Listes et tris

Objectifs

— Coder un algorithme de tri, avec la fonction d'ordre en paramètre
— Optimiser un algorithme de tri pour qu'il passe à l'échelle

Réalisations attendues

— Indispensable : Exercices 1 à 7
— Bonus "ludique" : Exercice 8
— Bonus + : Exercice 9

Rappel : tester à l'aide de Utop

— `dune utop` – lance utop
— `open Tp2 ;;` – charge le module `Tp2` i.e les fonctions contenues dans le fichier `tp2.ml`
— `insert (fun x y -> x<y) 3 [2;4;5];;` – par exemple pour tester la fonction `insert`
— `exit 0;;` – pour quitter

1 Algorithmes de tri

1.1 Tri par insertion

Le tri par insertion constitue un algorithme simple (quoiqu'inefficace) adapté aux listes. Son principe est de partir d'un ensemble résultat initialement vide, et d'y insérer successivement et dans l'ordre les éléments provenant d'un ensemble à trier.

▷ **Exercice 1** *Écrire une fonction qui prend en argument un ordre (fonction de comparaison des éléments), un élément e et une liste triée, et qui insère e à sa place. Le résultat sera également une liste triée.*

▷ **Exercice 2** *Écrire une fonction qui prend en argument un ordre et une liste et qui trie cette liste selon cet ordre, en utilisant la fonction d'insertion précédente.*

1.2 Algorithme du tri sur le principe «*split/join*»

De nombreux algorithmes de tri reposent sur le principe «*split/join*» qui consiste à décomposer la structure de données (liste, tableau, ou autre ...) à trier en deux sous-structures, sur lesquelles on applique récursivement la fonction de tri, puis à recomposer une structure triée à partir des deux résultats.

Algorithme du tri fusion L'algorithme de tri est un algorithme de la famille «*split / join*». La phase de décomposition consiste à couper la structure de données en deux sous-structures de taille égale sur lesquelles on applique récursivement la fonction de tri, puis la phase de recomposition intercale le contenu des deux résultats.

▷ **Exercice 3 (Décomposition)** *Écrire une fonction qui coupe une liste passée en argument en deux sous-listes. Pour l'efficacité de l'algorithme de tri, veiller à ce que les tailles des deux sous-listes soient les plus proches possibles.*

▷ **Exercice 4 (Recomposition)** *Le principe de la recomposition (appelée aussi "tri intercalaire" ou "merge sort") consiste à construire une nouvelle liste triée à partir de deux listes triées selon le même ordre en épuisant alternativement ces deux listes.*

Attention : il ne s'agit en aucun cas de faire l'union des deux listes mais de les parcourir simultanément et de construire le résultat au fur et à mesure.

▷ **Exercice 5 (Tri fusion)** *Écrire une fonction réalisant le tri fusion d'une liste en utilisant les fonctions de décomposition et de recomposition précédentes.*

▷ **Exercice 6** *Pourquoi la fonction de décomposition doit-elle couper la liste à trier en deux morceaux de même taille ?*

2 Tri et récursivité terminale

Les fichiers `lexer.mll` et `parser.mly` permettent d'analyser le fichier `nat2016.txt` contenant les statistiques de choix de prénoms entre 1900 et 2016. Ce fichier est accessible sur le site du gouvernement : `https://www.data.gouv.fr/fr/datasets/fichier-des-prenoms-edition-2016/` et de l'INSEE `https://www.insee.fr/fr/statistiques/2540004#consulter`.

La liste `listStat` est construite à partir de l'analyse du fichier `nat2016.txt`. C'est une liste de quadruplet :

- sexe : entier 1 (homme) ou 2 (femme)
- prénom
- année
- nombre de fois où ce prénom a été donné cette année-là

La Liste contient 605 463 éléments.

Sur le même principe `listStatHomme` contient 41 407 éléments qui sont les hommes ayant des prénoms commençant par A ou B.

▷ **Exercice 7** *Dans cet exercice nous allons tester et comparer l'efficacité des différents algorithmes de tri.*

1. *Utilisez votre fonction de tri par insertion pour trier la liste* `listStatHomme` *(les prénoms les plus donnés en premier). Que constatez-vous ?*
2. *Utilisez votre fonction de tri fusion pour tirer la liste* `listStatHomme` *(les prénoms les plus donnés en premier). Que constatez-vous ?*
3. *La complexité théorique du tri par insertion est* n^2 *et celle du tri fusion est* $n*log(n)$*. En utilisant la procédure* `time` *du module* `Sys`*, vérifiez que votre tri fusion est bien plus rapide que votre tri par insertion.*

4. *Utilisez votre fonction de tri fusion pour trier la liste* `listStat` *(les prénoms les plus donnés en premier). Que constatez-vous ?*
5. *Utilisez la fonction* `sort` *du module* `List` *pour trier la liste* `listStat` *(les prénoms les plus donnés en premier). Que constatez-vous ?*

Votre fonction de tri n'aboutit pas (`Stack overflow`) alors que la fonction proposée par OCaml n'a pas ce problème. Une des raisons de cet échec est qu'aucune des trois fonctions qui composent le tri n'est récursive terminale, alors que deux le sont dans la librairie OCaml. Le but de l'exercice suivant est des transformer votre tri fusion pour qu'il prenne moins de place en mémoire et soit donc capable de trier une liste de grande taille.

▷ **Exercice 8** *Transformer le tri fusion pour introduire de la récursivité terminale.*

1. *Écrire une nouvelle fonction qui scinde une liste en deux. Elle doit être récursive terminale et utilisera la longueur de la liste pour couper la liste au milieu.*
2. *Écrire une fonction qui fusionne deux listes triées selon un ordre en une liste unique triée selon la négation de l'ordre (triée "à l'envers"). Elle doit être récursive terminale et utilisera un accumulateur pour construire la liste triée.*
3. *Écrire une fonction de tri utilisant les deux fonctions précédentes. Elle se base sur deux fonctions auxiliaires, mutuellement récursives. L'une trie selon l'ordre et l'autre trie selon la négation de l'ordre. Noter que ces deux fonctions sont très proches et peuvent être remplacées par une unique fonction auxiliaire prenant un booléen en paramètre supplémentaire.*
4. *Vérifier que cette nouvelle fonction réussit à trier la liste* `listStat`*.*

3 Récupération d'informations dans une liste

Nous souhaitons maintenant pouvoir extraire des informations des listes.

▷ **Exercice 9** *Récupérer les n plus petits éléments d'une liste.*

1. *Écrire une fonction qui garde les n premiers éléments d'une liste.*
2. *En utilisant la fonction précédente, écrire une fonction qui renvoie, de façon naïve, les n premiers éléments d'une liste pour un ordre donné.*
3. *La fonction précédente trie toute la liste pour ne garder ensuite que les n premiers éléments. Écrire une fonction plus efficace. Elle se basera sur une liste accumulateur de taille fixe dans laquelle seront stockés (dans l'ordre) les n plus petits éléments rencontrés.*
4. *Écrire une fonction qui renvoie les n années où un prénom a été le plus donné.*
5. *Écrire une fonction qui renvoie les n prénoms les plus donnés une année donnée.*
6. *Vérifier que la fonction qui ne trie pas toute la liste est plus rapide que celle qui trie toute la liste pour ensuite la tronquer.*