TP 5

SUJET CCINP 2021 - FILIÈRE TSI

```
## Question 1
''' Le produit avec le plus de valeur est le 4 et l'on peut lui ajouter un autre produit.

    Les deux rapportant le plus sont 1 et 4 mais on peut encore ajouter le 3.

    Donc on ne maximise pas le profit avec 1 ou 2 objets.

    Si l'on prend les 4 objets alors on dépasse le poids maximal. '''



## Question 2
''' Avec 1, 2 et 3: on a un profit de 8.
    Avec 1, 2 et 4: on dépasse le poids autorisé.
    Avec 1, 3 et 4: on a un profit de 14.
    Avec 2, 3 et 4: on a un profit de 13. '''



## Question 3
''' Le choix maximal est constitué des objets 1, 3 et 4 (poids de 8) et le profit est 14. '''



## Question 4

def ListeProduit(n):
    return [k for k in range(1, n+1)]


def ListeProduit_bis(n):
    return list(range(1, n+1))  # moins l'esprit du programme


def ListeProduit_ter(n):
    L = []
    for k in range(1, n+1):
        L.append(k)
    return L



## Question 5

def Ratio(P, V):
    return [V[k]/P[k] for k in range(len(P))]
```

```
def Ratio_bis(P, V):
    L = []
    for k in range(len(P)):
        L.append(V[k]/P[k])
    return L



## Question 6

''' Puisque len(L) = 4, il y a 3 itérations de la boucle for.
    Après le cas i = 1: L = [3, 5, 2, 1].
    Après le cas i = 2: L = [2, 3, 5, 1].
    Après le cas i = 3: L = [1, 2, 3, 5]. '''



##  Question 7

''' Cela ne fonctionne pas avec des chaînes de caractères
    car aux lignes 7 et 9 il y a des modifications qui ne sont pas
    possibles sur une chaîne de caractères (type "non mutable"). '''



## Question 8

''' Il s'agit du tri par insertion.

    Notons n la longueur de L.

    Il y a n-1 itérations de la boucle for.
    À chaque itération:
        - il y a 3 affectations
        - et une boucle while avec un test j>0 et une comparaison x < L[j-1].

    On compte le nombre de comparaisons du type x < L[j-1].

    Dans le meilleur des cas, la liste est déjà triée donc la condition
        x<L[i-1] est toujours fausse et il n'y a qu'une seule comparaison
        par itération de la boucle for.

    Donc dans le meilleur des cas, complexité : O(n).

    Dans le pire des cas, j diminue systématiquement jusqu'à 0
        c'est le cas d'une liste totalement en sens inverse.
        Il y a alors 1+2+..+(n-1) comparaisons i.e. (n-1)*n/2.

    Dans dans le pire des cas, complexité : O(n**2).
'''



## Question 9

def Inverse(L):
    M = []
    n = len(L) - 1
    while n >= 0:
        M.append(L[n])
        n = n - 1
    return M


def Inverse_bis(L):
    M = []
    for k in range(len(L)):
        M.append(L[len(L)-k-1])
    return M


def Inverse_ter(L):
    return [L[len(L)-k-1] for k in range(len(L))]
```

```
## Question 10
''' La fonction Tri ne trie qu'une seule liste
    or on veut trier P et V en fonction du tri de V/P
    donc il faut réécrire la fonction de tri en triant
    en même temps toutes les listes. '''



## Question 11

def Tri2(P, V):
    L = Ratio(P, V)
    for i in range(1,len(L)):
        p, v, x = P[i], V[i], L[i]
        j = i
        while j>0 and x<L[j-1]:
            P[j], V[j], L[j] = P[j-1], V[j-1], L[j-1]
            j=j-1
        P[j], V[j], L[j] = p, v, x
    return Inverse(P), Inverse(V)



## Question 12

def Vmax(P, V, Pmax):
    Ptrie, Vtrie = Tri2(P, V)
    Pcumul, Vcumul = 0, 0
    i = 0
    while i < len(P) and Pcumul + Ptrie[i] <= Pmax:
        Pcumul += Ptrie[i]
        Vcumul += Vtrie[i]
        i += 1
    return Vcumul



## Question 13
''' Les ratios valent: 4/3, 3/2, 1 et 9/4.
    La fonction Tri2 donne pour les ratios [2.25, 1.5, 1.33, 1],
        pour les poids [4, 2, 3, 1] et pour les valeurs [9, 3, 4, 1].
    La fonction Vmax renvoie alors 12 (car 4 + 2 <= 8 mais 4 + 2 + 3 > 8).
    Cette solution n'est donc pas optimale.
'''



## Question 14

''' Pour i=0, il n'y a aucun produit donc le profit est nul.

    Pour i>0 et p_i>omega, le produit n°i a un poids qui dépasse la capacité
        donc on ne le prend pas et le profit est le même qu'avec les i-1
        premiers produits.

    Sinon, on peut prendre le produit n°i:
        - si on le prend alors le profit est alors v_i + le profit maximal
            réalisable avec les i-1 premiers produits et un poids maximal
            de omega - p_i;
        - sinon le profit maximal est le même qu'avec les i-1 premiers produits.
    On considère donc le maximum de ces deux cas.
'''



## Question 15

''' Cas de base : i = 0.
    À chaque appel récursif, le premier argument est diminué de 1 donc
        atteint le cas de base en un nombre fini d'étapes.
'''
```

```python
## Question 16

def Max(a, b):
    if a > b:
        return a
    else:
        return b



## Question 17

def recur(P, V, i, w):
    """ entrées : liste P des poids, liste V des valeurs (avec len(P)==len(V)),
                  i entier entre 1 et len(P), w entier
        sortie : entier S(i,w) correspondant au profit maximal
    """
    if i == 0:
        return 0
    elif P[i-1] > w: # le poids de l'objet i est P[i-1]
        return recur(P, V, i-1, w)
    else:            # et la valeur de l'objet i est V[i-1]
        return Max(recur(P, V, i-1, w), V[i-1] + recur(P, V, i-1, w-P[i-1]))



## Question 18

assert recur([3, 2, 1, 4], [4, 3, 1, 9], 4, 8) == 14



## Questions 19 et 20

''' On choisit de ne pas suivre l'énoncé (à ne pas faire au concours !!!)
    en exploitant plutôt un dictionnaire pour la mémoization. '''


dico_opti = {}

def recur2(P, V, i, w):

    if (i, w) not in dico_opti:

        if i == 0:
            res = 0
        elif P[i-1] > w:
            res = recur2(P, V, i-1, w)
        else:
            res = max(recur2(P, V, i-1, w), V[i-1] + recur2(P, V, i-1, w-P[i-1]))

        dico_opti[(i, w)] = res

    return dico_opti[(i, w)]
```