DS1 - INFORMATIQUE
CORRIGE
OPTIMISATION DU RENDEMENT D'UN ENTREPRISE DE LIVRAISON

Q1. Si un chargement contient uniquement le produit 1, 2 ou 4, on peut lui rajouter le produit 3 (on ne dépasse pas le poids maximal) et le profit est alors plus grand. Si il contient le produit 3, on peut rajouter le produit 4 et conclure de la même manière. Si un chargement est constitué de deux produits parmi 1, 2 et 4, on peut faire de même que pour un produit (on peut rajouter le produit 3). Si un chargement est constitué de deux produits dont le troisième, son poids vaut au maximum 5 et on peut toujours lui ajouter un produit. Pour finir, la somme des poids des 4 produits dépasse le poids maximal.

Q2. Voici les cargaisons sous la forme (cargaison, profit) :
- $((1, 2, 3), 8)$
- $((1, 3, 4), 14)$
- $((2, 3, 4), 13)$

Q3. C'est $(1, 3, 4)$ et le profit vaut 14.

Q4.

```
def ListeProduits(n):
    return list(range(1,n+1))
```

ou :

```
def ListeProduits(n):
    L=[]
    for i in range(1,n+1):
        L.append(i)
   return L
```

```
def ListeProduits(n):
    return [i for i in range(1,n+1)]
```

Q5.

```
def Ratio(P,V):
    R=[]
    for i in range(len(P)):
        R.append(V[i]/P[i])
    return R
```

ou :

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

Q6. La longueur de L est 4 donc i varie entre 1 et 3. Il y a donc 3 itérations de la boucle for. Voici la valeur de L à la fin de chaque itération :
- $i = 1$. $L = [3, 5, 2, 1]$.
- $i = 2$. $L = [2, 3, 5, 1]$.
- $i = 3$. $L = [1, 2, 3, 5]$.

Q7. Non car on ne peut pas modifier une chaine de caractères.

Q8. C'est le tri par insertion.
Dans le pire des cas, la liste est triée par ordre décroissant. Si n est la longueur de la liste L alors pour chaque $i \in \{1, \dots, n-1\}$, il y aura $2i$ comparaisons dans la boucle while (car le i-ième terme de la liste sera plus petit que tous les précédents). Au final, il y aura :

$$2(1 + 2 + \cdots + n - 1) = n(n-1) \underset{+\infty}{\sim} n^2$$

comparaisons ce qui donne une complexité quadratique.
Dans le meilleur des cas, la liste est triée par ordre croissant. Si n est la longueur de la liste L alors pour chaque $i \in \{1, \dots, n-1\}$, il y aura 2 comparaisons (et la deuxième sera fausse car le i-ième terme de la liste sera plus grand que tous les précédents). Au final, il y aura $n - 1$ comparaisons ce qui donne une complexité linéaire.

Q9.

```
def Inverse(L):
    M=[]
    for i in range(len(L)):
        M.append(L[len(L)-i-1])
    return M
```

ou

```
def Inverse(L):
    M=[]
    for i in range(len(L)):
        M=[L[i]]+M
    return M
```

Q10. Le problème est la fonction de tri. Si on trie la liste des ratios, il faut trier en même temps (et de la même manière) la liste des poids et des valeurs.

Q11.

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

Q12. Il faut se méfier dans la condition du while. Il faut comparer `i` avec la longueur des listes avant de tenter d'évaluer `P[i]`.

```
def Vmax(P,V,Pmax):
    P2,V2=Tri2(P,V)
    SP=0
    SV=0
    i=0
    while (i<len(P)) and (SP+P2[i]<=Pmax):
        SP=SP+P2[i]
        SV=SV+V2[i]
        i=i+1
    return SV
```

Q13. On a $P = [3, 2, 1, 4]$, $V = [4, 3, 1, 9]$. La liste des ratios (à approximation près) est $R = [1.33, 1.5, 1, 2.25]$. Les listes de poids et de valeurs triées deviennent $P = [4, 2, 3, 1]$ et $V = [9, 3, 4, 1]$. La méthode permet donc seulement de considérer le chargement constitué de deux produits dont la somme des poids fait 600 kg et la valeur associée vaut 1200 euros. On obtient pas la valeur maximale déjà calculée qui est 1400 euros mais nous ne sommes pas si loin...

Q14. On distingue les trois cas :

- Si $i = 0$, aucun objet n'est placé dans le camion donc la valeur du chargement est 0.
- Si $i > 0$ et $p_i > \omega$, on ne peut pas placer l'objet i dans le chargement car son poids dépasse le poids maximal ω. Ainsi, il revient au même de chercher la valeur maximale en considérant uniquement les $i - 1$ premiers objets.
- Si $i > 0$ et $p_i \leq \omega$, soit on maximise la valeur en prenant l'objet i (dans ce cas la valeur maximale vaut v_i auquel on ajoute la valeur maximale obtenue avec les $i - 1$ premiers objets sachant que le poids maximal vaut maintenant $\omega - p_i$), soit on maximise sans l'objet i (ce qui revient au point précédent). On choisit alors le maximum des deux valeurs.

Q15. Chaque évaluation de la fonction fait décroitre la valeur de i de 1. Or i vaut n au départ donc après n tours, i vaut 0 et on renvoie alors la valeur 0 (évidemment, un appel de S peut faire intervenir 2 appels de S mais dans ce cas, i est decrémenté dans les deux cas).

Q16.

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

Q17. Voici une proposition :

```
def recur(P,V,i,w):
    if i==0:
        return 0
    if P[i-1]>w:
        return recur(P,V,i-1,w)
    return max(recur(P,V,i-1,w), V[i-1]+recur(P,V,i-1,omega-P[i-1]))
```

Q18. Voici une proposition :

```
P=[3,2,1,4]
V=[4,3,1,9]
Pmax=8
print(recur(P,V,4,Pmax))
```

Q19. Voici une proposition :

```
Memoire=[[-1 for j in range(Pmax+1)] for i in range(n+1)]
```

ou encore :

```
Memoire=[[-1]*(Pmax+1)]*(n+1)
```

Q20. Voici une proposition :

```
def recur2(P,V,i,w,Memoire):
    if i==0:
        return 0
    if Memoire[i][w]>-1:
        return Memoire[i][w]
    if P[i-1]>w:
        Memoire[i][w]=recur2(P,V,i-1,w,Memoire)

    return Memoire[i][w]
else:
    if Memoire[i-1][w]==-1:
        Memoire[i-1][w]=recur2(P,V,i-1,w,Memoire)
    if Memoire[i-1][w-P[i-1]]==-1:
        Memoire[i-1][w-P[i-1]]=recur2(P,V,i-1,w-P[i-1],Memoire)
    a=max(Memoire[i-1][w],V[i-1]+Memoire[i-1][w-P[i-1]])
    Memoire[i][w]=a
    return Memoire[i][w]
```

Q25. 000100111.

Q26. Il faut convertir `Bin[i]` avec `int` et l'exposant de 2 doit être `len(Bin)-i-1`.

Q27. Voici une proposition :

```
def Identifiant2(Bin):
    '''Bin est est une chaine de caractères
     constituée de 0 et 1'''
    S=0
    Inv=Bin[::-1]
    K=1
    for i in range(len(Inv)):
        S=S+int(Inv[i])*K
        K=2*K
    return S
```

et une autre :

```
def Identifiant2(Bin):
    s=int(Bin[0])
    for i in range(1,len(Bin)):
        s=s*2+int(Bin[i])
    return s
```

Q28. Il faut 5 bits.

Q25. $163_{10} = 10100011_2 = A3_{16}$

Q26. Et 27

```
def Hex(L):
    H=['0000','0001','0010','0011','0100','0101','0110','0111','1000','1001','1010','1011','1100','1101','1110','1111']
    for i in range(len(H)):
        if H[i]==L :
            if i<9 : return str(i)
            else : return chr(i-10+65)
    return ''

def conversion(ch):
    s=''
    for i in range(0,len(ch),4):
        tamp=ch[-i-4:len(ch)-i]
        while len(tamp)<4:
            tamp='0'+tamp

        s=Hex(tamp)+s

    return s

#test
print(conversion('01010101010'))#2AA
print(conversion('11111111'))#FF
print(conversion('11'))#3
print(conversion('11100000001111000111101101'))#701E3ED
```