Algorithmique avancée

Module : Algorithmes et structures de données

Douglas Teodoro

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

2019-2020

Sommaire

PRÉSENTATION DU COURS

Organisation du module 633-1

2 parties

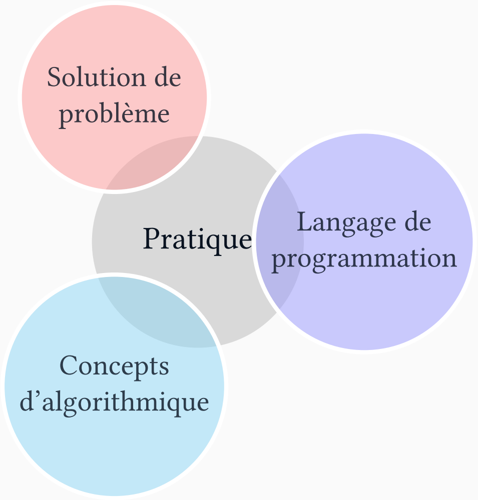
- Algorithmique avancée
- Structure de données avancées

4 intervenants

- Flávio Lindo (flavio.barreiro-lindo@hesge.ch)
- Nader Soukouti (nader.soukouti@hesge.ch)
- Christian Stettler (christian.stettler@hesge.ch)
- Douglas Teodoro (douglas.teodoro@hesge.ch)

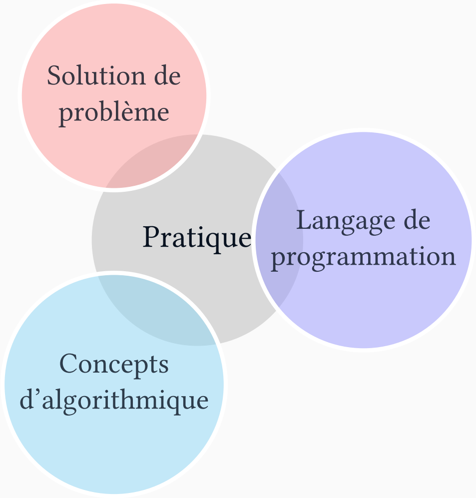
Objectif de l'unité Algorithmique avancée

- Développer un esprit d'analyse algorithmique nécessaire pour résoudre des problèmes pratiques



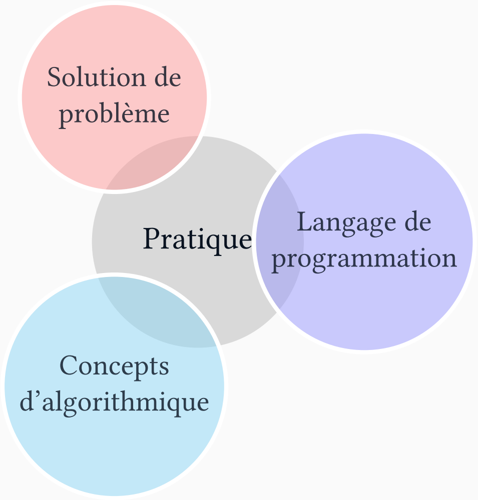
Objectif de l'unité Algorithmique avancée

- Développer un esprit d'analyse algorithmique nécessaire pour résoudre des problèmes pratiques
- Concevoir, réaliser et analyser la solution algorithmique d'un problème à l'aide des structures de données classiques



Objectif de l'unité Algorithmique avancée

- Développer un esprit d'analyse algorithmique nécessaire pour résoudre des problèmes pratiques
- Concevoir, réaliser et analyser la solution algorithmique d'un problème à l'aide des structures de données classiques
- Implanter dans un langage de programmation les principaux algorithmes liés aux structures de données étudiées



Programme du cours

- Introduction à l'analyse algorithmique
  - Analyse de complexité (itérations, ordres de grandeur)
  - Preuve et correction
  - La récursivité
- Algorithmes de tris et de recherche
  - Tris comparatifs et linéaires
  - Recherche dans un tableau trié
- Algorithmes pour les structures de données
  - Parcours de liste, arbre et graphe
  - Recherche et tris
  - Plus court chemin
- Principaux paradigmes
  - Diviser pour régner
  - Programmation dynamique
  - Algorithmes gloutons

Expérience des étudiant-e-s

Niveau de difficulté rencontré pour : Développer un algorithme pour résoudre un problème (sur papier ou conceptuellement)

RÉPONSE	MOYENNE	TOTAL
1 (Très difficile)	10%	3
2	39%	12
3	32%	10
4	16%	5
5 (Très facile)	3%	1
Total des réponses à la question	100%	31/31

Niveau de difficulté rencontré pour : Comprendre la définition d'un problème

RÉPONSE	MOYENNE	TOTAL
2	35%	11
3	29%	9
4	26%	8
5 (Très facile)	10%	3
Total des réponses à la question	100%	31/31

Méthodes pédagogiques

Organisation

- Durée du module : 15 semaines
- Deux heures de cours théoriques (50%) / pratiques (50%)
- Deux heures de laboratoire en conjunction avec l'unité « Structure de données avancées »
- Un assistant est à disposition dehors de ces séances sur rendez-vous

Méthodes pédagogiques

Forme

- **Partie théorique :** n'interdit nullement les questions et la participation des étudiant-e-s
- **Partie pratique :** repose essentiellement sur la **participation active** des étudiant-e-s
- **Consacre un temps** à la consolidation et à l'étude de ses notes et à la résolution des problèmes

Méthodes pédagogiques

Forme

- **Partie théorique :** n'interdit nullement les questions et la participation des étudiant-e-s
- **Partie pratique :** repose essentiellement sur la **participation active** des étudiant-e-s
- **Consacre un temps** à la consolidation et à l'étude de ses notes et à la résolution des problèmes

Philosophie

Les étudiant-e-s sont encouragées à **prendre en charge leur propre processus d'apprentissage (codez !)**

Mode d'évaluation - Intégrée avec Struct. de données

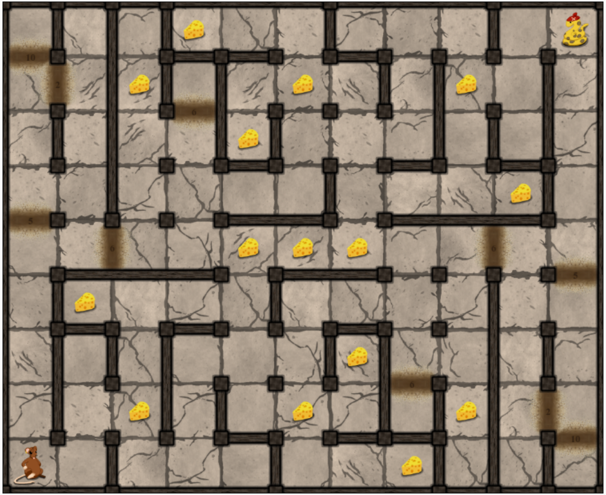
Contrôle continu - 50%

- Des travaux pratiques individuels (avec doc) (15%)
  - Rendu : semaines 4, 6, 8, 10, 13
- Un projet pratique en binôme (15%)
  - Rendu : semaine 14
- Un contrôle continue pratique individuel (sans doc) d'une durée approx. de 90 minutes (20%)
  - Jeudi 05 décembre 2019 à 17h15 (semaine 11)

Examen - 50%

- Un examen pratique interdisciplinaire (Algo&Struct) d'une durée de 180 minutes
  - L'examen aura lieu lors de la semaine du 20 janvier 2020 (semaine 16)
  - Rendu de 75% des travaux pratiques est exigé pour se présenter à l'examen

Projet - Python



Objectif : développer des algorithmes performants pour des structures de données avancées (graphes)

Mode d'évaluation

Note de l'unité de cours « Algorithmique avancée »

- Moyenne pondérée des notes des contrôles continus
- note = 20% CC $+15\%$ TPs $+15\%$ PROJ

Formation de la note du module

- Moyenne arithmétique des contrôles continus (Algo ; Struct) : 50%
- Note de l'examen : 50%

Références

Cormen, *Algorithmes Notions de base* (Dunod, 2013)
http://hesge.scholarvox.com

Cormen, Leiserson, Rivest et Stein, *Introduction à l'algorithmique* (Dunod, 2010)

Sedgewick et Wayne, *Algorithms* (Pearson Education, 2011)

Deitel et Deitel, *JAVA How to program* (Prentice Hall, 2012)

Evans et Flanagan, *Java in a Nutshell* (O'Reilly, 2015)

Cyberlean

- **Cours** : `19_HES-SO-GE_633-1 - ALGORITHMES ET STRUCTURES DE DONNÉES`
- **Mot de passe** : `633-1-2019`

Sommaire

CONCEPTION ET ANALYSE DES ALGORITHMES

Exemple d'algorithme - Calcul de la moyenne d'un tableau de notes

problème entant donné [`un tableau` A `de` n `nombres réels`],
on demande la [moyenne des nombres du tableau]

Exemple d'algorithme - Calcul de la moyenne d'un tableau de notes

problème entant donné [`un tableau` A `de` n `nombres réels`],
on demande la [moyenne des nombres du tableau]

$$moy = \mathtt{somme}(A)/n = (a_1 + a_2 + \ldots + a_{n-1} + a_n)/n$$

Exemple d'algorithme

$$moy = (a_1 + a_2 + \ldots + a_{n-1} + a_n)/n$$

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                          // résultat
i : entier = 0                      // indice
s : réel = 0.0                      // somme
pour i ← 1 à A.longueur faire       // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

Exemple d'algorithme

$$moy = (a_1 + a_2 + \ldots + a_{n-1} + a_n)/n$$

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                          // résultat
i : entier = 0                      // indice
s : réel = 0.0                      // somme
pour i ← 1 à A.longueur faire       // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

pour i=1 $\rightarrow s = 0.0 + a_1$ $\rightarrow$ 1 op

Exemple d'algorithme

$$moy = (a_1 + a_2 + \ldots + a_{n-1} + a_n)/n$$

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                       // résultat
i : entier = 0                   // indice
s : réel = 0.0                   // somme
pour i ← 1 à A.longueur faire    // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

pour i=1 $\rightarrow s = 0.0 + a_1$ $\rightarrow$1 op

pour i=2 $\rightarrow s = 0.0 + a_1 + a_2$ $\rightarrow$2 op

Exemple d'algorithme

$$moy = (a_1 + a_2 + \ldots + a_{n-1} + a_n)/n$$

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels

Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                       // résultat
i : entier = 0                   // indice
s : réel = 0.0                   // somme
pour i ← 1 à A.longueur faire    // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

pour i=1 $\rightarrow s = 0.0 + a_1$ $\rightarrow$1 op

pour i=2 $\rightarrow s = 0.0 + a_1 + a_2$ $\rightarrow$2 op

pour i=3 $\rightarrow s = 0.0 + a_1 + a_2 + a_3$ $\rightarrow$3 ops

Exemple d'algorithme

$$moy = (a_1 + a_2 + \ldots + a_{n-1} + a_n)/n$$

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                          // résultat
i : entier = 0                      // indice
s : réel = 0.0                      // somme
pour i ← 1 à A.longueur faire       // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

pour i=1 $\rightarrow s = 0.0 + a_1$ $\rightarrow$1 op

pour i=2 $\rightarrow s = 0.0 + a_1 + a_2$ $\rightarrow$2 op

pour i=3 $\rightarrow s = 0.0 + a_1 + a_2 + a_3$ $\rightarrow$3 ops

$\vdots$

Exemple d'algorithme

$$moy = (a_1 + a_2 + \ldots + a_{n-1} + a_n)/n$$

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                          // résultat
i : entier = 0                      // indice
s : réel = 0.0                      // somme
pour i ← 1 à A.longueur faire       // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

pour `i=1` $\longrightarrow s = 0.0 + a_1$ $\longrightarrow$ 1 op

pour `i=2` $\longrightarrow s = 0.0 + a_1 + a_2$ $\longrightarrow$ 2 op

pour `i=3` $\longrightarrow s = 0.0 + a_1 + a_2 + a_3$ $\longrightarrow$ 3 ops

$\vdots$

pour `i=n` $\longrightarrow s = 0.0 + a_1 + \cdots + a_n$ $\longrightarrow$ n ops

Exemple d'algorithme

$$moy = (a_1 + a_2 + \ldots + a_{n-1} + a_n)/n$$

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

moy : réel // résultat
i : entier $= 0$ // indice
s : réel $= 0.0$ // somme
pour $i \leftarrow 1$ **à** $A.longueur$ **faire** // $n = A.longueur$
    $s = s + A[i]$
$moy = s/A.longueur$

pour i=1 $\longrightarrow s = 0.0 + a_1$ $\longrightarrow$1 op
pour i=2 $\longrightarrow s = 0.0 + a_1 + a_2$ $\longrightarrow$2 op
pour i=3 $\longrightarrow s = 0.0 + a_1 + a_2 + a_3$ $\longrightarrow$3 ops
$\vdots$
pour i=n $\longrightarrow s = 0.0 + a_1 + \cdots + a_n$ $\longrightarrow n$ ops

Quel est le nombre d'opérations d'additions effectuées par cet algorithme ?

Différentes problématiques

terminaison terminera dans un temps fini

complexité en temps terminera dans un temps borné (raisonnable)

complexité en espace terminera en utilisant une quantité de mémoire bornée (raisonnable)

correction si l'algorithme termine en donnant une proposition de solution, alors cette solution est correcte

complétude pour un espace de problèmes donné, l'algorithme, s'il termine, donnera toujours des propositions de solutions

Analyse - Calcul de la moyenne

terminaison se terminera ?

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

```
moy : réel              // résultat
i : entier = 0          // indice
s : réel = 0.0          // somme
pour i ← 1 à A.longueur faire  // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

Analyse - Calcul de la moyenne

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels

Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                      // résultat
i : entier = 0                  // indice
s : réel = 0.0                  // somme
pour i ← 1 à A.longueur faire  // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

terminaison se terminera ?

- oui (`1 .. n` ou `0 .. n-1`)

Analyse - Calcul de la moyenne

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                          // résultat
i : entier = 0                      // indice
s : réel = 0.0                      // somme
pour i ← 1 à A.longueur faire       // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

terminaison se terminera ?

- oui (`1 .. n` ou `0 .. n-1`)

complexité en temps combien de temps ?

Analyse - Calcul de la moyenne

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                          // résultat
i : entier = 0                      // indice
s : réel = 0.0                      // somme
pour i ← 1 à A.longueur faire       // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

terminaison se terminera ?

- oui (`1 .. n` ou `0 .. n-1`)

complexité en temps combien de temps ?

- $c_t \times n$

Analyse - Calcul de la moyenne

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : *moy* : la moyenne des nombres du tableau

```
moy : réel                          // résultat
i : entier = 0                      // indice
s : réel = 0.0                      // somme
pour i ← 1 à A.longueur faire       // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

terminaison se terminera ?

- oui (`1 .. n` ou `0 .. n-1`)

complexité en temps combien de temps ?

- $c_t \times n$

complexité en espace combien d'espace ?

Analyse - Calcul de la moyenne

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels

Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                          // résultat
i : entier = 0                      // indice
s : réel = 0.0                      // somme
pour i ← 1 à A.longueur faire       // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

terminaison se terminera ?

- oui (`1 .. n` ou `0 .. n-1`)

complexité en temps combien de temps ?

- $c_t \times n$

complexité en espace combien d'espace ?

- $c_s \times n$

Analyse - Calcul de la moyenne

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels

Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                       // résultat
i : entier = 0                   // indice
s : réel = 0.0                   // somme
pour i ← 1 à A.longueur faire    // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

terminaison se terminera ?

- oui (`1 .. n` ou `0 .. n-1`)

complexité en temps combien de temps ?

- $c_t \times n$

complexité en espace combien d'espace ?

- $c_s \times n$

correction solution est correcte ?

Analyse - Calcul de la moyenne

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

moy : réel // résultat
i : entier $= 0$ // indice
s : réel $= 0.0$ // somme
pour $i \leftarrow 1$ **à** $A.longueur$ **faire** // $n = A.longueur$
    $s = s + A[i]$
$moy = s / A.longueur$

terminaison se terminera ?

- oui (`1 .. n` ou `0 .. n-1`)

complexité en temps combien de temps ?

- $c_t \times n$

complexité en espace combien d'espace ?

- $c_s \times n$

correction solution est correcte ?

- oui

Analyse - Calcul de la moyenne

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                         // résultat
i : entier = 0                     // indice
s : réel = 0.0                     // somme
pour i ← 1 à A.longueur faire      // n = A.longueur
    s = s + A[i]
moy = s / A.longueur
```

terminaison se terminera ?
- oui (`1 .. n` ou `0 .. n-1`)

complexité en temps combien de temps ?
- $c_t \times n$

complexité en espace combien d'espace ?
- $c_s \times n$

correction solution est correcte ?
- oui

complétude toute l'espace

Analyse - Calcul de la moyenne

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels

Résultat : moy : la moyenne des nombres du tableau

```
moy : réel                          // résultat
i : entier = 0                      // indice
s : réel = 0.0                      // somme
pour i ← 1 à A.longueur faire       // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

terminaison se terminera ?

- oui (`1 .. n` ou `0 .. n-1`)

complexité en temps combien de temps ?

- $c_t \times n$

complexité en espace combien d'espace ?

- $c_s \times n$

correction solution est correcte ?

- oui

complétude toute l'espace

- non ($n == 0$?)

Calcul de la moyenne - Implementation en Python

Algorithme : Calcul de la moyenne

Données : A : un tableau de n nombres réels
Résultat : s : la moyenne des nombres du tableau

```
moy : réel                       // résultat
i : entier = 0                   // indice
s : réel = 0.0                   // somme
pour i ← 1 à A.longueur faire    // n = A.longueur
    s = s + A[i]
moy = s/A.longueur
```

```python
#Calcul de la moyenne
moy:float              # résultat
A:list = [4.0,5.0,6.0]  # entrée
s:float = 0.0          # somme
for i in range(0, len(A)):
    s = s + A[i]
moy = s / len(A)
```

Correction et efficacité

Deux **aspects** nous intéressent principalement

- **correct** c'est-à-dire, qu'il donne toujours la bonne réponse quelles que soient les données d'entrée
- **efficace** c'est-à-dire, économe en temps de calcul et en ressources

PyCharm - semaine_1

exercice_1.py

PyCharm - Exercice 1

1. Coder un algorithme pour calculer la moyenne d’un tableau en utilisant une boucle `while`

PyCharm - Exercice 1

1. Coder un algorithme pour calculer la moyenne d'un tableau en utilisant une boucle `while`

2. Modifier l'algorithme pour qu'il n'incrémente pas i

PyCharm - Exercice 1

1. Coder un algorithme pour calculer la moyenne d'un tableau en utilisant une boucle `while`
2. Modifier l'algorithme pour qu'il n'incrémente pas i
   - Est-ce qu'il finit ?

PyCharm - Exercice 1

1. Coder un algorithme pour calculer la moyenne d'un tableau en utilisant une boucle `while`
2. Modifier l'algorithme pour qu'il n'incrémente pas i
   - Est-ce qu'il finit ?
3. Modifier l'algorithme initial pour qu'il divise la valeur par n dans la boucle au contraire de faire à la fin

PyCharm - Exercice 1

1. Coder un algorithme pour calculer la moyenne d'un tableau en utilisant une boucle `while`
2. Modifier l'algorithme pour qu'il n'incrémente pas i
   - Est-ce qu'il finit ?
3. Modifier l'algorithme initial pour qu'il divise la valeur par n dans la boucle au contraire de faire à la fin
   - Est-il correct ?

Correction - Invariant de boucle

Definition

invariant de boucle : c'est une propriété qui si elle est vraie en début de boucle, reste vraie en fin de boucle

```
...
// l'Invariant de boucle doit être vrai ici
while ( CONDITION DE TEST ) {
  // début de la boucle

  ...

  // fin de la boucle
  // l'Invariant de boucle doit être vrai ici
}
// Terminaison + Invariant de boucle = But
...
```

Un autre example

Problème de tri

Données : une suite de n nombres $(a_1, a_2, \dots, a_n)$
Résultat : permutation de la suite donnée en entrée $(a'_1, a'_2, \dots, a'_n)$
de telle sorte que $a'{}_1 \leq a'{}_2 \leq \cdots \leq a'_n$

Un autre example

Problème de tri

Données : une suite de n nombres $(a_1, a_2, \ldots, a_n)$
Résultat : permutation de la suite donnée en entrée $(a'_1, a'_2, \ldots, a'_n)$
de telle sorte que $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Principe du tri par insertion

On parcourt entièrement la liste en appliquant à chaque itération la stratégie suivante :
$\longrightarrow$ recherche de la place du i^{ème} élément

Tri par insertion - Principe

On veut trier la liste :

3	2	7	1	5

1. on part du constat que la `liste [1..1]` est triée

3	2	7	1	5

Tri par insertion - Principe

On veut trier la liste :

3	2	7	1	5

1. on part du constat que la `liste [1..1]` est triée

3	2	7	1	5

2. on met l'élément d'indice 2 à sa place dans la `liste [1..2]`

2	3	7	1	5

DEMO - tri insertion

Tri par insertion - Principe

On veut trier la liste :

3	2	7	1	5

1. on part du constat que la `liste [1..1]` est triée

3	2	7	1	5

2. on met l'élément d'indice 2 à sa place dans la `liste [1..2]`

2	3	7	1	5

3. la `liste [1..2]` est maintenant triée

DEMO - tri insertion

Tri par insertion - Principe

On veut trier la liste :

3	2	7	1	5

1. on part du constat que la `liste [1..1]` est triée

3	2	7	1	5

2. on met l'élément d'indice 2 à sa place dans la `liste [1..2]`

2	3	7	1	5

3. la `liste [1..2]` est maintenant triée
4. on recommence a l'étape 2 avec l'élément suivant

DEMO - tri insertion

Tri par insertion

Algorithme : Tri par insertion

Données : A : un tableau de n nombres à trier
Résultat : A : le tableau trié

```
i : entier      // indice de la partie tableau trié
j : entier      // indice du tableau
cle : élément du tableau      // élément en train d'être trié
pour j ← 2 à A.longueur faire
    i = j − 1
    cle = A[j]
    tant que i > 0 et A[i] > cle faire
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = cle
```

Tri par insertion

Algorithme : Tri par insertion

Données : A : un tableau de n nombres à trier
Résultat : A : le tableau trié

i : entier `// indice de la partie tableau trié`
j : entier `// indice du tableau`
cle : élément du tableau `// élément en train d'être trié`
pour $j \leftarrow 2$ **à** $A.longueur$ **faire**
  $i = j - 1$
  $cle = A[j]$
  tant que $i > 0$ ***et*** $A[i] > cle$ **faire**
    $A[i+1] = A[i]$
    $i = i - 1$
  $A[i+1] = cle$

```python
# Tri par insertion

A = [2,1,6,3,4,5]

for j in range(1, len(A)):
    i = j - 1
    cle = A[j]
    while i >= 0 and A[i] > cle:
        A[i+1] = A[i]
        i = i - 1

    A[i+1] = cle
```

Tri par insertion

Algorithme : Tri par insertion

Données : A : un tableau de n nombres à trier
Résultat : A : le tableau trié

i : entier `// indice de la partie tableau trié`
j : entier `// indice du tableau`
cle : élément du tableau `// élément en train d'être trié`
pour $j \leftarrow 2$ **à** $A.longueur$ **faire**
  $i = j - 1$
  $cle = A[j]$
  tant que $i > 0$ ***et*** $A[i] > cle$ **faire**
    $A[i+1] = A[i]$
    $i = i - 1$
  $A[i+1] = cle$

```python
# Tri par insertion

A = [2,1,6,3,4,5]

for j in range(1, len(A)):
    i = j - 1
    cle = A[j]
    while i >= 0 and A[i] > cle:
        A[i+1] = A[i]
        i = i - 1

    A[i+1] = cle
```

Invariant :
le sous-tableau `A[1..j-1]` contient les éléments du tableau original `A[1..j-1]` ordonnés

PyCharm - semaine_1

`exercice_2.py`

PyCharm - Exercice 2

1. Codez en Python l'algorithme «tri par insertion» pour trier un tableau `A` de taille `n`
2. Exécutez-le étape par étape et regardez les valeurs des variables `i`, `j` et `cle` et du tableau `A[1..j-1]`
3. Imprimez le nombre de fois que l'instruction `i=i-1` est exécutée pour les tableaux
   - `A = [2,1]`
   - `A = [3,2,1]`
   - `A = [4,3,2,1]`
   - `A = [5,4,3,2,1]`
   - `A = [6,5,4,3,2,1]`

Complexité d'un algorithme

Permet de quantifier les algorithmes

Deux types de complexité

- en espace : quelle quantité de place en mémoire va t-on utiliser ?
- en temps : combien d'opérations va t-on effectuer ?

Intérêt : comparer deux algorithmes

Mesurer le temps d'exécution

Complexité en temps

On cherche une fonction $T(n)$ représentant le temps d'exécution d'un algorithme en fonction de la taille de l'entrée n

Mesurer le temps d'exécution

Complexité en temps

On cherche une fonction $T(n)$ représentant le temps d'exécution d'un algorithme en fonction de la taille de l'entrée n

Le temps d'exécution dépend de l'entrée : par exemple, dans le cas d'un algorithme de tri, si le tableau est déjà trié

Mesurer le temps d'exécution

Complexité en temps

On cherche une fonction $T(n)$ représentant le temps d'exécution d'un algorithme en fonction de la taille de l'entrée n

Le temps d'exécution dépend de l'entrée : par exemple, dans le cas d'un algorithme de tri, si le tableau est déjà trié

On s'intéresse aux

- Meilleur des cas
- Cas moyen
- Pire de cas

Mesurer le temps d'exécution - tri par insertion

Algorithme : Tri par insertion

Données : A : un tableau de n nombres à trier
Résultat : A : le tableau trié

```
i : entier
j : entier
cle : élément du tableau
pour j ← 2 à A.longueur faire
    cle = A[j]
    i = j − 1
    tant que i > 0 et A[i] > cle faire
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = cle
```

coût	fois
c_4	n

Mesurer le temps d'exécution - tri par insertion

Algorithme : Tri par insertion

Données : A : un tableau de n nombres à trier
Résultat : A : le tableau trié

```
i : entier
j : entier
cle : élément du tableau
pour j ← 2 à A.longueur faire
    cle = A[j]
    i = j − 1
    tant que i > 0 et A[i] > cle faire
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = cle
```

coût	fois
c_4	n
c_5	$n-1$
c_6	$n-1$

Mesurer le temps d'exécution - tri par insertion

Algorithme : Tri par insertion

Données : A : un tableau de n nombres à trier
Résultat : A : le tableau trié

```
i : entier
j : entier
cle : élément du tableau
pour j ← 2 à A.longueur faire
    cle = A[j]
    i = j − 1
    tant que i > 0 et A[i] > cle faire
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = cle
```

coût	fois
c_4	n
c_5	$n-1$
c_6	$n-1$
c_7	$t_{j=2} + \ldots + t_{j=n}$

Mesurer le temps d'exécution - tri par insertion

Algorithme : Tri par insertion

Données : A : un tableau de n nombres à trier
Résultat : A : le tableau trié

```
i : entier
j : entier
cle : élément du tableau
pour j ← 2 à A.longueur faire
    cle = A[j]
    i = j − 1
    tant que i > 0 et A[i] > cle faire
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = cle
```

coût	fois
c_4	n
c_5	$n-1$
c_6	$n-1$
c_7	$t_{j=2} + \ldots + t_{j=n}$
c_8	$(t_{j=2} - 1) + \ldots + (t_{j=n} - 1)$
c_9	$(t_{j=2} - 1) + \ldots + (t_{j=n} - 1)$

Mesurer le temps d'exécution - tri par insertion

Algorithme : Tri par insertion

Données : A : un tableau de n nombres à trier
Résultat : A : le tableau trié

```
1  i : entier
2  j : entier
3  cle : élément du tableau
4  pour j ← 2 à A.longueur faire
5  |   cle = A[j]
6  |   i = j − 1
7  |   tant que i > 0 et A[i] > cle faire
8  |   |   A[i + 1] = A[i]
9  |   |   i = i − 1
10 |   A[i + 1] = cle
```

coût	fois
c_4	n
c_5	$n-1$
c_6	$n-1$
c_7	$t_{j=2} + \ldots + t_{j=n}$
c_8	$(t_{j=2} - 1) + \ldots + (t_{j=n} - 1)$
c_9	$(t_{j=2} - 1) + \ldots + (t_{j=n} - 1)$
c_{10}	$n-1$

Mesurer le temps d'exécution - tri par insertion

Algorithme : Tri par insertion

Données : A : un tableau de n nombres à trier
Résultat : A : le tableau trié

i : entier
j : entier
cle : élément du tableau
pour $j \leftarrow 2$ **à** $A.longueur$ **faire**
    $cle = A[j]$
    $i = j - 1$
    tant que $i > 0$ ***et*** $A[i] > cle$ **faire**
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = cle$

coût	fois
c_4	n
c_5	$n-1$
c_6	$n-1$
c_7	$t_{j=2} + \ldots + t_{j=n}$
c_8	$(t_{j=2} - 1) + \ldots + (t_{j=n} - 1)$
c_9	$(t_{j=2} - 1) + \ldots + (t_{j=n} - 1)$
c_{10}	$n-1$

Après la somme : $T(n) = a \cdot (t_{j=2} + t_{j=n}) \cdot n^2 + b \cdot n + c$

Mesurer le temps d'exécution - tri par insertion

Cas le plus favorable

- le tableau est déjà trié : `A=[1,2,3,4,5,6]`
- $j = 2..n$ on a $t_j = 1$
- temps d'exécution : $T(n) = a \cdot n - b$

```python
# Tri par insertion

A = [2,1,6,3,4,5]

for j in range(1, len(A)):
    i = j - 1
    cle = A[j]
    while i >= 0 and A[i] > cle:
        A[i+1] = A[i]
        i = i - 1

    A[i+1] = cle
```

Mesurer le temps d'exécution - tri par insertion

Cas le plus favorable

- le tableau est déjà trié : `A=[1,2,3,4,5,6]`
- $j = 2..n$ on a $t_j = 1$
- temps d'exécution : $T(n) = a \cdot n - b$

Cas le plus défavorable

- le tableau est trié dans l'ordre décroissant : `A=[6,5,4,3,2,1]`
- $t_j = j$ pour tout $j = 2..n$
- temps d'exécution : $T(n) = a \cdot n^2 + b \cdot n + c$

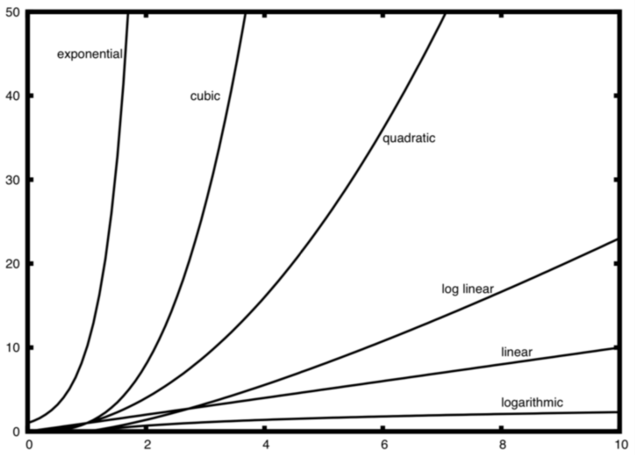
```
# Tri par insertion

A = [2,1,6,3,4,5]

for j in range(1, len(A)):
    i = j - 1
    cle = A[j]
    while i >= 0 and A[i] > cle:
        A[i+1] = A[i]
        i = i - 1

    A[i+1] = cle
```

Mesurer le temps d'exécution



T(n)	Nom
1	Constant
$\log n$	Logarithmique
n	Lineaire
$n \log n$	Log lineaire
n^2	Quadratique
n^3	Cubique
2^n	Exponentielle

Mesurer le temps d'exécution - Révision

T(n)	n = 10	n = 100	n = 1000
1			
$\log n$			
n			
$n \log n$			
n^2			
n^3			
2^n			

Concrètement

Supposons un processeurs que fait 10^6 opérations par seconde (1 MIPS)

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
10^2	$6.6\mu s$	0.1ms	0.6ms	10ms	1s	4×106 ans
10^3	$9.9\mu s$	1 ms	10 ms	1 s	16.6 min	
10^4	$13.3\mu s$	10 ms	0.1 s	1.6 min	11.6 j	
10^5	$16.6\mu s$	0.1 s	1.6 s	2.7 h	317 ans	
10^6	$19.9\mu s$	1 s	19.9 s	11.6 j	106 ans	
10^7	$23.3\mu s$	10 s	3.9 min	3.17 ans		
10^8	$26.6\mu s$	1.6 min	44.3 min	317 ans		
10^9	$29.9\mu s$	16.6min	8.3 h	31709 ans		

PyCharm - semaine_1

exercice_3.py

PyCharm - Exercice 3

Objective : voir en pratique le temps d'exécution pour les différentes complexités algorithmiques

1. Pour chaque fonction non implémentée dans le fichier `exercice_3.py`, l'implémenter de manière à respecter en fonction de la taille d'entrée n la complexité respective
   1.1 `comp_constant`
   1.2 `comp_lineaire`
   1.3 `comp_quadratique`
   1.4 `comp_cubique`
   - L'algorithme n'a pas à résoudre un problème
2. Augmenter la taille du paramètre d'entrée n ($5\times$, $20\times$, $50\times$, etc.) et relancer le programme
3. Quel est la complexité de la fonction `comp_x` ?

Quel langage ?

Langage de définition

Pseudo-code : langage algorithmique

Langage d'implementation

- Python
- Java
- C++
- R
- Pascal
- Perl
- Bash
- ...

Quel langage ?

Langage de définition

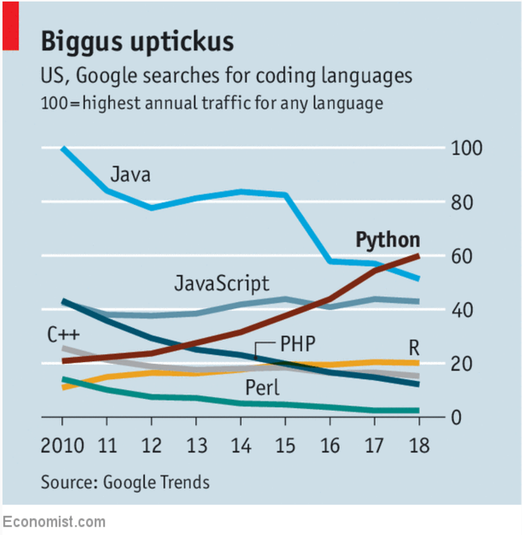
Pseudo-code : langage algorithmique

Langage d'implementation

- Python
- Java
- C++
- R
- Pascal
- Perl
- Bash
- ...

Pourquoi Python ?

Market share - relevance pour le marché de travail



Pourquoi Python ?

Simplicité et proximité du pseudo-code (*executable pseudo-code*)

```
def insertion_sort(A):
  for i in range(1, len(A)):
    key = A[i]
    j = i-1
    while j >=0 and key < A[j]:
      A[j+1] = A[j]
      j = j - 1
    A[j+1] = key

# Driver code to test
A = [68,5,120,35,90,13,6,47,1]
insertion_sort(A)
print("Sorted array is:")
print(A)
```

Pourquoi Python ? Comparaison avec Java

```
class InsertionSort {
  void sort(int[] A) {
    int n = A.length;
    for (int i=1; i<n; ++i) {
      int key = A[i];
      int j = i-1;
      while (j>=0 && A[j] > key) {
        A[j+1] = A[j];
        j = j-1;
      }
      A[j+1] = key;
    }
  }
  static void printArray(int[] A) {
    int n = A.length;
    System.out.println("Sorted array is:")
    for (int i=0; i<n; ++i) {
      System.out.print(A[i] + " ");
    }
  }
```

Pourquoi Python ? Comparaison avec Java

```java
class InsertionSort {
  void sort(int[] A) {
    int n = A.length;
    for (int i=1; i<n; ++i) {
      int key = A[i];
      int j = i-1;
      while (j>=0 && A[j] > key) {
        A[j+1] = A[j];
        j = j-1;
      }
      A[j+1] = key;
    }
  }
  static void printArray(int[] A) {
    int n = A.length;
    System.out.println("Sorted array is:")
    for (int i=0; i<n; ++i) {
      System.out.print(A[i] + " ");
    }
  }
```

```java
  // Driver method
  public static void main(String[] args) {
    int[] A = {12, 11, 13, 5, 6};
    InsertionSort ob = new InsertionSort();
    ob.sort(A);
    printArray(A);
  }
}
```

Pourquoi faire

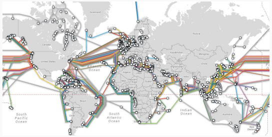
« *Algorithms* + *Data Structures* = *Programs* » **Niklaus Wirth**

« *I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code.* ***Good programmers worry about data structures*** *and their relationships.* » **Linus Torvalds** (creator of Linux)

Applications pratiques de l'algorithmique

Exemple

- Séquençage du génome humain
  - Plus longue sous-chaîne commune
  - Recherche d'un sous-chaîne
- Internet
  - Chemin le plus court
  - Distance entre les documents
- Commerce électronique
  - Cryptographie
- Finance
  - Blockchain : recherche dans une liste chaînée
  - Arbre de recherche

Objectifs du cours d'algorithmique avancée

- Concevoir et implementer des algorithmes de complexité moyenne et avancée
  - Algorithmes fondamentaux et appliqués
- Analyser la performance d'un algorithme : notion de complexité en temps et en espace)
- Maîtriser des algorithmes pour les structures de données classiques
  - séquentiels : tableaux, tableaux triés, listes chaînées
  - associatifs : tables de hachage
  - Graphes et arbres

Référence

Algorithmes Notions de base : Pages 1 - 9 Cormen, http://hesge.scholarvox.com

Cyberlearn : 18_HES-SO-GE_633-1 ALGORITHMES ET STRUCTURES DES DONNÉES

http://cyberlearn.hes-so.ch