

Les énoncés des travaux pratiques seront disponibles sur internet quelques jours avant la séance correspondante, à l'adresse indiquée ci-dessus. Les questions en rapport avec les TP, Caml Light ou l'informatique sont les bienvenues et peuvent être envoyées à victor.nicollet@ens.fr

L'objectif de cet énoncé est de présenter deux algorithmes de tri peu utilisés, qui sont le *tri par tas* et le *tri par base*.

1 TRI PAR TAS

Le tri par tas utilise une structure appelée tas, qui est en réalité un arbre binaire ayant deux propriétés:

- La racine de l'arbre est plus grande que tous ses autres éléments.
- Les sous-arbres gauche et droit sont des tas.

▷ **Question 1.** On suppose qu'un arbre vérifie la deuxième propriété mais pas la première. Sans écrire de code, proposer un algorithme de *percolation* qui transforme cet arbre en tas. Quelle est sa complexité? ◁

L'intérêt est que l'on peut trouver le plus grand élément d'un tas en temps constant, donc qu'on peut améliorer la performance d'un tri par sélection en utilisant un tas plutôt qu'une recherche linéaire. Néanmoins, il faut maintenir la structure de tas à chaque fois qu'on extrait un élément, de même que pouvoir transformer un tableau en tas.

On choisit de représenter les tas par des tableaux. La racine qui se trouve à l'indice i a pour fils gauche l'indice $2i$ et pour fils droit l'indice $2i + 1$. Cela est représenté par les fonctions ci-contre. Un tas de taille n est toujours de profondeur $\log_2 n$, par construction.

```
let p i = i/2;;  
let g i = i*2;;  
let d i = i*2+1;;  
let e n i = i >= 0 && i < n;;
```

Les éléments de 0 à $n - 1$ dans le tableau sont le tas, les éléments au-delà sont ceux déjà triés. Trier revient donc à prendre l'élément le plus grand du tas, à le mettre à la position $n - 1$, et à diminuer la taille du tas de 1 .

▷ **Question 2.** Implémenter une fonction récursive `percoler t n i` qui applique l'algorithme de percolation au sous-arbre du tas (t, n) enraciné en i . ◁

▷ **Question 3.** En déduire une fonction `transformer_en_tas t` qui transforme le tableau t en un tas, avec n égal à la taille du tableau. Quelle est sa complexité? ◁

▷ **Question 4.** Comment peut-on retirer un élément du tas en conservant la propriété de tas? Quelle est sa complexité? Programmer cet algorithme puis écrire l'algorithme de tri par tas et déduire sa complexité. ◁

▷ **Question 5.** Quelle critique pourrait-on faire à cet algorithme, par rapport à des algorithmes plus classiques (tri par insertion, tri rapide, tri fusion)? Quels en sont les avantages et les inconvénients? Quel choix faire dans quelle situation? ◁

2 TRI PAR BASE

Le tri par base utilise une propriété amusante de l'ordre lexicographique sur un alphabet fini. On prend une liste de mots de même taille, et on la partitionne suivant leur dernière lettre avant de recoller les partitions par ordre alphabétique. On recommence avant l'avant-dernière lettre, et ainsi de suite jusqu'à la première. Lorsqu'on a fini, la liste est triée.

▷ **Question 6.** S'en convaincre (c'est-à-dire le prouver). ◁

On va travailler ici sur les entiers considérés comme des mots sur l'alphabet $0, 1$ (puisque les entiers naturels en Objective Caml vont de 0 à $2^{30} - 1$, ce sont des mots de longueur 30. On rappelle que le k -ième chiffre d'un entier n écrit en base deux est $\text{land } (1 \text{ lsl } n)$.

▷ **Question 7.** Écrire une fonction `couper i l` qui sépare la liste d'entiers l en deux listes, suivant que le i -ième bit de chaque entier (le coefficient de 2^i dans son écriture en base 2) est égal à 0 ou à 1. L'ordre des entiers dans chaque sous-liste doit être le même que dans la liste d'origine. Quelle est sa complexité? ◁

En théorie, on devrait recoller les listes, puis les scinder à nouveau suivant le bit suivant. C'est lent: concaténation, donc temps de calcul linéaire. Au lieu de ça, on constate que recoller les listes puis les scinder est équivalent à scinder les listes puis les recoller.

▷ **Question 8.** Modifier la fonction `couper` pour qu'elle fasse le recollement en même temps qu'elle scinde la liste. La fonction sera de la forme `couper i l (a0, a1)` où `a0` et `a1` sont des listes déjà scindées suivant le bit i auxquelles il faut recoller le résultat de l'opération de découpage. ◁

▷ **Question 9.** Définir la fonction `tri_par_base`, qui utilise la fonction de découpage 30 fois pour regrouper nos éléments. Quelle est sa complexité? Quels sont les facteurs importants à prendre en compte? ◁

3 QUESTION DIFFICILE

▷ **Question 10.** Supposons que l'on veuille trier non pas des entiers, mais des chaînes de caractères sur un alphabet défini à l'avance. Quelles solutions proposer ? ◁