Lycée Jean Moulin **Draguignan**
NSI *Terminale* Année .....

DM - Programmation dynamique

Le problème à résoudre

On dispose d'une barre de longueur donnée que l'on peut revendre d'un seul tenant ou en morceaux.
L'objectif étant de maximiser le gain

Par exemple : la barre est de longueur 8 et le tableau ci-dessous donne les prix en fonction de la longueur du morceau.

Longueur du morceau	0	1	2	3	4	5	6	7	8
Prix du morceau	0	2	3	8	10	13	15	16	21

? Question 1:

Donner une ou plusieurs solutions optimales :

.....

.....

Trouver un algorithme - Approche naïve

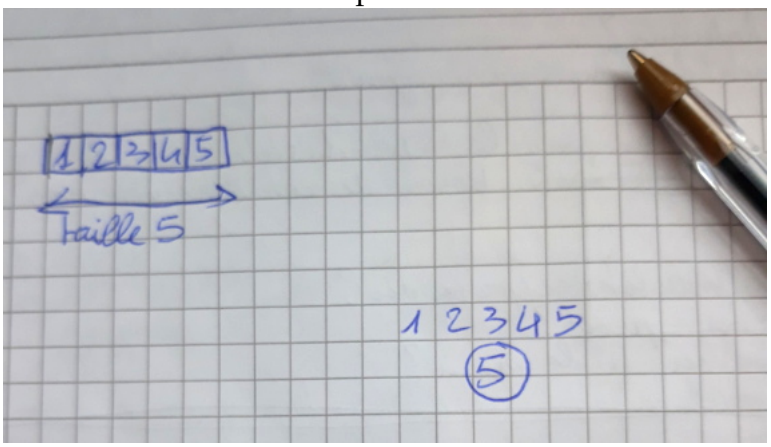
- Trouver toutes les découpes possibles.
- Calculer le gain pour chacune d'entre elles.
- Récupérer le plus grand.

Comment déterminer toutes les découpes possibles ?

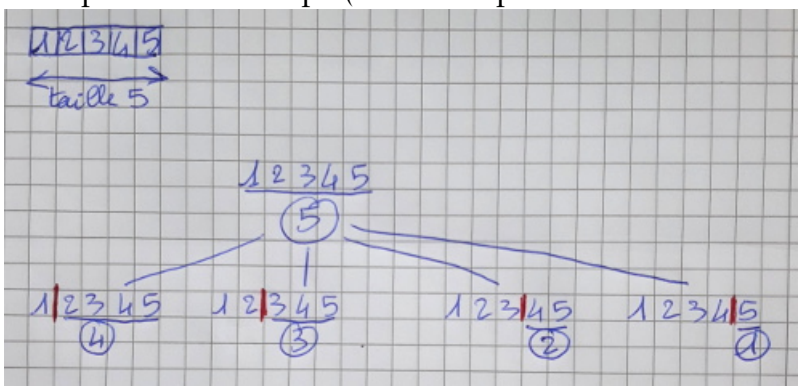
Un schéma vaut parfois mieux que de longues explications...

On considère une barre de longueur 5.

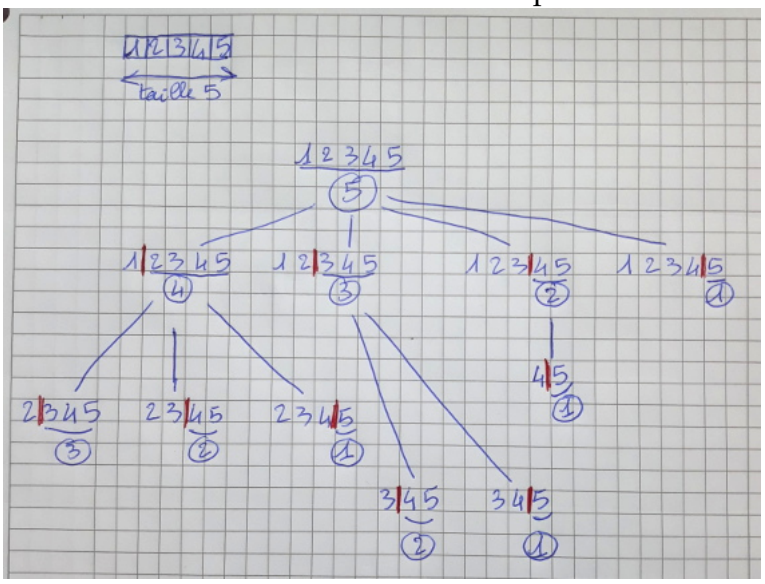
On commence par la barre entière



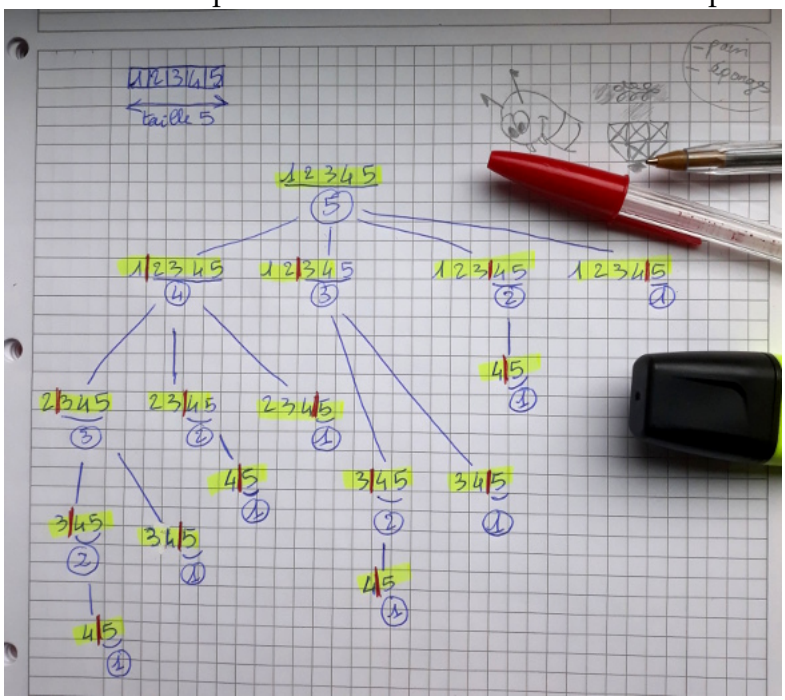
On fait une première découpe (toutes les possibilités sont envisagées)



Puis une seconde découpe

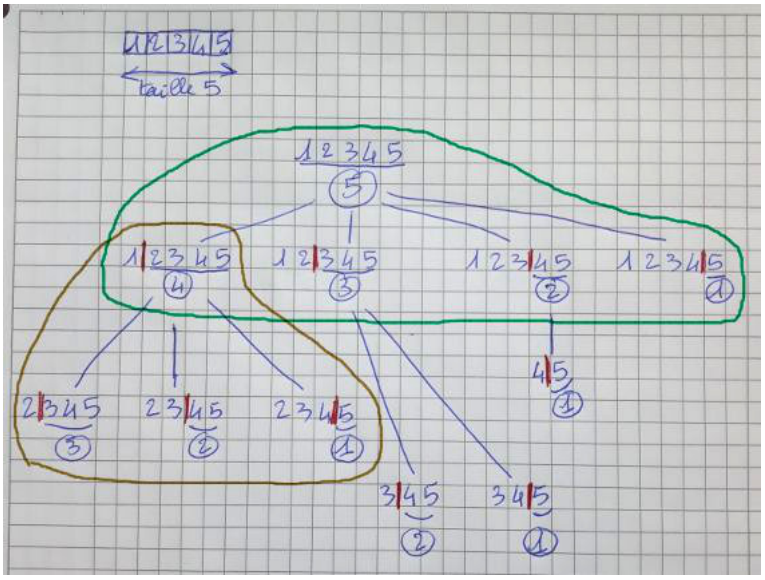


Finalement pour notre cas une troisième découpe



Pour une longueur de 5, il y a 16 possibilités.
Pour une barre de longueur n il y a 2^{n-1} possibilités.

Calcul du meilleur gain :



À chaque étage on a :

meilleur gain = max { Pour chaque découpe : prix de la partie découpée + le meilleur gain de ce que l'on peut faire du reste de la découpe.

Ce qui donne l'algorithme:

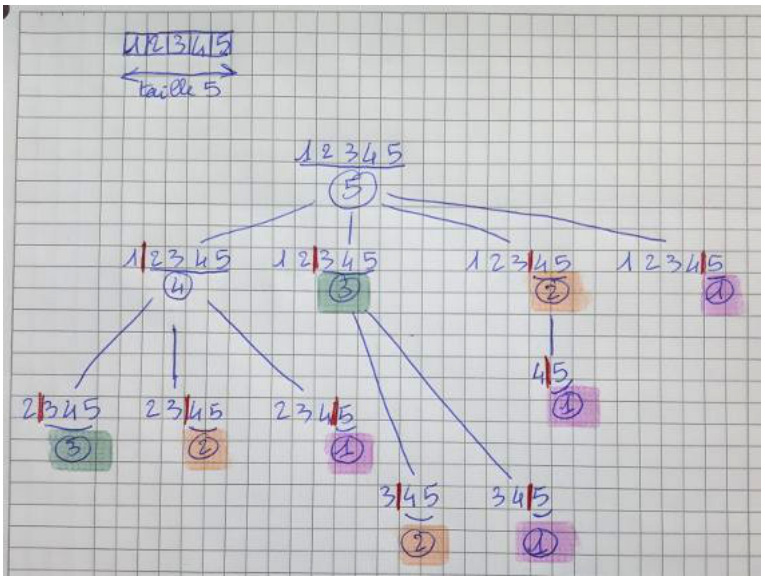
```
Données : taille est un entier
prix est une liste d'entiers
fonction coupe(taille,prix):
Si taille ≤ 0 alors
    renvoyer 0
meilleur ← 0
decoupe ← 1
Tant que decoupe ≤ taille faire
    meilleur ← max(meilleur,prix[decoupe] + coupe(taille-decoupe,prix))
    decoupe ← decoupe + 1
renvoyer meilleur
```

À FAIRE 1:

Écrire la fonction coupe et la tester sur notre exemple de départ.

On peut s'interroger sur l'efficacité de cet algorithme et le cas échéant l'améliorer.

En regardant de plus près...



- Le problème de taille 3 est calculé 2 fois.
- Le problème de taille 2 est calculé 3 fois.
- Le problème de taille 1 est calculé 4 fois.

La complexité est exponentielle...

La programmation dynamique

On va pouvoir améliorer cet algorithme en mémorisant les calculs pour pouvoir les réutiliser et ainsi diminuer le nombre d'appels récursifs.
L'idée est donc la suivante:

- On initialise un tableau de même taille que celui des prix (avec par exemple que des -1).
- Avant de calculer, on regarde si le résultat est déjà dans ce tableau.
- Si non, on le calcule et on le mémorise dans le tableau.

La complexité est alors en $O(n^2)$**.**

À FAIRE 2:

En vous inspirant de ce que l'on a vu dans le TP précédent et dans le cours. Écrire le programme qui réalise cette amélioration.

Prolongement possible...

On obtient bien le gain optimal, mais pas la manière de le réaliser...
Comment s'y prendre alors pour obtenir le gain maximal et une manière de l'obtenir ?

Un autre algorithme...pour les mordus...

Voici un programme déniché sur le net qui réalise cette recherche de gain optimal.

```
prices = [0, 2, 3, 8, 10, 13, 15, 16, 21]
memo_prices = [0, 0, 0, 0, 0, 0, 0, 0, 0]

def cut (size) :
    i = 0
    while i <= size :
        best = 0
        j = 1
        while j <= i :
            best = max (best, prices[j] + memo_prices[i - j])
            j = j + 1
        memo_prices[i] = best
        i = i + 1
    return memo_prices[size]
```

? Question 2:

Quelle différence avec celui que nous avons écrit ?