+1/1/60+

EPFL

1

Enseignant.e.s: G. Maatouk, L. Testa
ICS - CMS
8 janvier 2025
Durée : 105 minutes

Abra Kadabra

SCIPER : **987654**

Signature

☐ Absent.e

Attendez le début de l'épreuve avant de tourner la page. Ce document est imprimé recto-verso, il contient 10 questions et 12 pages, les dernières pouvant être vides. Le total est de 35 points. Ne pas dégrafer.

- Posez votre **carte d'étudiant.e** sur la table.
- Vérifiez votre nom et numéro SCIPER sur la première page et apposez votre **signature**.
- **Aucun** document n'est autorisé.
- Pour les questions à **choix unique**, on comptera :
  - les points indiqués si la réponse est correcte,
  - 0 point si il n'y a aucune ou plus d'une réponse inscrite,
  - 0 point si la réponse est incorrecte.
- Si une question est erronée, les enseignant.e.s se réservent le droit de l'annuler.
- Les algorithmes demandés sont à écrire sous forme de fonctions Python. Mettez votre code en forme en respectant les indentations: 4 carreaux = 1 tab.
- Vous n'avez pas besoin de commenter votre code mais vous pouvez le faire si vous pensez que cela aide à sa compréhension.
- Utilisez un **stylo** à encre **noire ou bleu foncé** et effacez proprement avec du **correcteur blanc** si nécessaire.
- Répondez dans l'espace prévu (**aucune** feuille supplémentaire ne sera fournie).
- Les brouillons ne sont pas à rendre: ils ne seront pas corrigés.

Respectez les consignes suivantes | Observe this guidelines | Beachten Sie bitte die unten stehenden Richtlinien

choisir une réponse | select an answer Antwort auswählen	ne PAS choisir une réponse | NOT select an answer NICHT Antwort auswählen	Corriger une réponse | Correct an answer Antwort korrigieren
☒ ☑ ■	☐	☐

ce qu'il ne faut **PAS** faire | what should **NOT** be done | was man **NICHT** tun sollte

Première partie, questions à choix unique

Pour chaque question, marquer la case correspondante à la réponse correcte sans faire de ratures. Il n'y a qu'**une seule** réponse correcte par question.

Question 1 (2 points)
Rappelons les implémentations des algorithmes de fusion et de tri par fusion.

```
def fusion(L, bas, milieu, haut):
    L1 = L[bas:milieu+1]
    L2 = L[milieu+1:haut+1]
    L1.append(float('inf'))
    L2.append(float('inf'))
    L1_index = 0
    L2_index = 0
    for i in range(bas, haut+1):
        if L1[L1_index] <= L2[L2_index]:
            L[i] = L1[L1_index]
            L1_index += 1
        else:
            L[i] = L2[L2_index]
            L2_index += 1
```

```
def tri_par_fusion(L, bas, haut):
    if haut - bas > 0:
        milieu = (bas + haut)//2
        tri_par_fusion(L, bas, milieu)
        tri_par_fusion(L, milieu+1, haut)
        fusion(L, bas, milieu, haut)
```

On considère la liste `L = [1, 3, 4, 6, -1, 8, 5, 2]`. Quelle est la quatrième paire de sous-listes à être fusionnée lors de l'appel `tri_par_fusion(L, 0, len(L)-1)`?

- ☐ `[1]` et `[3]`
- ☐ `[-1]` et `[8]`
- ☐ `[4]` et `[6]`
- ☐ `[1, 3]` et `[4, 6]`
- ☐ `[5]` et `[2]`
- ☐ `[1, 3, 4, 6]` et `[-1, 8, 5, 2]`
- ☐ `[1, 3, 4, 6]` et `[-1, 2, 5, 8]`
- ☐ `[-1, 8]` et `[2, 5]`

Question 2 (2 points)
On donne le code suivant.

```
def mafonc(n):
    s = 0
    i = n
    while i >= 1:
        s += i
        i //= 3
    return s
```

Quelle affirmation parmi les suivantes est correcte?

- ☐ `mafonc(9) = 3 * mafonc(3)`
- ☐ `mafonc(9) = mafonc(11)`
- ☐ `mafonc(9) = 9 + mafonc(2)`
- ☐ `mafonc(11) = 11 + mafonc(3)`

Question 3 (2 points)

On considère les deux fonctions suivantes définies sur les entiers strictements supérieurs à 1:

$$f(n) = \frac{1}{(\log_2 n)^3 + 2\log_2 n} \quad \text{et} \quad g(n) = \frac{10}{n \log_2 n}.$$

Laquelle des affirmations suivantes est vraie?

- ☐ $f(n) = \mathcal{O}(g(n))$ mais $g(n)$ n'est pas $\mathcal{O}(f(n))$
- ☐ On ne peut pas comparer l'ordre de croissance de f et de g
- ☐ $f(n) = \Theta(g(n))$
- ☐ $g(n) = \mathcal{O}(f(n))$ mais $f(n)$ n'est pas $\mathcal{O}(g(n))$

Question 4 (2 points)
Un tableau a été dérobé au musée du Louvre. La police cherche à identifier l'heure du crime et se base pour cela sur 225 images correspondant aux dernières 225 minutes, à raison d'une image par minute. Le tableau est présent sur la première image, mais n'apparait pas sur la dernière.
Combien d'images au minimum doivent parcourir les enquêteurs pour être certains de découvrir l'heure du crime à la minute près ?

☐ 225 ☐ 15 ☐ 64 ☐ 8

Question 5 (2 points)
Rappelons l'implémentation de l'algorithme de recherche binaire telle que vue en cours, à laquelle nous avons rajouté l'instruction `print(milieu, end = ' ')`.

```
def recherche_binaire(L, x):
    n = len(L)
    bas = 0
    haut = n-1
    while haut >= bas:
        milieu = (bas + haut)//2
        print(milieu, end=' ')
        if L[milieu] == x:
            return milieu
        elif L[milieu] > x:
            haut = milieu-1
        else:
            bas = milieu + 1
```

Soit la liste `L = [-10, -7, -4, -3, 0, 1, 5, 8, 9, 13, 17, 21, 25]`. Un appel à `recherche_binaire(L,x)` renvoie l'affichage suivant : `6 9 11 10`.

Que peut-on en déduire sur la valeur de `x`?

☐ $14 \leq x \leq 20$ ☐ $x = 17$ ☐ $14 < x \leq 17$ ☐ $17 \leq x < 20$

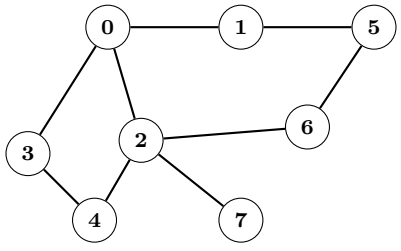
Question 6 (2 points)
On rappelle l'algorithme `BFS_chemins` vu en cours.

```
from collections import deque

def BFS_chemins(G,s):
    n = len(G)
    a_parcourir = deque([s])
    vu = [0 for u in range(n)]
    vu[s] = 1
    chemin = [[] for u in range(n)]
    chemin[s] = [s]
    while a_parcourir:
        sommet = a_parcourir.popleft()
        for u in G[sommet]:
            if not vu[u]:
                a_parcourir.append(u)
                vu[u] = 1
                chemin[u] = chemin[sommet].copy()
                chemin[u].append(u)
    return chemin
```

Sachant que l'appel `BFS_chemins(G, 0)` retourne la liste
`[[0], [0, 1], [0, 2], [0, 2, 3], [0, 1, 4], [0, 2, 3, 5]]`
lequel des dictionnaires ci-dessous **ne peut pas** être une représentation par listes d'adjacence de `G`? Il peut être utile de dessiner les graphes.

- ☐ `G = {0:[1, 2], 1:[0, 2, 4], 2:[0, 1, 3], 3:[2, 4, 5], 4:[1, 3], 5:[3]}`
- ☐ `G = {0:[2, 1], 1:[0, 3, 4], 2:[0, 3], 3:[1, 2, 5], 4:[1, 5], 5:[3, 4]}`
- ☐ `G = {0:[1, 2], 1:[0, 4], 2:[0, 3, 4], 3:[2, 5], 4:[1, 2], 5:[3]}`
- ☐ `G = {0:[1, 2], 1:[0, 4, 5], 2:[0, 3], 3:[2, 5], 4:[1], 5:[1, 3]}`

Question 7 (2 points)
On donne le graphe `H` ci-dessous.



Dans un parcours en profondeur de `H` à partir de `0`,

- ☐ si 2 apparaît avant 3 alors 7 apparaît avant 3
- ☐ 4 doit apparaître avant 6
- ☐ 3 doit apparaître avant 4
- ☐ si 5 apparaît avant 3 alors 2 apparaît avant 3

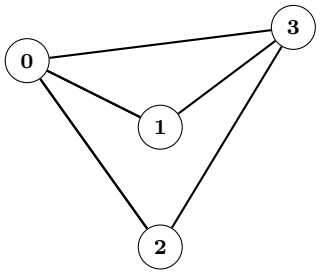
Deuxième partie, questions de type ouvert

Répondre dans l'espace dédié. Laisser libres les cases à cocher : elles sont réservées au correcteur.

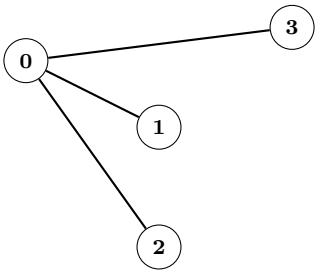
Question 8: *Cette question est notée sur 7 points.*

0 1 2 3 4 5 6 7 .5

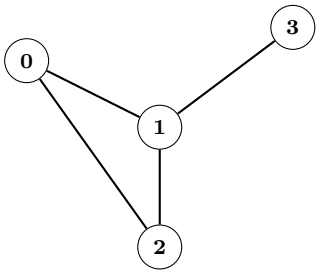
(a) Pour chacun des trois graphes ci-dessous, donnez sa représentation par listes d'adjacence sous forme d'un dictionnaire Python nommé `G` (`H1`, `H2` respectivement).



G



H1



H2

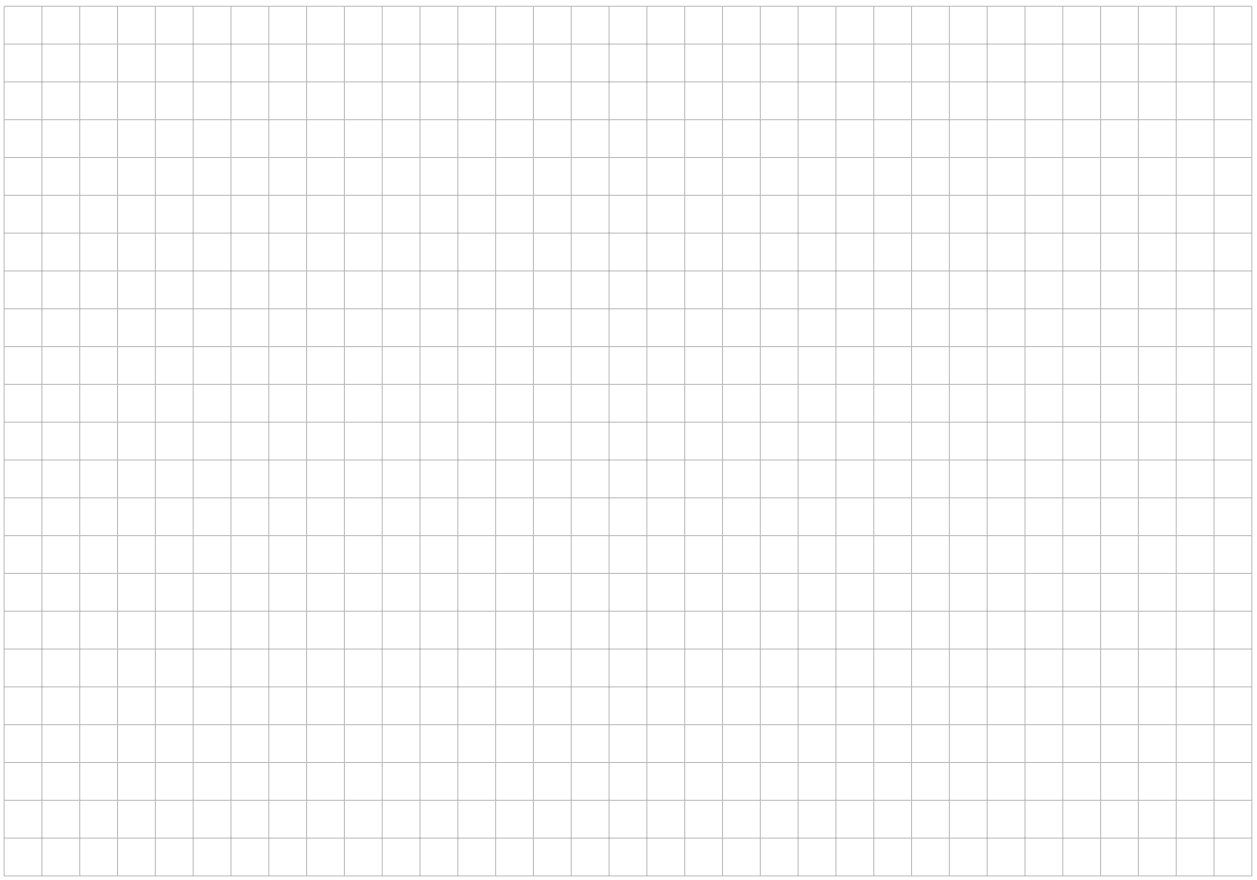
(b) La fonction ci-dessous prend en entrée deux graphes `G` et `H` définis sur le même ensemble de sommets et représentés sous forme de dictionnaires de listes d'adjacence.

```
def mystere(G, H):

    x = [0 for u in G]

    for u in G:
        for v in G[u]:
            x[v] = 1
        print(x)

        for v in H[u]:
            if x[v] != 1:
                print(f"({u},{v}) pas ok!")
                return False
            else:
                print(f"({u},{v}) ok")

        for v in G[u]:
            x[v] = 0
    return True
```

Qu'affiche chacun des appels suivants (où `G`, `H1` et `H2` sont les dictionnaires que vous avez définis au point (a))?

(i) `print(mystere(G, H1))`

(ii) `print(mystere(G, H2))`



(c) Expliquer en quelques mots ce que fait l'algorithme `mystere`.

Question 9: *Cette question est notée sur 7 points.*

Dans ce cours, nous avons étudié l'algorithme de tri par sélection et l'avons implémenté de manière itérative comme suit.

```
def tri_par_selection(L):
    n = len(L)
    for i in range(n):
        m = L[i]
        m_index = i
        for j in range(i+1,n):
            if L[j] < m:
                m = L[j]
                m_index = j
        L[i], L[m_index] = L[m_index], L[i]
```

Donnez une autre implémentation du tri par sélection sous forme d'un algorithme `selection_rec` **récursif**, sans boucle. Cet algorithme doit prendre en entrée une liste `L` de nombres non vide et un indice `start` valide (c'est-à-dire compris entre `0` et `len(L)-1` inclus), et il doit trier la liste `L` à partir de l'indice `start`.

Par exemple, les instructions

```
L = [4, 3, 2, 1]
selection_rec(L, 0)
print(f"{L=}")
```

doivent afficher:

```
L = [1, 2, 3, 4]
```

Votre algorithme peut faire appel à la fonction `get_min` donnée ci-dessous, qui prend en entrée une liste `L` de nombres non vide et un indice `start` valide, et qui renvoie le minimum de la liste `L[start:]` ainsi que l'indice correspondant à ce minimum.

```
def get_min(L,start):
    mi, m = start, L[start]
    for i in range(start,len(L)):
        if L[i] < m:
            mi, m = i, L[i]
    return m,mi
```

Remarque: votre programme n'a pas besoin de vérifier que la liste `L` et l'indice `start` correspondent bien aux spécifications de l'entrée (c'est-à-dire que `L` est non vide, contient des nombres, ou que `start` est un indice valide de `L`).

Question 10: *Cette question est notée sur 7 points.*

☐.5 ☐.5 ☐.5 ☐.5 ☐.5 ☐.5 ☐.5

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7

On définit une "liste à trou" `L` de taille $n \geq 1$ comme suit: elle contient les valeurs $0, 1, \ldots, n$ **sauf une**, triées. Par exemple, les listes à trou de taille 3 sont: `[1, 2, 3]`, `[0, 2, 3]`, `[0, 1, 3]` et `[0, 1, 2]`.

(a) Soit `L` une liste à trou où la valeur manquante est `k`. Que vaut `L[i] - i`, pour `i = 0, ..., n-1` ? Justifiez brièvement votre réponse.

(b) Ecrivez un algorithme `detect_gap` qui prend en entrée une liste à trou `L` et retourne la valeur manquante dans la liste. Par exemple, `detect_gap([0, 2, 3])` doit retourner la valeur `1`, et `detect_gap([1, 2, 3])` doit retourner la valeur `0`.

Pour une liste de taille n en entrée, votre algorithme doit avoir un temps de parcours $T(n)$ logarithmique en n, c'est-à-dire $T(n) = \Theta(\log_2(n))$. Justifiez brièvement ce temps de parcours.

Remarque 1: un algorithme correct mais moins efficace que ce qui est demandé obtiendra 1 point.

Remarque 2: votre programme n'a pas besoin de vérifier que la liste donnée en entrée est bien une liste à trou.