DS4

27 avril 2024

La calculatrice est interdite. L'usage de tout document est interdit. La rigueur, le soin, la présentation seront fortement pris en compte dans la notation. Les résultats de chaque question seront encadrés. Vous pouvez bien sûr utiliser des fonctions déjà écrites lors des questions précédentes. Veuillez indenter avec deux grands carreaux.

Primo : du tri ancien

Écrire une fonction `Fusion(Gauche:list,Droite:list)->list` qui, étant données deux listes triées `Gauche` et `Droite`, renvoie la fusion des deux listes de façon triée : si `Gauche=[1,4,8]` et `Droite=[-2,-1,0,1,2,3,9]`, la fonction renverra `[-2,-1,0,1,1,2,3,4,8,9]`.
Ne pas utiliser la commande `pop` et ne pas modifier `Gauche` et `Droite` par effet de bord.

Secundo : du tri nouveau

1. Écrire une fonction `Maximum(L:list)->float` qui, à une liste non vide de nombres, renvoie le plus grand élément de cette liste.
2. Étant donne un entier naturel `M`, quelle commande permet de créer une liste, notée `Liste` de longueur `M` contenant que des zéros ?
3. Écrire une fonction `Occurences(L:list)->dict` qui, à une liste, renvoie le dictionnaire des occurrences, c'est-à-dire que les clés du dictionnaires sont les éléments de `L` et pour chaque élément de `L`, la valeur associée à cette clé est le nombre d'occurrences de l'élément dans la liste `L`.
   Par exemple, si `L=[1,2,3,3,1,1,-5]`, cette fonction renvoie `{1:3,2:1,3:2,-5:1}`.
4. En déduire une fonction `Doublons(L:list)->bool` qui renvoie `True` si la liste `L` contient au moins deux éléments identiques et `False` sinon.
   Par exemple, `Doublons([3,1,2,2])` vaudra `True` contrairement à `Doublons([3,1,5])`.
   On veillera à ce que cette fonction ait une complexité linéaire (en la longueur de la liste) et on justifiera cette complexité.

La fonction `isinstance` (cette fonction n'est pas au programme) permet de savoir si un élément donné est de type entier. Par exemple :

- `isistance(3,int)` vaut `True`.
- `isistance(-3,int)` vaut `True`.
- `isistance(-3.0,int)` vaut `False` : -3.0 est considéré comme un nombre flottant (à virgule) et non comme un entier par Python.
- `isistance(3.14,int)` vaut `False`.

5. Écrire une fonction `ListeEntiersPositifs(L:list)->bool` qui renvoie `True` si tous les éléments de la liste sont des entiers (de type int) et sont tous positifs. Dans le cas contraire, cette fonction renvoie `False`.
6. Écrire une fonction `Histogramme(L:list)->list` où `L` est une liste d'entiers naturels qui renvoie l'histogramme. C'est-à-dire une liste `Liste` tel que `Liste[i]` contient le nombre d'occurrences de i dans la liste `L` et ce pour tout $i \in [\![0 ; M]\!]$ où M est le plus grand élément de la liste `L`.
   Par exemple, si `L=[0,1,1,2,3,1,6,3]` alors cette fonction renvoie `[1,3,1,2,0,0,1]`. En effet, il y a un 0 dans L, trois 1 dans L, 1 un 2 dans L, un 3 dans L, zéro 4 dans L, zéro 5 dans L et un 6 dans L.
   Veiller à ce que cette fonction ait une complexité linéaire (en la longueur de la liste) et justifier cette complexité.
7. Étant donné un histogramme d'une liste d'entiers naturels noté `Hist` (comme celui fourni par la fonction créée à la question précédente), écrire une fonction `Construction(Hist:list)->list` qui renvoie une liste en mettant chaque entier naturel autant de fois que l'historique l'impose de façon triée. Par exemple, si `Hist=[1,3,1,2,0,0,1]`, la fonction renvoie `[0,1,1,1,2,3,3,6]` : il y a un 0, trois 1, 1 un 2, un 3, pas de 4, pas de 5 et un 6.
8. En déduire de tout ce qui précède une fonction `TriComptage(Liste:list)->list` qui permet renvoyer la liste triée dès que `Liste` est une liste d'entiers naturels (condition que l'on vérifiera). Cette fonction aura une complexité linéaire (que l'on justifiera).

Cet algorithme s'appelle le tri par comptage et c'est un tri sans comparaison contrairement au tri à bulles, fusion ou rapide (où il y avait toujours des comparaisons entre éléments de la liste au moyen de `<`, `>`, `<=` ou `>=`).