# UE33 - Option 1
# Partiel d'Algorithmique

Arnaud Legrand

5 Novembre 2003

**Résumé**

La durée de l'examen est de 1 heure et 30 minutes, le polycopié, les notes de cours et de TD sont autorisés. *La notation prendra en compte la présentation et la clarté des explications (algorithmes commentés, dessins explicatifs, ...).* Les questions et sous-questions sont toutes *largement indépendantes*. L'examen sera noté sur 20 points et le barème est donné uniquement à titre indicatif. Il pourra être modifié lors de la notation finale.

▷ **Question 1. Procédures mystères (1 + 1 + 2 = 4 pts)** On s'intéresse à la fonction suivante :

```
Entier BLURB(Entier A[1..n], Entier n))
 1: Entier i, j
 2: Booléen B[1..n]
 3: Début
 4:   Pour i=1 à n :
 5:     B[i]← VRAI
 6:   Pour i=1 à n :
 7:     Pour j=i à n :
 8:       Si A[i] > A[j] Alors
 9:         B[j]← FAUX
10:       Si A[j] > A[i] Alors
11:         B[i]← FAUX
12:   i ← 1
13:   Tant que B[i] ≠ VRAI :
14:     i ← i + 1
15:   Renvoyer i
16: Fin
```

1. Détaillez l'exécution de cet algorithme sur l'entrée suivante.

   BLURB([1,-4,4,2,-4,4,1,3], 8)

2. Que fait cet algorithme et comment fonctionne-t-il ?
3. Calculez sa complexité.

*Réponse.* Cette fonction trouve le premier indice du plus grand élément du tableau $A$ en organisant un *tournoi* et en éliminant les éléments qui échouent. À la fin, il existe au moins un $i$ tel que $A[i]$ n'a jamais été battu (le maximum) et la boucle Tant que termine donc bien. Sa complexité est $O(n^2)$. □

▷ **Question 2. Plus longue coupe consécutive croissante (1 + 2 + 2 + 3 = 8 pts)**

1. Écrivez une fonction qui étant donné un tableau $A$ et deux indices $i$ et $j$, indique si $[A[i], A[i+1], \ldots, A[j]]$ est trié par ordre croissant.

*Réponse.* Pas de difficulté particulière.

```
Entier CROISSANTE(Entier A[1..n], Entier i,Entier j)
1: Entier k
2: Début
3:   Pour k = i à j − 1 :
4:     Si A[k + 1] > A[k] Alors
5:       Renvoyer FAUX
6:   Renvoyer VRAI
7: Fin
```

Son temps d'exécution est $O(j - i)$. □

2. Déduisez-en un algorithme qui, étant donné un tableau $A$, calcule la longueur du plus long sous-tableau de $A$ croissant. Par exemple le plus long sous-tableau de $[2, 1, 3, 4, 6, 2, 3]$ est $[1, 3, 4, 6]$ et est de longueur 4. L'algorithme devrait donc répondre 4 dans cette situation.

*Réponse.* Pas de difficulté particulière non plus.

```
Entier PLSTC(Entier A[1..n], Entier n)
1: Entier i, l
2: Booléen found
3: Début
4:   Pour l = 2 à n :
5:     found← FAUX
6:     Pour i = 1 à n + 1 − l :
7:       Si CROISSANTE(A, i, i + l − 1) Alors
8:         found← VRAI
9:     Si found = FAUX Alors
10:      Renvoyer (l − 1)
11:  Renvoyer n
12: Fin
```

□

3. Calculez la complexité de votre algorithme.

*Réponse.* Son temps d'exécution est $O(n^3)$ (il ne faut pas oublier de compter le coût de l'appel à CROISSANTE et cela correspond au pire cas où le tableau est déja trié. □

4. Proposez un algorithme dont la complexité est linéaire.

*Réponse.* Cette question est un peu plus délicate mais tout compte fait assez naturelle :

```
Entier PLSTC2(Entier A[1..n], Entier n)
 1: Entier i, l, max
 2: Booléen found
 3: Début
 4:    l ← 1
 5:    max ← 1
 6:    Pour i = 1 à n − 1 :
 7:       Si A[i] < A[i + 1] Alors
 8:          l ← l + 1
 9:       Sinon
10:          Si l > max Alors
11:             max ← l
12:          l ← 1
13:    Si l > max Alors
14:       max ← l
15:    Renvoyer max
16: Fin
```

La complexité de cet algorithme est clairement un $\Theta(n)$. □

▷ **Question 3. Drapeau hollandais (2 + 2 + 2 + 2 = 8 pts)** On se donne un tableau de $n$ éléments et une fonction COULEUR qui à chaque élément $x$ du tableau associe une couleur COULEUR($x$) codée par un nombre compris entre 1 et $k$ (par exemple bleu=1, blanc=2, rouge=3, noir=4, ...). On cherche à réarranger le tableau de manière à ce que les éléments de même couleur soient regroupés ensembles. Les seules opérations autorisées sont les échanges de deux éléments du tableau (grâce à l'appel ÉCHANGE(TAB,I,J) qui échange $tab[i]$ et $tab[j]$), et le test de la couleur d'un élément (grâce à l'appel de COULEUR(X)).

1. On suppose $k = 2$. Écrivez un algorithme qui sépare les éléments bleus des éléments blancs. Quelle est la complexité (en nombre de comparaisons) de votre solution ? Est-elle améliorable ?

   *Réponse.* L'idée est d'avoir un indice qui pointe vers le début du tableau et un autre qui pointe vers la fin, d'échanger les valeurs correspondantes si elles ne sont pas dans le bon ordre et de rapprocher les deux indices.

```
BLEU-BLANC(Entier A[1..n], Entier n)
 1: Entier debut, fin
 2: Début
 3:    debut ← 1
 4:    fin ← n
 5:    Tant que debut ≠ fin :
 6:       Tant que (COULEUR(A[debut]) =bleu) Et (debut < fin) :
 7:          debut ← debut + 1
 8:       Tant que (COULEUR(A[fin]) =blanc) Et (debut < fin) :
 9:          fin ← fin − 1
10:       ECHANGE(A, debut, fin)
11: Fin
```

   On regarde la couleur de chaque élément du tableau exactement une fois. Le nombre de comparaisons est donc égal à $n$. □

2. On suppose $k = 3$. En utilisant la solution précédente, proposez un algorithme qui sépare les différentes couleurs (*Indication : séparer d'abord les bleus et les blancs des rouges, puis ensuite séparer les bleus des blancs*) et calculez sa complexité (en nombre de comparaisons).

*Réponse.* En reprenant le même algorithme, on sépare donc les bleus et les blancs des rouges pour un coût égal à $n$. Il reste ensuite à séparer les bleus des blancs pour un coût égal au nombre de blancs et de bleus, ce qui peut être au pire égal à $n$. On peut donc séparer toutes les couleurs en au pire $2n$ comparaisons. □

3. En vous inspirant de la méthode pour $k = 3$, proposez un algorithme pour $k = 4$ et donnez sa complexité (en nombre de comparaisons).

   *Réponse.* En reprenant le même algorithme, on sépare donc les bleus, les blancs et les rouges des noirs pour un coût égal à $n$. En réutilisant l'algorithme précédant pour séparer les bleus, les blancs et les rouges, il nous faut au pire $2n$ comparaisons supplémentaires. On peut donc séparer toutes les couleurs en au pire $3n$ comparaisons. □

4. Proposez un algorithme pour le cas général de $k$ couleurs et donnez sa complexité.

   *Réponse.* En utilisant la même idée, on sépare $k$ couleurs en au pire $(k-1)n$ comparaisons. Je ne l'écris pas car le suivant marche sur le même principe et est plus instructif. □

5. (Bonus) Proposez une autre solution plus efficace $k = 4$ (en regroupant les couleurs différemment). Déduisez-en un algorithme plus efficace pour le cas général.

   *Réponse.* Pour séparer 4 couleurs, on peut en mettre deux d'un coté et deux de l'autre en $n$ comparaisons. Supposons qu'il y ait $n_1$ valeurs des deux premières couleurs et $n_2$ valeurs des deux dernières couleurs. On a donc $n = n_1 + n_2$. On peut spérarer les deux premières couleurs en $n_1$ comparaisons puis séparer les deux dernières coulers avec $n_2$ comparaisons supplémentaires. On peut donc séparer 4 couleurs avec $n + n_1 + n_2 = 2n$ opérations.
   En itérant l'idée, on sépare $8couleurs$ d'abord en deux groupes de 4 couleurs pour un coût égal à $n$ puis en réutilisant la même méthode, on trie l'ensemble des couleurs avec $2n$ comparaisons supplémentaires, soit avec $3n$ comparaisons au total. En fait, l'algorithme suivant sépare $k$ couleurs en temps $\lceil \log_2 k \rceil\, n$.

```
SÉPARATION(Entier A[1..n], Entier d, Entier f, Entier col_min, Entier col_max)
 1: Entier debut, fin, col_moy
 2: Début
 3:    col_moy ← ⌊(col_min + col_max)/2⌋
 4:    debut ← 1
 5:    fin ← n
 6:    Tant que debut ≠ fin :
 7:       Tant que (COULEUR(A[debut]) ∈ [col_min, col_moy]) Et (debut < fin) :
 8:          debut ← debut + 1
 9:       Tant que (COULEUR(A[fin]) ∈ [col_moy + 1, col_max]) Et (debut < fin) :
10:          fin ← fin − 1
11:       ECHANGE(A, debut, fin)
12:    Si col_moy > col_min Alors
13:       SÉPARATION(A, d, debut, col_min, col_moy)
14:    Si col_moy + 1 < col_max Alors
15:       SÉPARATION(A, fin, f, col_min + 1, col_max)
16: Fin
```

   Pour être tout à fait honnête, l'algo précédent, tel qu'il est écrit ne marche pas tout à fait... En fait, les bornes $[d, debut]$ et $[fin, f]$ ne sont pas bonnes (la preuve, $debut = fin$ à la sortie de la boucle principale). Il faut donc maintenir la séparation entre les deux zones plus précisément, mais ça ne change rien au principe de l'algorithme. □