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

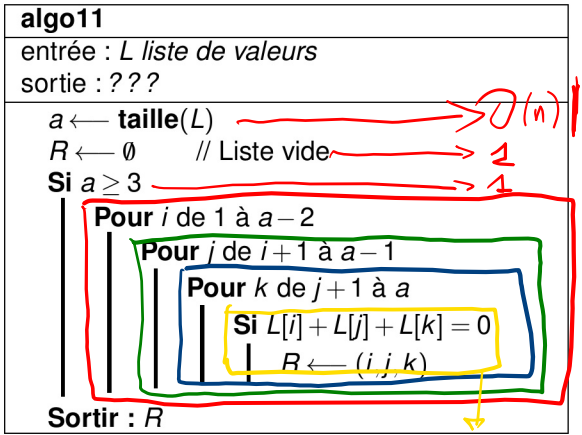
entrée : *L liste de valeurs*
sortie : *???*

```
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)

Examen 1 2018 Q11

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



- **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)

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] $\Theta(n^3)$ bcp bravo !

B] ~~$\Theta(2^n)$~~

C] ~~$\Theta(n^2)$~~

D] $\Theta(n^4)$ 2

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.

Écrivez un algorithme **itératif** (c.-à-d. non récursif, mais avec des boucles) pour résoudre ce problème.

Examen 1 2018 Q13

Entrée : L liste
Sortie : L « simplifiée »

n ← taille(L)
Si n ≤ 2
  Sortir L
L' ← (L[1], L[2])
Pour i de 3 à n
  Si L[i] ≠ L[i-1] ou L[i] ≠ L[i-2]
    L' ← L' ⊕ L[i]
Sortir L'

on ne veut pas
L[i] = L[i-1]
et L[i] = L[i-2]
négation

Examen 1 2018 Q13 – Erreurs fréquentes

- Les valeurs à supprimer doivent être **consécutives**. Ceci est donc **incorrect** :

  Pour i allant de 1 à $n-2$
  - **Pour** j allant de $i+1$ à $n-1$
    - **Pour** k allant de $j+1$ à n
      - ...

  Contre-exemple : (4,3,4,7,4,12,4,3,4,3)

- Pour supprimer le i-ième élément d'une liste (revoir la semaine passée) :
  $L \longleftarrow (L[1], ..., L[i-1], L[i+1], ..., L[n])$

- Ne modifiez **JAMAIS** une liste pendant son parcours : grands riques d'écrire un algorithme faux !
  - la taille de la liste modifiée change ;
  - les indices des ses éléments changent.
  - Notez qu'un « **Pour tout** x dans L » se fait **de toutes façons** comme L était au départ (*même* si L change ; ne confondez pas avec un « **Tant que** ... »)

  ☞ préférez créer une nouvelle liste

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 parcours 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

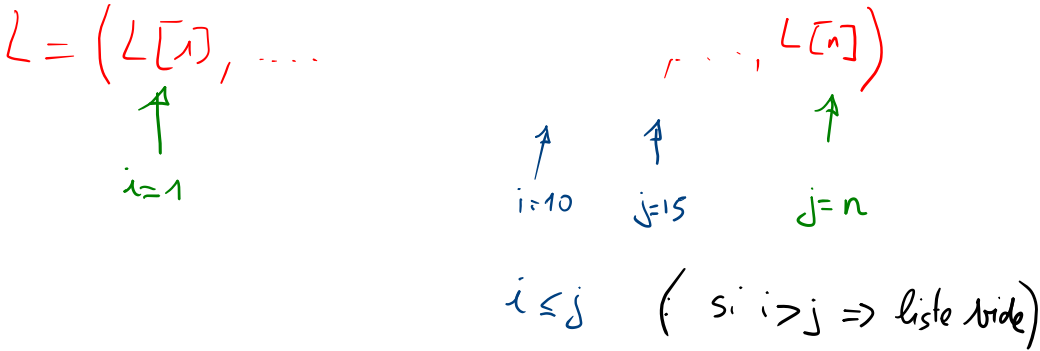
Entrée : L ordonnée, x une valeur

Sortie : oui/non ($x \in L$?)

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 ?



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)

Entrée : L, x, i, j

i : début, j : fin

Dicho

Entrée : L, x, i, j

Sortie : $x \in L$? (oui ou non)

Si $j < i$

  Sortir non

Si $L[i] = x$ ou $L[j] = x$

  Sortir oui

Si $L[\lfloor \frac{i+j}{2} \rfloor] \leq x$

  Sortir Dicho $(L, x, \lfloor \frac{i+j}{2} \rfloor, j)$

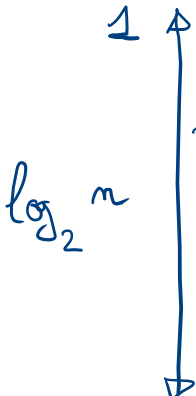
Sortir Dicho $(L, x, i, \lfloor \frac{i+j}{2} \rfloor)$

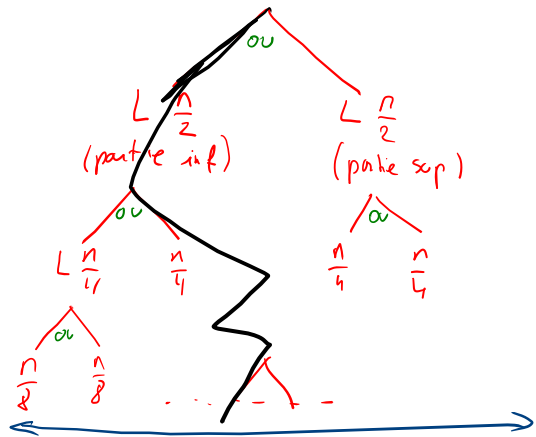
est-ce toujours correct ?

Complexité ?

méthode (1) : compter : cf vidéo

méthode (2) :





Pire cas

= 1 seul chemin

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 ?