

TP4 : Algorithmique sur les tris

Compétences

- Dessiner l'arbre d'appels d'un tri récursif
- Implémenter des tris
- Évaluer la complexité expérimentale de différents tris

Un peu d'histoire

Durant la seconde Guerre mondiale, La *Moore School of Engineering* à Philadelphie embauche des femmes pour effectuer des calculs de trajectoires. L'une d'elle Betty Holberton est vite remarquée pour ses aptitudes mathématiques et fut choisie pour être l'une des six programmatrices de l'ENIAC (acronyme de l'expression anglaise Electronic Numerical Integrator And Computer). Elle développe en 1952 le premier algorithme de tri connu.

In 1952, Holberton developed the Sort-Merge Generator" for the UNIVAC I which produced a program to sort and merge files. This was the first step toward actually using a computer to write programs (i.e., a precursor to the concept of a compiler) and was called "the first major *software routine* ever developed for automatic programming

Le XSort un tri insolite

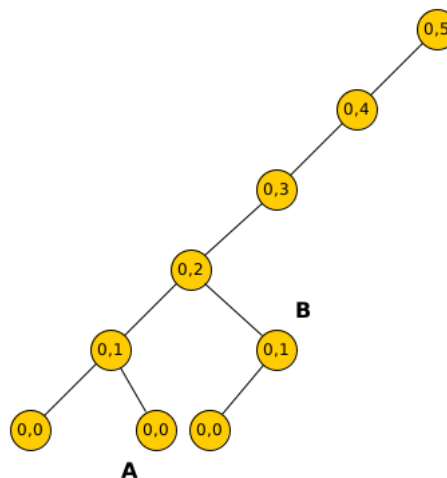
Les lignes de code d'un algorithme de tri ont été retrouvées dans un état qui ne permet pas de l'utiliser. Les seules indices que l'on possède pour essayer de retrouver le code d'origine est une partie de l'arbre des appels simplifié pour le tableau suivant : 3 1 2 7 8 2 et le contenu du tableau pour les noeuds notés A et B :

- A: 1 3 2 7 8 2
- B: 1 2 3 7 8 2 .

```

1 def XSort(t, i, j):
2
3   tmp = t[j-1]
4   XSort(t, i, j-1)
5   XSort(t, i, j-1)
6   return
7   if i >= j:
8   if t[j-1] > t[j]:
9   t[j-1] = t[j]
10  t[j] = tmp

```



1. À l'aide des indices donnés dans l'énoncé remettre dans l'ordre les lignes de code de la fonction **XSort**
2. Compléter l'arbre des appels.
3. En cherchant de l'information sur ce tri vous trouvez que dans le pire cas il est en $O(2^n)$ avec n la taille du tableau qu'en pensez-vous? Justifier.

Temps d'exécution sur des tableaux d'entiers avec tirage pseudo-aléatoires

Pour effectuer un tirage d'entiers pseudo-aléatoire on peut utiliser l'instruction suivante :

```

1 # import de la fonction randint avec l'alias rd
2 from random import randint as rd
3 # un tableau T de 5 valeurs entières dans l'intervalle [0, 10]
4 n = 5
5 T = array("l", [None]*n)
6 for i in range(n):
7     T[i] = rd(0, 10)

```

À l'aide d'une méthode empirique, remplir le tableau ci-dessous avec le temps moyen d'exécution de chaque algorithme de tris pour un tableau de taille n . Les calculs doivent être effectués pour un jeu de 10 expériences sur chaque tri.

n	Tri Insertion	Tri Fusion	XSort
1			
10			
100			
1000			
10000			
100000			
1000000			

Comparer les résultats obtenus avec les complexités théoriques.

Tri insertion modifié

Theorem 1 *Le tri par insertion a une durée d'exécution égale à $O(n)$ sur une séquence triée*

Ce théorème suggère que si des sous parties d'un tableau sont déjà triées, il peut être avantageux d'utiliser le tri par insertion qui a une complexité en $O(n)$ dans ce cas. Étudions un algorithme qui exploite ce théorème pour obtenir une complexité en $O(n^{1.5})$.

Principe

L'idée est d'appliquer plusieurs fois le tri par insertion à des sous séquences d'un tableau sur la base suivante :

1. Initialiser $d = 1$
2. Répéter l'étape 3 jusqu'à ce que $9d > n$
3. d prend la valeur $3d + 1$
4. Répéter les étapes 5 et 6 jusqu'à ce que $d = 0$
5. Appliquer le tri par insertion à chacune des d sous-séquences d'incrément d du tableau de départ.
6. Affecter la partie entière $d/3$ à d

Exemple

- Supposons que le tableau de départ soit composé de $n = 50$ éléments. La boucle de l'étape 2 serait alors répétée 2 fois ce qui ferait augmenter la valeur de départ de $d = 1$ à $d = 4, 13$.
- La boucle de l'étape 4 :
 - La première itération applique le tri insertion à chacune des 13 sous-séquences d'incrément 13 $[e_0, e_{13}, e_{26}, e_{39}], [e_1, e_{14}, e_{27}, e_{40}], \dots, [e_{11}, e_{24}, e_{37}, e_{50}], [e_{12}, e_{25}, e_{38}]$
 - L'étape 6 affecte $d = 4$
 - La deuxième itération applique le tri insertion à chacune des 4 sous-séquences d'incrément 4 $[e_0, e_4, e_8, \dots, e_{48}], [e_1, e_5, e_9, e_{13}, \dots, e_{49}], \dots$
 - L'étape 6 affecte $d = 1$
 - La troisième et dernière itération applique le tri insertion à tout le tableau.
- 1. Écrire une fonction `newSort (T, n)` implémentant l'algorithme décrit ci-dessus.
- 2. Comparer expérimentalement la complexité en temps de cet algorithme avec celui du tri insertion.