

## 1 Problématique

Lors du cours précédent nous sommes revenus sur plusieurs algorithmes de tris étudiés en Première et nous avons étudié leurs performances. Python fournit une fonction native *sorted* et la méthode de liste équivalente *sort*. Nous constatons que cette fonction propose des performances bien meilleures que celles des algorithmes que nous connaissons.

Quel algorithme de tri est implémenté dans la fonction *sorted* ?

## 2 Nouvelle approche

### 2.1 Résoudre des petits problèmes...

La propriété triviale suivante va nous permettre de construire une nouvelle méthode de tri :

« Une liste qui contient 0 ou 1 élément est triée. »



FIGURE 1 – Deux listes triées

Ainsi deux listes de un élément chacune peuvent être fusionnées en une liste triée de deux éléments.

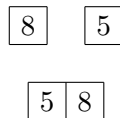


FIGURE 2 – Fusionner 2 listes de 1 élément

En résolvant des petits problèmes, nous pouvons remonter à des problèmes plus importants en appliquant le même principe.

### 2.2 ...pour solutionner un gros problème

Essayons de nous ramener à de petits problèmes. Considérons une liste non triée :

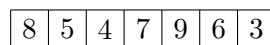


FIGURE 3 – Un gros problème

Pour se ramener à un problème plus petit, séparons la liste en deux listes :

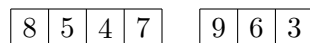


FIGURE 4 – Liste gauche et liste droite

Il suffit de répéter notre étape de séparation sur les sous-listes engendrées, jusqu'à obtenir des listes de un élément maximum.

## 2.3 Diviser pour régner : un algorithme récursif

Séparer un problème en sous-problèmes se construit très facilement de manière récursive en appelant la fonction de tri sur chaque sous-liste créée.

$$tri(liste) = \begin{cases} liste & \text{si la taille de liste} \leq 1 \\ fusionner(tri(liste \text{ gauche}), tri(liste \text{ droite})) & \text{sinon} \end{cases}$$

Cet algorithme est celui du *tri fusion*.

## 2.4 L'étape de fusion

Il est aisé de construire une liste triée à partir de deux listes déjà triées. Il suffit de prendre le plus petits éléments parmi nos deux listes et l'ajouter à la liste finale, et ce jusqu'à épuisement des listes.

**Activité 1 :** Écrire une fonction `fusionner(gauche : list, droite : list) → list` qui renvoie une liste triée composée des éléments de *gauche* et *droite* déjà triées.

## 2.5 Le code complet du tri fusion

**Activité 2 :** En s'appuyant sur la description récursive de l'algorithme au paragraphe 2.3, écrire une fonction `tri_fusion(l : list) → list` qui renvoie la liste *l* triée.

# 3 Performances du tri fusion

## 3.1 Comparaison

**Activité 3 :**

1. Créer un tuple *l* de 10000 entiers compris entre 0 et 1000.
2. En utilisant la fonction (*duree\_tri*) créée précédemment, effectuer une mesure de la durée d'exécution du tri fusion sur une liste dérivant du tuple *l*.
3. Effectuer la même mesure avec la fonction *sorted*.
4. Proposer plusieurs explications quant à la différence constatée.

## 3.2 Complexité

### 3.2.1 Découper en sous-listes

À chaque appel de la fonction *tri\_fusion* nous divisons la liste en deux. La question à se poser est de savoir combien de fois faut-il couper la liste en deux pour obtenir des listes de un élément. Mathématiquement nous cherchons *a* tel que :

$$\frac{n}{2^a} = 1$$

Le logarithme base 2 noté  $\log_2$  se définit :  $\log_2(2^x) = x$ .

**Activité 4 :** Déterminer la valeur de *a* en fonction de *n*.

### 3.2.2 Fusionner deux listes

La fonction *fusionner* réalise *n* comparaisons pour assembler deux listes de taille  $\frac{n}{2}$ .

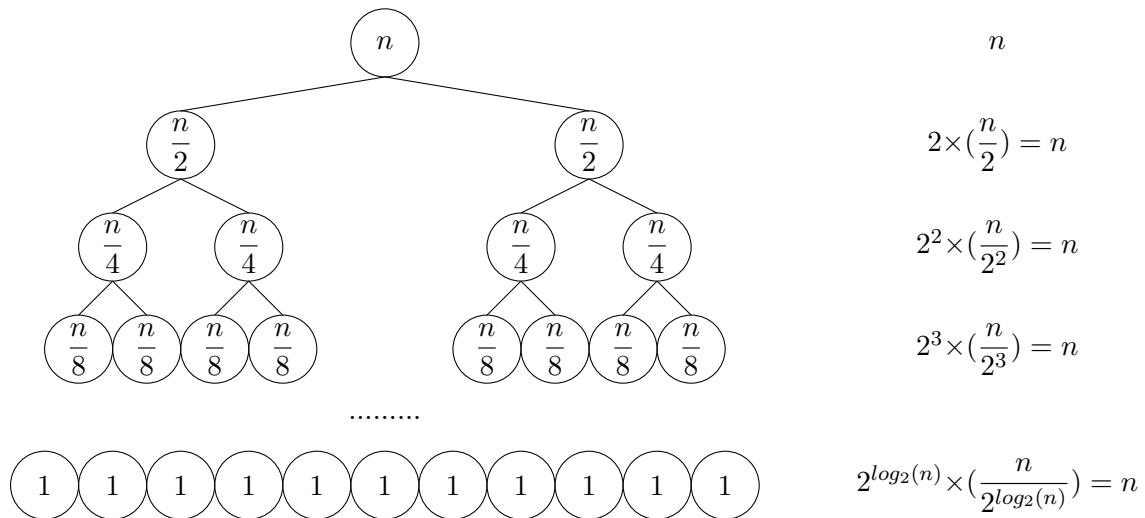


FIGURE 5 – Nombre de comparaisons

### 3.2.3 Complexité du tri fusion

Chaque niveau de fusion a un coup de  $n$  et il y a  $\log_2(n)$  niveaux.

La complexité du tri fusion est en  $O(n \times \log_2(n))$ .

### 3.3 Stabilité

On dit qu'un algorithme de tri est stable s'il ne modifie pas l'ordre initial des clés identiques.

#### Activité 5 :

1. À la main sur quelques exemples, vérifier si le tris par sélection et par insertion sont stables.
2. Vérifier la stabilité du tri fusion.

## 4 La fonction native *sorted*

Elle implémente l'algorithme *Timsort* mis au point par Tim Peters en 2002. C'est un algorithme hybride de plusieurs tris.

**Activité 6 :** Réaliser une présentation de l'algorithme *Timsort*. Il n'est pas demandé d'effectuer une étude théorique précise mais d'expliquer le fonctionnement général et les choix de Tim Peters.