NOM : [redacted]

Prénom : [redacted]

IUD

Groupe

ALGO-AR2 - exam B5 - 21/10/2025 **Réponses**

***Réponses 1* (Level up – *5 points*)**

15,5

Spécifications :

`level_up(T:Tree)` vérifie que chaque niveau de l'arbre T contient strictement plus de nœuds que le niveau précédent.

3,5

```
def level_up(B):
    q = Queue()
    q.enqueue(B)
    q.enqueue(None)
    current_level = []
    levels = []
    while not q.isempty():
        n = q.dequeue()
        if n is None:
            if len(levels) > 0 and len(levels[-1]) > len(current_level):
                return False
            levels.append(current_level)
            current_level = []
            if not q.isempty():
                q.enqueue(None)
        else:
            current_levels.append(n)
            for c in n.children:
                q.enqueue(c)
    return True:
```

Tu pourrais juste compter au lieu de stocker tout ! Ça ferait 2 variables au lieu d'une liste de listes pouvant contenir de cases que la taille de l'arbre

Réponses 2 **(Check sum – *6 points*)**

Spécifications :

`check_sum(B:TreeAsBin)` vérifie si dans l'arbre `B` la clé de chaque nœud interne est égale à la somme des clés de ses fils.

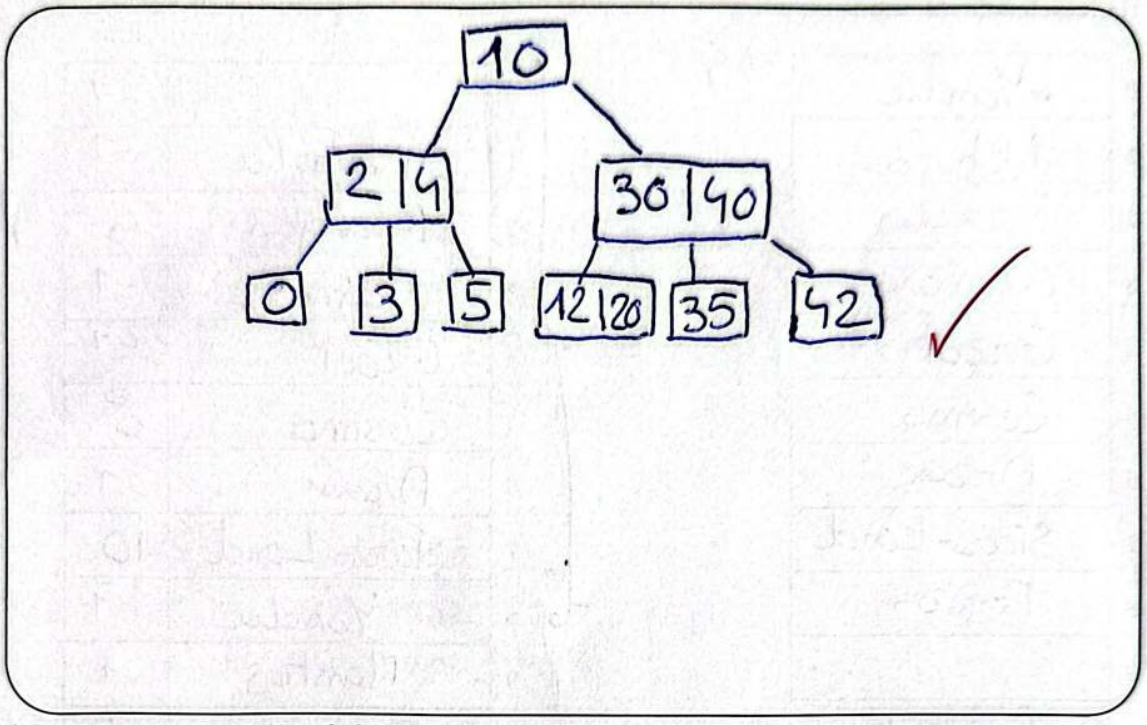
```
def check_sum(B):
    S = B.key
    c = B.child
    while c != None and check_sum(c):
        S -= c.key
        c = c.sibling
    return c == None and S == 0
```

6

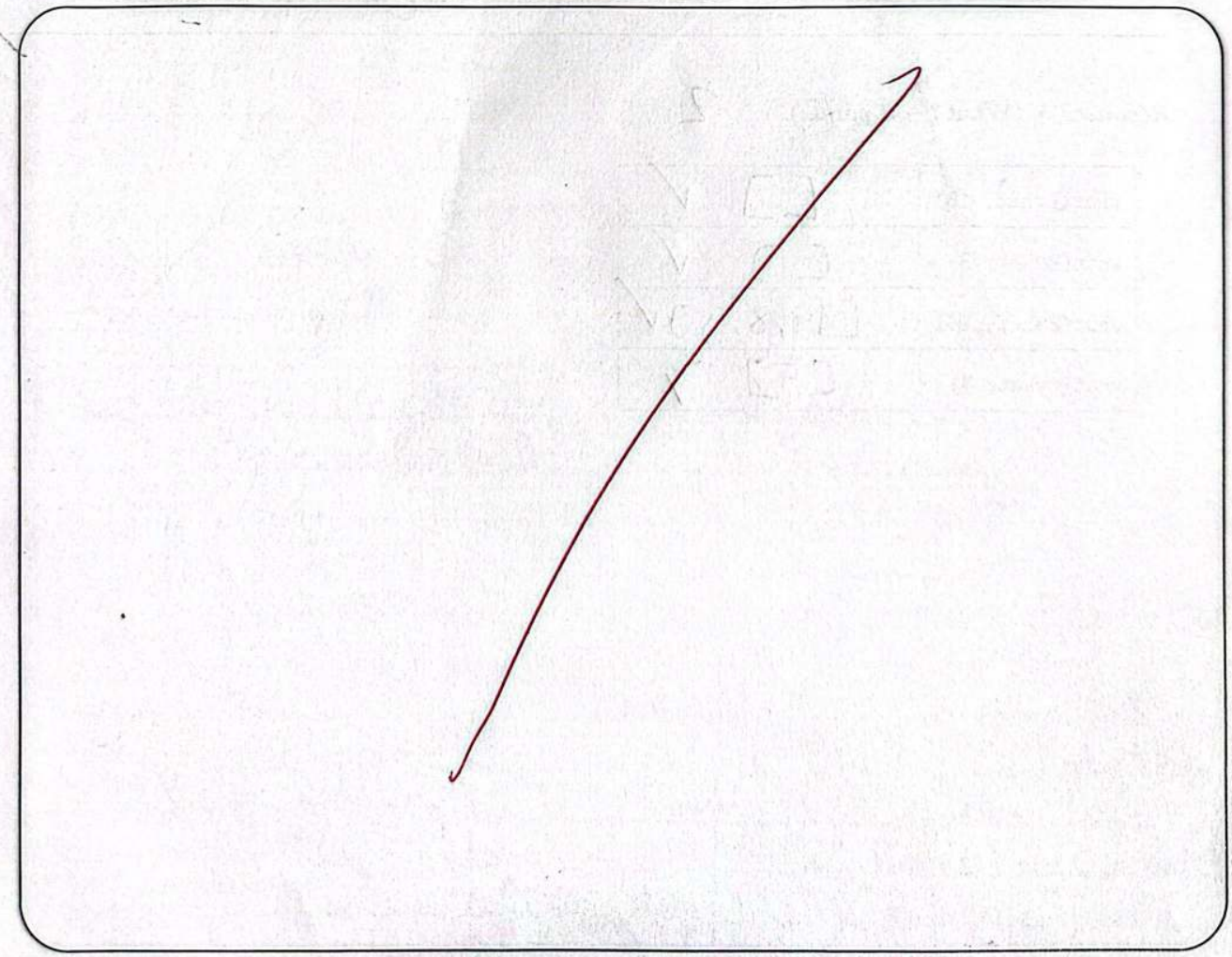
```
def check_sum(B):
    if B.child is None:
        return True
    else:
        S = 0
        c = B.child
        while c != None and check_sum(c):
            S += c.key
            c = c.sibling
        return c == None and S == B.key
```

Réponses 3 **(B-arbre : insertions et suppression –** ***3 points*****)**

1. L'arbre B1 après insertions des valeurs 42, 12 et 2 :



2. Arbre B2 après suppression de la valeur 20 :



Réponses 4 **(We are Groot – *3 points*)**

1. Hachage linéaire ($d = 3$) :

0	Yondu
1	Nebula
2	Kraglin
3	Gamora
4	Groot
5	Cosmo
6	Drax
7	Star-Lord
8	Mantis
9	
10	Rocket

1,5

2. Hachage coalescent :

1

	elts	liens
0		-1
1	Nebula	-1
2	Kraglin	9
3	Gamora	-1
4	Groot	-1
5	Cosmo	8
6	Drax	-1
7	Star-Lord	10
8	Yondu	-1
9	Mantis	-1
10	Rocket	-1

R

Utiliser la valeur -1 comme "fin de chaînage".

Réponses 5 **(What ? – *3 points*)** 2

`what(Bwhat, 15)`	[] ✓
`what(Bwhat, 8)`	[] ✓
`what(Bwhat, 10)`	[14,8,9] ✓
`what(Bwhat, 4)`	[5] X

NOM : _____ IUD

***Réponses 4* (Euler – 7 *points*)**

Spécifications :

La fonction `Euler(G)`, avec G un graphe non orienté simple, retourne
- si G possède une chaîne eulérienne, une liste contenant ses deux extrémités ;
- si G possède un cycle eulérien, une liste vide ;
- sinon, la valeur `None`.

```
def aux_Euler(G, i, M):
    M[i] = len(G.adjlists[i])
    for y in G.adjlists[i]:
        if M[y] == None:
            aux_Euler(G, y, M)

def Euler(G):
    N = G.order
    M = [None] * N
    E = []
    aux_Euler(G, 0, M) # On part du sommet 0
    for i in range(N):
        if M[i] == None:
            return None
        if M[i] % 2 == 1: # degré impaire
            E.append(M[i])
    return E if len(E) == 0 or len(E) == 2 else None
```

b

& retire

```
def aux_Euler(G, i, M, E):
    L = len(G.adjlists[i])
    M[i] = L
    if L % 2 == 1:
        if len(E) > 2:
            return False
        else:
            E.append(L)
    for y in G.adjlists[i]:
        if M[y] is None:
            if not aux_Euler(G, y, M, E):
                return False
    return True
```

Récupérer dans le MS aussi

```
def Euler(G):
    n = G.order
    M = [None] * n
    E = []
    if not aux_Euler(G, 0, M):
        return None
    for i in range(n):
        if M[i] == None:
            return None
    if len(E) == 0 or len(E) == 2:
        return E
    return None
```

test inutile tu aurais déjà renvoyé None si pb dans ce c