Lycée Vaugelas MPSI 2021 - 2022
TD d'informatique n° 11 du 29-30/11/2021 **Tris quadratiques**

Un **tri par comparaison** repose sur l'existence d'un **préordre** (relation binaire réflexive et transitive) **total** (deux éléments sont toujours comparables) sur l'ensemble des valeurs des éléments d'un tableau. Nous pouvons trier des entiers pour la relation d'ordre $\leq$, mais aussi des chaînes de caractères pour l'ordre lexicographique. Une fonction de tri peut souvent être **polymorphe**, c'est-à-dire qu'elle accepte des tableaux de différents types. Un préordre antisymétrique est une relation d'ordre.

Caractéristiques importantes d'un algorithme de tri

- La complexité temporelle en moyenne ou dans le pire des cas, souvent assimilée au nombre $T(n)$ d'opérations élémentaires nécessaires pour trier le tableau de taille n. On se limite dans un premier temps au nombre de comparaisons, mais le nombre de lectures, et surtout d'écritures, de valeurs en mémoire est aussi important.
- La complexité spatiale dans le pire des cas
- Le caractère « en place » du tri ou inversement la préservation du tableau initial.
- Le caractère stable d'un tri : l'ordre de deux éléments équivalents est préservé. Si le tableau est trié selon une partie de la donnée seulement, c'est-à-dire que l'on utilise une relation de préordre : on peut avoir $x \leq y$ et $y \leq x$ sans avoir $x = y$. Par exemple la relation $(x_1, y_1) \leq (x_2, y_2) \Leftrightarrow x_1 \leq x_2$ est un préordre total sur $\mathbb{N}^2$.
- La possibilité de paralléliser les opérations de tri pour bénéficier des capacités multitâches des processeurs.

Les complexités s'étudient en fonction de la taille n du tableau à trier (ordre de grandeur ou majoration asymptotique). Les tris par comparaison classiquement présentés dans le cours d'algorithmique se divisent alors en deux catégories :

- Les **tris quadratiques**, comme le tri à bulle, le tri par insertion et le tri par sélection. Leur complexité temporelle est en $\Theta(n^2)$ dans le pire des cas.
- Les **tris quasi-linéaires**, comme le tri fusion, le tri rapide, le tri par tas. Leur complexité temporelle est en $\Theta(n \ln n)$ dans le pire des cas.

Le tri d'un tableau peut constituer un traitement préalable en vue d'optimiser des opérations ultérieures. Ainsi, pour rechercher de nombreuses valeurs dans un tableau, il peut s'avérer algorithmiquement efficace de commencer par trier entièrement le tableau, car le coût d'une recherche dichotomique est logarithmique. **L'usage assez systématique de tableaux pour implémenter les algorithmes de tri provient de la propriété de lire ou d'écrire une valeur en temps constant.** La nature mutable d'un tableau permet de réaliser le tri en place lorsque l'algorithme le permet, ce qui réduit la quantité de mémoire nécessaire.
L'utilisation de structures persistantes, notamment pour la programmation fonctionnelle, rend nécessaire la construction de nouvelles données en mémoire et la notion de tri en place n'a plus de sens : la fonction de tri renvoie un nouveau tableau, de même taille que le tableau dont la référence est transmise en argument.

Consignes pour l'exercice

Les seules opérations autorisées dans l'exercice suivant sont la comparaison de deux éléments par $\leq$ et la permutation de deux éléments t_i et t_j d'un tableau $t = [t_0, \cdots, t_{n-1}]$, pour $0 \leq i, j < n$. Les fonctions qui nécessitent un parcours du tableau ne sont pas autorisées (comme `min` et `max`).

Exercice 1 (Tri par sélection (*selection sort*))

On recherche le plus petit élément du tableau que l'on échange avec le premier. On applique la même méthode au sous-tableau restant, tant qu'il contient au moins deux éléments. C'est le tri utilisé pour la photo de groupe : on place le plus petit de ceux qui ne sont pas encore installés juste derrière ceux qui sont déjà installés.

1. Écrire une fonction `permute (t,i,j)` qui permute les éléments d'indices `i` et `j` d'un tableau `t`.
2. Écrire la fonction `minimum(t,j)` qui retourne l'indice du minimum de la partie du tableau `t` comprise entre les indices `j` et `n-1`, où n est la taille du tableau.
3. Donner l'invariant de boucle qui justifie la correction de la fonction `minimum`.
4. Écrire l'algorithme de tri par sélection en pseudo-code, en utilisant les fonction précédente. Réaliser préalablement les schémas nécessaires à la visualisation de l'algorithme.
5. Écrire la fonction `tri_selection (t)` qui trie le tableau `t` **en place** par la méthode du tri par sélection.
6. Proposer un invariant de boucle qui justifie la correction de la fonction `tri_selection (t)`. Démontrer cette propriété ainsi que la correction de la fonction de tri.
7. Calculer le nombre de comparaisons réalisées par le tri par sélection sur un tableau de taille n dans le meilleur des cas, le cas moyen, le pire des cas.
8. Quel est le nombre d'échanges réalisé par l'algorithme, dans le pire des cas, le meilleur des cas ?
9. Quelle est la complexité temporelle du tri par sélection ?
10. Quelle est la complexité spatiale de l'algorithme ?
11. Étudier la stabilité de cet algorithme de tri. Le cas échéant, comment modifier l'algorithme pour qu'il devienne stable ?
12. Quels avantages/inconvénients peut-on retenir pour cet algorithme. Dans quel cas le conseiller ?
13. Calculer le nombre d'échanges réalisés dans le cas moyen par le tri par sélection.
14. L'approche récursive du tri par sélection est naturelle, ce tri a d'ailleurs été défini de manière inductive.

    Écrire des fonctions récursives `minimum` et `tri_selection`. Ne pas chercher à rendre la récursivité terminale, mais veiller à ne pas dégrader la complexité temporelle.
15. Vérifier que l'implémentation récursive présente la même complexité temporelle que l'implémentation impérative, sinon la modifier.
16. Vérifier les fonctions écrites en triant des tableaux d'entiers aléatoires. Les erreurs seront plus facilement détectées sur de plus petits tableaux que sur de grands tableaux. Il est judicieux de vérifier le comportement pour des cas particuliers bien choisis, avec une attention particulière pour le premier et le dernier élément du tableau.

L'idée du tri par sélection est reprise dans le **tri par tas** où l'on utilise une structure de donnée (un tas) qui permet d'extraire le plus petit élément en $\Theta(\ln n)$ au lieu de $\Theta(n)$ pour un tableau. La complexité temporelle est alors réduite à $\Theta(n \ln n)$, optimale pour un tri par comparaison. Le tri par sélection est utile dans le contexte de dispositifs limités en mémoire où les opérations d'écriture sont parfois lentes et diminuent la durée de vie. Le défaut de ne pas trier plus rapidement un tableau presque-trié est parfois très gênant.