TP nº 6 - Des Tris

Tous les sujets et les corrigés sont disponibles aux adresses suivantes :
`http://www.lif.univ-mrs.fr/~vpoupet/enseignement/matlab09.php`
`http://www.lif.univ-mrs.fr/~pvanier/?q=cours`

Exercice 1. *marrants thons.*

Dans cet exercice, nous allons voir comment rechercher le plus petit ou le plus grand élément d'un tableau d'entiers.

1. Générez un tableau `tab` de 100 cases contenant des entiers entre 1 et 1000 au hasard.
Indication : Servez-vous de l'aide sachant qu'en anglais « aléatoire » se dit « random ».

Réponse : La fonction `rand(n, m)` renvoie un tableau de dimension $n \times m$ contenant des nombres aléatoires entre 0 et 1. On peut alors multiplier ces nombres par 1000 pour avoir des nombres entre 0 et 1000, et arrondir au dessus pour avoir des entiers entre 1 et 1000 :

```
>> tab = ceil(1000 * rand(1, 100));
```

Attention à l'ordre d'application des fonctions. Si on arrondit avant de multiplier par 1000, on obtient un tableau qui ne contient que des 1000 (pas très aléatoire...). Par ailleurs si vous voulez arrondir à l'entier inférieur il faut utiliser la fonction `floor` (*ceiling* signifie plafond et *floor* signifie plancher, pour les moins anglophones...).

2. Écrivez une fonction `minimum` qui, étant donné un tableau en entrée, renvoie son plus petit élément, puis une fonction `maximum` qui renvoie le plus grand élément d'un tableau.

Réponse :

```
function y = minimum(tab)
y = tab(1);
for i = 2 : size(tab,2)
    if tab(i) < y
        y = tab(i);
    end
end
```

3. Sur un tableau de taille n, combien de comparaisons les fonction `minimum` et `maximum` effectuent-elles ?

Réponse : Chaque élément (sauf le premier) est comparé au minimum/maximum courant. On effectue donc $(n-1)$ comparaisons.

4. Si l'on veut trouver à la fois le plus grand et le plus petit élément d'un tableau, une solution consiste à appeler la fonction `minimum` puis la fonction `maximum`.
Il est toutefois possible de faire une fonction plus efficace en prenant les éléments du tableau deux par deux, en les comparant entre eux puis en comparant le plus grand des deux au maximum courant et le plus petit au minimum courant.
Écrivez la fonction `minmax` qui renvoie le plus petit et le plus grand élément d'un tableau de cette manière. Combien de comparaisons effectue-t-elle sur un tableau de taille n ?

Réponse :

```
function r = minmax(tab)
n = size(tab, 2);
m = tab(n);
```

```
M = tab(n);

for i = 1 : 2 : n-1
    if tab(i) < tab(i+1)
        if tab(i) < m
            m = tab(i);
        end
        if tab(i+1) > M
            M = tab(i+1);
        end
    else
        if tab(i+1) < m
            m = tab(i+1);
        end
        if tab(i) > M
            M = tab(i);
        end
    end
end
r = [m M];
```

Exercice 2. *Tri naïf*

On veut maintenant trier le tableau du plus petit élément au plus grand.

1. Ecrivez une fonction qui trie un tableau de manière naïve :
– on cherche le plus petit élément et on le place en position 1 ;
– on cherche le deuxième plus petit élément que l'on le place en position 2 ;
– etc.
Remarque : On pourra écrire une fonction `minfin` qui prend en argument un tableau `tab` et un entier `i` et qui renvoie l'indice du plus petit élément de `tab` à partir de l'indice `i` (on ignore le début du tableau).

2. Combien de comparaisons doit-on effectuer pour trier un tableau de taille n par cette méthode ?

Exercice 3. *Tri bulle*

Nous allons voir maintenant un des tris les plus faciles à mettre en oeuvre. Le tri bulle consiste à parcourir le tableau en échangeant les positions de deux éléments adjacents si ils ne sont pas dans le bon ordre. Lorsque l'on arrive au bout, on repart du début et l'on traverse le tableau de nouveau, jusqu'à ce que le tableau soit trié.

1. Expliquez pourquoi le tableau est nécessairement trié après au plus $(n-1)$ passages.

2. Ecrivez une fonction `tri_bulle` qui implémente le tri bulle.

3. Combien de comparaisons effectue-t-on en fonction de la taille du tableau en entrée ?

Exercice 4. *Tri fusion*

Les deux tris précédents étaient simples et intuitifs. Ils ne sont cependant pas très efficaces (ils font beaucoup de comparaisons). Nous allons maintenant voir un algorithme bien plus performant (mais aussi assez nettement plus compliqué).
L'idée est d'appliquer le principe « diviser pour régner » :
– si un tableau est de longueur 0 ou 1, il est forcément trié ;
– si un tableau est de longueur supérieure ou égale à 2, on le sépare en deux tableaux, on trie séparément chaque moitié puis on les *fusionne* en un grand tableau trié.

L'idée qui fait fonctionner l'algorithme est que la fusion de deux tableaux triés est assez simple à exécuter et ne demande pas beaucoup de comparaisons.
Intermède : Si vous n'avez pas vraiment compris comment fonctionne le tri fusion (ce qui est assez compréhensible), regardez votre chargé de TP faire une démonstration avec un jeu de cartes.

1. Ecrivez une fonction `fusion(tab1,tab2)` qui fusionne les deux tableaux `tab1` et `tab2` que l'on suppose triés, de manière à obtenir un grand tableau trié.

2. Combien de comparaisons effectue la fonction `fusion` ?

3. Ecrivez la fonction `tri_fusion(tab)` qui trie récursivement un tableau.
Indication : Avec la fonction `fusion` déjà écrite, c'est assez facile. Si le tableau en entrée est de longueur 0 ou 1, on le renvoie directement, sinon on le coupe en deux tableaux, puis on trie les deux morceaux par un appel récursif à la fonction `tri_fusion` (rien à écrire, c'est la magie de la récursivité) et l'on renvoie la fusion des deux tableaux triés.

4. Estimez l'ordre de grandeur du nombre de comparaisons nécessaires pour trier un tableau de taille n par la méthode du tri fusion.