

## But de ces séances

**PRÉREQUIS** : avoir vu la vidéo

### Buts :

- ▶ Améliorer/renforcer votre apprentissage
  - ▶ Répondre à vos questions
  - ▶ Approfondir des sujets (à votre demande)
- ☞ Vous faire **gagner du temps** de révision / de mise en pratique sur les exercices

### Déroulement

- ▶ Qu'avez-vous retenu ? / Points importants (rapide car pas le but de redire le cours)
- ▶ Questions / approfondissements / points difficiles
- ▶ Propositions d'« études de cas »
- ▶ Pratique libre (exercices) + réponse (libre) aux questions (= commencer ensemble la série d'exercices)

## Les conseils de Valentine (PH-BA3)

- ▶ Adaptez votre méthode de travail
  - ☞ Repérer les parties du cours **adaptées à votre niveau et à votre manière de travailler**
- ▶ Regarder les examens **au plus tôt**, réviser avec les anciens examens
  - ☞ Les series ne sont pas suffisantes pour tout le monde pour être prêts aux examens
- ▶ Garder en tête que **le but est de comprendre et d'acquérir des compétences** et non de terminer les séries
- ▶ Faire des **choix d'organisation et planifier ses priorités** surtout les semaines de midterm ou de rendu

## Leçon I.1b (algorithmes, complexité) – Points clés

2 grandes parties :

- ① comment écrire un algorithme
- ② complexité d'un algorithme

## Leçon I.1b (algorithmes, complexité) – Points clés ①

### ① comment écrire un algorithme :

#### ▶ « méthodologie » :

1. **comprendre** le problème : question ? entrée(s) ? sortie(s) ?  
(écriture de l'algorithme : ne pas oublier de bien spécifier l'/les entrée(s) et la/les sortie(s))
2. trouver *un* algorithme correct (vérifier qu'il est bien correct ; penser à vérifier les cas limites)
3. trouver un algorithme *efficace*  
☞ savoir calculer la complexité d'un algorithme

#### ▶ « boîte à outils » :

- ▶ affectation ( $x \leftarrow 18$ ) et autres opérations usuelles
  - ▶ 3 structures de contrôle : branchements, boucles, itérations
  - ▶ mot-clé « **Sortir :** » pour terminer et indiquer quelle est la sortie de l'algorithme
  - ▶ *appel* d'autres (sous-)algorithmes
- ▶ trois problèmes « clés » : recherche, tri, plus court chemin

## Leçon I.1b (algorithmes, complexité) – Points clés ①

### ① comment écrire un algorithme :

#### ▶ « méthodologie » :

1. **comprendre** le problème : question ? entrée(s) ? sortie(s) ?  
(écriture de l'algorithme : ne pas oublier de bien spécifier l'/les entrée(s) et la/les sortie(s))
2. trouver *un* algorithme correct (vérifier qu'il est bien correct ; penser à vérifier les cas limites)
3. trouver un algorithme *efficace*  
☞ savoir calculer la complexité d'un algorithme

#### ▶ « boîte à outils » :

- ▶ affectation ( $x \leftarrow 18$ ) et autres opérations usuelles
- ▶ 3 structures de contrôle : branchements, boucles, itérations
- ▶ mot-clé « **Sortir :** » pour terminer et indiquer quelle est la sortie de l'algorithme
- ▶ *appel* d'autres (sous-)algorithmes

#### ▶ trois problèmes « clés » : recherche, tri, plus court chemin

*taille(L)*

## Leçon I.1b (algorithmes, complexité) – Points clés ②

### ② complexité d'un algorithme :

- ▶ est une fonction de la taille de l'entrée
- ▶ compte le nombre d'opérations élémentaires nécessaires

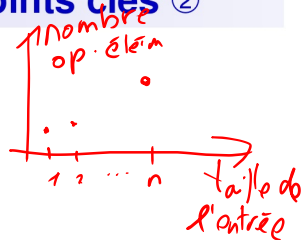
▶ dans le pire des cas ← d'avoir identifié

- ▶ on s'intéresse à son comportement à l'infini (c'est-à-dire quand la taille de l'entrée augmente) :

on utilise donc une notation d'ensemble :  $\Theta(f(n))$  :

ensemble des fonctions qui « croissent à l'infini comme  $f(n)$  »

- ▶ meilleure complexité des algorithmes résolvant les trois problèmes « clés » :
  - ▶ recherche : liste ordonnée :  $\Theta(\log n)$  ; liste quelconque :  $\Theta(n)$
  - ▶ tri :  $\Theta(n \log n)$
  - ▶ plus court chemin : ? (semaine prochaine)



$$f \in \Theta(g) \Leftrightarrow \begin{cases} f \in O(g) \\ \text{et} \\ g \in O(f) \end{cases}$$

## Leçon I.1b (algorithmes, complexité) – Question(s)

QUESTIONS?

$$\Theta(\log(n))$$

Rappel semaine passée :

1. Quelle est la meilleure complexité ?

A.  $\Theta(\log(n))$

B.  $\Theta(n)$

C.  $\Theta(n \cdot \log(n))$

D.  $\Theta(n^2)$

La *meilleure* complexité est la plus petite :  
on préfère un algorithme qui prend *moins* de temps

Question fréquente : *dans quelle base est le log ?*

$$\log_b(n) = \frac{\log_a(n)}{\log_a(b)}$$

## Trouver tous les éléments maximaux dans une liste

(3, 5, 18, 18)

(3, 18, 4, 18, 5)

### Éléments maximaux

entrée :  $L$  une liste (non vide ??) de nombres

sortie : (liste des positions des) éléments maximaux de la liste

$n \leftarrow \text{taille}(L)$

$L \leftarrow \text{tri}(L)$

$L' \leftarrow (n)$  // liste à 1 seul élément

Tant que  $n > 1$  et  $L(n-1) = L(n)$

|  $n \leftarrow n-1$

|  $L' \leftarrow L' \oplus (n)$  // ajout de la valeur  $n$  à la liste

Sortir :  $L'$

la Valeur max : 18

les éléments max :  
le 2<sup>ie</sup> et le 4<sup>e</sup>

(2, 4)

# Trouver tous les éléments maximaux dans une liste

pire cas : (18, 18, 18, 18)

## Éléments maximaux

entrée :  $L$  une liste (non vide ??) de nombres

sortie : (liste des positions des) éléments maximaux de la liste

$n$  ← **taille**( $L$ ) → ? 1 ou  $n$   
 $L$  ← **tri**( $L$ ) →  $n \log n$   
 $L' \leftarrow (n)$  // liste à 1 seul élément → 1

Tant que  $n > 1$  et  $L(n-1) = L(n)$

$n \leftarrow n-1$   
 $L' \leftarrow L' \oplus (n)$  // ajout de la valeur  $n$  à la liste

Sortir :  $L'$

Complexité  $\Theta(n \log n)$

Peut-on faire mieux ?

?  
nb fois  $\times \Theta(1)$

pire cas :  $n$  fois

→  $\Theta(n)$

$$\Theta(1) + \Theta(n \log n) + \Theta(n) \rightarrow \Theta(n \log n)$$

(2,4)



# Trouver tous les éléments maximaux dans une liste

$L : (3, 18, 1, 18, 14) \xrightarrow{\text{tri}} (1, 3, 14, 18, 18)$

Éléments maximaux	
entrée : $L$ une liste (non vide ??) de nombres	
sortie : (liste des positions des) éléments maximaux de la liste	
$n \leftarrow \text{taille}(L)$	$n$
$L \leftarrow \text{tri}(L)$	$5$
$L' \leftarrow (n)$ // liste à 1 seul élément	$(5)$
<b>Tant que</b> $n > 1$ <b>et</b> $L(n-1) = L(n)$	
$n \leftarrow n-1$	$4$
$L' \leftarrow L' \oplus (n)$ // ajout de la valeur $n$ à la liste	$(5, 4)$
<b>Sortir</b> : $L'$	

(5,4)

Complexité ?

☞  $\Theta(n \log n)$ , mais...  
...il est FAUX!!

Peut-on faire mieux ?

# Trouver tous les éléments maximaux dans une liste

## Éléments maximaux

entrée :  $L$  une liste (non vide ??) de nombres

sortie : (liste des positions des) éléments maximaux de la liste

$n \leftarrow \text{taille}(L)$

$L \leftarrow \text{tri}(L)$

$L' \leftarrow (n)$  // liste à 1 seul élément

**Tant que**  $n > 1$  **et**  $L(n-1) = L(n)$

|  $n \leftarrow n - 1$

|  $L' \leftarrow L' \oplus (n)$  // ajout de la valeur  $n$  à la liste

**Sortir** :  $L'$

Complexité ?

☞  $\Theta(n \log n)$ , mais...  
...il est FAUX!!

Peut-on faire mieux ?

☞ **oui**  
non seulement *juste*  
mais aussi  
à moindre coût

## Trouver tous les éléments maximaux dans une liste

### Éléments maximaux v2

entrée :  $L$  une liste (non vide ??) de nombres

sortie : (liste des positions des) éléments maximaux de la liste

$n \leftarrow \text{taille}(L)$

$L' \leftarrow ()$  // liste vide

$x_{\max} \leftarrow -\infty$

Pour  $i$  allant de 1 à  $n$

Si  $L(i) > x_{\max}$

$x_{\max} \leftarrow L(i)$

$L' \leftarrow (i)$

// liste à 1 seul élément

Sinon, si  $L(i) = x_{\max}$

$L' \leftarrow L' \oplus (i)$

// ajout de la valeur  $i$  à la liste

Sortir :  $L'$

Si  $n = 0$   
Sortir ()  
 $x_{\max} \leftarrow L(1)$

(érase)

désigne l'élément

# Trouver tous les éléments maximaux dans une liste

## Éléments maximaux v2

entrée :  $L$  une liste (non vide ??) de nombres

sortie : (liste des positions des) éléments maximaux de la liste

$n \leftarrow \text{taille}(L)$  → 1 sur  $n$   
 $L' \leftarrow ()$  ← // liste vide 1  
 $x_{\max} \leftarrow -\infty$

Pour  $i$  allant de 1 à  $n$

**Si**  $L(i) > x_{\max}$

$x_{\max} \leftarrow L(i)$

$L' \leftarrow (i)$

// liste à 1 seul élément

**Sinon, si**  $L(i) = x_{\max}$

$L' \leftarrow L' \oplus (i)$

// ajout de la valeur  $i$  à la liste →  $\Theta(1)$

Sortir :  $L'$

Complexité ?

$n$  fois  $\Theta(n)$  →  $\Theta(n)$

Peut-on faire mieux ?

# Trouver tous les éléments maximaux dans une liste

## Éléments maximaux v2

entrée :  $L$  une liste (non vide ??) de nombres

sortie : (liste des positions des) éléments maximaux de la liste

$n \leftarrow \text{taille}(L)$

$L' \leftarrow ()$  // liste vide

$x_{\max} \leftarrow -\infty$

**Pour**  $i$  allant de 1 à  $n$

**Si**  $L(i) > x_{\max}$

$x_{\max} \leftarrow L(i)$

$L' \leftarrow (i)$  // liste à 1 seul élément

**Sinon, si**  $L(i) = x_{\max}$

$L' \leftarrow L' \oplus (i)$  // ajout de la valeur  $i$  à la liste

**Sortir** :  $L'$

Complexité ?

☞  $\Theta(n)$

Peut-on faire mieux ?

☞ **non**

## Questions fréquentes

toutes sont  
op. élémentaires

- ▶ Comment ajouter un élément à une liste ?

$L \leftarrow L \oplus (x)$  // ajout en fin

$L \leftarrow (x) \oplus L$  // ajout au début

- ▶ Comment ajouter un élément  $x$  à une position  $i$  dans une liste ?

$L \leftarrow (L[1], \dots, L[i-1], x, L[i], \dots, L[n])$

( $n$  : taille de  $L$ )

- ▶ Comment vider une liste ?

$L \leftarrow ()$  // affectation de la liste vide

- ▶ Comment supprimer un élément d'une liste ?

$L \leftarrow (L[1], \dots, L[i-1], L[i+1], \dots, L[n])$

(où la taille de la liste aura préalablement été affecté à  $n$ )

à la position  $i$

## Leçon I.1b (algorithmes, complexité) – Recherche

Recherche dans une liste : qu'est-ce qui est le mieux ?

A] faire une recherche linéaire

B] commencer par trier la liste, puis faire une recherche par dichotomie

$$\Theta(n)$$

$$\Theta(n \log n)$$

ensuite  $\Theta(\log n)$

## Leçon I.1b (algorithmes, complexité) – Recherche

Recherche dans une liste : qu'est-ce qui est le mieux ?

A] faire une recherche linéaire

B] commencer par trier la liste, puis faire une recherche par dichotomie

Cela dépend de ce que l'on veut :

soit  $K$  le nombre de fois que l'on fait la recherche,  
et soit  $n$  taille de la liste ;

A :  $K \cdot f(n)$ , avec  $f \in \Theta(n)$

B :  $g(n) + K \cdot h(n)$ , avec  $g \in \Theta(n \log n)$  et  $h \in \Theta(\log n)$