Exercices algorithmiques

⚠️ **Consignes principales :**

- Les exercices doivent être faits dans des fonctions JavaScript

- Ils sont exécutés sur Node → Pas de fichier HTML !
- Nous valorisons la mise en place de tests unitaires avec plusieurs cas

Algorithme : Rends l'argent ! (facile)

Soit une liste **L** de nombre décroissants représentant les valeurs des billets et pièces disponibles dans un pays (ex: 500, 200, 100, 50, 20, 10, 5, 2, 1, 0.5, ...). Ecrire un algorithme qui à partir d'un montant donné **M**, donne le nombre le plus petit **P** de billets et de pièces dont la somme vaut ce montant, ou -1 si ce n'est pas possible d'atteindre ce montant avec ces billets et pièces.

Bien vérifier des cas de tests simples (autres que ceux donnés)...

▼ Specs & Exemple

L : un tableau de flottants strictement positifs, strictement décroissants, non vide.
M : un montant positif.
P : résultat

```
// Exemple 1
T = [500, 200, 100, 50, 20, 10, 5, 2, 1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01]
M = 626.5
P = 6         (500 + 100 + 20 + 5 + 1 + 0.5)

// Exemple 2
T = [500, 200, 100, 50, 20, 10, 5]
M = 626.5
P = -1        (500 + 100 + 20 + 5, mais impossible de faire le reste)
```

Algorithme : Des 1 et des 0 à devenir fou ! (moyen)

À partir d'un nombre entier positif **N**, donner la longueur **L** de la séquence la plus longue de 0 qui compose sa forme binaire. Par exemple pour 296, sa forme binaire est 100101000, et le résultat attendu est 3.

> 🔥 Il n'est pas autorisé d'utiliser la fonction interne `Number.toString(base)` pour traduire le nombre en binaire, ou alors seulement si blocage il y a. Ne pas oublier les cas limites !

▼ Specs & Exemple

M : entier positif (0 ≤ M ≤ 2^32)
L : résultat

```
// Exemple 1
M = 123456
L = 6 (car 11110001001000000 en binaire)

// Exemple 2
M = 65535
L = 0 (car 1111111111111111 en binaire)
```

Algorithme : Pic pic pic ! (difficile)

Une poule est sur un échiquier **E** de hauteur **H** et de largeur **L**. Elle est placée sur la case en haut à gauche d'indice (0,0). Sur chaque case d'indice **(i,j)** il y a un certain nombre de graines **Eij**. Elle peut faire seulement deux mouvements, aller sur la case en dessous, ou sur celle à droite. Écrire un algorithme qui permet de savoir quel est le nombre maximal **M** de graines que la poule peut manger en rejoignant la case en base à droite d'indice (H-1, L-1).

Une attention particulière doit être faite sur la performance de l'algorithme. Il existe une solution récursive mais ce n'est pas la plus simple. Un pseudo code sera toujours apprécié !

▼ Specs & Exemple

E = (Eij) avec i, j > 0 et Eij ≥ 0 (E est un tableau à deux dimensions)
M : résultat

```
// Exemple 1
E = [[1, 2, 3],
     [4, 5, 6],
     [7, 0, 9]]
M = 25  (car 1 -> 4 -> 5 -> 6 -> 9)

// Exemple 2
E = [[0, 0, 0, 0],
     [0, 1, 0, 0],
     [0, 0, 1, 0],
     [0, 0, 0, 1]]
M = 3   (car 0 -> 0 -> 1 -> 0 -> 1 -> 0 -> 1)
```