Quicksort

1 Tri rapide en C

Pour passer un sous-tableau à une fonction en C, on peut passer un pointeur vers la première case de la partie qui nous intéresse et la longueur de la partie. Pour obtenir un pointeur vers la partie qui commence à l'indice `i`, il suffit de prendre `&t[i]` (ce n'est pas idiomatique du tout, mais c'est la manière de procéder en restant dans les bornes du programme).

1. Écrire une fonction `partition` ayant le prototype suivant :

```c
int partition(int *arr, int len);
```

   Cette fonction partitionnera le tableau passé en argument en utilisant le premier élément comme pivot.

2. Écrire une fonction `quicksort` ayant le prototype suivant :

```c
void quicksort(int *arr, int len);
```

   Contrairement à ce que nous avons fait en OCaml, il n'y aura pas de fonction auxiliaire ici.

1. À l'aide des fonctions présentes dans le squelette, écrire une fonction `test_quicksort` vérifiant la correction de `quicksort` sur divers tableaux aléatoires de petite taille (entre 1 et 100 éléments, disons).
2. Observer le comportement de `quicksort` sur des tableaux plus grands (dizaines ou centaines de milliers d'éléments), dans le cas d'un tableau aléatoire d'entiers entre 0 et 1 000 000 et dans le cas d'un tableau trié.

2 Quickselect

Le problème de la *sélection* est le suivant :
— on nous donne un tableau t de n éléments (d'un type totalement ordonné, des entiers par exemple), et un entier k vérifiant $0 \leqslant k < n$;
— on doit renvoyer l'élément de t qui serait à l'indice k si t était trié.

1. Que doit renvoyer `select(t, 0)` ? `select(t, n - 1)` ?
2. Comment peut-on calculer une médiane d'un tableau si l'on dispose d'une fonction *select* ? *Une médiane d'un tableau de taille n est un élément x de ce tableau tel que au moins $n/2$ éléments de t soient inférieurs ou égaux à x et au moins $n/2$ soient supérieurs ou égaux.*

1. Proposer une méthode permettant de réaliser la sélection en temps $O(n \log n)$.

Pour faire mieux, on peut adapter l'algorithme du tri rapide : c'est l'algorithme *quickselect*. L'idée est de réutiliser la fonction `partition` écrite précédemment (sans la modifier), mais de ne pas trier entièrement le tableau : un seul appel récursif est nécessaire à chaque étape.

1. Écrire une fonction `quickselect_aux` de prototype

```c
int quickselect_aux(int *arr, int k, int len);
```

   Cette fonction renverra le $(k+1)$-ème plus petit élément du tableau (c'est-à-dire `select(arr, k)`) et pourra avoir un effet secondaire quelconque sur le contenu de ce tableau.

2. Écrire une fonction `quickselect` ayant le même prototype que `quickselect_aux` et la même valeur de retour, mais ne modifiant pas le tableau qu'on lui passe en paramètre.
3. Analyser la complexité temporelle de `quickselect` :
   — dans le pire cas ;
   — dans le cas où le pivot choisi partage équitablement le tableau à chaque étape.

3 Introsort

Écrire une fonction `heapsort` ayant le prototype suivant :

```
void heapsort(int *arr, int len);
```

Cette fonction triera le tableau `arr` par ordre croissant en utilisant l'algorithme du tri par tas. On réfléchira aux fonctions auxiliaires utiles (on n'a pas besoin de toutes les fonctions sur les tas).

1. Écrire une fonction `ilog` calculant la partie entière du logarithme en base 2 de son argument (que l'on pourra supposer strictement positif).

   ```
   int ilog(int n);
   ```

2. Dans le cas où le pivot partage équitablement le tableau à chaque étape, quelle est la profondeur de récursion maximale de `quicksort` ?
3. Écrire une fonction `introsort` de prototype

   ```
   void introsort(int *arr, int len);
   ```

   Cette fonction triera le tableau passé en argument, en place, en commençant par un tri rapide mais en basculant vers un tri fusion dès que la profondeur de récursion est deux fois supérieure à ce que l'on pourrait attendre dans un « bon » cas. *Une fonction auxiliaire sera nécessaire.*