

## TD R1.01.P2 – Semaine 49

### Objectifs du TD

Algorithmes de tris (tri par comptage de fréquence).

### Exercice 1. L'algorithme de tri par comptage de fréquences

Etant donné l'exemple qui vous a été donné en cours, écrivez en Java l'algorithme qui effectue ce tri particulier.

La signature de la méthode est :

```
void triParComptageFreq ( int[] leTab, int nbElem )
```

où :

- `leTab` est le tableau initial à trier par ordre croissant
- `nbElem` est le nombre de valeurs que contient le tableau

#### Première étape

L'algorithme se base sur une hypothèse (forte) de départ : le tableau ne contient que des entiers de valeurs positives (ou égales à zéro) et sur une plage de valeurs faibles ( $\leq 50$  par exemple). Par contre, aucune restriction concernant le nombre d'éléments total dans le tableau.

Ci-dessous, un exemple de tableau qui peut typiquement se trier selon la méthode du comptage de fréquences.

45	0	45	10	15	0	5	10	45
----	---	----	----	----	---	---	----	----

La première étape consiste à compter pour chaque entier du tableau le nombre de fois que ce dernier apparaît : c'est ce qu'on appelle la fréquence d'apparition. Ce comptage de fréquences sera mémorisé dans un nouveau tableau dont la taille est celle de la valeur maximum du tableau initial (*max*) plus un pour tenir compte du zéro (dans l'exemple, la taille de ce nouveau tableau est :  $45 + 1 = 46$ ).

Ecrire cette méthode qui crée et remplit le tableau des fréquences (on l'appellera *tabFreq*) :

```
int[] creerTabFreq ( int[] leTab, int nbElem )
```

où :

- `leTab` est le tableau initial à trier par ordre croissant
- `nbElem` est le nombre de valeurs que contient le tableau
- le retour est un tableau de taille *max+1* qui contient les fréquences d'apparition

Le tableau *tabFreq* renvoyé est tel que *tabFreq[i]* contient le nombre d'apparitions de la valeur « i » dans le tableau initial (*leTab*). Les cas d'erreurs possibles seront traités en TP.

Dans l'exemple du tableau à trier ci-dessus, le tableau des fréquences (*tabFreq*) sera :

0	...	5	...	10	...	15	...	45
2	0	1	0	2	0	1	0	3

On remarque que ce tableau est une image du futur tableau trié car :

- ces indices sont précisément les valeurs que l'on retrouvera dans le tableau trié
- ces indices sont forcément organisés par ordre croissant des valeurs

### Deuxième étape

Ecrire la méthode de tri par comptage de fréquence `void triParComptageFreq ( int[] leTab, int nbElem )`.

Cette méthode commencera par appeler *creerTabFreq(...)* pour récupérer le tableau des fréquences. A partir de ce tableau *tabFreq*, que l'on parcourt de *zéro* à *max* avec un indice « i » :

- récupérer le nombre d'exemplaires de l'entier « i » : *tabFreq[i]* (= *nbExemplaires*)
- écrire dans chaque case à la suite dans le tableau à trier *leTab*, *nbExemplaires* \* l'entier « i »

Au final, on obtiendra bien un tableau *leTab* trié par ordre croissant :

0	1	2	3	4	5	6	7	8
0	0	5	10	10	15	45	45	45

### Exercice 2. Efficacité de l'algorithme de tri par comptage de fréquences

Par un relevé du nombre d'opérations élémentaires, calculez *f* le nombre **approximatif** d'opérations élémentaires exécutées par l'algorithme. Constatez que cette fonction *f* dépend de 2 variables (dont *n* la taille du tableau de départ mais pas que...).

En déduire l'ordre de grandeur  $\theta$  de l'algorithme dans différents cas (à vous de les trouver).

Ecrire la méthode `void testTriParComptageFreqEfficacite()` en suivant les mêmes principes que pour le tri rapide.