ITC - PC

TD - manipulation de listes 1

Exercice 1

Écrire une fonction `est_dedans(t,x)` qui prend en entrée une liste `t` et une valeurs `x` et qui renvoie `True` si `x` est un élément de `t` (`False` sinon).
Le but de cette fonction est de recréer `x in t`, on n'utilisera donc pas cette instruction.

Exercice 2

Écrivez une fonction qui détermine le maximum d'une liste, une autre qui détermine la position de ce maximum (Bien entendu on n'utilisera pas la fonction max).
Et si le maximum est pris plusieurs fois dans la liste, lequel est renvoyé par cette seconde fonction ?
Comment changer ce comportement ?
Ecrire une fonction renvoyant les deux plus grands éléments d'une liste.

Exercice 3

Écrivez une fonction prenant une liste en entrée et renvoyant le booléen `True` si la liste est croissante (`False` sinon).

Exercice 4

Écrivez une fonction prenant en entrée une liste, et renvoyant la liste des sommes cumulées (depuis le premier terme). La complexité devra être en $O(n)$ où n est le nombre d'éléments dans la liste.

Exercice 5

Écrire une fonction renvoyant les deux valeurs les plus proches dans une liste donnée en entrée.

Exercice 6

Écrire une fonction qui permet de trier une liste. Le tri pourra être en place ou pas, l'algorithme choisi est laissé libre : tri par insertion, tri fusion ou tri rapide.

Remarque 1

il en existe plein d'autres : tri à bulle, tri par sélection, et celui utilisé par par la fonction `sorted` de Python : le timsort inventé par Tim Peter en 2002 et qui est un dérivé du tri fusion et du tri par insertion et qui part du principe que dans un ensemble de données une parti est déjà triée, sa complexité est en O(N) dans le meilleur des cas et en O(N ln(N)) dans le pire et aussi en moyenne.

Exercice 7 (*Algorithme glouton*)

Un algorithme est dit glouton lorsqu'il résout un problème d'optimisation en une succession d'étapes et où à chaque étape il choisit la solution optimale, cependant il n'y a aucune garantie pour que la solution finale soit optimale. Par exemple si on désire dans une rue visiter toutes les boutiques en minimisant la distance parcourue un algorithme glouton consisterait à choisir à chaque étape la boutique la plus proche, il est facile de voir que dans ce cas là, la solution n'est pas nécessairement optimale (par exemple si on commence à la deuxième boutique de la rue, que la plus proche est la troisième etc. on devra à la fin revenir à la première boutique qu'on a pas visitée).

Un exemple classique est celui du rendu de monnaie, on considère qu'on a des pièces (et des billets) de 1,2,5,10,50 et 100 euros et qu'on doit rendre la monnaie en utilisant un minimum de pièce, si on doit rendre 42 euros on va clairement rendre 4 billets de 10 et une pièce de 2, quand on fait cela on applique un algorithme glouton : on choisit la plus grande valeur de la monnaie inférieure à 42, etc.

1. Écrire une fonction `pieces_a_rendre(somme_a_rendre , systeme_monnaie)` qui prend en entrée la somme à rendre et la liste des pièces possible et qui retourne la liste des pièces à rendre.
2. Que peut-on dire de l'algorithme pour un système de monnaie qui a des pièces de 1, 3 et 4 (on peut aussi considérer le système anglais d'avant sa réforme en 1971) ?