TP 4 informatique

BCPST 1 2019-2020

Algorithmes de tris

Exercices

Exercice 1. Écrire une fonction `tri_bulle(L)` qui effectue le tri à bulle de L. A l'issue de la fonction, la liste L sera trié et on ne fera pas de copie de L.

Exercice 2. Écrire une fonction `tri_insertion(L)` qui effectue le tri par insertion de L. A l'issue de la fonction, la liste L sera trié et on ne fera pas de copie de L.

Exercice 3. $\star$

On veut implémenter un algorithme de tri rapide sans effectuer de copies de listes. Pour cela, on utilise une fonction auxiliaire `tri_aux(L,debut,fin)` et deux variables auxiliaires `debut` et `fin` qui indiquent la partie de la liste que l'on veut trier. Par exemple, `tri_aux(L,2,5)` trie la partie $L[2:6]$. Voici une stratégie générale qui permet d'effectuer un tri rapide :

1. si debut>=fin, on ne fait rien.
2. on choisit `L[debut]` comme pivot.
3. on utilise deux indices, a=debut+1,b=fin pour placer respectivement les éléments plus petit et plus grand que le pivot.
4. Dès que $a = b$, on compare $L[a]$ avec le pivot, et on place le pivot au bon endroit.
5. En notant p la position du pivot, étant maintenant bien placé, il nous reste à faire le tri entre `debut` et $p-1$ et $p+1$ et `fin`.

Par exemple, si $L = [4, 1, 2, 5, 8, 3, 2, 4]$, en choisissant le premier 4 comme pivot, `debut=0`, `fin=7`, décrivons ce qui se passe dans la première passe :

1. $[\mathbf{4}, \underline{1}, 2, 5, 8, 3, 2, \overline{3}]$ car $1 < 4$.
2. $[\mathbf{4}, 1, \underline{2}, 5, 8, 3, 2, \overline{3}]$ car $2 < 4$.
3. $[\mathbf{4}, 1, 2, \underline{5}, 8, 3, 2, \overline{3}]$.
4. $[\mathbf{4}, 1, 2, \underline{3}, 8, 3, \overline{2}, 5]$ car $5 > 4$.
5. $[\mathbf{4}, 1, 2, 3, \underline{8}, 3, \overline{2}, 5]$.
6. $[\mathbf{4}, 1, 2, 3, \underline{2}, 3, \overline{8}, 5]$.
7. $[\mathbf{4}, 1, 2, 3, \underline{2}, \overline{3}, 8, 5]$.
8. $[\mathbf{4}, 1, 2, 3, 2, \underline{\overline{3}}, 8, 5]$.
9. $[3, 1, 2, 3, 2, \underline{\overline{4}}, 8, 5]$.

1. En vous aidant de la description précédente, écrire une fonction récursive `tri_aux(L,debut,fin)` qui permet de trier la partie $L[debut : fin + 1]$ de la liste.
2. En déduire une fonction `tri(L)` qui trie la liste.