ALGO1 🚀

Bienvenue au cours d'algorithmique 😃 !

↪ 📝 Remplissez le sondage svp 🙏

Objectifs du cours

- Étude de l'**algorithmique** = science des algorithmes
  Étude **théorique** et **empirique**
- *Pas* une étude de la calculabilité (cf. autre cours)
- Maîtriser différents axes :
  - types abstraits & structures de données
  - des algorithmes "importants" et typiques
  - paradigmes de conception d'algorithmes
  - spécifications : correction…
  - garanties : terminaison, complexités (temps & mémoire)…
- **Algorithmes** : en pseudo-code et implémentations en Python 

Organisation du cours

- 📜 Page web du cours
  `perso.crans.org/besson/teach/info1_algo1_2019`
- 10 séances de cours + 10 séances de TD
  (planning sur ADE)
- Votre travail :
  - attentif·ve·s en cours 📝
  - actif·ve·s en TD 🏋️
  - lire les implémentations proposées pour chaque cours
  - finir les TD à la maison 🏠
  - un DM & un DS 📝
  - lire les références 📚
- ⚠️ Présence obligatoire aux CM et TD !

Organisation des *Cours Magistraux*

- 👋 *Par* : Lilian Besson ↪ `Lilian.Besson @ Inria.fr` ✉️
- 🏛️ *Lieu* : ENS de Rennes, salle 07 ou autre (à voir sur ADE)
- 📅 *Date* : les mardis
  - 14h - 16h généralement
  - ou 13h30 - 15h30 *quand il y a un séminaire du département* (17 sept., 01 & 15 oct., 05 & 12 & 26 nov.)
    ⚠️ Présence obligatoire aux séminaires (en amphi)
- ⏰ *Durée* : 2 heures, 5-10 minutes de pause au milieu

- Support de cours *pas mis en ligne* : ⚠️ prenez des notes !

Organisation des *Travaux Dirigés*

- 👋 *Par* : Raphaël Truffet ↪ `Raphael.Truffet @ IRISA.fr` ✉️
- 🏢 *Lieu* : Beaulieu (salles à voir sur ADE)
- 📅 *Date* : les jeudis, 16h15 - 18h15
- ⏰ *Durée* : 2 heures, 5-10 minutes de pause au milieu

- Feuille de TD distribuée pendant le TD
- A terminer pour le TD suivant

🎓 Évaluations

1. ⚠️ **Présence obligatoire à tous les cours *et* les TD** ⚠️

2. Un DM écrit $\Longrightarrow$ **1/3 de la note**
   - ⚠️ seul (pas de travail de groupe)
   - sujet donné mi octobre, à rendre début novembre (après Toussaint)
   - des preuves à rédiger sur papier libre (ou en LaTeX/…)
   - un peu de code à écrire (Python / OCaml)

3. Un DS écrit $\Longrightarrow$ **2/3 de la note**
   - pendant les examens mi décembre
   - des preuves à rédiger sur papier libre (ou en LaTeX)
   - un peu de code à écrire (pseudo-code, ou Python / OCaml)

📚 Des références bibliographiques ?

- **Introduction à l'algorithmique**, Cormen et al ; Dunod.
- **Éléments d'algorithmique**, Beauquier, Berstel, Chrétienne ; Masson.
  (en libre accès sur Internet !)
- **Types de données et algorithmes**, Froidevaux et al.
- **Programmation efficace : Les 128 algorithmes qu'il faut avoir compris et codés dans sa vie**, Vie & Dürr ; Ellispes ; `TryAlgo.org`
- 🇬🇧/🇺🇸 Cours diffusés librement sur Internet, en anglais :
  - `Algorithms.wtf`
  - `OpenDataStructures.org`
  - et plein d'autres…

Définitions… 📝 à définir ensemble

- Problème de calcul…

- Algorithmes…

- Types de données abstrait *vs* implémentation…

- Mesures de performance…

- Temps de calcul
- Mémoire
- Mais aussi : batterie, bande passante, nombre lecture mémoire etc…

💻 Illustrations des exemples en cours ?

→ **en pseudo code (au tableau)**

→ **en Python 🐍 (sur l'écran 🎦)**

📝 Illustrations des exemples en TD/exam ?

→ **en pseudo code**

→ **en Python 🐍 *ou* en OCaml 🐫**

Rappels sur les types de bases

Domaine D, représentation, taille de stockage, etc

- Booléen : $D = \{\mathtt{true}, \mathtt{false}\}$, 1 bit
- Entiers : $D = \mathbb{Z}$ ou $D = \mathbb{N}$, 32 ou 64 bits

  ⚠ généralement non exacts
- Flottants : $D = \mathbb{R}$, 32 ou 64 bits

  ⚠ généralement non exacts
- Chaînes de caractères, selon l'alphabet Σ
  - sur l'alphabet $\Sigma = \mathrm{ASCII}$, 7 ou 8 bits par caractères
  - $\Sigma = \mathrm{Unicode}$, 2 à 4 octets par caractères (octet = byte = 8 bit)
  - $D = \Sigma^*$

⚠ Représentations non exactes… des entiers

- En C, C++, Javascript, OCaml : les entiers bouclent !
- Les opérations de bases ne sont pas associatives ! Et pas exactes !

```
# 4611686018427387904 = (-4611686018427387904);;
- : bool = true
# (4611686018427387900 + 10) - 10 = 4611686018427387900;;
- : bool = false
# 4611686018427387900 + (10 - 10) = 4611686018427387900;;
- : bool = false
```

- ⚠ on va ignorer tout cela dans les algorithmes étudiés en cours.
- On suppose pouvoir représenter les entiers $\mathbb{Z}$ en $\mathcal{O}(1)$ temps et effectuer des opérations dessus en temps constant $\mathcal{O}(1)$

⚠️ Représentations non exactes… des flottants

- En C, C++, Javascript, OCaml, Python : les flottants utilisent la norme IEEE 754, qui donne plein d'erreurs possibles 💥
- Les opérations de bases ne sont pas associatives ! Et pas exactes !
- Les tests d'égalités sur les flottants ne sont pas "fiables" !

```
>>> (0.1 + 0.1 + 0.1) == 0.3
False
>>> (0.1 + 0.1 + 0.1) - 0.1 == 0.2
False
>>> 0.1 + 0.1 + (0.1 - 0.1) == 0.2
True
```

- ⚠️ on va ignorer tout cela dans les algorithmes étudiés en cours.
- On suppose pouvoir représenter les nombre décimaux $\mathbb{D}$ en $\mathcal{O}(1)$ temps et effectuer des opérations dessus en temps constant $\mathcal{O}(1)$

Rappels : structures "linéaires" de données

- Espace mémoire $\propto n$
- Taille *fixée*, `n = longueur(T)` ou $n = |T|$, calcul en $\mathcal{O}(1)$
- Contient des données généralement d'un même type
  - obligatoire en OCaml (typage statique)
    types `'a array`, `'a list`, etc (`'a arbre`)
  - aucune limitation en Python (typage dynamique)
- En pratique :
  - **tableaux statiques**, **tableaux dynamiques**
  - **listes simplement** ou **listes doublement chaînées**
  - **file**, **pile**, **file de priorité** et d'autres

Tableau (statique)

- 💥 Construction avec n valeurs en $\mathcal{O}(n)$ (taille connue à l'avance)
- 🚀 Accès au `i` ème élément en $\mathcal{O}(1)$

📝 En pseudo-code

Initialisation : $T_{1:n}$, accès T_i ou $T[i]$, écriture $T[i] \leftarrow x$

🐫 En OCaml : accès `t.(i)` , écriture t.(i) <- x

```
# let t = [| 0; 1; 2; 3 |];;
val t : int array = [|0; 1; 2; 3|]
```

🐍 En Python : accès `t[i]` , écriture `t[i] = x`

```
>>> tableau = [ 0, 1, 2, 3 ]
```

Liste simplement chaînée

- Taille *dynamique*
- 🚀 Construction initiale en $\mathcal{O}(1)$
- 🚀 Ajout de 1 élément en tête de liste en $\mathcal{O}(1)$
  $\Longrightarrow$ 💥 Ajout de n éléments en $\mathcal{O}(n)$!
- 🚀 Accès au `i` ème élément en $\mathcal{O}(i)$

🐫 **En OCaml : tête `hd l`, queue `tl l`**

```
# let l = [ "tete"; "suite"; "de"; "la"; "liste" ];;
val l : string list = ["tete"; "suite"; "de"; "la"; "liste"]
```

🐍 ⚠️ **En Python n'existe pas ! Le type `list` est un mélange entre les tableaux et les listes… (tableau dynamique)** ⚠️

Liste simplement chaînée

En Python

↪ implémentation "manuelle" avec deux petites classes

- Cf. notebook Python sur
  `perso.crans.org/besson/info1_algo1_2019/notebooks/`

Liste *doublement* chaînée

🐫 En OCaml

- Avec un type paramétrique
- Vous pouvez essayer à la maison

🐍 En Python

↪ implémentation "manuelle" avec deux petites classes

- Cf. notebook Python sur
  `perso.crans.org/besson/info1_algo1_2019/notebooks/`

Étude du tri par file de priorité (tri par tas)

1. File de priorité ?

2. Implémentation par tas binaire

3. Tri par tas

Dans ce cours : des algorithmes…

… expliqués et prouvés au tableau

… illustrés sur des exemples

↪ vous pouvez revoir les exemples à la maison sur
`www.cs.usfca.edu/~galles/visualization/Algorithms.html`

… codés en Python 3 🐍

- Montrés dans des *notebooks Jupyter* (cf. `Jupyter.org`)
  - Disponibles sur la page du cours (`/notebooks`)
  - et sur `GitHub.com/Naereen/ALGO1-Info1-2019/`
- Des fois de façon interactive avec *Python Tutor* (cf. `PythonTutor.com`)
- Étude empirique de complexités temps/mémoire etc.

Fin du cours 1/10

Merci de votre attention.