I Rendu de monnaie

1. On détermine la plus grande pièce utilisable puis la monnaie à rendre pour les reste récursivement et on rajoute (concaténation de listes) la pièce initialement trouvée.

```
def glouton(s,P) :
    if s == 0 :
        return []
    else :
        i = 0
        while P[i] > s :
            i += 1
        return [P[i]] + glouton(s-P[i],P)
```

2. On trouve respectivement `[10,10,10,10,5,2,2]` et `[30,12,6,1]` qui n'est pas optimal dans le second cas puisque $49 = 24 + 24 + 1$ donc on pouvait rendre la monnaie avec 3 pièces seulement

3. $N_s = 1 + \min\{N_{s-p_i}, i \in [\![1,k]\!], s - p_i \geqslant 0\}$

4. Une fonction inefficace par récursivité multiple : vous pouvez la tester, l'exécution de `rendu(49,P1)` avec `P1=[10,5,2,1]` prend plusieurs minutes

```
def rendu(s,P) :
    if s == 0 :
        r = 0
    else :
        m = s # le futur minimum, il est <= s
        for k in P :
            if k <= s :
                m1 = rendu(s-k,P)
                if m1 < m :
                    m = m1
        r = 1+m
    return r
```

5. On introduit un dictionnaire dont les clefs sont les valeurs de s pour éviter les calculs multiples des valeurs de N_s dans la fonction précédente : cette fois l'exécution de `renduMemo(49,P1)` avec `P1=[10,5,2,1]` devient instantanée

```
def renduMemo(s,P) :
    d = {}
    def f(s,P) :
        if s not in d :
            if s == 0 :
                r = 0
            else :
                L = []
                for k in P :
                    if k <= s :
                        L.append(f(s-k,P))
                r = 1+min(L)
            d[s] = r
        return d[s]
    return f(s,P)
```

6. On remplit `L` au fur et à mesure :

```
def BasEnHaut(s,P) :
    L=[0]
    for i in range(1,s+1) :
        m = i # la futur valeur de L[i] = min ...
        for k in P :
            if k <= i :
                if L[i-k] < m :
                    m = L[i-k]
        L.append(1+m)
    return L[s]
```

7. Si on note k le nombre de pièces alors $C(s) = O(ks)$ donc linéaire si on considère le nombre de pièces fixé.

II Suite de Syracuse

1. On suit la définition de (u_n) :

```
def Syracuse(x,n) :
    if n == 0 :
        return x
    else :
        y = Syracuse(x,n-1)
        if y%2 == 0 :
            return y//2
        else :
            return 3*y+1
```

2. La boucle `while` s'arrête si on admet la conjecture de Syracuse :

```
def tempsVol(x) :
    n = 0
    while Syracuse(x,n) != 1 :
        n += 1
    return n
```

3. On calcule le temps de vol maximal au fur et à mesure :

```
def tempsVolMax(N) :
    M = 0
    for k in range(1,N+1) :
        t = tempsVol(k)
        if t > M :
            M = t
    return M
```

4. La fonction précédente n'est pas efficace car elle calcule plusieurs fois le même temps de vol : on calcule par exemple le temps de vol en $x = 3$ que l'on recalculera dans le temps de vol en $x = 6$ puisque si $u = 0 = 6$ alors $u_1 = 3$. On a la relation suivante sur les temps de vol :

$$tv(x) = \begin{cases} 0 & \text{si } x = 1 \\ 1 + tv(\varphi(x)) & \text{sinon} \end{cases},$$

si f est la fonction telle que $u_{n+1} = \varphi(u_n)$. On peut donc calculer le temps de vol récursivement et utiliser la mémoïsation pour éviter les calculs redondants :

```
def tempsVolMaxMemo(N) :
    d = {}
    def f(x) :
        if x not in d :
            if x == 1 :
                r = 0
            else :
                if x%2 == 0 :
                    y = x//2
                else :
                    y = 3*x+1
                r = 1+f(y)
            d[x] = r
        return d[x]
    M = 0
    for x in range(1,N+1) :
        if f(x) > M :
            M = f(x)
    return M
```

5. Le problème vient du fait qu'on peut avoir $u_{n+1} > u_n$ donc si on veut remplir une liste `L` de sorte que `L[i]` soit le temps de vol en `i`, cette valeur ne dépend pas que des valeurs de `L[j]` avec $j \leqslant i-1$ donc on pourrait être amené à calculer `L[3+i+1]` qui obligerait à compléter la liste `L` avec des valeurs peut être inutiles (il faudrait rajouter des valeurs nulles par exemples entre les indices $i+1$ et $3i$, si elles n'ont pas déjà été calculées). Le problème vient du fait que l'on ne sait pas avant de faire le calcul quelle va être la longueur de la liste à utiliser pour déterminer le temps de vol en x.