Algorithmique et Programmation – CPI1

TD6 : Tableaux et Tris - CORRIGE

1. Appliquer un tri lent pour trier le tableau d'entiers suivant : [4, 8, 2, 10, 1, 9, 7, 6, 3, 5].
2. Écrire une fonction qui vérifie si les éléments d'un tableau d'entiers donné sont triés ou pas.

```
Fonction TableauTrie(A[n]: Tableau de Entier) : Booléen
Variables i : Entier
Début
Si n > 2 Alors
        Si (A[1] <= A[n]) Alors
                Pour i ← 1 à n-1
                        Si (A[i]>A[i+1]) Alors
                                retourner faux
                        FinSi
                FinPour
        Sinon
                Pour i ← 1 à n-1
                        Si (A[i] < A[i+1]) Alors
                                retourner faux
                        FinSi
                FinPour
        FinSi
FinSi
retourner vrai
Fin
```

3. Écrire une fonction qui insère une nouvelle valeur dans un tableau trié d'entiers. Évaluer le nombre d'affectations nécessaires à votre fonction dans le pire des cas, et en déduire sa complexité.

```
Fonction Insérer(A[n] : Tableau de T,  val : T) : Tableau de T
Variables i, pos, B[n+1] : Entier
Début
        pos ← 1
        Tantque A[pos] < val
                pos ← pos+1
        FinTantque
        Pour i ← 1 à (pos – 1)
                B[i] ← A[i]
        FinPour
        B[pos] ← val
        Pour i ← pos+1 à n+1
                B[i] ← A[i-1]
        FinPour
        retourner B
Fin
```

4. Réécrire le tri par insertion vu en cours pour trier dans l'ordre décroissant. Est-il possible d'améliorer le coût de cet algorithme en appliquant une recherche dichotomique pour l'insertion d'une valeur dans le sous tableau déjà trié ?

```
Procédure TriInsertion(ES A[n] : Tableau de T)
Variables i, j : Entier, temp : T
Début
       Pour i ← 2 à n
              j ← i
              Tantque j > 1 et (A[j] > A[j-1])
                     temp ← A[j]
                     A[j] ← A[j-1]
                     A[j-1] ← temp
                     j ← j-1
              FinTantque
      FinPour
Fin
```

Complexité :

Nombre de comparaisons :
- meilleur cas (tableau trié dans bon ordre) : n-1
- pire cas (tableau trié en ordre décroissant) : 1 + 2 + ... + n-1 = n(n-1)/2

Nombre d'échanges :
- meilleur cas (tableau trié dans bon ordre) : 0
- pire cas (tableau trié en ordre décroissant) : 1 + 2 + ... + n-1 = n(n-1)/2

Est-il possible d'améliorer le coût de cet algorithme en appliquant une recherche dichotomique pour l'insertion d'une valeur dans le sous tableau déjà trié ?

Dans le pire des cas cela ne change rien pour le coût car, même si la recherche de l'emplacement où il faut insérer est plus rapide (log(n) au lieu de n), l'insertion dans un tableau trié nécessite un décalage 1 qui lui est forcément linéaire. Avec une structure permettant une insertion en temps constant (exemple listes chaînées), on obtiendrait un gain de performances.

5. Écrire un algorithme qui trie un tableau de booléens de sorte que tous les FAUX se trouvent à gauche du tableau, et tous les VRAI à droite du tableau. Attention, votre algorithme doit avoir la complexité O(n), et non pas O(n*log n) voire $O(n^2)$.

```
Procedure triBool (ES tab[n]: Tableau de Booléen)
// trie tous les VRAI vers la gauche, complexité: O(n)
Debut
	i_inf <- 1
	i_sup <- n;
	TantQue (i_inf < i_sup) Faire
		Si tab[i_inf] = VRAI Alors
			i_inf <- i_inf + 1
		Sinon
			Echanger(tab, i_inf, i_sup)
			i_sup <- i_sup - 1
		FinSi
	FinTantQue
Fin
```

6. « Tri drapeau » (difficile) : un tableau contient des éléments rouges, blancs et bleus. Triez ce tableau pour que les éléments rouges soient au début, les blancs au milieu et les bleus à la fin. De nouveau, l'algorithme doit être O(n), c'est-à-dire qu'il faut ne tester qu'une fois la couleur d'un élément !

```
Procedure triDrapeau (ES tab[n]: Tableau de Couleur)
// trie un tableau de trois couleurs (Noir -> Gris -> Blanc), complexité: O(n)
Debut
	i_Noir <- 0
	i_inf <- 1
	i_sup <- N
	TantQue (i_inf <= i_sup) Faire
		Selon tab[i_inf]
			Cas Gris:
				i_inf <- i_inf + 1
			Cas Blanc:
				Echanger(tab, i_inf, i_sup)
				i_sup <- i_sup - 1
			Cas Noir:
				i_Noir <- i_Noir + 1
				Echanger(tab, i_Noir, i_inf);
				i_inf <- i_inf +1;
		FinSelon
	FinTantque
Fin
```