Tri et recherche

1. Tri par insertion

[1] Soit `L` une liste de nombres entiers dont les `n` premiers éléments sont triés par ordre croissant. Écrire une fonction `insertion(L,n,x)` qui effectue l'insertion du nombre entier `x` dans ces `n` premiers éléments. La recherche de l'emplacement d'insertion sera faite de manière linéaire, en parcourant les `n` éléments par indice décroissant.

[2] Écrire une fonction `tri_insertion(L)` qui effectue le trie par insertion d'une liste de nombres. Tester la fonction avec le code suivant :

```
import random
L = []
N= 50
for k in range(N):
    L.append(random.randint(0,N))
L=tri_insertion(L)
print(L)
```

2. Recherche d'un élément dans une liste

On recherche un nombre entier dans une liste de nombres entiers déjà triée. Il s'agit de déterminer les indices des différentes occurences de ce nombre dans la liste.

On commence par la méthode naïve consistant à parcourir les éléments de la liste dans l'ordre jusqu'à la première occurence du nombre cherché.

[3] Écrire une fonction `recherche(liste,x)` qui effectue la recherche de toutes les occurences du nombre `x` dans la liste triée `liste`. Cette fonction doit renvoyer la liste des indices dans l'ordre croissant. Tester la fonction sur la liste `L` obtenue précédemment.

[4] Quel est l'ordre de grandeur de la complexité temporelle de cette méthode naïve (dans le pire des cas) ?

La recherche peut être rendue plus efficace par une méthode dichotomique récursive. La liste est coupée en deux parties à peu près égales. Une comparaison est effectuée pour savoir si le nombre cherché se trouve dans la première ou dans la seconde partie, puis la recherche est effectuée récursivement sur cette partie. Il faudra aussi traiter le cas où le nombre cherché se trouve en plusieurs occurences dans les deux parties.

Soit `liste` la liste de nombre entiers dans laquelle on fait la recherche du nombre `x`. Une partie de cette liste sera définie par les indices `debut` et `fin` : le premier élément de la partie est `liste[debut]`, le dernier élément de la partie est `liste[fin-1]`.

[5] Écrire une fonction `extraction_indices(liste,x,i)` qui renvoie la liste des indices (dans l'ordre croissant) des occurences du nombre `x`, en supposant que l'indice `i` soit celui d'une occurence déjà trouvée.

[6] Écrire une fonction `recherche_dichotomique(liste,x,debut,fin)` qui effectue la recherche dichotomique récursive dans une partie de `liste` définie par les indices `debut` et `fin`. La fonction doit renvoyer la liste des indices dans l'ordre croissant. Tester la fonction sur la liste `L` obtenue précédemment.

[7] Pour évaluer la complexité temporelle de la méthode dichotomique, on suppose que sa longueur s'écrit $N = 2^p$, où p est un nombre entier. Combien faut-il faire d'appels récursifs

pour arriver à une partie ne contenant qu'un seul élément ? En déduire que l'ordre de grandeur maximal de la complexité est $O(ln(N))$. La complexité dépend-elle de l'ordre des nombres dans la liste initiale ? Comparer à la méthode naïve. Dans quel cas la méthode naïve pourrait être plus efficace ?

3. Tri par fusion

On cherche à implémenter le tri par fusion de manière itérative (sans utiliser la récursivité). On suppose que la taille de la liste à trier s'écrit $N = 2^p$, où p est un nombre entier.

[8] Faire un schéma montrant les différentes fusions à réaliser, dans le cas $p = 3$. On appelle niveau de fusion l'ensemble des fusions de listes de même taille. Combien y-a-t-il de niveaux de fusion ? Pour une fusion de niveau n, quelle est la taille des deux listes à fusionner ?

[9] Soit `taille` la taille des deux listes à fusionner. Combien y-a-t-il de fusions à faire pour un niveau ? Indiquer la position des deux listes à fusionner dans la liste principale.

[10] Il n'est pas nécessaire de recopier les deux listes à fusionner, qui sont deux sous-listes de la liste principale en cours de tri. Est-il nécessaire de créer un tableau temporaire pour effectuer la fusion de ces deux sous-listes ?

[11] Écrire une fonction `tri_fusion(liste,p)` qui effectue ce tri par fusion de manière itérative.