Tris Récursifs

1 Tri rapide (quicksort)

1.1 Présentation

La méthode consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en 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.

Cette opération s'appelle le partitionnement. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété **récursivement**, jusqu'à ce que l'ensemble des éléments soit trié.

Concrètement, le principe pour trier une liste L est le suivant :

- Si L est vide ou admet un seul élément elle est triée (c'est notre critère d'arrêt).
- sinon :
  - On choisit un élément e $\in$ L quelconque, que l'on retire de L
  - On construit deux sous-listes : Li formée des éléments inférieurs à e et Ls formée des éléments plus grands.
  - Le résultat est la concaténation de Li, récursivement triée, de [e] et de Ls, elle aussi récursivement triée.

1.2 Exemple

Exercice 1. Trier à l'aide du tri rapide en choisissant comme pivot le premier élément de la liste la liste suivante : L=[16,4,32,8,2,64,1,128]

1.3 Implémentation en Python

Nous choisissons ici le premier élément de la liste comme pivot.

Exercice 2. Écrire une fonction `trirapide` qui réalise le tri rapide.

1.4 Complexité dans le pire des cas

On admet que le pire cas du tri rapide, est, lorsque à chaque appel récursif, une des deux listes est vide et l'autre contient un élément de moins que lors de l'appel récursif. Dans le cas où le premier élément est systématiquement le pivot choisi, cela revient à dire que l'on travaille sur une liste déjà triée.

Exercice 3. Montrer avec ces hypothèses que le tri rapide a pour complexité dans le pire des cas en $O(n^2)$.

1.5 Complexité dans le meilleur des cas

On admet que le meilleur des cas du tri rapide est, lorsque à chaque appel récursif, les deux listes inférieures ou supérieures contiennent autant d'éléments ou un élément d'écart seulement (ce n'est pas si simple à montrer).

Exercice 4 (Complexité du tri rapide dans le meilleur des cas)**.** On étudie ici un preuve informelle de la complexité dans le meilleur des cas du tri rapide.

On note $T(n)$ le nombre de comparaisons nécessaires pour trier un tableau de taille n dans le meilleur des cas. On admet que la fonction T est croissante et on étudie le cas $n = 2^m - 1$

1. Montrer que $T(2^m - 1) \leqslant 2T(2^{m-1} - 1) + 2^m - 2$
2. Que vaut $T(1)$? Et $T(2)$?
3. Montrer que $T(n) = O(n \log n)$.

.

Théorème 1 (Complexité du tri rapide dans le meilleur des cas)**.** *Le tri rapide a pour complexité* $O(n \log n)$ *dans le meilleur des cas pour un tableau de taille* n*.*

1.6 Complexité en moyenne

La complexité en moyenne est le nombre moyen d'opérations nécessaires (asymptotiquement en fonction de n) pour trier une liste de longueur n.

Elle n'est pas directement au programme, néanmoins il est bon de la garder à l'esprit : un algorithme qui serait dans le meilleur des cas en $\Theta(n)$ et dans le pire des cas en $\Theta(n^2)$ peut ainsi être très efficace si il est en moyenne en $\Theta(n)$ ou beaucoup moins efficace si il est en moyenne en $\Theta(n^2)$.

Les complexités en moyenne des trois tris du programme dépendent d'outils probabilistes : elles seront étudiées en TD de Mathématiques.

Théorème 2 (Complexité du tri rapide dans le meilleur des cas ; admis)**.** *Le tri rapide a pour complexité* $O(n \log n)$ *en moyenne pour un tableau de taille* n*.*

1.7 Complexité en mémoire

Une mise en œuvre naïve du tri rapide utilise un espace mémoire proportionnel à la taille du tableau dans le pire cas. Il est néanmoins possible de limiter la quantité de mémoire à $O(\log n)$

2 Tri fusion

2.1 Présentation

Le tri fusion repose sur une stratégie de type "diviser pour régner". Le principe est le suivant :

- Pour rassembler deux tableaux T1 et T2 déjà triés, on peut les interclasser de manière suivante : on compare les plus petits éléments de chacun d'eux, on place le plus petit des deux dans un nouveau tableau T et on poursuit cette opération jusqu'à épuisement de l'un des deux tableaux. On complète alors T en ajoutant à la fin de celui-ci les éléments du tableau non vide.
- Le tri fusion est alors défini de la manière suivante :
  - Si le tableau T a au plus un élément il est déjà trié

– Si le tableau T a deux éléments ou plus, on partage T en deux sous-tableaux de même taille lorsque possible, on appelle récursivement la fonction sur chacun des sous-tableaux et on interclasse les copies triées.

2.2 Exemple

Exercice 5. Trier à l'aide du tri fusion la liste suivante : L=[16,4,32,8,2,64,1,128]

2.3 Implémentation

L'implémentation du tri fusion va reposer sur deux fonctions : une première fonction `interlac` qui permet d'inter-lacer deux tableaux L1 et L2 déjà triés pour en faire un tableau trié unique et une fonction fusion qui va réaliser la procédure récursive.

La fonction interlac va utiliser deux compteurs i et j, qui correspondront aux positions locales dans le parcours des deux listes L1 et L2.

Exercice 6.
1. Écrire une fonction `interlac(L1,L2)` qui prend en argument deux listes `L1` et `L2` triées et qui renvoie une liste unique triée contenant les éléments de `L1` et `L2`.
2. Écrire une fonction `fusion(L)` qui réalise (récursivement) le tri fusion.

2.4 Complexités

Théorème 3 (Complexités du tri fusion)**.** *La complexité du tri fusion, tant dans le pire des cas, qu'en moyenne ou dans le meilleur des cas est en* $O(n \log n)$

Le tri fusion nécessite $O(n)$ espace mémoire pour être réalisé sur des tableaux.

3 Comparaison des méthodes de tris

Le tableau suivant résume les résultats sur les méthodes de tris étudiées :

Algorithme	Nature	Pire des cas	Meilleur des cas	Moyenne	Mémoire
Tri par insertion	Itératif	$O(n^2)$	$O(n)$	$O(n^2)$	En place
Tri rapide	Récursif	$O(n^2)$	$O(n\log n)$	$O(n\log n)$	$O(n)$ mais $O(\log n)$ possible
Tri fusion	Récursif	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$

4 Application au calcul de la médiane d'une liste de nombres

La médiane m d'un tableau T de nombres est le nombre unique lorsqu'il existe tel qu'il y ait autant d'éléments de T qui soient supérieurs que d'éléments qui sont inférieurs à m.

Lorsque T a un nombre pair d'éléments, la médiane n'est en général pas unique. On parle alors d'une valeur médiane.

4.1 Médiane d'une liste triée

Soit T un tableau de nombres (entiers ou réels) *trié*. Soit n=len(T). T=[T[0],...,T[n-1]]

Si n est impair, on a alors n=2p+1 et m=T[p].

Lorsque n est pair, on a n=2p et n'importe quel nombre compris entre T[p-1] et T[p] est valeur médiane de T. On peut définir dans ce cas la médiane basse comme T[p-1].

Le coût en temps de la recherche de la médiane dans un tableau trié est $O(1)$. Comme on sait trier une liste en $O(n\log n)$, on dispose donc d'algorithmes de recherches de la médiane en $O(n\log n)$

4.2 Liste d'entiers : tri comptage

Lorsqu'on travaille sur un tableau T d'entiers bornés (dont on connaît la borne : ils seront considérés comme compris entre O et M-1), il est possible de faire un tri qui n'effectue pas des comparaisons et qui a une complexité meilleure que $O(n\log n)$. L'extraction de la médiane est ainsi plus rapide.

Le tri comptage s'effectue en deux temps : on crée un tableau C de longueur M contenant des zéros, et on parcourt le tableau T à trier, en ajoutant un à C[i] chaque fois qu'on lit i dans T.

Ensuite, le parcours du tableau C, en ajoutant C[j] fois l'élément j dans une liste TTrie initialement vide permet de trier T

Exercice 7. Écrire les fonctions comptage et tri

Exercice 8. Quelle est la complexité en temps et en mémoire du tri comptage ?

4.3 Et d'autres algorithmes récursifs

Il est possible d'écrire des algorithmes récursifs s'appuyant sur la technique "diviser pour régner" pour trouver une médiane. La recherche de la médiane est alors en temps linéaire sans effectuer de tris, et même pour des données qui ne sont pas de type entier.