique la carte courante (la clé) qu'on veut insérer
- ▶ Ici  $j$  comment à 2 -> 5 (indice 1) étant déjà trié puisque seul !
- ▶ On décale les valeurs (flèches grises) jusqu'à tant que les valeurs sont supérieures à la clef
- ▶ Puis on déplace la clé (flèches noires) (Cette place est provisoire !)
- ▶ On remarque pour le cas (d) qu'on a 4 décalages pour un déplacement
- ▶ Pour le cas (a) un décalage pour un déplacement



# FONCTION EN PYTHON

```
def tri_insertion(tableau):
    for i in range(1, len(tableau)):
        en_cours = tableau[i]
        j = i
        # décalage des éléments du tableau
        while j>0 and tableau[j-1]>en_cours:
            tableau[j]=tableau[j-1]
            j = j-1
        # on insère l'élément à sa place
        tableau[j]=en_cours
```



MERCI POUR  
VOTRE ATTENTION