LYCEE CHARLEMAGNE

Lundi 9 janvier

M.P.S.I.2

2022 2023

Info00

Lycee Charlemagne | MPSI2 | Annee 2021/22

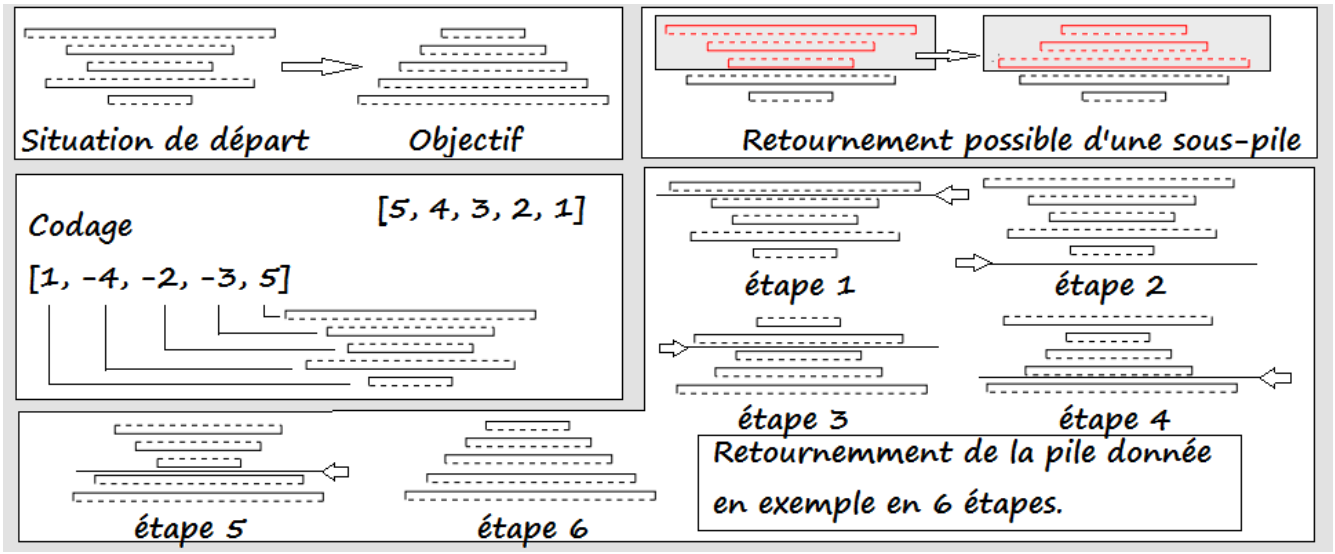
Chandeleur

On se propose d'étudier un algorithme de tri sur des listes, dit « algorithme du crêpier ». Oui, du marchand de crêpes.
Le crêpier fabrique des crêpes qui ont des diamètres tous différents, et de plus chacune a deux faces : une face dorée, et une face blanche, pas assez cuite. Il les a posées dans le désordre, et chacune dans n'importe quel sens.

Son objectif : une belle pile de crêpes, toutes orientées dans le même sens, face blanche vers le haut, de la plus grande à la plus petite.

Ce qu'il a le droit de faire : attraper le dessus du paquet de crêpes, sur une certaine épaisseur, puis retourner ce sous-paquet.
(les n crêpes de ce paquet sont alors changées de sens, et la sous-pile elle même est renversée).

Sur le dessin, on donne un exemple de retournement (optimal ?) d'une pile de cinq crêpes.



Une pile de crêpes sera codée par une liste d'entiers relatifs tous distincts en valeur absolue, de la crêpe du bas à la crêpe du haut. La valeur absolue de l'entier donne son diamètre, et le signe donne son orientation (deux bonnes raisons qu'il n'y ait pas de crêpe codée 0).
La pile finale devra donc être faite d'entiers positifs, du plus grand au plus petit.

I∼0) Donnez le codage de chacune des six piles de l'exemple représenté dans le cadre ci dessus. 1 pt.

I∼1) Écrivez une procédure `Melange` qui prend en entrée un entier n et retourne une pile de n crêpes, désordonnée. 3 pt.
Exemple : `Melange(5)` pourra donner `[1, -4, -2, -3, 5]`.
Rappel : le module `randrange` contient les fonctions `randrange`, `random` et `shuffle`.

`shuffle(L)` mélange une liste `L` : exemple
```
L = [3, 6, 4, 2, 0]
shuffle(L)
print(L) : [0, 3, 2, 4, 6]
```
`randrange(a,b)` donne un entier entre `a` (inclus) et `b` (exclu)
`random()` donne un flottant au hasard entre `0` et `1`.

Un bonus si vous écrivez une procédure qui n'utilise pas `shuffle` mais juste des `randrange`.

I∼2) Écrivez une procédure `Test` qui prend en entrée une liste de crêpes `L` et retourne un booléen pour dire si la pile est triée, avec ses crêpes dans le bon sens. 3 pt.

Exemple : `Test( [6, 4, 3, 2, 1])` donne `True`,
`Test( [7, -5, 4, 3, 2, -1])` donne `False`,
`Test([7, 4, 5, 3, 2, 1])` donne `False`.

I∼3) Écrivez une procédure `Retourne` qui prend en entrée une liste `L` et un index `n` et retourne la pile de crêpes `L` retournée à partir de l'indice `n`. 3 pt.

Exemple : `Retourne([6, 7, -5, 4, 2, -3], 3)` donne `[6, 7, -5, 3, -2, -4]`.

Rappel : on peut couper une liste d'un indice `a` (inclus) à un indice `b` (exclu), de plus une option indique avec quel pas on avance dans la liste (par défaut, c'est bien sûr 1).

Exemple : `L = [1, 3, 5, 7, 8, 2, 6, 5, 9]`
`L[2 : 5]` donne `[5, 7, 8]`
`L[ : 4]` donne `[1, 3, 5, 7]`
`L[2 : 7 : 2]` donne `[5, 8, 6]`
`L[9 : 1 : -2]` donne `[9, 6, 8, 5]`
`L[1 : 9 : -2]` donne `[ ]`

```
def Scooby(L) :
....M, i = abs(L[0]), 0
....for k in range(len(L)) :
........if abs(L[k]) > M :
............M, i = L[k], k
....return(i)
```

I∼4) Que fait cette procédure : `Scooby` ?
Que retournera `Scooby([2, 5, -8, 7, 1, 3, 4, -6])` ? 2 pt.

I∼5) Complétez 2 pt.

	[-7, 2, 5, 1, 3, 4, -6]
Retourne(L, 0)	*[6, -4, -3, -1, -5, -2, 7]*
Retourne(L, 6)	*[6, -4, -3, -1, -5, -2, -7]*
Retourne(L,)	*[7, 2, 5, 1, 3, 4, -6]*
Retourne(L,)	*[7, 6, -4, -3, -1, -5, -2]*
Retourne(L, 5)	
Retourne(L,)	*[7, 6, -4, -3, -1, 2, -5]*
Retourne(L,)	*[7, 6, 5, , , ,]*
Retourne(L, 6)	*[7, 6, 5, , , ,]*
	[7, 6, 5, 4, 3, 2, 1]

I∼6) Quelle est la pile de cinq crêpes qui sera la plus longue à retourner par algorithme du crêpier, et en combien de retournements ? (la plus courte est la pile déjà triée `[5, 4, 3, 2, 1]`). 3 pt.

I∼7) Écrivez un script qui prend en entrée une liste `L` et la trie par algorithme du crêpier. 4 pt.

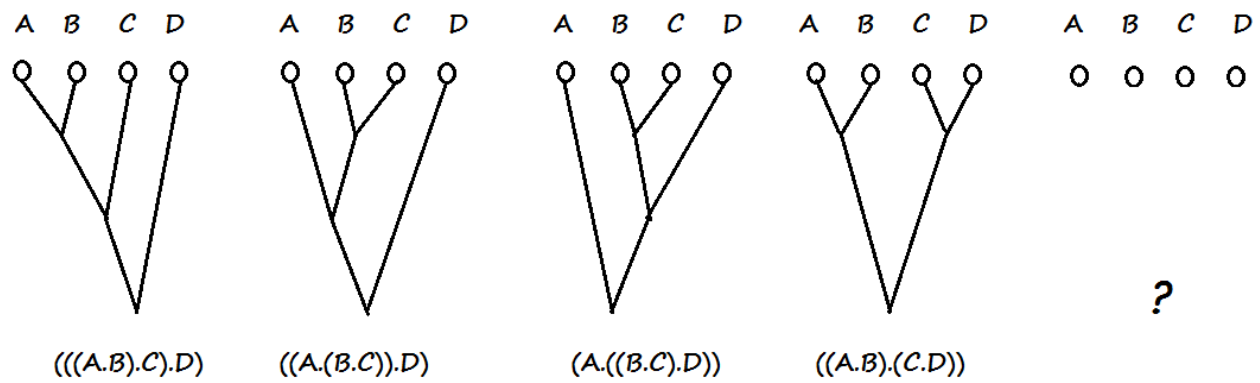
Lycee Charlemagne — MPSI2 — Annee 2021/22

. Eugène Catalan .

On envisage de calculer un produit de $n+1$ matrices (dont les formats sont compatibles) : $A_0.A_1.A_2 \dots A_n$.
Mais on ne peut multiplier que deux matrices à la fois (la multiplication est associative). Par exemple, pour $n = 3$, on a les possibilités suivantes :

$(((A_0.A_1).A_2).A_3)$	$((A_0.(A_1.A_2)).A_3)$	$(A_0.((A_1.A_2).A_3))$	$((A_0.A_1).(A_2.A_3))$	.

avec A, B, C et D pour alléger



Par exemple, pour $((A_1.(A_2.A_3)).A_4)$, on commence par le produit $A_2.A_3$. On le multiplie ensuite à gauche par A_1 et à la fin, on multiplie à droite par A_4.

♢ 0 ♢ Il reste une case vide. A vous de la compléter. Et comme la question ne mérite pas un point entier, pour le demi point qui manque, écrivez 2023 en base 8. [1 pt.]

♢ 1 ♢ Donnez les 14 possibilités pour cinq matrices (soit sous forme parenthésée (appelez alors vos matrices A, B, C, D et E plutôt que A_k), soit sous forme d'arbre). [3 pt.]
On note C_n le nombre de façons de placer les parenthèses pour $n+1$ matrices.
Les C_n s'appellent nombres de Catalan, mais ont été étudiés initialement par Euler.

> Eugène Charles Catalan naît le 30 mai 1814 à Bruges, en Belgique. Son père, Joseph Catalan, un joailler parisien, le reconnaît en 1821. Eugène vit à Paris à partir de 1825 ; il intègre l'École polytechnique (Promotion X1833). Il est expulsé de l'école l'année suivante pour son activisme républicain (il avait déjà pris une part active aux journées de 1830). Il est autorisé en 1835 à reprendre ses études à Polytechnique et en sort diplômé, puis enseigne les mathématiques à l'École des Arts et Métiers de Châlons-sur-Marne. Son ami Liouville l'aide à obtenir en 1838 un poste d'enseignant en géométrie descriptive à Polytechnique. Mais ses activités politiques très à gauche mettent un frein à sa carrière. Il enseigne pendant plusieurs années au lycée Charlemagne (ou Saint-Louis selon les biographies). En 1844, dans une lettre à l'éditeur du Journal de Crelle, Catalan écrit sa célèbre conjecture :
>
> « Je vous prie, Monsieur, de vouloir bien énoncer, dans votre recueil, le théorème suivant, que je crois vrai, bien que je n'aie pas encore réussi à le démontrer complètement : d'autres seront peut-être plus heureux : Deux nombres entiers consécutifs, autres que 8 et 9 ne peuvent être des puissances exactes ; autrement dit : l'équation $x^p - y^q = 1 x^p - y^q = 1$ dans laquelle les inconnues sont entières et positives, n'admet qu'une seule solution. »
> Il étudia les C_n comme le nombre de parenthésage d'expressions pendant son étude du problème des tours de Hanoï.

♢ 2 ♢ Justifiez par un argument de dénombrement : $C_0 = C_1 = 1$ et $C_{n+1} = \sum_{k=0}^{n} C_k.C_{n-k}$ (et donc, sans récurrence !). [4 pt.]

♢ 3 ♢ Vrai ou faux : $C_7 = 429$. [1 pt.]

♢ 4 ♢ Écrivez un script Python qui pour n donné va alors retourner la liste $[C_0,\ C_1,\ C_2, \dots C_n]$. [3 pt.]

♠ 0 ♠ On définit : $f = x \longmapsto (1-4.x)^{1/2}$. Prouvez $f^{(n)} = x \longmapsto -2.\dfrac{(2.n-2)!}{(n-1)!}.(1-4.x)^{(1-2.n)/2}$ pour tout n, puis calculez $f^{(n)}(0)$. [3 pt.]

♠ 1 ♠ Donnez alors le développement limité de f d'ordre $n+1$ en 0. [2 pt.]

♠ 2 ♠ Déduisez le développement limité d'ordre n en 0 de $x \longmapsto \dfrac{1-f(x)}{2.x}$ (application notée c) sous la forme $c(x) = \sum_{k=0}^{n} \gamma_k.x^k + o(x^n)$, et calculez γ_k pour k de 0 à 5. [3 pt.]

♠ 3 ♠ Comparez $c(x)$ et $1 + x.(c(x))^2$. [1 pt.]

♠ 4 ♠ Déduisez par récurrence forte : $\gamma_k = C_k$ pour tout k. [3 pt.]

♠ 5 ♠ Maintenant qu'on sait : $C_n = \frac{1}{n+1}.\binom{2.n}{n}$, écrivez un script Python qui prend en entrée n et retourne C_n (nombre entier, pas flottant, on est en maths !). [3 pt.]

♠ 6 ♠ Juste pour faire plaisir au prof de maths, calculez le déterminant $\begin{vmatrix} C_0 & C_1 & C_2 \\ C_1 & C_2 & C_3 \\ C_1 & C_3 & C_4 \end{vmatrix}$. [1 pt.]

♠ 7 ♠ Et pour faire plaisir au matheux, on admet que $n!$ est équivalent à $\left(\frac{n}{e}\right)^n.\sqrt{2.n.\pi}$ quand n tend vers l'infini.

Montrez que C_n est équivalent à $\frac{4^n}{n.\sqrt{n.\pi}}$ quand n tend vers l'infin. [2 pt.]

Rappel : « a_n équivalent à α_n quand n tend vers l'infini » signifie $\frac{a_n}{\alpha_n}$ tend vers 1 quand n tend vers l'infini,

vous pouvez multiplier les équivalents entre eux (c'est à dire que si vous avez a_n équivalent à α_n et b_n équivalent à β_n alors vous avez $a_n.b_n$ équivalent à $\alpha_n.\beta_n$)

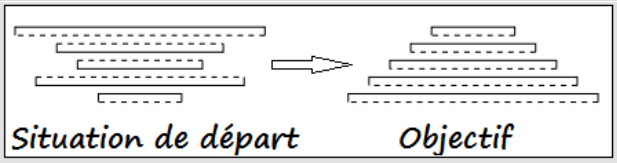
vous pourrez utiliser (mais aussi démontrer en passant au logarithme et avec un D.L.) que $\left(\frac{n+1}{n}\right)^n$ tend vers e quand n tend vers l'infini

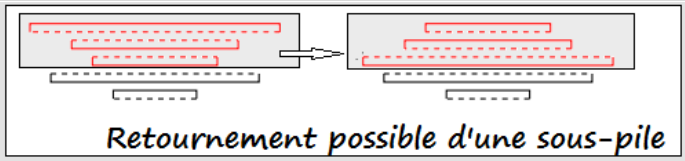
LYCEE CHARLEMAGNE — Lundi 9 janvier — *M.P.S.I.2*

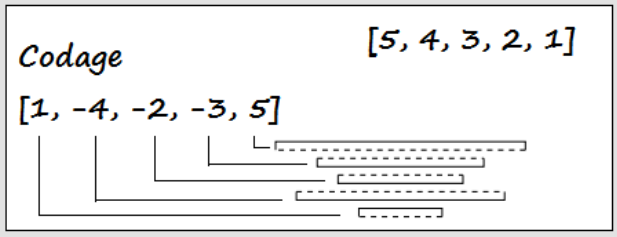
2022 2023

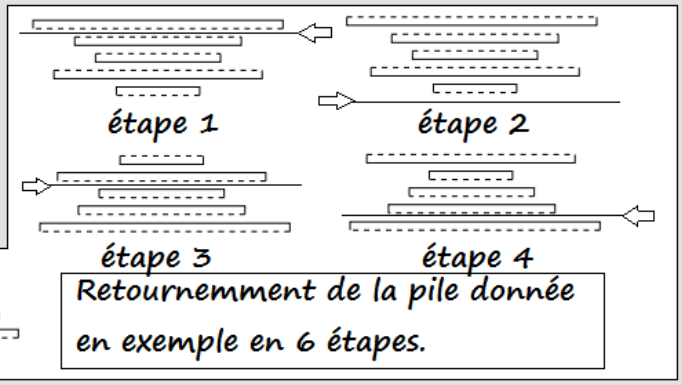
Info00
CORRECTION

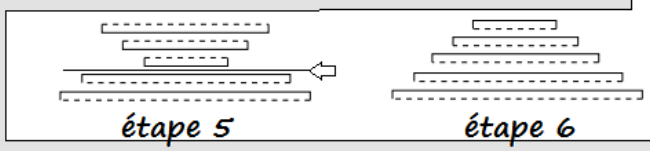
Info00











étape 0	étape 1	étape 2	étape 3	étape 4
`[1, -4, -2, -3, 5]`	`[1, -4, -2, -3, -5]`	`[5, 3, 2, 4, -1]`	`[5, 3, 2, 1, -4]`	`[5, 4, -1, -2, -3]`
`[1, -4, -2, -3] + [5]`	`[ ] + [1, -4, -2, -3, -5]`	`[5, 3, 2] + [4, -1]`	`[5] + [3, 2, 1, -4]`	`[5, 4] + [-1, -2, -3]`
			étape 5	`[5, 4, 3, 2, 1]`

Une ligne a été ajoutée pour comprendre où on coupe.

Mélange. Info00

Comme on veut des entiers distincts, on ne va pas se prendre la tête : `range(1, n+1)` (attention, on commence à 1 et on doit donc terminer à n inclus).
On pourrait commencer avec `[k for k in range(1, n+1)]`.

On veut un signe plus ou moins : `randrange(-1, 2, 2)` prend un entier entre -1 et 2 (exclu) mais avec pas de 2 : bref : -1 ou 1.
On reprend donc avec `[randrange(-1, 2, 2)*k for k in range(1, n+1)]`.

On veut la mélanger ? Un petit coup de `shuffle`.

```
def Melange(n) :
....L = [randrange(-1, 2, 2)*k for k in range(1, n+1)]
....shuffle(L)
....return(L)
```

```
def Melange(n) :
....L = [ ]
....for k in range(1, n+1) :
........if randrange(2) == 0 :
............L.append(k)
........else :
............L.append(-k)
....shuffle(L)
....return(L)
```

```
def Melange(n) :
....L, LL = [ ], [ ]
....for k in range(1, n+1) :
........if randrange(2) == 0 :
............L.append(k)
........else :
............L.append(-k)
....while len(L) > 0 :
........a = L.pop(randrange(len(L))
........LL.append(a)
....return(LL)
```

Test. Info00

On peut le faire en deux fois. On teste d'abord si toutes les crêpes sont dans le bon sens

```
for crepe in L :
....if crepe =< 0 :
........return(False)
```

Et surtout pas

```
for crepe in L :
....if crepe =< 0 :
........return(False)
....else :
........return(True)
```

En effet, avec cette option, vous sortez dès la première crêpe étudiée, avec `True` ou `False`, sans étudier les autres crêpes.
Ce qu'il faut, c'est répondre `False` si on croise une crêpe dans le mauvais sens, c'est tout.

Sinon, vous pouvez aussi utiliser

```
for k in range(len(L)) :
....if L[k] =< 0 :
........return(False)
```

c'est pareil.

Ensuite, on regarde si chaque crêpe est plus grande que la crêpe suivante.

```
for k in range(len(L)-1) :
....if L[k+1] > L[k] :
........return(False)
```

Attention, même chose, on ne met le `return(True)` que si on a parcouru toute la liste sans erreur.
De plus, comme on compare `L[k]` et `L[k+1]`, il faut arrêter `k` un index plus tôt que d'habitude.

```
def Test(L) : #list of int -> boolean
....for crepe in L :
........if crepe =< 0 :
............return(False)
....for k in range(len(L)-1) :
........if L[k+1] > L[k] :
............return(False)
....return(True) #tout s'est bien passé
```

Variante en une fois (on teste « positif et plus grand que le suivant » dont la négation est connue, et il reste le dernier à regarder).

```
def Test(L) :
....for k in range(len(L)-1) :
........if L[k] =< 0 or L[k+1} > L[k] :
............return(False)
....return(L[-1]>0) #reste à tester juste si le dernier est positif
```

Retournement. Info00

On coupe la liste en deux justement à partir de l'indice n.
Le paquet du bas ne contiendra que les termes de 0 (valeur par défaut) à n (exclu).
Le paquet du haut contiendra les termes de l'indice n (inclus) jusqu'à la fin (valeur par défaut).

```
Bas = L[0 : n]
Haut = L[n : len(L)]
```

```
Bas = L[ : n]
Haut = L[n : ]
```

On retourne `Haut` soit avec méthode `reverse`,
soit par parcours avec un pas de `-1` : `Haut = Haut[ : : -1]` (si si, ça marche).
On change les signes de tout le monde : `[-x for x in Haut]`.
On recolle les deux : `Bas + Haut`.

```
def Retourne(L) :
....Bas, Haut = L[ : n], L[n : ]
....tuaH = [-x for x in Haut[ : : -1]]
....return(Bas + tuaH)
```

On peut aussi recopier la fin de la liste et changer ensuite les valeurs des éléments de la liste `L`.

```
def Retourne(L, n) : #list, int -> list
....#retourne la liste L à partir de la position i, en changeant le signe de chaque crêpe
....Pile = L[n : ] #copie de la fin de la liste
....for k in range(n, len(L)) :
........L[k] = -Pile[n-k-1] #remplacement d'un élément de L par l'opposé de l'élément de Pile lu depuis la fin
....return(L)
```

On vérifie : pour `k = i`, on va chercher l'élément d'indice `-1` dans `Pile`, c'est le dernier.
pour `k = i+1`, on va chercher l'élément d'indice `-2` dans `Pile` : l'avant dernier
pour `k = len(L)-1`, on va chercher l'élément d'indice `n-Len(L)` dans `Pile`, et sachant que `Pile` a `Len(L)-n` éléments, c'est le premier.
Oui étrangement, `L[len(L)]` n'existe pas (`index out of range` trop long) mais `L[-len(L)]` existe, et c'est `L[0]`.

Procédure Scooby. Info00

Cette procédure ressemble fort à la recherche du maximum d'une liste.

```
def Vera(L) : #list of *** -> ***
....M = L[0] #on initialise
....for k in range(len(L)) : #on parcourt la liste
........if L[k] > M : #si on trouve un nouveau record
............M = L[k] #c'est lui qu'on enregistre
....return(M)
```

Ce qui change ici : on met des valeurs absolues. On cherche donc le plus grand terme en valeur absolue.
De plus, on mémorise le record, mais aussi son index k. Et c'est lui qu'on retourne.

On cherche donc l'index du terme le plus grand en valeur absolue.

`Scooby([2, 5, -8, 7, 1, 3, 4, -6])` va donc retourner 2 puisque c'est `L[2]` qui est le plus grand en valeur absolue.

Un exemple. Info00

J'ajoute une colonne pour la compréhension

		[-7, 2, 5, 1, 3, 4, -6]
Retourne(L, 0)	[ ] + [-7, 2, 5, 1, 3, 4, -6]	[6, -4, -3, -1, -5, -2, 7]
Retourne(L, 6)	[6, -4, -3, -1, -5, -2] + [7]	[6, -4, -3, -1, -5, -2, -7]
Retourne(L, 0)	[ ] + [6, -4, -3, -1, -5, -2, -7]	[7, 2, 5, 1, 3, 4, -6]
Retourne(L, 1)	[7] + [2, 5, 1, 3, 4, -6]	[7, 6, -4, -3, -1, -5, -2]
Retourne(L, 5)	[7, 6, -4, -3, -1} + [-5, -2]	[7, 6, -4, -3, -1, 2, 5]
Retourne(L, 6)	[7, 6, -4, -3, -1, 2} + [5]	[7, 6, -4, -3, -1, 2, -5]
Retourne(L, 2)	[7, 6] + [-4, -3, -1, 2, -5]	[7, 6, 5, -2, 1, 3, 4]
Retourne(L, 6)	[7, 6, 5, -2, 1, 3] + [4]	[7, 6, 5, -2, 1, 3, -4]
Retourne(L, 3)	[7, 6, 5] + [-2, 1, 3, -4]	[7, 6, 5, 4, -3, -1, 2]
Retourne(L, 4)	[7, 6, 5, 4] + [-3, -1, 2]	[7, 6, 5, 4, 2, 1, 3]
Retourne(L, 6)	[7, 6, 5, 4, 2, 1] + [3]	[7, 6, 5, 4, 2, 1, -3]
Retourne(L, 4)	[7, 6, 5, 4] + [2, 1, -3]	[7, 6, 5, 4, 3, -1, -2]
Retourne(L, 5)	[7, 6, 5, 4, 3] + [-1, -2]	[7, 6, 5, 4, 3, 2, 1]

Il me semble bien que c'est la pile [-5, -4, -3, -2, -1] qui sera la plus longue à retourner.
Les crêpes sont dans le bon ordre, mais toutes à l'envers.
[-5, -4, -3, -2, -1]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, -5]
[5, -4, -3, -2, -1]
[5, 1, 2, 3, 4]
[5, 1, 2, 3, -4]
[5, 4, -3, -2, -1]
[5, 4, 1, 2, 3]
[5, 4, 1, 2, -3]
[5, 4, 3, -2, -1]
[5, 4, 3, 1, 2]
[5, 4, 3, 1, -2]
[5, 4, 3, 2, -1]
[5, 4, 3, 2, 1]
Mais je m'y prends put être mal. Le nombre de fois où je retourne juste la crêpe du haut du paquet !

Algorithme. Info00

Un algorithme prend forme.
Une partie de la liste est triée, mais pas encore la liste entière.
Regarder ce qu'il reste.
Trouver dans ce qu'il reste la plus grand crêpe.
La placer au dessus du paquet en renversant à partir de sa position.
Si elle est dans le « mauvais sens », retourner juste cette crêpe.
Retourner la pile pour mettre alors la plus grande crêpe à la bonne place.

Par exemple, avec [6, 5, 3, -4, 2, 1].
[6, 5] est déjà trié.
On regarde [3, -4, 2, 1].
Le plus grand élément en valeur absolue est -4, avec index (global) 2.
On renverse la liste initiale à partir de l'indice 2 :

$$[6,5,3,-4,2,1] -> [6,5,3] + [-4,2,1] -> [6,5,3] + [1,-2,4] -> [6,5,3,1,-2,4]$$

On retourne la dernière crêpe avec Renverse(L, 6) :

$$[6,5,3,1,-2,4] -> [6,5,3,1,-2] + [4] -> [6,5,3,1,-2] + [-4] -> [6,5,3,1,-2,-4]$$

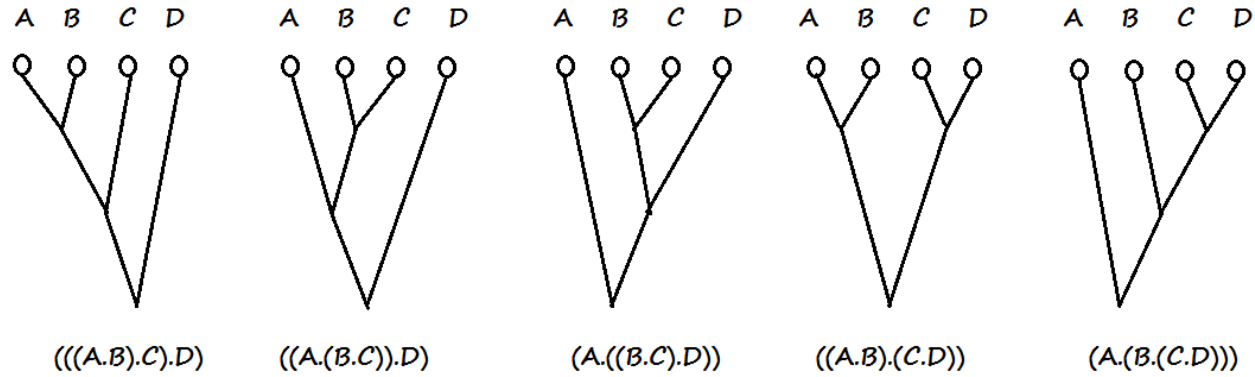
On renverse à partir de l'indice 2 : Renverse(L, 2) :

$$[6,5,3,1,-2,-4] -> [6,5] + [3,1,-2,-4] -> [6,5] + [4,2,-1,-3] -> [6,5,4,2,-1,-3]$$

Premiers nombres de Catalan. Info00

On complète la dernière case, et le graphe associé :

$(((A_0.A_1).A_2).A_3)$	$((A_0.(A_1.A_2)).A_3)$	$(A_0.((A_1.A_2).A_3))$	$((A_0.A_1).(A_2.A_3))$	$(A_0.(A_1.(A_2.A_3))$
avec A, B, C et D				



Et sinon, en base 8 : $\boxed{2023 = 3.8^3 + 7.8^2 + 4.8 + 7}$ donc $2023_{10} = 3747_8$

On peut chercher avec les multiples de 8 connus : 1, 8, 64, 512.
Combien de fois 512 trois fois mais pas 4 et il reste 487.
Dans 487 : 7.64 et il reste 39.
Et 39 est égal à $4.8 + 7$.

On retrouve les premiers nombres de Catalan.
Si il n'y a qu'une matrice, un seul calcul possible : (A) : $C_0 = 1$.

S'il n'y a que deux matrices (et donc une multiplication) : $(A.B)$ et C_1 vaut encore 1.

S'il y a trois matrices : $(A.(B.C))$ et $((A.B).C)$, suivant par quel regroupement on commence. $C_2 = 2$.

On a calculé C_3 et trouvé 5.
D'ailleurs, si on lit plus loin, $C_n = \dfrac{1}{3+1}.\binom{6}{3}$.

On est prié de trouver effectivement $C_4 = 14$.
On peut donner les 14 formes de $(A.(B.(C.(D.E))))$ à $(((((A.B).C).D).E)$ en passant par $((A.B).((C.D).E))$.

On peut aussi donner les arbres.

On peut aussi le prouver en ayant une jolie piste pour la formule $C_{n+1} = \sum_{k=0}^{n} C_k.C_{n-k}$.
Si on doit calculer $A.B.C.D.E$ en mettant nos parenthèses et priorités, la question peut être : quelle sera la dernière multiplication effectuée ?
Elle coupera le produit en deux : des termes d'un côté, des termes de l'autre, comme $(A.B) \times (C.D.E)$.
On a une façon de calculer $(A.B)$, mais on en a deux de calculer $(C.D.E)$.
D'où 1.2 façons quand on coupe ainsi.
On aurait pu couper aussi en $(A) \times (B.C.D.E)$ avec 1 façon pour (A) et 5 façons pour $(B.C.D.E)$.

découpage	$(A)\times(B.C.D.E)$	$(A.B)\times(C.D.E)$	$(A.B.C)\times(D.E)$	$(A.B.C.D)\times(E)$	
premier terme	1	1	2	5	
second terme	5	2	1	1	
total	1.5	1.2	2.1	5.1	$=14$
	$(A).\Big(B.(C.(D.E))\Big)$	$(A.B).\Big((C.D).E\Big)$	$\Big((A.B).C\Big).(D.E)$	$\Big(A.(B.(C.D))\Big).(E)$	
	$(A).\Big(B.((C.D).E)\Big)$	$(A.B).\Big(C.(D.E)\Big)$	$\Big(A.(B.C)\Big).(D.E)$	$\Big(A.((B.C).D)\Big).(E)$	
	$(A).\Big((B.C).(D.E)\Big)$			$\Big((A.B).(C.D)\Big).(E)$	
	$(A).\Big(B.(C.D)).E\Big)$			$\Big((A.(B.C)).D\Big).(E)$	
	$(A).\Big((B.C).D).E\Big)$			$\Big(((A.B).C).D\Big).(E)$	

La formule $C_{n+1}=\sum_{k=0}^{n} C_k.C_{n-k}$. se démontre par un argument de dénombrement.

Elle ne se démontre pas par récurrence.
En revanche, elle servira ensuite à des récurrences.

On veut estimer C_{n+1} par dénombrement.
On doit calculer un produit de $n+2$ facteurs.
On se demande : quelle sera la dernière multiplication effectuée. Disons qu'elle se placera en $k+1^{ieme}$ position.
Elle aura à sa gauche un produit de $k+1$ termes et à sa droite un produit de $n+2-k$ termes : $(A_0.A_1\dots A_k)\times(A_{k+1}\dots A_{n+1})$.

Il y aura C_k façons de calculer $(A_0.A_1\dots A_k)$
C_{n-k} façons de calculer $(A_{k+1}.A_{k+2}\dots A_{n+1})$
et donc $C_k.C_{n-k}$ façons de calculer $(A_0.A_1\dots A_k)\times(A_{k+1}\dots A_{n+1})$
Par disjonction de cas, on somme sur toutes les valeurs de k : $C_n=\sum_{k=0}^{n} C_k.C_{n-k}$

Pour ce qui est de C_7, on utilise la formule $C_{n+1}=\sum_{k=0}^{n} C_k.C_{n-k}$.
On sait $C_7=C_6.C_0+C_5.C_1+C_4.C_2+C_3.C_3+C_2.C_4+C_1.C_5+C_0.C_6$
$C_6=C_5.C_0+C_4.C_1+C_3.C_2+C_2.C_3+C_1.C_4+C_0.C_5$
$C_5=C_4.C_0+C_3.C_1+C_2.C_2+C_1.C_3+C_0.C_4$
$C_4=C_3.C_0+C_2.C_1+C_1.C_2+C_0.C_3$
$C_3=5, C_2=2, C_1=C_0=1$
On effectue à partir de la fin :

$$\begin{array}{|l}\hline C_4=5.1+2.2+1.5=14\\ C_5=14.1+5.1+2.2+1.5+1.14=42\\ C_6=42.1+14.1+5.2+2.5+1.14+1.42=132\\ C_6=132.1+42.1+14.2+5.5+2.14+1.42+1.142=429\\ \hline\end{array}$$

Après, on peut aussi tricher et lire la suite : $C_5=\frac{1}{6}.\binom{10}{5}=\frac{1}{6}.\frac{10.9.8.7.6}{5.4.3.2.1}=42$
et recommencer : $C_7=\frac{1}{8}.\frac{14.13.12.11.10.9.8}{1.2.3.4.5.6.7}$ et on efface directement :

$$C_7=\frac{1}{*}.\frac{14.13.12.11.10.9.*}{1.2.3.4.5.6.7}=\frac{14.13.12.11.*.9}{1.*.3.4.*.6.7}=\frac{14.13.*.11.9}{1.*.*.6.7}=\frac{*.13.11.9}{1.*.*}=\frac{13.11.9}{1.3}=13.11.3=429$$

Quiconque effectue le produit de tout le numérateur, de tout le dénominateur et ne simplifie qu'ensuite ne pourra jamais faire de maths ni de physique...

Procédure Python. Info00

On va calculer les termes de proche en proche.
On initialise une liste avec `1` : `C = [1]` (c'est `C[0]`).
On avance en calculant à chaque fois une somme `S` des `C[i]*C[k-i]` par une boucle sur `i`.
On ajoute ce nouveau terme en bout de liste `C.append(S)`.

On effectue ce calcul `n` fois pour avoir un total de $n+1$ termes.

```
def Catalan(n) : #int -> list of int
....C = [1] #premier terme de la suite
....for k in range(n) : #il reste n termes à calculer
........S = 0 #initialisation de la somme
........for i in range(k+1) : #attention au nombre de termes
............S += C[i]*C[k-i] #formule de récurrence
........C.append(S) #ajouter le nouveau terme à la liste
....return(C) #la liste est complète
```

Des développements limités. Info00

La formule $f^{(n)} = x \longmapsto -2.\dfrac{(2.n-2)!}{(n-1)!}.(1-4.x)^{(1-2.n)/2}$ se démontre par récurrence sur n.

On a en effet $f = x \longmapsto (1-4.x)^{1/2}$ d'où $f' = x \longmapsto \dfrac{1}{2}.(-4).(1-4.x)^{-1/2}$

puis $f' = x \longmapsto \dfrac{1}{2}.(-4).\dfrac{-1}{2}.(-4).(1-4.x)^{-3/2}$.

On valide la formule aux premiers rangs.

On la suppose vraie à un rang n quelconque donné : $f^{(n)} = x \longmapsto -2.\dfrac{(2.n-2)!}{(n-1)!}.(1-4.x)^{(1-2.n)/2}$. On redérive

$$f^{(n+1)} = x \longmapsto -2.\frac{(2.n-2)!}{(n-1)!}.\frac{1-2.n}{2}.(-4).(1-4.x)^{\frac{1-2.n}{2}-1}$$

$$f^{(n+1)} = x \longmapsto -2.\frac{(2.n-2)!}{(n-1)!}.(2.n-1).2.(1-4.x)^{\frac{1-2.n-2}{2}}$$

$$f^{(n+1)} = x \longmapsto -2.\frac{(2.n-2)!}{(n-1)!}.(2.n-1).\frac{2.n}{n}.(1-4.x)^{\frac{1-2.(n+1)}{2}}$$

On aboute les factorielles : $(2.n-2)!.(2.n-1).(2.n) = (2.n)!$ et $(n-1)!.n = n!$. La formule est validée au rang $n+1$.

Abouter : verbe transitif du premier groupe : joindre bout à bout.

On notera qu'elle n'est pas valable au rang 0.

On peut calculer les dérivées successives en 0 et utiliser la formule de Taylor avec reste... beh non, sans reste :

$$f(0+h) = \sum_{k=0}^{n+1} \frac{f^{(k)}(0)}{k!}.h^k + o(h^{n+1}) = f(0) + \sum_{k=1}^{n+1} -2.\frac{(2.k-2)!}{(k-1)!}.(1-4.0)^{(1-2.k)/2}.\frac{h^k}{k!} + o(h^{n+1})$$

On regroupe ce qu'on peut :

$$f(h) = 1 - \sum_{k=1}^{n+1} 2.\frac{(2.k-2)!}{(k-1)!}.\frac{h^k}{k!} + o(h^{n+1})$$

On passe à la fonction c en mettant un signe moins $-f(h) = -1 + \sum_{k=1}^{n+1} 2.\dfrac{(2.k-2)!}{(k-1)!}.\dfrac{h^k}{k!} + o(h^{n+1})$

En ajoutant 1 : $1 - f(h) = \sum_{k=1}^{n+1} 2.\dfrac{(2.k-2)!}{(k-1)!}.\dfrac{h^k}{k!} + o(h^{n+1})$

En divisant par $2.h$: $\dfrac{1-f(h)}{2.h} = \sum_{k=1}^{n+1} \dfrac{(2.k-2)!}{(k-1)!}.\dfrac{h^{k-1}}{k!} + o(h^n)$

Oui, $o(h^{n+1})$ ou la moitié, ça ne change rien, puisque par définition, c'est une histoire de quotient qui tend vers 0. Vous faites une différence entre « tendre vers 0 » ou tendre vers $\dfrac{0}{2}$ » ?

De même, $\frac{o(h^{n+1})}{h} = o(h^n)$.

En effet, par définition, si on nomme $g(h)$ la fonction $\frac{o(h^{n+1})}{h}$, on a justement $\frac{g(h)}{h^n} = \frac{o(h^{n+1})}{h^n.h} = \frac{o(h^{n+1})}{h^{n+1}}$ et ceci tend vers 0 quand h tend vers 0.

On décale les indices : $\frac{1-f(h)}{h} = \sum_{i=1}^{n} \frac{(2.(i+1)-2)!}{((i+1)-1)!}.\frac{h^{i+1-1}}{i+1} + o(h^n)$ et on a

$$c(h) = \sum_{i=0}^{n} \frac{(2.i)!}{i!.(i+1)!}.h^i + o(h^n)$$

On nous demande une formule du type $\sum_{k=0}^{n} \gamma_k.h^k + o(h^n)$? C'est bon, avec $\gamma_k = \frac{(2.k)!}{k!.(k+1)!}$.

On calcule les premiers, non sans surprise :

γ_0	γ_1	γ_2	γ_3	γ_4	γ_5
$\frac{0!}{0!.1!} = 1$	$\frac{2!}{1!.2!} = 1$	$\frac{4!}{2!.3!} = 2$	$\frac{6!}{3!.4!} = 5$	$\frac{8!}{4!.5!} = 14$	$\frac{10!}{5!.6!} = 42$

Le truc génial : $1 + x.(c(x))^2 = c(x)$.
C'est juste du calcul.

$$1 + x.(c(x))^2 = 1 + x.\frac{1 + (1-4.x) - 2.\sqrt{1-4.x}}{4.x^2} = 1 + \frac{1 - 2.x - \sqrt{1-4.x}}{2.x} = \frac{1-\sqrt{1-4.x}}{x}$$

On montre par récurrence forte sur n que les γ_k sont égaux aux C_k.
Le phénomène est initialisé.

On se donne n quelconque, et on suppose $C_k = \gamma_k$ pour tout k de 0 à n.

On sait alors $C_{n+1} = \sum_{k=0}^{n} C_k.C_{n-k}$.

Qu'en est il de γ_{n+1} ?
C'est le coefficient de x^{n+1} dans le développement limité de $c(x)$.
C'est donc aussi le coefficient de x^{n+1} dans le développement limité de $1 + x.(c(x))^2$.

On va donc développer $1 + x.\Big(\sum_{k=0}^{n} \gamma_k.x^k + o(x^n)\Big)^2$ (il faut bien que les questions servent à quelquechose).

Mais on va plutôt développer $1 + x.\Big(\sum_{k=0}^{n} \gamma_k.x^k + o(x^n)\Big).\Big(\sum_{i=0}^{n} \gamma_i.x^i + o(x^n)\Big)$.

On trouve $1 + \sum_{\substack{k\leqslant n \\ i\leqslant n}} \gamma_k.\gamma_i.x^{k+i} + x.o(x^n).\Big(\sum_{i=0}^{n} \gamma_i.x^i\Big) + x.o(x^n).\Big(\sum_{k=0}^{n} \gamma_k.x^k\Big) + x.o(x^n).o(x^n)$ si on fait du zèle.

Mais sinon, on se contente de $1 + \sum_{\substack{k\leqslant n \\ i\leqslant n}} \gamma_k.\gamma_i.x^{k+i} + o(x^{n+1})$ en jetant tout dans une poubelle unique.

$\sum_{j=0}^{n+1} \gamma_k.x^{j+1} + o(x^{n+1}) = c(x) = 1 + x.(c(x))^2 = 1 + \sum_{\substack{k\leqslant n \\ i\leqslant n}} \gamma_k.\gamma_i.x^{k+i} + o(x^{n+1})$
On identifie le coefficient de x^{n+1} de chaque côté : γ_{n+1} est égal à la somme de tous les termes en $\gamma_i.\gamma_k$ donnant du x^n.
La condition est donc $i + k = n$. Ou même $i = n - k$.
On a donc $\gamma_{n+1} = \sum_{k=0}^{n} \gamma_k.\gamma_{n-k}$.

On résume : $\gamma_k = C_k$ pour tout$k$ de 0 à n,

$$C_{n+1} = \sum_{k=0}^{n} C_k.C_{n-k} \text{ et } \gamma_{n+1} = \sum_{k=0}^{n} \gamma_k.\gamma_{n-k}$$

On, déduit $C_{n+1} = \gamma_{n+1}$.
La récurrenceà forte hérédité s'achève.

Le déterminant $\begin{vmatrix} 1 & 1 & 2 \\ 1 & 2 & 5 \\ 2 & 5 & 14 \end{vmatrix}$ vaut 1.

Et le plus fort, ce sera que $\begin{vmatrix} 1 & 1 \\ 1 & 2 \end{vmatrix}$ et $\begin{vmatrix} 1 & 1 & 2 & 5 \\ 1 & 2 & 5 & 14 \\ 2 & 5 & 14 & 42 \\ 5 & 14 & 42 & 132 \end{vmatrix}$ valent aussi 1.

On effectue le quotient $C_n = \frac{1}{n+1}.\binom{2.n}{n} = \frac{(2.n)!}{(n+1)!n!}$ puis

$$\frac{C_n}{\frac{4^n}{n.\sqrt{n.\pi}}} = \frac{(2.n)!.n.\sqrt{n.\pi}}{(n+1)!.n!.4^n}$$

et on tente d'y retrouver nos quotients utiles comme $\frac{n!}{\left(\frac{n}{e}\right)^n.\sqrt{2.n.\pi}}$ mais aussi $\frac{(n+1)!}{\left(\frac{n+1}{e}\right)^{n+1}.\sqrt{2.(n+1).\pi}}$ et $\frac{(2.n)!}{\left(\frac{2.n}{e}\right)^{2.n}.\sqrt{4.n.\pi}}$ qui tendent tous vers 1.

Autant le faire efficacement : les équivalents passent au produit :

$$\frac{(2.n)!}{(n+1)!n!} = \frac{\left(\frac{2.n}{e}\right)^{2.n}.\sqrt{4.n.\pi}}{\left(\frac{n+1}{e}\right)^{n+1}.\sqrt{2.(n+1).\pi}.\left(\frac{n}{e}\right)^n.\sqrt{2.n.\pi}}$$

On regarde l'exposant de e : $-2.n$ en haut et $-(n+1)-n$ en bas. Il va nous rester du e au numérateur.
On regarde l'exposant de n : $2.n$ en haut et n en bas. Il va nous rester du n^n.
On regarde l'exposant de $n+1$: $n+1$ en bas et c'est tout.
On regarde l'exposant de π : $\frac{1}{2}$ en haut et $2\times\frac{1}{2}$ en bas. Il reste du $\frac{1}{\sqrt{\pi}}$.
On regarde l'exposant de n sous des racines : $\frac{1}{2}$ en haut et $2\times\frac{1}{2}$ en bas. Il reste du $\frac{1}{\sqrt{n}}$.
On regarde l'exposant de 2 : $2.n+1$ en haut (il y a $\sqrt{4}$), et 2 en bas. Il reste $2^{2.n}$ en haut.

Bilan : $\frac{e.n^n.4^n}{(n+1)^{n+1}\sqrt{\pi.n}}$. On semble encore loin du résultat voulu.

Mais on peut faire mieux : $\frac{e.n^n.4^n}{(n+1)^n(n+1).\sqrt{\pi.n}}$ et même $\frac{e.4^n}{\left(\frac{n+1}{n}\right)^n(n+1).\sqrt{\pi.n}}$ en se laissant influencer par l'énoncé.

Et justement, $\left(\frac{n+1}{n}\right)^n$ tend vers e. Les e s'en vont. Il reste $\frac{4^n}{(n+1).\sqrt{n.\pi}}$.

Et $n+1$ est équivalent à n (le quotient tend vers 1). On a bien $\frac{4^n}{n.\sqrt{n.\pi}}$ comme demandé.

Et, et comment prouver que la forme indéterminée $\left(\frac{n+1}{n}\right)^n$ tend vers e ?
En prenant son logarithme dont on va utiliser le développement limité en 0 : $\ln(1+x) = 0+x+o(x)$:

$$\ln\left(\left(\frac{n+1}{n}\right)^n\right) = n.\ln\left(\frac{n+1}{n}\right) = n.\ln\left(1+\frac{1}{n}\right) = n.\left(0+\frac{1}{n}+o\left(\frac{1}{n}\right)\right) = 1+o(1)$$

C'est gagné, ce logarithme tend vers 1.
Le retour à l'exponentielle nous fait tendre vers e.