

# cours\_\_1

March 10, 2021

Algorithmique des tableaux

Christophe Saint-Jean

Transparents du cours

Code du cours

Année 2020-2021

## 1 Organisation de l'UE

### 1.1 Mentions Légales

Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions 4.0 International.

### 1.2 L'équipe enseignante

- **Christophe Saint-Jean** (CM/TD/TP - Resp.)
- Laurent Mascarilla (TD/TP)
- El Hadi Zahzah (TD/TP)

### 1.3 Communication

- Questions pédagogiques : [Moodle](#)
  - Approfondissement/Questions
  - Organisation de l'UE/Planning
- Questions administratives (Secrétariat)
  - Appartenance groupes TD/TP
  - Absences/Justifications

### 1.4 Dispositif horaire

On essaie tout en **synchrone**:

- 6 cours de 1,5 heures
- 4 TDs de 1,5 heures
- 5 TP de 1,5 heures
- 4 créneaux de 1,5h de TEA - 2 sujets

## 1.5 Evaluation

$$S_1 = \frac{CC_1 + CC_2}{2}$$

$$S_2 = CC_3$$

Les CC se passent *a priori* en TP 3 et 5 (idem semestre 1).

Attention à la règle sur les absences.

## 1.6 Environnement de travail

- Cours:
  - En direct: notebook
  - Hors-ligne: pdf, html (parfois des pbs de conversion).
- TD:
  - Support papier
  - Eventuellement un IDE (Thonny)
- TP/TEA:
  - Un IDE (thonny)

## 1.7 Les objectifs de cet enseignement

- Initiation à l'algorithmique
  - Qu'est ce qu'un algorithme ?
  - Différence algorithme / programme (cf. TD)
  - Initiation à l'analyse d'un algorithme.
- Approfondir vos connaissances sur :
  - les listes, tableaux, fonctions.
  - le langage Python.
- Découvrir des algorithmes simples et les analyser.
- Initiation à la récursivité.

# 2 Généralités sur l'algorithmique

## 2.1 Analogie algorithme/recette

1. Mettez la farine dans un saladier avec le sel et le sucre.
2. Faites un puits au milieu et versez-y les œufs légèrement battus à la fourchette.
3. Commencez à incorporer doucement la farine avec une cuillère en bois. Quand le mélange devient épais, ajoutez le lait froid petit à petit.
4. Quand tout le lait est mélangé, la pâte doit être assez fluide, si elle vous paraît trop épaisse, rajoutez un peu de lait. Ajoutez ensuite le beurre fondu, mélangez bien.
5. Faites cuire les crêpes dans une poêle chaude.
6. Répétez jusqu'à épuisement de la pâte.

On attend:

- Langage de description intelligible

- Instructions séquentielles, fonctions, répétitives.

On distingue bien:

- la recette (l'algorithme)
- de sa mise en oeuvre (traduction dans un langage de programmation =: Implémentation)
- aux moyens d'ustensiles de cuisine (variables, structures de données, fonctions, ...)

## 2.2 Algorithme (Définition)

Une définition formelle:

Un algorithme est la description d'une méthode de calcul qui, à partir d'un ensemble de données d'entrée (problème) et une suite finie d'étapes, produit un ensemble de données en sortie (solution).

Un algorithme est la description d'une méthode de calcul ...

### *Description*

On doit décrire chaque fonction (sous-algorithmes) non triviale et structures de données employées

### *Méthode de calcul*

Il ne peut résoudre que des problèmes calculables. On démontre que certains problèmes ne sont pas calculables (décidables).

Ex.: [Problème de l'arrêt](#) (-> L3)

Un algorithme est .... et une suite finie d'étapes, produit un ensemble de données en sortie (solution).

### *Une suite finie d'étapes*

Attention, ne pas confondre:

- Une séquence d'instructions qui se termine.
- Une description de longueur finie possibilité de boucle infinie.

### *Solution*

L'algorithme apporte t'il une solution au problème posé ?

## 2.3 Algorithmique (Définition)

L'algorithmique est la science qui étudie les algorithmes pour eux-même indépendamment de tout langage de programmation.

d'après "Al Khwarizmi", surnom du mathématicien arabe [Muhammad Ibn Musa](#) (IX siècle).

### 2.3.1 Algorithmique (Histoire de calculs)

L'algorithmique et les algorithmes sont bien antérieurs à l'informatique:

- Abaques grecques, romaines, chinoises (arithmétique simple, [racines carrées](#))
- PGCD d'[Euclide](#), [Crible d'Ératosthène](#) (IIIème siècle av. J.-C.)
- [Boulier japonais](#) (XIIIème siècle)

- [Pascaline](#) (1646)
- [Métier à tisser programmable Jacquard](#) (18ième siècle)
- Algorithme de cryptographie, Traitement des données, Algorithmique optique/quantique

### 2.3.2 Questions de l'algorithmique

1. L'algorithme A est t'il correct pour un problème P (toutes instances) ?
2. L'algorithme A se termine t'il ?
3. L'algorithme A est t'il plus efficace qu'un algorithme B ?

Parallèlement, des questions plus fondamentales:

- Est il possible de trouver un algorithme qui résoud un problème P ? (Décidabilité)
- Si, oui existe t'il un algorithme efficace résoud un problème P ? (Classes de complexité)

### 2.3.3 Exemples d'algorithmes (cf. S1)

---

#### Algorithme 1: Plus grand élément d'un tableau

---

**Données :** T un tableau  $n$  de valeurs comparables par  $<$

**Résultat :** La plus grande valeur de T

```

1 max ← T[1];
2 pour i ← 2 à n faire
3   si max < T[i] alors
4     max ← T[i];
5   fin
6 fin
7 retourner max
```

---



---

#### Algorithme 2: Approximation de $\pi$

---

**Données :**  $\pi_c$  une valeur approchée précise de  $\pi$ ,  $p$  un entier positif

**Résultat :** Une approximation  $\hat{\pi}$  de  $\pi_c$  à  $10^{-p}$  près

```

1 som ← 0 ;                               // Initialiser som à 0
2  $\hat{\pi} \leftarrow 0$  ;                       // Initialiser  $\hat{\pi}$  à 0
3 tant que  $|\hat{\pi} - \pi_c| \geq 10^{-p}$  faire
4    $som \leftarrow som + \frac{-3^k}{2^{k+1}}$  ;      // Ajouter  $\frac{-3^k}{2^{k+1}}$  à som
5    $\hat{\pi} \leftarrow \sqrt{12} \text{ som}$ ;
6    $k \leftarrow k + 1$  ;                      // Incrémenter k
7 fin
8 retourner  $\hat{\pi}$ 
```

---

## 2.4 Description d'un algorithme

On utilisera un langage de description d'un algorithme appelé **pseudo-code**.

### Caractéristiques du pseudo-code

- Il ne doit pas être attaché la syntaxe d'un langage informatique particulier.
- Il doit être lisible par un non-programmeur.
- Être capable de décrire les structures de contrôle des langages impératifs (If, While, For, ...).

#### 2.4.1 L'interface de l'algorithme

Quelques sont les entrées attendues par l'algorithme ?

- Type : Nombre, Tableau, Liste, Arbre, etc ...
- Taille : nombre de bits, nombre d'éléments du tableau, nombre de feuilles, ...
- Propriétés : entiers positifs, tableau trié, ...

Que fait/produit l'algorithme ?

- *Idem* que sur les entrées
- Description textuelle (éventuelle) de l'algorithme

#### 2.4.2 Les types utilisables

Types de données élémentaires :

- Variables simples : booléen, entier, réel, caractère
- Tableaux
- *Pointeurs*

Cela permet de définir des structures de plus haut niveau:

- Chaîne de caractères
- Ensemble, Collection
- Liste, *table de hachage*
- *Graphes*, ...

#### 2.4.3 Quelques instructions du pseudo-code

- Affectation : `<-`
- Test : `=`
- Opérations arithmétiques : `+, -, *, /`
- Séparateur d'instructions : `;`
- Elements d'un tableau T : `T[i]` (convention 1..n !!)
- Adresse d'une variable `"@"`
- Instruction de retour : `Retourner <val. sortie>`
- Le branchement conditionnel :

```
Si <condition> alors
    <blocsi>
sinon
    <blocsinon>
```

fin

- Les itératives et les répétitives:

```
Pour i <- 1 à n [par pas de 1] faire
    <bloc>
fin
```

```
Tant que <condition> faire
    <bloc>
fin
```

#### 2.4.4 Langage de description et Python

- Le langage Python a été conçu pour être le plus lisible et naturel possible.
- On est très proche du langage de description (raccourci en TD)
- **Un algorithme devient une fonction !!**

```
def max(T):
    maxi = T[0]
    for Ti in T[1:]:
        if maxi < Ti:
            maxi = Ti
    return maxi
```

En C++

```
template<class T> T max(const T* data, int size) {
    T result = data[0];
    for(int i = 1 ; i < size ; i++)
        if(result < data[i])
            result = data[i];
    return result;
}
```

En Assembleur

```

01 .MODEL SMALL
02
03 .STACK 100H
04
05 .DATA
06
07     array DB 2,8,9,5,7
08
09 .CODE
10
11
12 MAIN PROC
13
14     MOV AX,@DATA
15     MOV DS,AX
16
17     MOV CX,5
18     MOV DI,0
19
20     SUB AL,AL
21
22     BIG:
23     CMP AL,array[DI]
24
25     JA NEXT
26
27     MOV AL,array[DI]
28
29     NEXT:
30     INC DI
31     LOOP BIG
32
33     MOV AH,2
34     ADD AL,30H
35     MOV DL,AL
36     INT 21H
37
38
39
40
41
42     MAIN ENDP
43 END MAIN

```

### 2.4.5 De l'importance d'une bonne description

---

**Algorithme 3:** Calcul de l'élément maximal d'un tableau trié

---

**Données :**  $T$  un tableau trié dans l'ordre croissant de  $n$  entiers

**Résultat :**  $i_{\max}$  et  $\max$  tel que  $T[i_{\max}] = \max$  avec  $\max$  comme la plus grande valeur de  $T$

```
1  $i_{\max} \leftarrow n$ ;  
2  $\max \leftarrow T[n]$ ;
```

---

## 2.5 Analyse: Preuve de terminaison

- Vérifier que chaque instruction simple se termine:
  - calcul simple, affectation OK
  - affichage OK
  - Appel de fonction -> vérifier la fonction
- Instruction Si : vérifier la condition et les deux branches possibles
- Pour les boucles for, s'assurer que la séquence parcourue est taille finie.
- Pour la répétitive While, s'assurer que dans tous les cas que la condition de continuation sera fausse au moins une fois (critère math. parfois).
- Pour les algorithmes récursifs (plus tard), on doit s'assurer que la récursion se termine

---

**Algorithme 1:** Plus grand élément d'un tableau

---

**Données :**  $T$  un tableau  $n$  de valeurs comparables par  $<$

**Résultat :** La plus grande valeur de  $T$

```
1  $\max \leftarrow T[1]$ ;  
2 pour  $i \leftarrow 2$  à  $n$  faire  
3   | si  $\max < T[i]$  alors  
4   |   |  $\max \leftarrow T[i]$ ;  
5   | fin  
6 fin  
7 retourner  $\max$ 
```

---

## 2.6 Analyse: Preuve de correction

Il est question de prouver que l'algorithme fait ce qu'il dit faire !

On utilise souvent un invariant de boucle et la preuve par récurrence.

Pour des algorithmes concernant un tableau  $T[1..n]$ , la démarche (simplifiée) est généralement la suivante:

- Soit  $i$  l'indice du parcours de  $T$  (c.a.d.  $T[i]$ )
- Avant l'itération, quelle propriété vérifie la variable qui sera retournée par rapport à  $T[1..i-1]$  ?



- Après l'itération, cette propriété reste t'elle vraie pour  $T[1..i]$  ?
- L'initialisation de la variable est elle compatible à la propriété ?

On a donc les deux éléments d'une récurrence: Initialisation et Hérédité.

---

### Algorithme 1: Plus grand élément d'un tableau

---

**Données :**  $T$  un tableau  $n$  de valeurs comparables par  $<$

**Résultat :** La plus grande valeur de  $T$

```

1  $\max \leftarrow T[1]$ ;
2 pour  $i \leftarrow 2$  à  $n$  faire
3   | si  $\max < T[i]$  alors
4   |   |  $\max \leftarrow T[i]$ ;
5   | fin
6 fin
7 retourner  $\max$ 
```

---

## 2.7 Analyse : Complexité algorithmique (asymptotique)

L'algorithme est il rapide ?

Pour un tableau  $T$  de taille  $n$ , la rapidité *devrait* dépendre de:

- $n$  la taille  $T$  (des données).
- d'une propriété, du contenu de  $T$ .
- du langage de programmation ?
- des ressources machine (CPU, mémoire, ...)

**Les outils du jour:**

- Mesurer le temps d'exécution du programme implémentant l'algorithme (module time)
- Tracer une courbe (module matplotlib)
- Le module tqdm

```
[1]: from time import time, sleep
```

```

debut = time()
# votre code ici
# sleep(2)
fin = time()
duree = fin - debut
print("Duree: ", duree)
```

Duree: 2.0041921138763428

```
[2]: def max(T):
    maxi = T[0]
    for Ti in T[1:]:
```

```
        if maxi < Ti:
            maxi = Ti
    return maxi
```

```
[4]: from time import time, sleep
      from random import randint

      T, n = [], 10_000_000
      for _ in range(n):
          T.append(randint(1, 100_000))

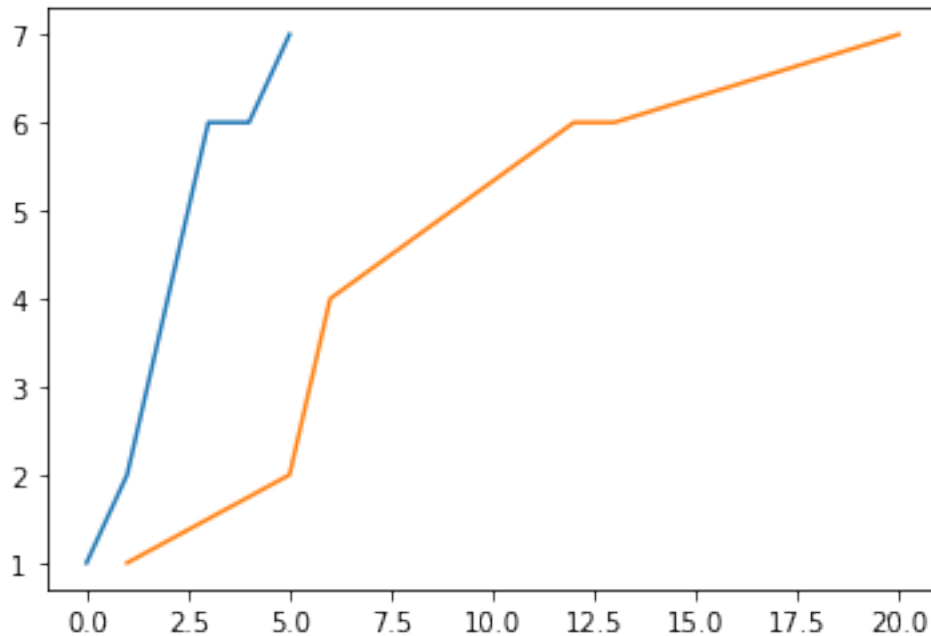
      debut = time()
      _ = max(T)
      fin = time()
      duree = fin - debut
      print("Duree: ", duree)
```

Duree: 0.5535769462585449

```
[8]: import matplotlib.pyplot as plt # a installer

      y = [1, 2, 4, 6, 6, 7]
      x1 = list(range(len(y)))
      x2 = [1, 5, 6, 12, 13, 20]

      plt.plot(x1, y)
      #plt.show()
      plt.plot(x2, y)
      plt.show()
```



```
[10]: # tqdm a installer
#from tqdm import tqdm
from tqdm.notebook import tqdm
#from tqdm.gui import tqdm
from time import sleep
from random import randint

# programme python classique
# dans un notebook
# pour thonny

for i in tqdm(range(5), desc='Boucle sur i'):
    for j in tqdm(range(30), desc=f'Boucle sur j', leave=False):
        duree_sommeil = randint(1, 2)
        sleep(duree_sommeil)
```

```
Boucle sur i:  0%|          | 0/5 [00:00<?, ?it/s]
Boucle sur j:  0%|          | 0/30 [00:00<?, ?it/s]
Boucle sur j:  0%|          | 0/30 [00:00<?, ?it/s]
```

```
-----
KeyboardInterrupt                                Traceback (most recent call last)
<ipython-input-10-2a35daf1202a> in <module>
      9     for j in tqdm(range(30), desc=f'Boucle sur j', leave=False):
     10         duree_sommeil = randint(1, 2)
--> 11         sleep(duree_sommeil)

KeyboardInterrupt:
```

```
[11]: !./run.sh
```

```
[NbConvertApp] Converting notebook complet.ipynb to notebook  
[NbConvertApp] Writing 178269 bytes to complet.nbconvert.ipynb  
[NbConvertApp] Converting notebook complet.ipynb to pdf  
[NbConvertApp] Writing 530316 bytes to complet.pdf  
[NbConvertApp] Converting notebook cours_1.ipynb to pdf  
[NbConvertApp] Writing 488391 bytes to cours_1.pdf  
[NbConvertApp] Converting notebook cours_2.ipynb to pdf  
[NbConvertApp] Writing 89820 bytes to cours_2.pdf  
[NbConvertApp] Converting notebook complet.ipynb to html  
[NbConvertApp] Writing 759266 bytes to complet.html  
[NbConvertApp] Converting notebook cours_1.ipynb to html  
[NbConvertApp] Writing 621031 bytes to cours_1.html  
[NbConvertApp] Converting notebook cours_2.ipynb to html  
[NbConvertApp] Writing 646641 bytes to cours_2.html
```

```
[ ]:
```