

1. Contexte

Un cambrioleur possède un sac à dos d'une contenance maximale de 30 Kg.

Au cours d'un de ses cambriolages, il a la possibilité de dérober les objets suivants :



Objet	Poids (en Kg)	Valeur (en €)
Argenterie	13	700
Bijou	12	400
Chandelier	8	300
Drageoir	10	300

Tableau 1. Liste des objets

Le problème est de déterminer les objets que le cambrioleur aura intérêt à dérober, sachant que :

- Tous les objets dérobés devront tenir dans le sac à dos (30 Kg maxi),
- Le cambrioleur cherche à obtenir un **gain maximum**.

2. Principe de l'algorithme « brute force »

L'algorithme classique, dit « **brute force** », pour résoudre ce problème consiste à étudier l'**ensemble des combinaisons possibles** de sélection d'objets à mettre dans le sac.

Question 1 : Lister l'ensemble des combinaisons possibles de sélection d'objets.

Question 2 : Pour un nombre n d'objets disponibles, combien de combinaisons sont à tester pour résoudre le problème ? En déduire la complexité de l'algorithme « brute force » en fonction de n .

3. Principe de l'algorithme glouton

L'algorithme **glouton** repose sur le principe suivant :

- Classer les objets par **ordre décroissant** de leur valeur massique vm (rapport entre la valeur et le poids d'un objet),
- Remplir le sac en prenant les objets dans l'ordre, de telle sorte que **la capacité du sac ne soit pas dépassée**.

Question 3 : Calculer la valeur massique des objets du tableau 1 en complétant le tableau suivant.

Cours NSI	Thème : Algorithmique - Glouton TD – Problème du sac à dos	Date :
-----------	---	--------

Objet	Valeur massique
Argenterie	<input type="text"/>
Bijou	<input type="text"/>
Chandelier	<input type="text"/>
Drageoir	<input type="text"/>

 Question 4 : Quel est le contenu du sac en appliquant l'algorithme glouton sur les objets du tableau 1 ?

 Question 5 : La solution trouvée est-elle optimale ? Pourquoi ?

4. Implémentation en Python

Soit la spécification et le corps **incomplet** de la fonction `glouton_backpack`.

```
def glouton_backpack(objets, capacite):
    """
    :param objets: (list) Un tableau d'objets. Chaque objet est un dictionnaire, doté des
    clés nom, poids et valeur
    :param capacite: (int) La capacité maximale du sac
    :return: (tuple) (Le sac doté des objets (list), gain (int))
    """
    n = len(objets)
    .....# Tri des objets par valeur massique décroissante
    sac = []
    poids = 0
    gain = 0
    for objet in objets:
        if poids + ..... <= .....:
            sac.....
            poids = .....
            gain = .....
    return (sac, gain)
```

 Question 6 : Compléter la fonction `glouton_backpack`.

Il sera possible de tester la fonction avec le tableau d'objets suivant :


```
objets = [
    { "nom" : "Argenterie", "poids" : 13, "valeur" : 700},
    { "nom" : "Bijou", "poids" : 12, "valeur" : 400},
    { "nom" : "Chandelier", "poids" : 8, "valeur" : 300},
    { "nom" : "Drageoir", "poids" : 10, "valeur" : 300}
];
```

Aide : La méthode `sort` permet de trier un tableau selon un critère donné sous forme de fonction `lambda`.

Exemple d'utilisation : `objets.sort(key=lambda x: x[poids], reverse=True)`

Cet appel a pour effet de trier le tableau `objets` par poids décroissant.

 Question 7 : Exprimer la complexité de l'algorithme glouton en fonction du nombre n d'objets.

 Question 8 : Quel est le résultat obtenu si on rajoute l'objet Encrier d'un poids de 6 Kg et d'une valeur de 350 € ? La solution trouvée est-elle optimale ?