C’est quoi le bleu ?

		MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 8-10	cours prog. 45 min. Jeudi 10-11	cours théorie 90 min. Vendredi 13-14	cours théorie 90 min. Vendredi 14-15	exercices théorie 45 min. Vendredi 15-16	
1	11.09.25	--	-1	prise en main	Bienvenue/Introduction	Introduction + Algo 1		Algo 1	12.09.25
2	18.09.25	1. variables	0	variables / expressions	variables / expressions	Algorithmes 1 (suite)		Algo 1 suite	19.09.25
3	25.09.25	2. if	0	if – switch	if – switch	Algo 1	Algo 2 (stratégies)	Algo 2	26.09.25
4	02.10.25	3. for/while	0	for / while	for / while	Algo 2 (stratégies)	Calculabilité	Calculabilité	03.10.25
5	09.10.25	4. fonctions	0	fonctions (1)	fonctions (1)	Calculabilité	Représentations numériques	Représentations numériques	10.10.25
6	16.10.25		1	fonctions (2)	fonctions (2)	Représentations numériques	Signaux + Filtrage	Révisions	17.10.25
-	23.10.25								
7	30.10.25	5. tableaux (vector)	1	vector	vector	Examen 1 (1h45)			31.10.25
8	06.11.25	6. string + struct	1	array / string	array / string	Th. d’échantillonnage		Signaux–Echantillonnage	07.11.25
9	13.11.25		2	structures	structures	Signaux–Echantillonnage	Compression 1	Compression 1	14.11.25
10	20.11.25	7. pointeurs	2	pointeurs	pointeurs	Compression 1	Compression 2	Compression 2	21.11.25
11	27.11.25		-	entrées/sorties	entrées/sorties	Compression 2	Architecture des ordinateurs	Architecture des ordinateurs	28.11.25
12	04.12.25		-	erreurs / exceptions	erreurs / exceptions	Architecture des ordinateurs	Stockage/Réseaux	Stockage/Réseaux	05.12.25
13	11.12.25		-	révisions	théorie : sécurité	Stockage/Réseaux	Sécurité	Révisions	12.12.25
14	18.12.25	8. étude de cas	-	révisions	Révisions	Examen final (2h45)			19.12.25
					(ne sont pas sur le MOOC)	(prép. examen)	(« classe inversée » : rép. questions + compléments)		

Examen 1 2018 Q11

Quelle est la sortie de l'algorithme suivant sur l'entrée $L = (-3,5,12,-4,3,8,-1,-6,4)$:

algo11

entrée : *L liste de valeurs*
sortie : *???*

$a \longleftarrow$ **taille**(L)
$R \longleftarrow \emptyset$ // Liste vide
Si $a \geq 3$
  Pour i de 1 à $a-2$
    Pour j de $i+1$ à $a-1$
      Pour k de $j+1$ à a
        Si $L[i]+L[j]+L[k]=0$
          $R \longleftarrow (i,j,k)$
Sortir : R

Sortir R

$R \leftarrow R \oplus (i,j,k)$

$R \leftarrow (L[i], L[j], L[k])$

A] $\emptyset$ (liste vide)

B] $(2,4,7)$

C] $(5,-4,-1)$

D] $(1,7,9)$

E] $(1,7,9,2,4,7)$

F] $(7,4,2)$

Examen 1 2018 Q11

Quelle est la sortie de l'algorithme suivant sur l'entrée $L = (-3,5,12,-4,3,8,-1,-6,4)$:

algo11

entrée : *L liste de valeurs*
sortie : *???*

```
a ⟵ taille(L)
R ⟵ ∅        // Liste vide
Si a ≥ 3
    Pour i de 1 à a−2
        Pour j de i+1 à a−1
            Pour k de j+1 à a
                Si L[i]+L[j]+L[k] = 0
                    R ⟵ (i,j,k)
Sortir : R
```

A] ∅ (liste vide)
***B]** (2,4,7)
C] (5,−4,−1)
D] (1,7,9)
E] (1,7,9,2,4,7)
F] (7,4,2)

n^2 1

n^3 plein

n^4 0

2^n 0

Examen 1 2018 Q12

Si l'on note n la taille de la liste L, quelle est la complexité de l'algorithme de la question précédente ?

A] $\Theta(n^3)$
B] $\Theta(2^n)$
C] $\Theta(n^2)$
D] $\Theta(n^4)$

Examen 1 2018 Q11

algo11

entrée : *L liste de valeurs*
sortie : *? ? ?*

$a \longleftarrow$ **taille**(L)
$R \longleftarrow \emptyset$ // Liste vide
Si $a \geq 3$
  Pour i de 1 à $a-2$
    Pour j de $i+1$ à $a-1$
      Pour k de $j+1$ à a
        Si $L[i]+L[j]+L[k]=0$
          $R \longleftarrow (i,j,k)$
Sortir : R

n : taille de L

$\Theta(1)$
$\Theta(1)$
$O(n)$

$$\sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n} c + \partial(n) \in \Theta(n^3)$$

? $n-2$ fois
? $i+1 \to n-1$? fois
? $j+1 \to n$ fois
$\Theta(1)$

Examen 1 2018 Q12

Si l'on note n la taille de la liste L, quelle est la complexité de l'algorithme de la question précédente ?

*A] $\Theta(n^3)$
B] $\Theta(2^n)$
C] $\Theta(n^2)$
D] $\Theta(n^4)$

Examen 1 2018 Q13

On s'intéresse ici à raccourcir les répétitions de trois ou plus valeurs identiques successives ; par exemple à produire la liste $(6,6,4,4,12,4,6)$ à partir de la liste $(6,6,4,4,4,12,4,6)$ en supprimant le 4 en cinquième position car il est présent trois fois consécutives.
À noter que :

- les seules valeurs supprimées sont celles qui sont répétées successivement trois fois ou plus (l'une à la suite de l'autre) ; on ne garde alors que deux de ces valeurs (cf la valeur 4 ci-dessus) ;
- toute valeur présente une ou deux fois successivement est préservée, et l'on conserve l'ordre de la liste ;
- en sortie on ne peut donc pas avoir plus de deux valeurs identiques consécutives.

Ecrivez un algorithme itératif (c.-à-d. non récursif, mais avec des boucles) résolvant ce problème.

Examen 1 2018 Q13

Entrée : une liste L ...
Sortie : liste L' ...

n ← taille(L)
L' ← (L[1] L[2])
Pour i = 3 à n
  Si L[i] ≠ L[i-1] OU L[i] ≠ L[i-2]
    L' ← L' ⊕ L[i]

Sortir L'

Ne Pas recopier
SI
L[i] = L[i-1]
et L[i] = L[i-2]

Négation

Attention aux pièges suivants :

- Surtout pas de
  Pour i de 1 à n-2
  Pour j de i+1 à n-1
  Pour k de j+1 à n
  ici : car i, j et k ne sont pas consécutifs

  → contre-exemple : (4, 3, 4, 7, 4, 12, 4, 3, 4, 8, 4)

- Attention à $L[i]=L[j]=L[k]$: sa négation est ambigüe

  est-ce $L[i]\neq L[j]$ ou/et $L[j]\neq L[k]$ ou/et $L[i]\neq L[k]$?

  préférez $L[i]=L[j]$ et $L[i]=L[k]$

- Pour supprimer un élément d'une liste :

  voir semaine passée : $L \leftarrow (L[1], \dots, L[i-1], L[i+1], \dots, L[n])$

  JAMAIS de ∈ (notation ambigüe)

- Ne modifiez JAMAIS une liste pendant son parcours
  - sa taille change
  - les indices de ses éléments changent
  - un « Pour tout ... » se fait de toutes façons (sans remise en question)
    → ne pas confondre avec « Tant que »

→ toutes les chances d'écrire un algorithme faux !

Examen 1 2018 Q14

Déterminez la complexité de votre algorithme.
Justifiez votre réponse.

Leçon I.2 (conception d'algorithmes) – Points clés

- approche descendante : **DÉCOMPOSEZ** le problème
- algorithmes récursifs :
  - ramener le problème à la résolution du *même* problème sur moins de données
  - penser à la condition d'arrêt
- programmation dynamique :
  stocker/mémoriser au lieu de recalculer
- problèmes de plus courts chemins
  complexité polynomimale : $\Theta(n^3)$, $\Theta(n^2)$ ou $\Theta(n)$ en fonction de la nature du problème
  (nombre de villes de départ/d'arrivée fixées)

Leçon I.2 (conception d'algorithmes) – Difficultés connues (1/2)

- **concevoir un algorithme récursif :**
  1. essayer de voir « le schéma qui se répète »
  2. pensez à la/aux condition(s) d'arrêt(s)
  3. si 1 est trop difficile : essayez, de partir d'à peine plus compliqué que 2 (conditions d'arrêt), et d'avancer « d'un cran de plus » pour arriver aux conditions d'arrêt
     pendant quelques pas (pour avoir une idée de 1)
- **méthodes usuelles pour avancer « d'un cran de plus » :**
  - enlever un
  - couper en deux

Leçon I.2 (conception d'algorithmes) – Difficultés connues (2/2)

- **calculer la complexité d'algorithmes** (en particulier récursifs)

  Vous avez trois moyens « pratiques » :

  1. ***Compter* les instructions**
     l'inconvénient est que cela conduit à une équation sur la complexité (fonction),
     qu'il est parfois (souvent ?) difficile de résoudre

  2. ***dessiner* le graphe des appels** depuis la taille n jusqu'à toutes les terminaisons et compter alors le nombre d'arcs (pour le parcourt du pire cas)

     (revoir l'exemple du cours des appels du calcul récursif des coefficients du binôme)

  3. utiliser la méthode « **incrémenter et compter** » :
     de combien augmente la complexité si j'augmente la taille de l'entrée de 1 ?

     ☞ cela donne une estimation de la dérivée de la complexité (fonction)

Leçon I.2 (conception d'algorithmes) – Dichotomie

Écrire complètement l'algorithme de recherche par dichotomie dans une liste *ordonnée* :

- spécifier le problème

entrée : L, x
sortie : $x \in L$? (bool)

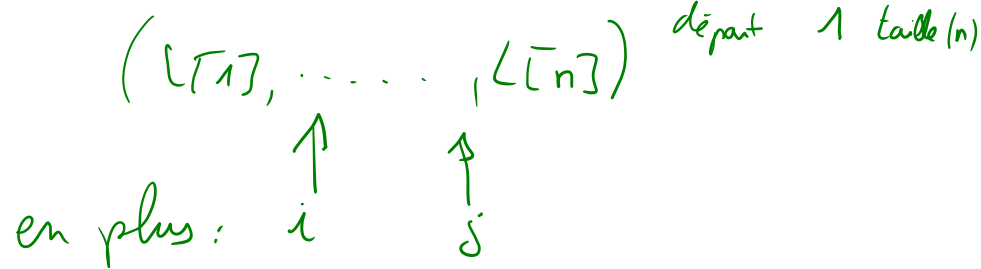
Leçon I.2 (conception d'algorithmes) – Dichotomie

Écrire complètement l'algorithme de recherche par dichotomie dans une liste *ordonnée* :

- spécifier le problème
- « couper la liste en deux » : comment faire ?

entrée : L, x, i, j

Sortie : $x \in L$?



Leçon I.2 (conception d'algorithmes) – Dichotomie

Écrire complètement l'algorithme de recherche par dichotomie dans une liste *ordonnée* :

- spécifier le problème
- « couper la liste en deux » : comment faire ?
  - ☞ je vous impose de passer 2 paramètres supplémentaires en entrée : indice de début de recherche et indice de fin de recherche (inclus)

Dicho

Entrée : L (une liste), x, i (début), j (fin)

Sortie : $x \in (L[i], \dots, L[j])$?

Si $i \geq j$

sortir faux

Si $i = j$

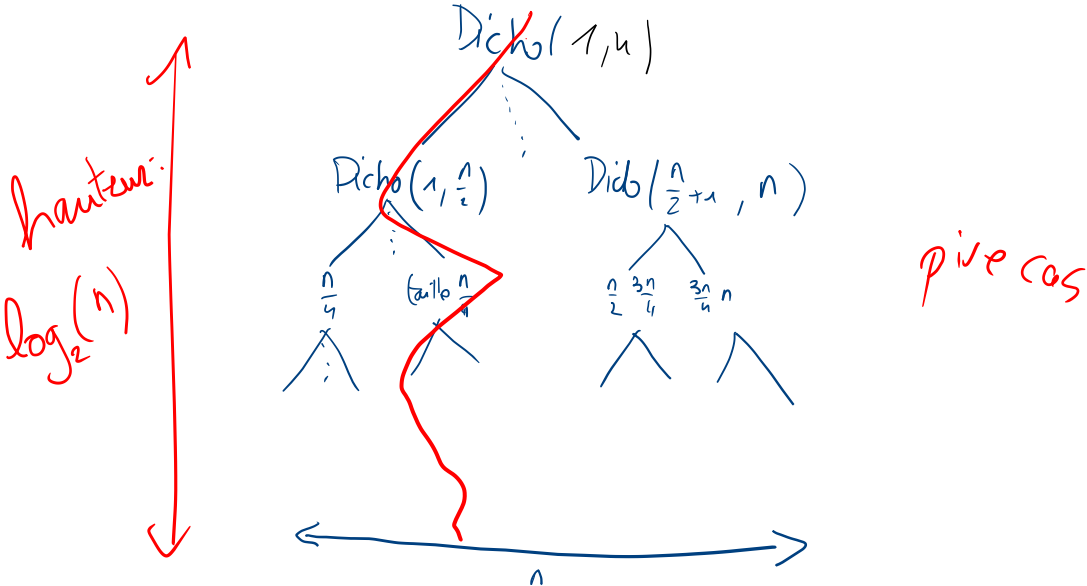
Sortir $L[i] = x$ (bool)

Sinon $k \leftarrow \left\lfloor \frac{i+j}{2} \right\rfloor$

Si $L[k] > x$ sortir Dicho(L, x, i, k)

[Sinon] sortir Dicho(L, x, k, j)

est-ce correct ?

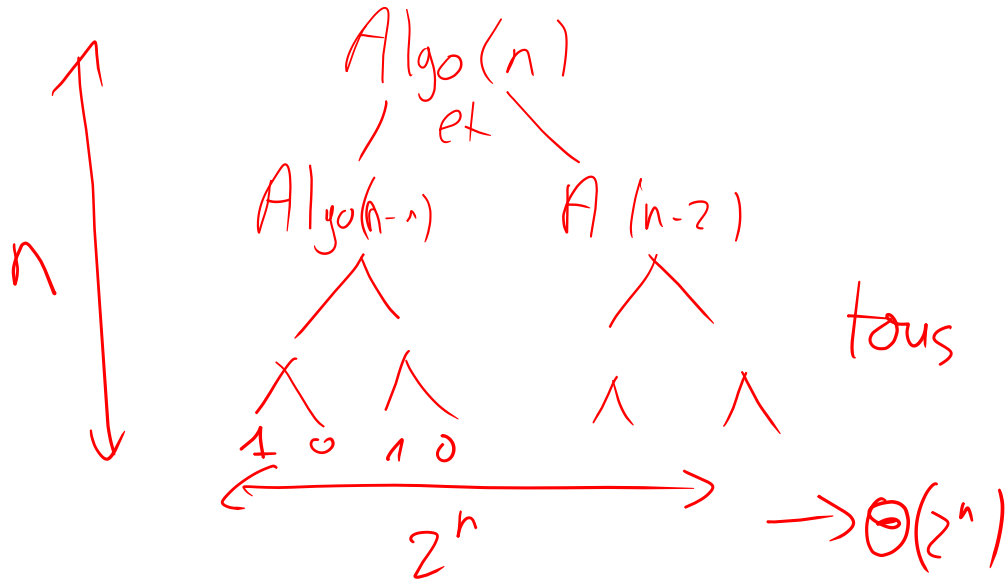

hauteur
$\log_2(n)$
Dicho(1, n)
Dicho(1, $\frac{n}{2}$)
Dicho($\frac{n}{2}+1$, n)
$\frac{n}{4}$
taille $\frac{n}{4}$
$\frac{n}{2}$
$\frac{3n}{4}$
$\frac{3n}{4}$ n
n
pire cas

Leçon I.1b (algorithmes, complexité) – Que fait-il ?

Algo
entrée : *entier naturel n* sortie : *??*
Si $n \leq 1$ **Sortir :** $n+2$ **Sortir :** $2 \cdot$ **Algo**$(n-1) -$ **Algo**$(n-2)$

1. Que calcule cet algorithme ?
2. Quelle est sa complexité ?

n	Algo(n)
0	2
1	3
2	$2 \cdot 3 - 2 = 4$
3	$2 \cdot 4 - 3 = 5$
n	$n+2$


Algo(n)
et
Algo(n-1)
A(n-2)
n
tous
1 0 1 0
2^n
$\rightarrow \Theta(2^n)$