nbcm-l8dn001-quicksort

March 17, 2021

[1]:
```python
# Fonction utilitaire d'échange, utilisée par de nombreux algorithmes de tri.
# Rq. En python, on peut écrire directement : T[i], T[j] = T[j], T[i]
# C'est plus compact mais ça consomme au moins autant de mémoire
def echange(T,i,j):
    prov = T[i]
    T[i] = T[j]
    T[j] = prov
```

[2]:
```python
# partition(L,deb,fin,pivot) : réorganise la liste L[deb:fin] pour faire
# en sorte que tous les éléments inférieurs au pivot soient *avant*
# tous les éléments supérieurs (ou =)
# la liste est modifiée "sur place",
# et la valeur renvoyée est la position du premier élément >= au pivot
# on suppose la fonction "echange()" définie

def partition(L,deb,fin,pivot):
    gauche = deb
    droite = fin-1
    while (True):
        echange(L,gauche,droite)
        while (L[gauche] < pivot):  gauche += 1
        while (L[droite] >= pivot): droite -= 1
        if gauche >= droite: break
    return gauche

# principes de programmation:
# les indices gauche et droite désignent la première (resp. dernière)
# case dont la valeur doit être examinée
# par contraste, les indices 'deb' et 'fin' sont employés "à la python":
#       'deb' est l'indice de la première case de la liste à considérer
#       'fin' est le premier indice à _ne pas_ considérer
# la boucle principale est une boucle avec test en queue:
#    elle est simulée avec une boucle infinie et un break
# on commence par faire un échange des deux valeurs extrêmes du tableau
# avant de faire avancer (resp. reculer) les indices 'gauche' et 'droite'
# cet échange peut ne pas être pertinent au premier tour,
# mais procéder ainsi simplifie l'écriture du corps de la boucle
```

[3]:
```python
# choix_pivot(L, i,j) : cette fonction vise deux buts distincts:
# - renvoyer une valeur de pivot pour la partition (la plus appropriée)
# - repérer les cas où il faut considérer la liste comme triée
# On travaille sur la sous-liste L[i:j],
# et on considère que L[i] existe.
# la valeur renvoyée n'est pas le pivot, mais son indice dans la liste
# (ou -1 si la liste est déjà triée)

def choix_pivot(L,i,j):
    premCle = L[i]
    k = i + 1
    while (k < j and L[k] == premCle):
        k += 1
    if k == j: return -1
    if L[k] > premCle: return k
    if L[k] < premCle: return i

# La boucle initiale vérifie qu'on a au moins deux valeurs différentes
# dans la liste (sinon, elle est déjà triée)
# Ensuite, on renvoie la valeur la plus grande entre les deux premières
# valeurs différentes
```

[9]:
```python
# Quicksort : programme principal, qui lance le tri récursif
# en utilisant partition() et choix_pivot()
# On travaille sur L[i:j] (j = dernier indice + 1)
# Le tri se fait "sur place" (dans la liste), il n'est donc pas nécessaire
# de renvoyer (par return) la liste triée.

def quicksort(L,i,j):
    i_pivot = choix_pivot(L,i,j)
    if i_pivot == -1 : return
    k = partition(L,i,j,L[i_pivot])
    quicksort(L,i,k)
    quicksort(L,k,j)

# "Wrapper" : fonction qui sert juste à simplifier l'appel à quicksort
def w_quicksort(L):
    if L != []:
        quicksort(L,0,len(L))
```

[10]:
```python
# génération d'une liste d'entiers aléatoire (tous différents)
import random
def genere_liste(Min=1,Max=50,N=20):
    "génère une liste de N valeurs aléatoires comprises en Min et Max"
    return random.sample(range(Min, Max), N)
```

[11]:
```
L = genere_liste()
print(L)
w_quicksort(L)
print(L)
```

```
[41, 7, 13, 20, 24, 17, 37, 16, 3, 5, 36, 30, 21, 46, 39, 49, 2, 28, 44, 11]
[2, 3, 5, 7, 11, 13, 16, 17, 20, 21, 24, 28, 30, 36, 37, 39, 41, 44, 46, 49]
```