

## Entraînement : conception d'algorithme (partie 1)

Les solutions peuvent être rédigées en **pseudo-code, python, C, C++ ou Java**. La syntaxe du langage n'a pas d'importance tant que celle-ci reste **cohérente** et **compréhensible**. (Dans les exemples, les solutions sont données en pseudo-code).

### Grille d'évaluation

|        |   |
|--------|---|
| A (20) | Les deux algorithmes répondent correctement aux problèmes posés, ils sont écrits de façon claire et compacte et sont de complexité optimale.  |
| B (16) | Le principe général des deux algorithmes est le bon pour obtenir une complexité optimale cependant certains cas ont été oubliés ou des éléments non essentiels rajoutés (tests inutiles, écriture compliquée) ou bien on trouve des "petites" erreurs type indice, signes, etc. |
| C (11) | Le principe général du premier algorithme est le bon pour obtenir une complexité optimale mais pas le second.   |
| D (8)  | Il y a une tentative de réponse sur au moins le premier algorithme avec une tentative de complexité améliorée mais la solution est fausse ou la complexité non optimale.  |
| E (1)  | Soit les deux algorithmes sont faux ou manquant soit ils sont justes mais avec une complexité non optimale sans aucune tentative d'amélioration.  |

### Exercice 1.

Soit  $T$  un tableau trié d'entiers, par exemple

|   |    |     |     |     |     |     |     |     |      |
|---|----|-----|-----|-----|-----|-----|-----|-----|------|
| 2 | 45 | 120 | 150 | 220 | 250 | 368 | 410 | 582 | 1000 |
|---|----|-----|-----|-----|-----|-----|-----|-----|------|

Donner les deux algorithmes suivant avec complexité optimale. (Le second ne compte que si vous avez écrit le premier)

- (1) Un algorithme qui prend en entrée le tableau trié  $T$  de taille  $n$  et un entier  $v$  et qui renvoie le plus petit indice  $i$  tel que  $T[i] > v$ . Par exemple, pour le tableau ci-dessus et  $v = 160$ , l'algorithme répond 4. Si toutes les valeurs sont plus inférieures ou égales à  $v$ , l'algorithme renvoie  $n$ .
- (2) Un algorithme qui prend en entrée le tableau trié  $T$  et deux entiers  $min$  et  $max$  et qui renvoie le nombre de valeurs  $v$  du tableau telle que  $min < v \leq max$ . Par exemple, pour  $min = 200$  et  $max = 300$ , l'algorithme doit répondre 2. Pour  $min = 0$  et  $max = 1000$ , l'algorithme doit répondre 10.

### Solution

(Remarque : je donne la solution itérative mais ça marche aussi en récursif)

```

Algo1
Input : Un tableau d'entier trié T de taille n, un entier v
Procédé :
    i <- 0
    j <- n
    Tant que i < j :
        m <- (i+j)/2
        Si v < T[m] :
            Si m = 0 ou T[m-1] <= v :
                Retourner m
            j <- m

```

```

    Sinon :
        i ← m+1
    Retourner n

```

Algo2

Input : Un tableau d'entier trié T de taille n, deux entiers min et max

Procédé :

```

    a ← Algo1(T, min)
    b ← Algo1(T, max)
    Retourner b - a

```

La complexité optimale est donc  $O(\log(n))$ , on résout le problème par dichotomie.

Exemple de "petites" erreurs type  $B$  : vous avez oublié de retourner  $n$  à la fin, vous avez oublié de tester  $m = 0$ , vous avez inversé les comparaisons.

### Exercice 2.

Le problème est le suivant : on a accès en lecture à un certain tableau de données qui contient une suite de 0 suivi d'une suite de 1. Exemple : un tableau de taille 10 qui contiendrait trois 0 suivi de sept 1. Une fois qu'on a lu un 1, il n'y aura plus que des 1. **On cherche à connaître la position du premier 1.**

Pour les deux cas suivant, donnez un algorithme optimal qui donne la position du premier 1.

- (1) **Premier cas** : on prend entier le tableau  $Tab$  ainsi qu'un entier  $n$  qu'on suppose être la taille du tableau. Les positions sont indexées à partir de 0.

Cas particuliers : si le tableau ne contient que des 0, on retourne sa taille, s'il ne contient que des 1, on retourne 0.

- (2) **Deuxième cas** : on considère que le tableau est de taille infinie, c'est-à-dire que  $Tab[i]$  donne toujours une valeur, même si  $i$  est très grand.

Cas particulier : si le tableau ne contient que des 1, on retourne 0. On a la certitude que le tableau ne contient pas que des 0.

### Solution

Algo1

Input : Tab, n

Procédé :

```

    i ← 0
    j ← n
    Tant que i < j :
        m ← (i+j)/2
        Si T[m] = 1 :
            Si m = 0 ou T[m-1] = 0
                Retourner m
            j ← m
        Sinon :
            i ← m+1
    Retourner n

```

Algo2

Input : Tab

Procédé :

```

    i ← 1
    Tant que Tab[i] = 0 :
        i ← i*2
    Retourner Algo1(Tab, i)

```

Remarque : dans les deux cas la complexité est  $O(\log(n))$ .