Université Paris XII
IUT de Fontainebleau
Département informatique
Licence SIL
Structures de données abstraites
2005-2006

PARTIEL 2

Seuls sont autorisés, à titre de documents, les listings comprenant explicitement le nom de l'étudiant imprimé (et celui-ci seulement) et les notes manuscrites (pas de photocopie) comprenant le nom de l'étudiant sur chaque page.

Il est conseillé d'écrire les exercices en langage C++. L'écriture en Java complique les entrées-sorties et l'écriture en langage C allonge le programme (ce qui ne rapportera pas de point supplémentaire dans un cas comme dans l'autre).

Exercice.- (**Tri par seaux**)

L'idée fondamentale du **tri par seaux** *(*bucketsort *en anglais) est de se procurer un certain nombre de seaux* $S_1, \ldots S_m$*, de parcourir le tableau non trié et de jeter chaque élément dans un seau de façon telle que tous les éléments du seau* S_i *soient plus petits que tout élément du seau* S_j *si* $i < j$*. Ensuite on trie les éléments de chaque seau et on regroupe tout dans un tableau (qui sera donc trié) contenant d'abord les éléments du seau* S_1 *dans l'ordre, puis ceux du seau* S_2 *et ainsi de suite jusqu'au seau* S_m*.*

La seconde idée est de considérer un grand nombre de seaux contenant peu d'éléments. La façon de trier les éléments de chaque seau n'est donc pas très importante. On pourra utiliser une méthode simple telle que le tri insertion ou le tri à bulles.

La façon de distribuer les éléments dans chaque seau se fait en choisissant une fonction de hachage.

Cette méthode a surtout un intérêt lorsque l'on sait que les éléments à trier sont, par exemple, des entiers compris entre 0 et un certain entier MAX*.*

- 1^o) Définir une constante MAX (par exemple 100 000) et une classe `seau`, dont les attributs sont un tableau d'au plus 16 entiers et un entier *length* spécifiant la dimension effective utilisée et pour méthodes un constructeur par défaut (qui initialise la longueur à zéro), `insert(int)` qui place un élément dans celui-ci, `sort()`, qui trie la partie utile du tableau (grâce à un tri sélection), `len()` qui renvoie la longueur effective du tableau et `elem(int i)` qui renvoie le i-ième élément du tableau.

- 2^o) Définir une méthode :

```
void bucketsort(int *tab, int n);
```

qui trie le tableau d'entiers naturels `tab` de longueur n par seaux.

[*On définira un tableau S de $MAX/16$ pointeurs sur des seaux, que l'on initialisera à* `NULL` *pour tous les éléments. On parcourera le tableau non trié et on placera l'élément en cours dans le bon seau : si celui-ci est vide, il faut le créer ; dans tous les cas, il faut insérer l'élément dans le seau. On triera les seaux non vides. On concaténera enfin les seaux non vides pour obtenir le tableau trié.*]

- 3^o) Écrire un programme principal qui demande la longueur du tableau, qui génère un tableau d'entiers compris entre 0 et $MAX - 1$ de façon aléatoire, qui affiche ce tableau (au plus les cent premiers éléments), qui trie ce tableau par la méthode des seaux et qui affiche le tableau trié.

[*Rappelons que la constante entière :*

```
RAND_MAX
```

est la valeur maximum renvoyée par la fonction :

```
int rand(void)
```

qui renvoie un entier pseudo-aléatoire de l'intervalle [0, RAND_MAX] et que toutes les deux sont déclarées dans le fichier en-tête :

`stlib.h`]

- 4^o) Indiquer les améliorations que voyez.