UFR de mathématique et d'informatique
Université de Strasbourg

Algorithmique et Programmation 1 - Examen

- Durée : 2 heures
- Aucun document autorisé

---

1. Listes et Tuples (4 questions, 5 points)

Question 1.1 (1 pt)

Écrire en python, une fonction `insere_milieu`, qui admet deux arguments : une liste `lst` et un élément `x`. Si `lst` est la liste vide, alors la fonction retourne `[x]`. Sinon, elle retourne une autre liste contenant les mêmes éléments que `lst`, dans le même ordre, mais avec un élément supplémentaire de valeur `x` inséré au milieu de `lst` (si `lst` a un nombre pair d'éléments) ou juste avant le milieu de `lst` (si `lst` a un nombre impair d'éléments). Cette fonction devra laisser la liste initiale inchangée. Par exemple :

`insere_milieu([1,2,3,4,5],10)` vaut `[1,2,10,3,4,5]`

`insere_milieu([1,2,3,4],10)` vaut `[1,2,10,3,4]`

Indication : vous n'avez pas nécessairement besoin d'utiliser une boucle.

Question 1.2 (1,5 pt)

Écrire, en python, une fonction `extrait_multiples` qui admet un seul argument de type liste de couples d'entiers `lst`. Un *couple* d'entiers est un tuple composé de deux entiers. Cette fonction devra retourner une liste qui contient l'ensemble de tous les couples de `lst` dont le premier élément est divisible par le second élément. Si `lst` ne contient aucun élément qui vérifie cette propriété, alors la fonction devra retourner la liste vide. Cette fonction ne devra pas modifier `lst`.

Par exemple : `extrait_multiples([(2,7), (8,2), (7,6), (10,5)])` vaut `[(8,2),(10,5]` car 8 est divisible par 2 et 10 par 5.

Question 1.3 (1 pt)

Écrire, en python, une fonction `somme_elements_positifs` qui admet un seul argument `lstlst` de type liste de listes de nombres. Tous les éléments de `lstlst` n'ont pas nécessairement le même nombre d'éléments. Cette fonction devra additionner tous les nombres positifs contenus dans `lstlst` et retourner la somme obtenue. Cette fonction ne devra pas modifier `lstlst`.

Par exemple : `somme_elements_positifs([[8,-9,-6,3],[5,0,-2,7],[1,2,-3,-4,-5]])` vaut 26.

Question 1.4 (1,5 pts)

Écrire, en python, une fonction `lisse_liste` qui admet un seul argument `lst` de type liste de nombres. Cette fonction ne devra pas modifier `lst`. Elle devra retourner une autre liste de nombres de même longueur que `lst`. Si

`lst` est vide, alors la fonction devra retourner la liste vide. Sinon, dans la liste retournée :

- le premier élément devra être égal à `lst[0]` ;
- l'élément d'indice `i` devra être égal à la moyenne des éléments `lst[i]` et `lst[i-1]`.

Par exemple :

`lisse_liste([-1,1,-1,1,3,5,1])` vaut `[-1, 0.0, 0.0, 0.0, 2.0, 4.0, 3.0]`

---

2. Récursivité (3 questions, 5 points)

Question 2.1 (1 pt)

Etant donné cette image illustrant la construction d'un triangle de Sierpiński à différentes étapes de construction. Le premier triangle est à l'étape 0, le deuxième à l'étape 1, etc.

On veut calculer l'aire d'un triangle (représentée par la surface en noir) à une étape donnée de construction. En admettant que l'aire du premier triangle soit 1, l'aire du deuxième triangle sera $\frac{3}{4}$. Le rapport d'aire d'un triangle au suivant est toujours $\frac{3}{4}$.

Ecrivez une fonction récursive `aire_sierpinski` prenant un paramètre entier `n`, positif ou nul, qui est l'étape et qui renvoie l'aire du triangle à cette étape.

Par exemple, `aire_sierpinski(0)` renverra `1`, `aire_sierpinski(3)` renverra `0.421875` (ce qui est $(\frac{3}{4})^3$).

Question 2.2 (2 pt)

Ecrire une fonction récursive `indice_lst` qui prend en paramètre une liste `lst` et un élément `a_trouver`. La fonction retourne l'indice du premier élément `a_trouver` présent dans la liste `lst`. Si l'élément `a_trouver` n'est pas dans la liste, la fonction retourne `-1`.

Question 2.3 (2 pt)

Nous supposons dans cette question qu'un ensemble d'entiers est représenté par une liste dont les **valeurs sont rangées par ordre croissant**. Cette liste ne contient jamais deux fois le même élément.

Ecrire une fonction `union` qui prend en paramètres deux ensembles E et F, et qui retourne un ensemble qui soit l'union de E et F, notée $E \cup F$. Par exemple pour $E = [3, 5, 6, 7]$ et $F = [2, 5, 6]$, la fonction doit renvoyer $[2, 3, 5, 6, 7]$.

La fonction doit être récursive. Elle ne doit pas contenir de boucle et ne doit pas utiliser l'opérateur `in`.

---

3. Décomposition fonctionnelle (3 questions, 5 points)

Les exercices successifs ci-dessous vous permettront d'écrire cette fonction progressivement. Sauf indication contraire, **il est très important d'utiliser les fonctions déjà écrites**. Vous pouvez utiliser une fonction d'un exercice

précédent même si vous n'avez pas réussi à l'écrire.

Question 3.1 (1,5 pt)

1. Ecrivez une fonction `myAbs` qui calcule et renvoie la valeur absolue d'un argument entier.

   Par exemple : `myAbs(-2)` renvoie `2`

2. Ecrivez une fonction `myAbsList` qui renvoie une copie d'une liste d'entiers passée en argument mais dans laquelle tous les nombres négatifs ont été remplacés par leur valeur absolue.

   Par exemple : `myabslist([-1,0,1,-5])` renvoie `[1,0,1,5].`

Question 3.2 (1,5 pts)

1. Ecrivez une fonction `equalLists` qui étant données deux listes d'entiers passées en paramètre renvoie `True` si les deux listes sont identiques (contiennent exactement les mêmes éléments dans le même ordre), ou `False` sinon. NB: cette fonction ne réutilise pas les précédentes.

   Par exemple :

   `equalLists([-1,0,1,-5], [-1,0,1,-5])` renvoie `True`

   `equalLists([-18,4,12,8], [-1,0,-1,-5])` renvoie `False`

   `equalLists([-18,4,12,8], [12,4,-18,8])` renvoie `False`

2. Ecrivez une fonction `isPositiveList` qui à partir d'une liste d'entiers renvoie `True` si la liste ne contient que des nombres positifs ou nuls, et `False` si elle contient au moins un nombre négatif. Attention, on ne devra **pas** utiliser les symboles d'inégalité `>` ou `<` dans cette fonction.

   Par exemple :

   `isPositiveList([-1,0,1,-5])` renvoie `False`

   `isPositiveList([1,0,1,5])` renvoie `True`

Question 3.3 (2 pts)

1. Ecrivez une fonction `mirror` qui à partir d'une liste d'entiers, renvoie son miroir. La primitive python `reverse` ne *doit pas* être utilisée.

   NB: cette fonction ne réutilise pas les précédentes.

   Par exemple : `mirror([6,4,1,3])` renvoie `[3,1,4,6]`

2. Ecrivez une fonction `isMirrorWhateverSign` qui à partir de deux listes d'entiers renvoie `True` si les deux listes sont le miroir l'une de l'autre sans tenir compte du signe des éléments, et `False` sinon.

   Par exemple :

   `isMirrorWhateverSign([1,2,3,4], [1,2,3,4])` renvoie `False`

   `isMirrorWhateverSign([1,2,3,4], [4,3,2,1])` renvoie `True`

   `isMirrorWhateverSign([1,-2,3,-4], [4,-3,2,1])` renvoie `True`

---

4. Tri d'une liste (2 questions, 5 points)

On supposera connue la formule $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

Dans les sous-questions 2, 3 et 4 des deux exercices ci-dessous, on ne vous demande pas un simple résultat, mais de *justifier vos réponses* par une explication.

Question 4.1 (2,5 pts)

Le tri par sélection consiste trier une liste entre les indices `0` et `k` :

- en la parcourant des indices `0` à `k` pour trouver l'indice d'un maximum, puis en échangeant ce maximum avec la valeur à l'indice `k`,
- et en recommençant pour la liste de l'indice `0` jusqu'à l'indice `k-1`, jusqu'à ce qu'on arrive à `k = 0`.

Pour trier une liste de longueur `n` on commencera avec `k = n-1`. On a alors l'algorithme itératif suivant :

```
tri_selection(liste lst)
    n ← longueur de lst
    pour k de n-1 à 1 pas -1
      imax ← 0
      pour j de 1 à k
        si lst[j] > lst[imax] alors imax ← j
      fin pour
      si imax != k alors échanger lst[k] et lst[imax]
    fin pour
```

1. Programmez la fonction correspondant à cet algorithme en Python.
2. On appelle boucle interne, la boucle faisant varier l'indice `j` de `1` à `k` : lors d'une exécution complète de la boucle interne, combien fait-on de comparaisons ? (justifiez)
3. Combien de fois est-ce que cette boucle interne est exécutée dans la boucle externe, lorsque `k` varie de `n-1` à `1` ? (justifiez)
4. En déduire le nombre de comparaisons en fonction de `n`, la longueur de la liste `lst` (jutifiez).
5. En prenant l'opération de comparaison comme référence, quelle est la complexité dans le meilleur des cas ? et dans le pire des cas ?

Question 4.2 (2,5 pts)

On veut améliorer l'algorithme précédent en calculant dans la même boucle interne les indices du minimum *et* du maximum. Ainsi, la taille de la liste traitée diminuera par les deux bouts, une borne `inf` qui augmente et une borne `sup` qui diminue. Pour une liste `lst` de longueur `n`, on commencera avec `inf = 0` et `sup = n-1` :

```
tri_selection2(liste lst)
    n ← longueur de lst
    sup ← n-1
    inf ← 0
    tant que inf < sup faire
      imax, imin ← inf, inf
      pour j de inf+1 à sup
        si lst[j] > lst[imax], alors imax ← j
        si lst[j] < lst[imin], alors imin ← j
      fin pour
      si imax != sup alors échanger lst[sup] et lst[imax]
      # si on a déplacé imin il faut remettre le bon indice :
      si imax != sup et imin = sup alors imin ← imax
```

```
    si imin != inf alors échanger lst[inf] et lst[imin]
    inf, sup ← inf+1, sup-1
  fin pour
```

1. Programmez la fonction correspondant à cet algorithme en Python.
2. Lors d'une exécution complète de la boucle interne, combien fait-on de comparaisons ? (justifiez)
3. Combien de fois cette boucle interne est-elle exécutée par la boucle externe ? (justifiez)
4. En déduire le nombre de comparaisons en fonction de `n`, la longueur de la liste `lst` (justifiez).
5. Comparez avec la fonction de l'exercice précédent et concluez.