Équilibrage d'un bateau lors de l'embarquement des passagers

On fixe pour tout le problème n un entier naturel non nul, ie $n \in \mathbb{N}^*$.
On souhaite faire embarquer dans un bateau un groupe `w` de n individus.
Les personnes se positionnent à bâbord ou à tribord.
L'objectif est de répartir au mieux les masses entre les deux côtés.
On désignera par liste d'individus toute liste `w` de taille n formée d'entiers strictement positifs.
Pour tout $i \in [\![0, n-1]\!]$, `w`$[i]$ représentera la masse de l'individu i.
Un embarquement `emb` d'une liste d'individus `w` désignera une liste de taille n composée de 0 ou de 1. Soit $i \in [\![0, n-1]\!]$:

- `emb`$[i] = 0$ signifiera que l'individu i monte à bâbord.
- `emb`$[i] = 1$ signifiera que l'individu i monte à tribord.

De plus :

- `mb` désignera la masse totale des personnes positionnées à `babord`.
- `mt` désignera la masse totale des personnes positionnées à `tribord`.

1 Tri fusion (merge sort)

Le principe du tri fusion est ici le suivant :
◇ Partitionner une liste en deux listes de tailles proches.
◇ Trier récursivement les sous-listes.
◇ Fusionner correctement les deux sous-listes triées .

Question 1 :

Écrire une fonction **récursive** `fusion_rec(lst1 : list, lst2 : list) -> list` prenant en entrée deux listes `lst1` et `lst2` supposées triées et retournant une liste triée obtenue en fusionnant `lst1` et `lst2`.

Question 2 :

Ecrire une fonction **récursive** `tri_fusion_rec(lst : list) -> list` prenant en entrée une liste `lst` et retournant la liste obtenue en triant par ordre croissant les éléments `lst` en utilisant l'algorithme de tri fusion décrit ci-dessus.

2 Équilibrages naïfs

Une première stratégie est d'équilibrer « à la volée ». Les personnes montent à bord du bateau une à une, et sont envoyées du côté où le poids total est le plus faible.

On souhaite obtenir une fonction `equilibrage1` prenant entrée une liste d'individus `w` et retournant le triplet `emb`, `mb`, `mt` où `emb` est l'embarquement de `w` obtenu en équilibrant « à la volée » sachant que la première personne monte à bâbord.

```
>>> equilibrage1([8,7,5,2,19,5])
([0, 1, 1, 0, 0, 1], 29, 17)
```

Question 3 :

Écrire `equilibrage1(w : list) -> (list ,int ,int)`.

Question 4 :
Que retourne `equilibrage1`$([3,5,6,1])$? Le bateau aurait-il pu être mieux équilibré ?

Améliorons l'algorithme en faisant embarquer les personnes, toujours « à la volée » mais par poids décroissants. Le classement du groupe d'individus provoque un renommage (sans importance) des personnes.

Question 5 : En utilisant cette méthode, écrire une fonction `equilibrage2(w : list) -> (int ,int)` prenant entrée une liste d'individus `w` et retournant le couple `mb`, `mt` .
On utilisera impérativement `tri_fusion_rec` et `equilibrage1`.

Question 6 :
Proposer une liste w_2 de cinq individus rangés par ordre de poids décroissants telle que `equilibrage2`(w_2) ne soit pas un rangement optimal.

3 Équilibrage et Force brute

Question 7 :
Écrire une fonction `calcul_des_masses(w : list, emb : list) -> (int, int)` prenant entrée une liste d'individus `w`, un embarquement `emb` de `w` et retournant (`mb`, `mt`)

```
>>> calcul_des_masses([2,3,48,5], [1,0,0,1])
(51, 7)
```

Question 8 :
Écrire une fonction **récursive** `listes_binaires(n : int) -> list` prenant entrée un entier $p \in \mathbb{N}$ et retournant la liste de toutes les listes binaires à p éléments classée par l'ordre lexicographique.

```
>>> listes_binaires(0)
[[]]
>>> listes_binaires(3)
[[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1],
 [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]
```

Question 9 :
Donner la longueur de la liste `liste_embarquements`(n).

Question 10 :
En déduire une fonction `liste_embarquements(n : int) -> list` prenant entrée un entier $n \in \mathbb{N}^*$ non nul et retournant la liste des listes de tous les embarquements, **débutant à bâbord**(au moins une personne monte à tribord), pour un groupe de n individus.

```
>>> liste_embarquements(1)      [[0]]
>>> liste_embarquements(2)      [[0, 1]]
>>> liste_embarquements(3)      [[0, 0, 1], [0, 1, 0], [0, 1, 1]]
>>> liste_embarquements(4)
[[0, 0, 0, 1], [0, 0, 1, 0], [0, 0, 1, 1], [0, 1, 0, 0],
 [0, 1, 0, 1], [0, 1, 1, 0], [0, 1, 1, 1]]
```

Question 11 :

Écrire une fonction `force_brute (w : list -> (list, int, int)` prenant en entrée une liste d'individus `w` et retournant le triplet `emb`, `mb`, `mt` où `emb` est l'embarquement optimal de `w` obtenu en examinant tous les embarquements possibles (débutant par convention à bâbord).

```
>>> force_brute([8,7,5,2,19,5])
([0, 0, 0, 0, 1, 1], 22, 24)
```

Question 12 :

Donner la complexité de force brute et commenter.

4 Équilibrage et Programmation dynamique bottom-up/tabulation

On utilise ici une méthode de programmation dynamique consistant à diviser un problème en sous- problèmes qui se chevauchent et en mémorisant ce qui a déjà été calculé.
On a ici une approche ascendante (bottom-up), les solutions de sous-problèmes de plus en plus grands sont obtenus dans un certain ordre, gardés en mémoire (tabulation).
La solution du problème d'origine (top solution) est obtenue en fin de chaîne.

`w` est une liste de n d'individus. Notons $m = \sum_{i=0}^{n-1} \texttt{w}[i]$ et posons $\boxed{md = m//2}$.

md représente la masse idéale du groupe de personnes embarquant d'un côté.

Question 13 :

Écrire une fonction `masse_ideale(w : list) -> int` prenant en entrée une liste d'individus `w` et retournant la valeur de md.

A chaque `w`, on peut associer un unique tableau de booléens `M` à n lignes et $md+1$ colonnes tel que, pour tout $(i,j) \in [\![0, n-1]\!] \times [\![0, md]\!]$, `M`$[i,j]$ vaut `True` s'il existe un sous-ensemble de $\{0,1,...,i\} = [\![0,i]\!]$ dont le poids total est exactement j et `False` sinon, ie :

$$\boxed{\texttt{M}[i,j] = \texttt{True} \iff \exists I \subset [\![0,i]\!] : \sum_{k\in I} \texttt{w}[k] = j}$$

On appellera `creation_M` la fonction prenant en entrée une liste d'individus `w` et retournant le tableau `M` de booléens associé.

Question 14 :

Que vaut `creation_M`$([4,2,2,1,2])$? On pourra abréger `True` en `T` et `False` en `F`.

Question 15 :

Soit $(i,j) \in [\![1, n-1]\!] \times [\![0, md]\!]$. Écrire une relation de récurrence exprimant le booléen `M`$[i,j]$ en fonction de deux booléens de la ligne $M[i-1]$ et de $(j \geqslant w[i])$.

Question 16 :

Écrire la fonction `creation_M(w : list) -> list`. prenant en entrée une liste d'individus `w` et retournant le tableau `M` de booléens associé.
On pourra prévoir un test d'arrêt lorsque toutes les lignes du tableau ont été mises à jour **ou** lorsque que l'on a trouvé un groupe de personnes dont le poids total est exactement md.

Question 17 :

Écrire une fonction `recuperation_couple(M : list) -> (int, int)` prenant en entrée un tableau `M` de booléens à n lignes dont la dernière ligne contient au moins une fois la valeur `True` et retournant le couple $(n-1, jj)$ tel que :

- $\texttt{M}[n-1, jj] = \texttt{True}$
- $\forall k > jj$, $\texttt{M}[n-1, k] = \texttt{False}$.

En fait $jj = \max(j \in [\![0, md]\!] : \texttt{M}[n-1, j] = \texttt{True})$.

On considère avoir obtenu un couple $(ii, jj) \in [\![0, n-1]\!] \times [\![0, md]\!]$ tel que :

- $\texttt{M}[ii, jj] = \text{True}$
- $\forall k > jj$, $\forall \ell \geqslant ii$, $\texttt{M}[\ell, k] = \text{False}$
- $\texttt{M}[ii-1, jj] = \text{False}$ si $ii > 0$.

(ii, jj) correspond à la position dans M la plus à droite et la plus basse valant `True`.
On souhaite ensuite construire la liste `lst` des couples (`personne_k`, `masse_k`) embarquant à tribord pour équilibrer au mieux le bateau.
Précisément, la liste `lst` est formée des couples $(i, \texttt{w}[i])$ où $i \in I$ avec I sous-ensemble de $\{0, 1, ..., ii\}$ dont le poids total est exactement jj.
La personne ii fera partie du sous groupe I.
On cherche ensuite le **plus petit** $i' \in [\![0, ii-1]\!]$ tel que $\texttt{M}[i', jj - w[ii]] = \text{True}$.
i' fera lui aussi partie de I et ainsi de suite...

Question 18 :

Recopier et compléter :

```
def rech_partitions_equilibrees((w : list) -> list):
    lst = []
    md = #COMPLETER      #definition de md
    M = #COMPLETER   #M est le tableau de booléens de w
    ii , jj = #COMPLETER  # M[n - 1, jj] est vrai
    #A ce stade M[ii - 1, jj] peut etre vrai aussi
    #recherche de la premiere ligne ii telle que M[ii , jj] vraie
     while jj > 0:
         while ii > 0  and M[ii - 1, jj]:
             #COMPLETER (avec une seule ligne)
        lst. append((ii , w[ii]))
        # mise à jour de jj , ii
        jj #COMPLETER
        ii #COMPLETER
    return lst
```

Remarque 1 :

En devoir, on pensera à changer de **couleur** pour mettre en valeur les parties de code complétées.