Algorithmique et Programmation 2 – CM1

*Transparents réalisés avec **Quarto** et **Revealjs**.*

Organisation

Cours magistraux

- Transparents (disponibles sur Célène)
- Notes sur tablette (disponibles sur Célène)
- Notes aux tableaux, tutoriels, démos, … **uniquement sur place**
- Consulter régulièrement l'EDT sur ADE

Évaluation

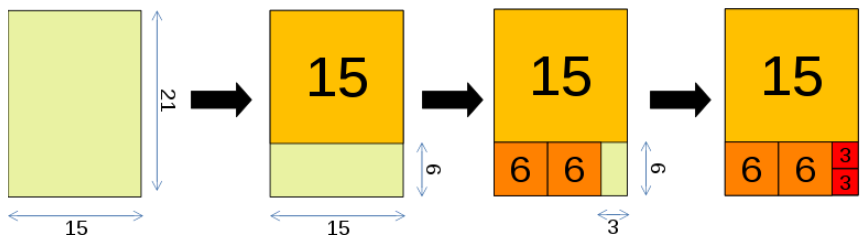
- Contrôle continu le 27/02
- Contrôle terminal pendant la période d’examens
- Principalement basés sur les éléments de TD

Retour sur le S1

Fonctions

Plus grand diviseur commun

```python
def pgcd(a,b):
    # Pré-condition : a > b
    pgcd = a
    r = b

    while (r != 0):
        t = r
        r = pgcd%r
        pgcd = t

    return pgcd
```



Fonction d'Euler

Définition

$$\begin{array}{rcccl} \varphi & : & \mathbb{N}^* & \to & \mathbb{N}^* \\ & & n & \mapsto & \left|\{m \in \mathbb{N}^* \mid m \leqslant n \text{ et } m \text{ premier avec } n\}\right| \end{array}$$

```python
def euler(n):
    zn = 0
    # 0 n'est pas dans Zn
    for i in range(1,n):
        if(pgcd(n,i) == 1):
            zn = zn + 1
    return zn

print(euler(99))
```

```
60
```

Listes et dictionnaires

Création de listes

```python
def creation(n):
    liste = []
    for i in range(1,n) :
        liste.append(i)
    return liste
```

```python
def diviseurs(n):
    liste_diviseurs = []
    for i in range(1,n) :
        if n % i == 0 :
            liste_diviseurs.append(i)
    return liste_diviseurs
```

Dictionnaires

```python
enseignants = {
    # clé: valeur,
    "TD1": "Marcilio de Souto",
    "TD2": "Mathieu Guilbert",
    "TD3": "Wadoud Bousdira-Semmar",
    "TD4": "Catherine Julié-Bonnet",
}
```

```python
for groupe in enseignants: # équivalent à enseignants.keys()
    print(f"{groupe}", end=", ")

for enseignant in enseignants.values():
    print(f"{enseignant}", end=", ")

for groupe, enseignant in enseignants.items():
    print(f"{enseignant} est en charge du groupe {groupe}")
```

Récursivité

Un premier exemple

```
1 def fonction(n):
2     if(n==0):
3         return 0
4     if(n==1):
5         return 1
6     else:
7         return n + fonction(n-1)
8 
9 print(fonction(10))
```

```
55
```

```
1 print(fonction(100))
2 print(fonction(1000))
```

```
5050
500500
```

Classes et objets

Définition d’une classe

```
class Piece :
    def __init__(self, nom, L, l):
        self.nom = nom
        self.L = L
        self.l = l

    # Retourne une chaîne de caractères affichant l’objet
    def __str__(self):
        modele = "{0} : L = {1} m, l = {2} m, surface = {3} m².".
        return modele.format(self.nom, self.L, self.l, self.surface())

    def surface(self):
        return self.L * self.l
```

Utilisation d’une classe dans une autre

```
class Maison :
    def __init__(self):
        self.pieces = []

    def __str__(self):
        ch = f"{self.surface()} m² et {len(self.pieces)} pièces :\n"
        for piece in self.pieces :
            ch += f" - {piece.__str__()}\n"
        return ch

    def ajouter_piece(self, piece):
        if piece not in self.pieces:
            self.pieces.append(piece)

    def supprimer_piece(self, piece):
        if piece in self.pieces:
            self.pieces.remove(piece)
    …
```

Utilisation d’une classe dans une autre

```
class Maison :
    …

    def ajouter_piece(self, piece):
        if piece not in self.pieces:
            self.pieces.append(piece)

    def supprimer_piece(self, piece):
        if piece in self.pieces:
            self.pieces.remove(piece)

    def surface(self) :
        s = 0
        for piece in self.pieces :
            s += piece.surface()
        return s
```

Manipulation de classes

```
1 m = Maison()
2 m.ajouter_piece(Piece("chambre",4,3))
3 print(m)
4 print(f"La maison a une surface de {m.surface()} m2")
```

```
12 m² et 1 pièces :
 - chambre : L = 4 m, l = 3 m, surface = 12 m².

La maison a une surface de 12 m2
```

Création de listes

```python
def create_append(n):
    L = []
    for i in range(n):
        L.append(i)
    return L

def create_init(n):
    L = [0]*n
    for i in range(n):
        L[i] = i
    return L
```

```
Pour n=500000
avec create_append : 4.180050159979146s
avec create_init : 3.599399237020407s
```

Compréhension de listes

```python
def create_comprehension(n):
    # Création d'une liste composée des entiers i dans l'intervalle [0,n-1]
    return [i for i in range(n)]
```

```
avec create_append : 4.1800501599791 46s
avec create_init : 3.599399237020407s
avec create_comprehension : 2.4883745859842747s
```

(Plus que du) sucre syntaxique (mais...)

```
1 def euler(n):
2     zn = 0
3     for i in range(1,n):
4         if(pgcd(n,i) == 1):
5             zn = zn + 1
6     return zn
7     return len([i for i in range(n) if pgcd(n,i) == 1])
```

Fonctions prédéfinies

```
1 def somme(n) :
2     return sum([i for i in range(n+1)])
3
4 print(somme(1000))
```

```
500500
```

Algorithmes de tri

Tri par sélection

```python
def tri_selection(liste):
    for i in range(len(liste)-1):
        indice = i
        for j in range(i+1, len(liste)):
            if (liste[j]<liste[indice]):
                indice = j

        if (indice!=i):
            tmp = liste[i]
            liste[i] = liste[indice]
            liste[indice] = tmp
```

Tri par insertion

```python
def tri_insertion(liste):
    for i in range(1, len(liste)):
        x = liste[i]
        j = i

        while (j > 0) and (x < liste[j-1]):
            liste[j] = liste[j-1]
            j -= 1

        liste[j] = x
```

En python

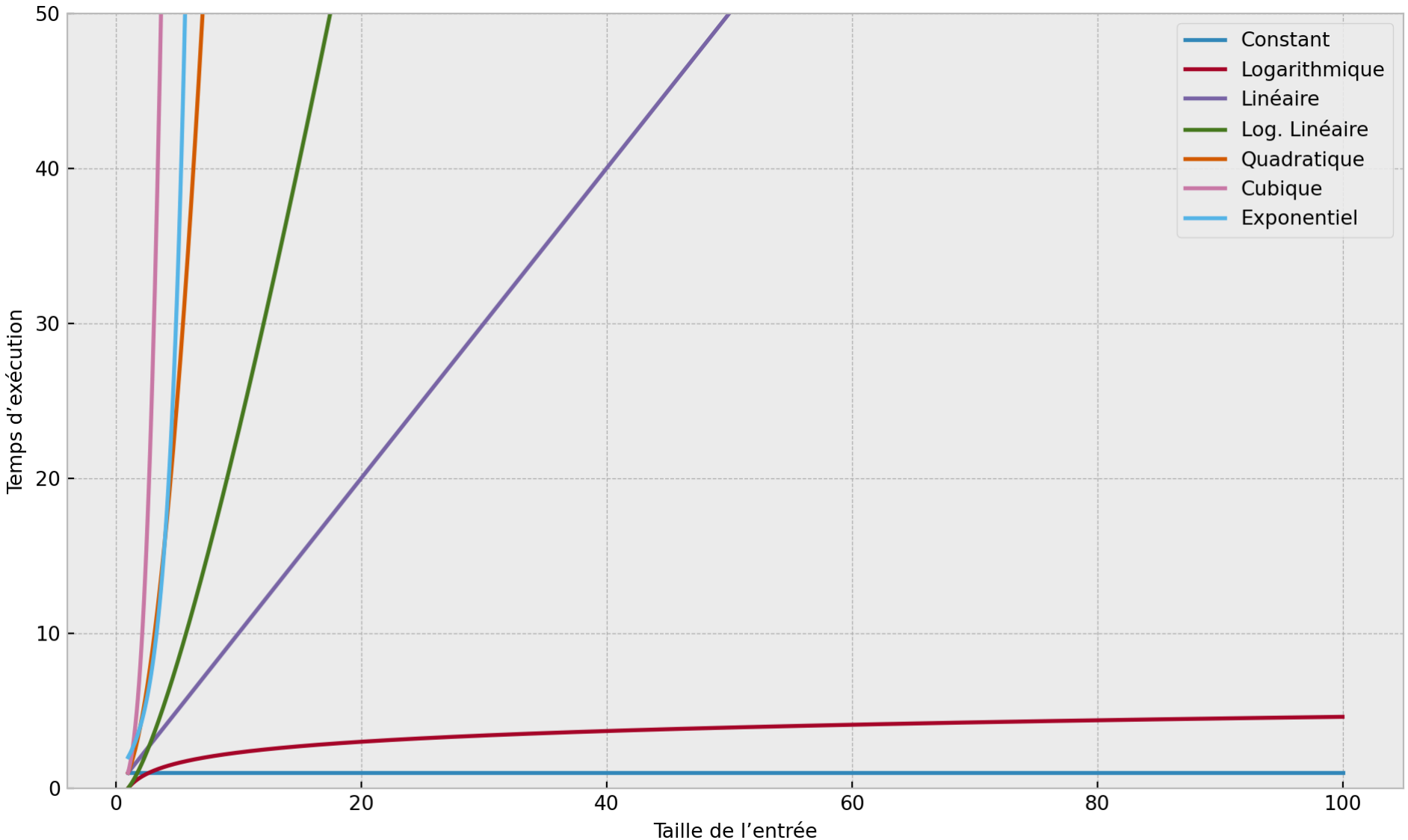
```python
import random

liste = [random.randrange(1,10) for i in range(10)]
print(liste)

print(sorted(liste))
```

```
[3, 5, 7, 3, 4, 9, 4, 7, 8, 7]
[3, 3, 4, 4, 5, 7, 7, 7, 8, 9]
```

Quel algorithme est utilisé ? (la **doc** aide-t-elle ?)

Notions de complexité

Temps d’exécution



Pour les tris

> **Note**
>
> - Tri séléction : $O(n^2)$
> - Tri insertion : $O(n^2)$
> - Tri fusion : $O(n \cdot log_2(n))$

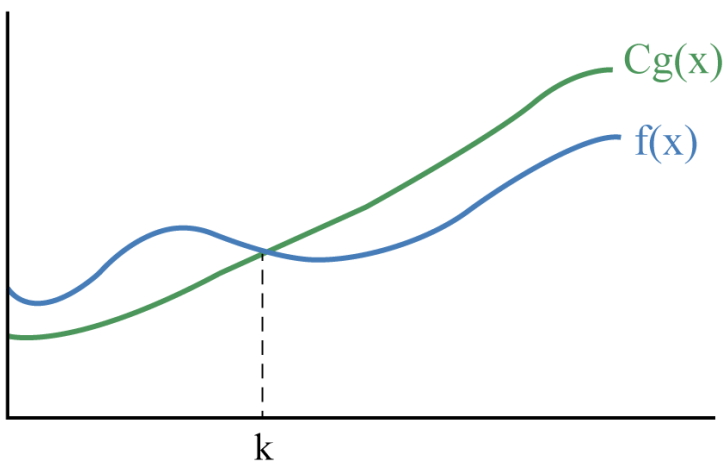
Mais encore ?

La notation $O(\cdot)$

Définition formelle

$$f(x) = O(g(x)) \equiv \exists C > 0, k \geqslant 0 : |f(x)| \leqslant c \cdot |g(x)|$$
$$\forall\, x \geqslant k$$

Mais encore ?



Calcworkshop.com

La suite de l'UE

PREMIÈRE PARTIE

1. (Retour sur) la récursivité et les listes chaînées
2. Files et piles
3. Programmation Orientée Objet
4. Arbres (binaires ((de recherche)))
5. Exceptions, tests unitaires, (in)variants, … (Hors CC)
6. **Contrôle Continu**

DEUXIÈME PARTIE

1. Graphes
2. Algorithmes de graphes et tas
3. Algorithmique du texte
4. Ouverture

Pseudo-code

Description des algorithmes en langage naturel

```
1 Fonction resultat (n)
2 Debut
3     Si (n > 1)
4         Retourner n * resultat(n-1)
5     Sinon
6         Retourner 1
7     FinSi
8 Fin
```

Pseudo-code