Les Structures de Données Python
Cours 5 : Les algorithmes de tri

Halim Djerroud

ESIEE
PARIS

révision : 0.1

Plan

Les listes 2D :

1. Tri par sélection
2. Tri par insertion
3. Tri par bulles
4. Tri par fusion
5. Tri rapide

Tri par sélection

- rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0
- rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1
- continuer de cette façon jusqu'à ce que le tableau soit entièrement trié

Tri par sélection

```
lst = [2,8,45,1,34,22,48,15,44,12,11,15,13,7,41,32]

def selectionSort(lst):
    for i in range (len(lst) - 1):
        min = i
        for j in range (i+1, len(lst)):
            if lst[j] < lst[min]:
                min = j
        if min != i :
            tmp = lst[i]
            lst[i] = lst[min]
            lst[min] = tmp


selectionSort(lst)
print(lst)
#[1, 2, 7, 8, 11, 12, 13, 15, 15, 22, 32, 34, 41, 44, 45, 48]
```

Tri par insertion

Le tri par insertion considère chaque élément du tableau et l'insère à la bonne place parmi les éléments déjà triés

Tri par insertion

```
lst = [2,8,45,1,34,22,48,15,44,12,11,15,13,7,41,32]

def insertion_sort(lst):
    for i in range (1, len(lst)):
        for j in range (i-1, -1, -1):
            if(lst[j+1] > lst[j]):
                break
            tmp = lst[j]
            lst[j] = lst[j+1]
            lst[j+1] = tmp


insertion_sort(liste)
print(liste)
```

Tri par bulles

- Il consiste à comparer répétitivement les éléments consécutifs d'un tableau, et à les permuter lorsqu'ils sont mal triés.

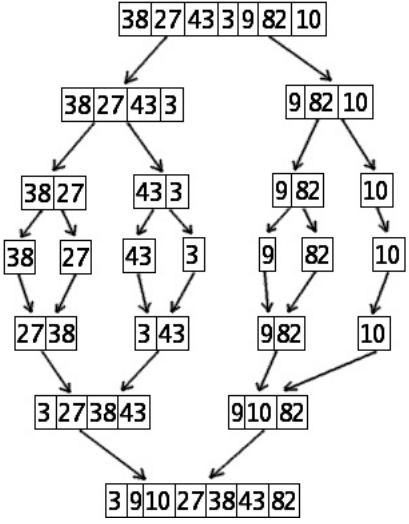
Tri par bulles

```
lst = [2,8,45,1,34,22,48,15,44,12,11,15,13,7,41,32]

def bubbleSort(lst):
    for i in range (len(lst), 0, -1):
        for j in range (0, i-1):
            if lst[j] > lst[j+1]:
                tmp = lst[j]
                lst[j] = lst[j+1]
                lst[j+1] = tmp

bubbleSort(lst)
print(lst)
```

Tri par fusion

- Si le tableau n'a qu'un élément, il est déjà trié.
- Sinon, séparer le tableau en deux parties à peu près égales
- Trier récursivement les deux parties avec l'algorithme du tri fusion
- Fusionner les deux tableaux triés en un seul tableau trié

Tri par fusion



Tri par fusion

```
def mergeSort(arr):
    if len(arr) > 1:
         # Trouver le milieu du tableau
        mid = len(arr)//2

        # Diviser les elements du tableau
        L = arr[:mid]
        R = arr[mid:]

        # Trier les deux parties
        mergeSort(L)
        mergeSort(R)

        i = j = k = 0
```

```
        # Copier les donnees dans les
        # tableaux temporaires L[] et R[]
        while i < len(L) and j < len(R):
            if L[i] <= R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        # Verifier s'il reste un element
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
```

Tri rapide

- Placer un élément du tableau (appelé pivot) à sa place définitive
- permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite (partitionnement)

Tri rapide

```
def partitionate(tab, f, l, p):
    max =l
    while f < l:
        while tab[f] <= tab[p] and f < max:
            f += 1
        while tab[l] >= tab[p] and l > p :
            l -= 1
        if f<l:
            tab[f], tab[l] = tab[l], tab[f]
    tab[p], tab[l] = tab[l], tab[p]
    return l
```

```
def quicksort(tab, f, l):
    if(f>l):
        return
    p = partitionate(tab, f, l, f)
    quicksort(tab,f, p-1)
    quicksort(tab,p+1, l)

t = [8,5,7,12,4,54,3,48,20]
quicksort(t, 0, len(t)-1)
print(t)
```