Correction TD4

1. On a $S = s_1 + s_2$ donc $|s_1 - s_2| = 2|s_1 - S/2|$

2. Il y a 2^n sous ensembles de `E` donc une complexité exponentielle.

3. À cause de la récursivité double, on a $C_n \geqslant 2C_{n-1}$ donc à nouveau une complexité exponentielle.

4. a) Il faut faire attention à avoir `j-E[i]` $\geqslant 0$

```
def tableau(E) :
    S = sum(E)
    T = [[False for _ in range(S+1)] for _ in range(len(E)+1)]
    T[0][0] = True
    for i in range(len(E)) :
        v = E[i]
        for j in range(S+1) :
            T[i+1][j] = T[i][j] or (j-v>=0 and T[i][j-v])
    return T
```

b) On ne peut pas obtenir la somme `m` avec les éléments de `E[:i]` mais on peut l'obtenir avec ceux de `E[:i+1]` donc en utilisant l'élément `E[i]` ; les autres éléments sont à chercher dans `E[:i]`, afin d'obtenir ensuite la somme `m-E[i]`.

c)
```
def partition(E) :
    T = tableau(E)
    L = []
    # détermination de m
    m = sum(E)//2
    while not T[len(E)][m] :
        m -= 1
    # reconstruction de l'ensemble E1
    while m > 0 :
        # recherche du premier élément de E1
        i = 1
        while not T[i][m] :
            i += 1
        v = E[i-1]
        L.append(v)
        # on recommence avec la somme m-E[i]
        m -= v
    return L
```

d) La complexité de la création du tableau est $O(n \times S)$, la première boucle `while` en $O(S)$ et la seconde (2 boucles imbriquées) en $O(n \times S)$ car `m` $\leqslant S/2$ et `i` $\leqslant n$. La complexité totale est donc $O(n \times S)$.

5. La fonction `p` utilise deux paramètres donc on utilise un dictionnaire dont les clés sont les couples `(E,S)` mais il faut faire attention qu'une liste n'est pas hachable (donc pour contourner le problème, on peut placer les listes dans une chaîne de caractère, même s'il y aurait plus efficace)

```
def partition2(E) :
    d = {}
    S = sum(E)/2
    def part(E,S) :
        if (str(E),S) not in d :
            if len(E) == 1 :
                r = E
            else :
                E1 = part(E[1:],S)
                s1 = sum(E1)
                E2 = part(E[1:],S-E[0])
                s2 = sum(E2)+E[0]
                if abs(s1-S)<abs(s2-S) :
                    r = E1
                else :
                    r = [E[0]]+E2
            d[(str(E),S)] = r
        return d[(str(E),S)]
    return part(E,S)
```