TD 31 : Tri rapide

Le tri rapide (*quicksort*) est basé sur la stratégie « diviser pour régner » qui réduit la résolution d'un problème à la résolution de plusieurs sous-problèmes de tailles inférieures. Sa complexité moyenne en $\Theta(n \ln n)$ est optimale pour un tri par comparaison et c'est l'un des tris les plus rapides en pratique sur les tableaux de grande taille.

Ce tri a été créé par Tony Hoare en 1960, à qui l'on doit la logique de Hoare pour montrer la correction des programmes. Il est également le premier auteur d'un compilateur complet ; il a reçu le prix Turing en 1980. Le tri rapide est quadratique dans le pire des cas, mais cela ne pose pas de problème en pratique.

L'efficacité du tri repose sur la possibilité de partitionner rapidement, et en place, un tableau en deux tableaux sur lesquels portent les appels récursifs. Aucun travail n'est requis après les 2 appels récursifs, contrairement au tri fusion.

Principe du tri
Soit un tableau `t` à trier, de taille `n` :
- On choisit un élément `p` dans le tableau, appelé **pivot**.
- On réarrange le tableau (partitionnement) en trois parties consécutives : les éléments inférieurs ou égaux à `p` , le pivot `p` et les éléments supérieurs à `p`.
- On trie récursivement les deux sous-tableaux des éléments respectivement inférieurs ou égaux et supérieurs à `p` . Le tableau est alors trié.

Les propriétés évidentes du tri rapide :
- Le tri rapide n'est pas stable : les éléments égaux à p situés après p se retrouveront avant le pivot.
- L'efficacité du tri rapide dépend du choix du pivot.
- Le tri rapide peut être réalisé en place.
- Le tri rapide n'opère pas plus rapidement sur un tableau presque trié.

Le choix du pivot est généralement réalisé selon l'une des méthodes suivantes :
- prendre le premier élément de la partie du tableau à partitionner (choix le plus simple)
- prendre l'élément en milieu de tableau (meilleur choix pour un tableau déjà trié)
- prendre un élément aléatoirement (meilleur choix en moyenne).
- prendre un élément proche de la médiane du tableau, si on la connaît et si on peut trouver un tel élément sans sur-coût.

La difficulté d'implémentation d'une version en place du tri rapide réside dans la bonne gestion des indices de début et de fin de la partie de tableau en train d'être triée et la détermination de la position finale du pivot dans cette partie.
Pour rendre le tri rapide performant, il faut réaliser le moins d'échanges possibles dans le partitionnement, d'où différentes variantes classiques afin d'optimiser ce partitionnement.

On remarquera que le second appel récursif est terminal puisqu'aucun travail n'est réalisé ensuite. Une astuce classique consiste donc à réaliser cet appel sur la partie de plus grande taille. Cette idée permet également d'éviter le débordement de pile potentiel de la fonction de tri (la taille de la pile est alors logarithmique en la taille du tableau dans le pire des cas).
Le tri rapide gagne à être remplacé par un tri quadratique lorsque l'appel porte sur un tableau de très petite taille (moins de 5 éléments).

Exercice 1 (Tri rapide d'un tableau)

On effectue le tri en place d'un tableau (pas de copie du tableau). La fonction de partition prend en argument le tableau `t` et deux indices délimitant la portion du tableau à trier : les indices `g` et `d`. Le partitionnement en deux parties est réalisé avec le premier élément `t[g]` de la tranche comme pivot.

1. Ecrire une fonction `partition(t, g, d)` qui partitionne la tranche du tableau compris entre les indices `g` et `d`. Cette fonction :
   - utilise le premier élément de cette tranche comme pivot : `p = g` (attention, il est essentiel ensuite de prendre $p = g + 1$ pour la position frontière).
   - compare `t[i]` à `t[g]`, pour tous les indices `i` compris entre `g+1` et `d`. Si `t[i]<=t[g]`, on incrémente l'indice `p` qui représente la position frontière et on réalise l'échange des éléments d'indice i et `p`. On a toujours $g \leq p \leq i \leq d$.
   - On place `t[g]`.
   - retourne la position `p` du pivot
2. Ecrire une fonction `tri_rapide(t)` appelant une fonction locale `tri(g, d)` qui, si `g < d` réalise le partitionnement, puis s'appelle récursivement sur les deux parties du tableau. Pour finir, la fonction `tri_rapide(t)` appelle la fonction `tri` sur l'ensemble du tableau à trier (attention aux indices négatifs ou hors de bornes, on pourra écrire $\max(0, ...)$ pour affecter 0 en cas de valeurs négatives).
3. Tester votre fonction sur des tableaux d'entiers.

   (On comptera les comparaisons et les échanges dans la suite pour la complexité)
4. Montrer que la complexité temporelle est quadratique dans le pire des cas (on admet qu'il s'agit du cas où le partitionnement contient systématiquement une liste de taille 1)
5. Quelle est la complexité temporelle dans le meilleur des cas ? (on admet qu'il s'agit du cas où le partitionnement se fait systématiquement en deux listes de tailles égales (ou "presque"))
6. Modifier la fonction de tri rapide (en une nouvelle fonction `tri_rapide2`) pour utiliser comme pivot un élément choisi au hasard dans le tableau. On évite ainsi la complexité temporelle dans le cas d'un tableau déjà ordonné.

Exercice 2 (Version de Hoare du tri rapide)

La méthode décrite par Hoare pour le partitionnement est différente. On parcourt les éléments du tableau à l'aide de 2 indices `g` et `d`. L'indice de gauche est incrémenté tant que l'on rencontre des valeurs inférieures ou égales au pivot. L'indice de droite est décrémenté tant que l'on rencontre des valeurs supérieures au pivot. Les valeurs `t[g]` et `t[d]` sont alors permutées puisqu'elles sont toutes les 2 mal placées, puis on incrémente `g` et on décrémente `d`. Et on recommence jusqu'à ce que les indices se croisent. On place enfin le pivot à sa bonne place en l'échangeant avec l'élément d'indice `d`.

1. Ecrire une fonction `partition2(t, g, d)` qui partitionne la tranche du tableau `t` comprise entre les indices `g` et `d` selon la méthode de Hoare.
2. Testez le tri rapide avec cette méthode, on pourra mesurer le nombre de comparaisons réalisées pour tester la pertinence de cette approche.