Lycée Louis Thullier 2025-2026
BCPST2

Travaux Pratiques 02 : Algorithmes de tris

L'objectif de ce TP est d'une part de revoir les différents tris vu en première année, et d'autre part de découvrir un algorithme de tri récursif.

On s'intéressera ensuite à la rapidité de chacun de ses algorithmes.

Objectif :

— Connaître le principe des algorithmes de tris présentés en première et en seconde année.

— Être capable de programmer un algorithme de tri (celui de son choix) sans indication.

Pour comparer l'efficacité des différents algorithmes on s'intéressera au temps nécessaire pour trier des listes d'entiers de 1 à n.

Il est également possible de s'intéresser au nombre de comparaisons effectuer par chacun de ses algorithmes. Dans ce cas, il est possible de démontrer que, sans conditions particulière sur la liste à trier, les meilleurs algorithmes effectuent un nombre de comparaison en $O(n\ln(n))$ où n est la taille de la liste à trier.

1 Deux tris de première année

1.1 Tri par insertion

Le principe du tri par insertion est le suivant : On se donne une liste `L` à trier et une liste vide `T`. On insère le premier élément de `L` dans `T` . Puis on insère le second élément `L` dans `T` de telle sorte que les éléments de `T` soient rangés dans l'ordre croissant. On réitère ce procédé jusqu'à avoir parcouru toute la liste `L`.

```
Tant que la liste L est non vide
    Retirer le premier élément de L
    L'insérer à la bonne place dans la liste T
Renvoyer la liste T
```

1.1.1. Écrire la valeur de `T` au cours des différentes étapes du tri par insertion de la liste $[4,7,6,1,3]$.

1.1.2. Compléter le programme ci-dessous qui dans la liste (déjà ordonnée) `T` insère à la bonne place l'élément `x`. Ce programme renvoie une nouvelle liste.

```
def insertion (T,x):
    k=...
    while ... and ... :
        ...
    return(T[:k]+...+...)
```

On rappelle que `T[:k]` permet de créer une copie des k premiers termes de la liste `T`.

1.1.3. Tester le programme `insertion` avec la liste `T=[1,3,7]` et `x=2`. Le programme doit renvoyer `[1,2,3,7]`.

1.1.4. Tester maintenant le programme avec

(a) `T=[]`, `x=1`,

(b) `T=[1,2,3]`, `x=4`,

(c) `T=[1,2,3]`, `x=0`.

1.1.5. Compléter le programme du tri par insertion :

```
def tri_insertion (L):
    T=[]
    for l in ...:
            T=insertion(...,... )
    return(...)
```

1.1.6. Tester le programme sur la liste $[4, 7, 6, 1, 3]$.

1.2 Tri par sélection

Le principe du tri par sélection sur une liste `L` est le suivant : On cherche le plus petit élément de `L`. Ensuite on l'échange avec le premier élément de `L`. On recommence ensuite en cherchant le plus petit élément de `L[1:]` et en l'échangeant avec le second élément de `L` et ainsi de suite...

```
Pour k parcourant les indices de la liste L
    Rechercher le plus petit élément de la liste L[k:]
    L'échanger avec le k-ème élément de L
La liste L est triée
```

1.2.1. Écrire l'état de la liste `L` au cours des différentes étapes du tri par sélection de la liste `[4,7,6,1,3]`.

1.2.2. Écrire un programme `mini` permettant de renvoyer *la position* du minimum d'une liste `L`. On testera le programme sur la liste `[4,7,6,1,3]`.

1.2.3. Compléter le programme du tri par sélection :

```
def selection (L):
        for k in range(...):
                pos=mini(L[k:])+k
                #On rajoute k pour prendre en compte
                #le décalage d'indice dans la liste extraite.
                L[k],L[...]=...
```

1.2.4. Tester le programme sur la liste `[4,7,6,1,3]`.

2 Des tris récursifs

2.1 Tri fusion

Le principe du tri fusion sur une liste `L` est le suivant : On commence par couper en deux parties de longueur égale la liste `L`, trier chacune des moitiés grâce au tri fusion puis fusionner (en créant une liste triée) les deux listes triées ensemble.

```
Si la liste est vide ou réduite à un élément on renvoie cette liste
Sinon :
   On sépare la liste en deux
   On effectue un tri fusion sur chaque moitié
   On fusionne les deux sous-listes triées
   On renvoie la liste obtenue
```

On remarque que pour effectuer un tri fusion le programme est obligé d'effectuer lui-même un tri fusion sur une liste plus petite (d'où le fait que le programme est récursif).

2.1.1. Dans un premier temps on va programmer la fonction permettant de fusionner deux listes triées $L1$ et $L2$ pour en obtenir une liste trié $L3$. L'idée du programme est la suivante :

- On part d'une liste vide $L3$
- On compare le premier élément des listes $L1$ et $L2$,

— Si le premier élément de la liste $L1$ est le plus petit on le retire de la liste $L1$ et on le place dans la liste $L3$.
— Sinon c'est le premier élément de la liste $L2$ que l'on retire de la liste $L2$ et que l'on rajoute dans $L3$.
— Quand l'une des deux listes $L1$ ou $L2$ est vide on rajoute le contenu de la liste non vide à la fin de la liste $L3$.

Plutôt que de retirer les éléments des listes $L1$ et $L2$ on peut les parcourir avec une boucle et comparer les éléments `L1[i]` et `L2[j]`.

Compléter le programme suivant pour obtenir le résultat voulu

```
def fusion(L1,L2):
    L3=[]#on commence avec la liste vide.
    n1,n2=len(L1),len(L2)
    i,j=0,0#compteur de position dans la liste L1,L2
    while i<n1 and j<n2:# On parcourt les listes L1 et L2
        #On rajoute la plus petite valeur entre L1 et L2 dans L3 et on
        #met à jour le compteur correspondant
        if ...<=...:
            L3.append(...)
            i+=1
        else:
            L3.append(...)
            j+=1
    #Une des listes est entièrement inséré dans L3 on rajoute maintenant
    #ce qui reste de l'autre liste
    if i<n1:
        return(...+...)
    else:
        return(...+...)
```

2.1.2. Tester le programme `fusion` sur les listes
— `[1,4,7]`, `[2,5,8,10]`
— `[1,4,7,11]`, `[2,3,6]`

2.1.3. Écrire le programme `trifusion` et le tester sur la liste `[2,4,1,5,7]`.

2.2 Tri rapide

Le principe du tri rapide est de séparer la liste en deux en utilisant un pivot. Dans l'une des deux sous-listes on placera tous les éléments plus petit que le pivot et dans l'autre tout les éléments plus grand que le pivot. Puis on applique un tri rapide à chaque sous-liste et on concatène les listes obtenues.

```
Si la liste est vide ou réduite à un élément on renvoie cette liste
Sinon :
   On sépare la liste en deux en utilisant le premier élément comme pivot
   On effectue un tri rapide sur chaque sous-liste
   On regroupe les sous-listes
   On renvoie la liste obtenue
```

2.2.1. Écrire une fonction `Pivot(L,x)` qui renvoie deux listes `Lp,Lg` tel que `Lp` contiennent les éléments de `L` plus petit que `x` et `Lg` les éléments plus grands que `x`.

2.2.2. Tester le programme précédent avec comme liste `L=[3,4,6,1,0,10]` et comme pivot respectivement 4, -1, et 11.

2.2.3. Compléter le programme `trirapide` ci-dessous :

```
def trirapide(L):
        if len(L)<= 1:
                return(...)
        else:
                Lp,Lg=Pivot(L[1:],L[0]) #On coupe la liste par rapport à son
                #premier élément
                Lp=...#On trie les sous-listes
                Lg=...
                return(...+[L[0]]+...)#On retourne la liste finale
```

2.2.4. Tester ce programme avec la liste `[8,4,1,5,-1]`.

3 Comparaison de rapidité d'exécution

Pour comparer la vitesse d'exécution de chaque tri on va utiliser la bibliothèque `time`. Celle-ci possède une fonction `time()` qui permet d'obtenir le temps passé depuis un point d'origine. En appelant cette fonction une fois avant un algorithme de tri et une fois après on peut obtenir le temps passé à exécuter le programme.

3.0.1. Compléter les lignes suivantes pour obtenir une liste contenant les temps d'exécutions du tri par insertion sur des listes contenant les nombres de 1 à n.

```
import time
import random as rd

ListeTempsI=[]
for k in range(1,1000):
        L=[i for i in range(k)]# Création de la liste
        rd.shuffle(L)# Mélange
        t1=time.time()# Temps initial
        tri_insertion(...)# Tri
        t2=...#Temps final
        ListeTempsI.append(...)# Ajout du temps passé pour trier
```

3.0.2. Reprendre le programme précédent pour créer une liste `ListeTempsS` contenant les temps d'exécutions du tri par sélection sur des listes contenant les nombres de 1 à n.

3.0.3. Compléter les lignes suivantes pour afficher le temps d'exécution de chaque tri en fonction de la longueur de la liste : `plt.plot(X,Y,'r*',label='legende')` permet de tracer les points de `X` en fonction de ceux de `Y`, en rouge.

```
import matplotlib.pyplot as plt

N=...

plt.figure('Comparaison tri sélection vs insertion')
plt.plot(N,...,'r*',label='Temps tri sélection')
plt.plot(...,...,'b*',label='Temps tri insertion')

plt.suptitle('')
plt.xlabel('Longueur de la liste')
plt.ylabel('Temps (s)')
legend = plt.legend(loc='upper left', shadow=True, fontsize='medium')

plt.show()
```

Que peux-t-on en déduire ?

3.0.4. Comparer les performances du tri par insertion avec le tri rapide et le tri fusion.

4 Quelques liens pour visualiser des tris

Plein de tris : https://www.youtube.com/watch?v=kPRA0W1kECg

Tri par insertion : https://www.youtube.com/watch?v=ROalU379l3U

Tri par sélection : https://www.youtube.com/watch?v=Ns4TPTC8whw

Tri rapide : https://www.youtube.com/watch?v=ywWBy6J5gz8