

Année 2024-2025 – Examen session 1

9 décembre 2024 – Durée 2 h 30

Une feuille A4 manuscrite recto-verso autorisée. Les calculatrices et les téléphones (incluant smartphone, tablettes,... tout ce qui contient une interface réseau) sont interdits.

Les dictionnaires pour les personnes de langue étrangère sont autorisés.

Le barème est indicatif.

Comme d'habitude, vous êtes vivement encouragés à illustrer vos recherches et vos réponses par des schémas.

1 Le problème du k -ième élément

On dispose d'un tableau T contenant n valeurs, et on cherche à déterminer la valeur de **rang** k dans T , autrement dit la k -ième plus petite de ces valeurs.

Dans tout cet examen, nous allons mettre au point et comparer plusieurs méthodes pour résoudre ce problème.

Par exemple, si $T = [E, B, R, E, I, R, B, C]$ et $k = 5$ (on considère ici l'ordre alphabétique), alors la réponse attendue est E puisque les premières valeurs du tableau (en ordre croissant) sont B, B, C, E, E, I, \dots

Pour $k = 3$ avec le même tableau il faudrait renvoyer C .

Quelques précisions :

- On suppose toujours que $1 \leq k \leq n$.
- Le type des valeurs contenues dans le tableau n'est pas connu à l'avance, on sait juste qu'on a une opération $<$ pour comparer deux valeurs.
- On s'autorise à modifier le contenu du tableau T pendant l'exécution de l'algorithme.
- **Toutes les complexités demandées dans ce problème devront être exprimées en fonction de n et/ou de k .**

2 Comparaison de quatre méthodes (9 points)

1. Méthode naïve

- (a) (2 points) Écrivez un algorithme qui résout le problème naïvement, en déterminant successivement la première, deuxième, \dots , k -ième valeur de T .
- (b) (1 point) Calculez la complexité de cet algorithme.

2. Avec un tas min

On ne demande pas d'implémenter les fonctions du tas, vous pouvez le manipuler comme une structure abstraite (file à priorités).

- (a) (1 point) Écrivez un algorithme qui résout le problème en utilisant un tas *min* (dans lequel les valeurs sont ordonnées de façon croissante de la racine vers les feuilles), **vide initialement**.

Prenez le temps de chercher comment effectuer le moins possible d'opérations sur le tas !

- (b) (1 point) Calculez la complexité de cet algorithme.

3. Avec un tas max

- (a) (1 point) Écrivez un algorithme qui résout le problème en utilisant un tas *max* (dans lequel les valeurs sont ordonnées de façon décroissante de la racine vers les feuilles), **vide initialement**.

À nouveau, prenez le temps de chercher comment effectuer le moins possible d'opérations sur le tas !

- (b) (1 point) Calculez la complexité de cet algorithme.

4. (1 point) Méthode par tri

On propose la méthode suivante :

```
Trier( $T$ )
renvoyer  $T[k - 1]$ 
```

Quelle est sa complexité ? (Vous avez le choix de l'algorithme de tri à utiliser.)

- 5. (1 point) Concluez : laquelle des 4 méthodes précédentes vous semble la plus efficace et pourquoi ?

3 Un algorithme inspiré du tri rapide (13 points)

On propose l'algorithme ci-dessous pour résoudre le **même** problème qu'en première partie, le but de cette partie est de démontrer (en partie) qu'il est correct :

```

1   $g := 0$ 
2   $d := n - 1$ 
3  tant que  $g < d$  faire
4  |    $i := g + 1$ 
5  |    $j := d$ 
6  |   tant que  $i \leq j$  faire
7  |   |   si  $T[i] < T[g]$  alors
8  |   |   |    $i = i + 1$ 
9  |   |   sinon
10 |   |   |   Échanger  $T[j]$  et  $T[i]$ 
11 |   |   |    $j = j - 1$ 
12 |   Échanger  $T[g]$  et  $T[j]$ 
13 |   si  $j < k - 1$  alors
14 |   |    $g := j + 1$ 
15 |   sinon
16 |   |    $d := j$ 
17 renvoyer  $T[g]$ 

```

On remarque ici que $T[g]$ joue à chaque fois le rôle d'un *pivot* autour duquel on réordonne partiellement la zone de tableau $T[g..d]$.

1. On s'intéresse dans un premier temps à la boucle interne, lignes 6 à 11.

On donne un invariant pour cette boucle (juste avant la ligne 7) :

Les éléments d'indices entre $g + 1$ et $i - 1$ (inclus) sont $< T[g]$, et les éléments d'indices entre $j + 1$ et d (inclus) sont $\geq T[g]$.

- (1 point) Représentez cet invariant sous forme de schéma.
- (1 point) Justifiez que cet invariant est vérifié après la ligne 5 de l'algorithme.
- (2 points) Démontrez que la boucle interne maintient effectivement cet invariant.
- (1 point) Quelle postcondition peut-on démontrer à l'aide de l'invariant en sortie de boucle **et après** l'exécution de la ligne 12 ?
- (1 point) Justifiez que cette boucle interne se termine.

2. On s'intéresse maintenant à la boucle externe, lignes 3 à 16.

On **admet** que cette boucle se termine.

La postcondition qu'on cherche à démontrer est que la valeur renvoyée par l'algorithme ($T[g]$) est bien la k -ième valeur du tableau, autrement dit qu'il y a exactement $k - 1$ valeurs dans T inférieures à $T[g]$.

(a) (2 points) Exprimez **deux** invariants vérifiés par la boucle externe de cet algorithme (juste avant la ligne 4) et permettant de démontrer la postcondition donnée ci-dessus :

- le premier invariant exprime une relation entre les valeurs de g , k et d ;
- le second invariant exprime des inégalités entre les valeurs contenues dans certaines zones de T .

(b) (1 point) Présentez également le second invariant sous forme de schéma.

Il n'est **pas demandé** de démontrer ces invariants.

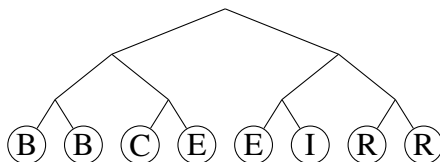
(c) (2 points) Démontrez la postcondition de l'algorithme complet en utilisant vos deux invariants en sortie de boucle externe.

3. (2 points) (*difficile*) Si le pivot choisi est à chaque fois proche de la valeur médiane des valeurs restant à étudier, quelle sera la complexité de cet algorithme ?

4 Avec un arbre (3 points)

Dans cette dernière partie, on suppose que le nombre de valeurs n est une puissance de 2, et qu'on a stocké les n valeurs dans les feuilles d'un arbre binaire complet (dont tous les nœuds ont 2 fils non vides, sauf au dernier niveau), en ordre croissant de gauche à droite.

Par exemple, avec les valeurs données en début d'énoncé, l'arbre serait :



Écrivez un algorithme qui prend en argument :

- l'arbre A décrit ci-dessus
- le nombre n de feuilles dans A
- et le rang k

renvoie le k -ième élément avec une complexité $\mathcal{O}(\log_2(n))$.

Il n'est pas demandé de démontrer cette complexité.