COMPARAISON EXPÉRIMENTALE DES DIFFÉRENTS TRIS

Pour comparer expérimentalement les fonctions de tris, nous avons besoin de grands tableaux ou listes quelconques, qu'il est évidemment pénible de créer à la main.

Nous allons donc utiliser un module CAML, nommé « random », qui contient des fonctions de génération aléatoires de nombres, etc. Un module est un paquet de fonctions qui ne sont pas chargées automatiquement au démarrage de CAML pour ne pas encombrer la mémoire (comme les paquets Maple).

Si une fonction est dans un module, on peut l'utiliser en donnant son nom complet (on dit aussi nom qualifié) `module__fonction` : entre le nom du module et celui de la fonction, vous placez un double caractère sousligné (underscore).

Autre solution : charger le module complet en mémoire par l'instruction `#open "module";;` et utiliser les fonctions par leur nom simple. Pour décharger la mémoire si vous n'avez plus besoin du module : `#close "module";;`.

1 Module "random"

La fonction `random__int n`, de paramètre un entier relatif n, génère un entier aléatoire compris entre 0 et $n-1$.

De même, la fonction `random__float x`, de paramètre un flottant z, génère un flottant aléatoire compris entre 0. et z.

Pour ajouter au caractère aléatoire de la création, on peut initialiser un paramètre appelé la graine du générateur aléatoire : vous choisissez une valeur entière arbitraire `valeur` et vous exécutez l'instruction `random__init valeur;;`.

Pour créer un tableau aléatoire de k entiers compris entre 0 et n, vous pouvez donc utiliser la fonction suivante :

```
let tab_alea k n =
    let t = make_vect k 0 in
    for i = 0 to (k - 1) do
        t.(i) <- (random__int n)
    done;;
    t;;
```

Pour une liste aléatoire :

```
let liste_alea k n =
    let rec hasard_list k =
        if k = 0 then []
        else (random__int n) :: hasard_list (k - 1)
    in
    hasard_list k;;
```

2 Mesure du temps de calcul

Le module "sys" contient des fonctions qui permettent d'utiliser les ressources de la machine. Entre autres, il contient la fonction `sys__time ()`, de type `unit -> float`, qui donne le temps de calcul total depuis le début de la session. Elle permet donc d'évaluer le temps réel de calcul d'un objet.

Malheureusement, selon le système, elle donne une mesure du temps qui est quantifiée : on obtient un nombre flottant qui est un multiple d'un temps élémentaire, donc en-dessous de ce temps élémentaire, elle rend `0.0`, ce qui n'est pas très pratique pour travailler avec précision. On peut contourner ce défaut en allongeant artificiellement le temps de calcul d'une fonction, en glissant au milieu du code des boucles inutiles qui ne servent qu'à perdre du temps : bien sûr, si on veut comparer des fonctions, il faut glisser ce même bout de code dans toutes les fonctions à comparer, au niveau le plus interne (au niveau des opérations élémentaires) pour multiplier le temps de calcul par un facteur constant.

3 Comparaison du tri par insertion et du tri fusion

Écrivez les fonctions nécéssaires pour le tri par insertion et le tri fusion, soit pour les tableaux, soit pour les listes.

Générez des structures aléatoires de grande taille (dans un ordre croissant, par exemple de longueur 100, puis 500, puis 1000,...), relevez les temps approximatifs de tri et calculez les rapports entre ces temps. D'après le cours (pas tout à fait... en étant plus précis que nos simples majorations), on devrait trouver un résultat du type

$$\frac{\text{temps par insertion}(n)}{\text{temps par fusion}(n)} \sim K\frac{n}{\ln n}$$

où K est une constante.

Remarque : selon la machine à votre disposition, cela peut être difficile à réaliser en pratique. En effet, si votre machine est rapide mais a peu de mémoire, vous ne pourrez trier que des listes courtes en un temps qui sera trop petit pour être sensible. En revanche, si votre machine est lente mais avec de la mémoire, vous pourrez trier de longues listes, mais il faudra être patient... le mieux serait que la machine soit moyennement rapide avec beaucoup de mémoire.

4 Comparaison du tri fusion et du tri rapide

Faîtes de même pour les deux tris fusion et rapide sur des listes.

Pour des listes aléatoires, on doit trouver un rapport entre les temps de tris sensiblement constant. Mais nous avons vu que le tri rapide n'est pas toujours rapide, si la structure est quasiment triée.

Écrivez une fonction qui construit de grandes listes triées de longueur n et comparez les temps de tris. Cette fois-ci, on devrait trouver un rapport de temps qui est du même type que précédemment.