

Examen : Algorithmique et Complexité (Durée : 2h00)

Questions : (6 pts)

Choisir **la** bonne réponse (une seule) :

1. Une complexité temporelle de $O(1)$ signifie que :
 - A. L'algorithme effectue une seule opération indépendamment de la taille de l'entrée.
 - B. L'implémentation de l'algorithme prend exactement une unité de temps indépendamment de la taille de l'entrée.
 - C. L'algorithme effectue un nombre d'opérations constant indépendant de la taille de l'entrée.
2. Dans un arbre binaire de recherche, quel est le parcours qui permet d'obtenir les éléments dans l'ordre ?
 - A. Parcours en largeur.
 - B. Parcours infixe.
 - C. Parcours préfixe.
3. Quelle est la complexité temporelle de l'algorithme de tri rapide (quick sort) dans le pire des cas ?
 - A. $O(n)$
 - B. $O(n \log n)$
 - C. $O(n^2)$
4. L'algorithme de tri par sélection (selection sort) a une complexité temporelle de :
 - A. $O(n)$ comparaisons et $O(n^2)$ échanges.
 - B. $O(n^2)$ comparaisons et $O(n)$ échanges.
 - C. $O(n^2)$ comparaisons et $O(n^2)$ échanges.
5. Parmi les algorithmes de tri suivants, lequel a une complexité spatiale de $O(n)$?
 - A. Tri par fusion (merge sort).
 - B. Tri par insertion (insertion sort).
 - C. Tri rapide (quick sort).
6. Dans un arbre binaire, une feuille est un noeud qui :
 - A. N'a pas de fils.
 - B. N'a pas de père.
 - C. N'a pas de frère.

Exercice 1 : (4 pts)

Soit la fonction suivante :

```
int doingSomething(int T[], int n) {
    int S = 0;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (T[n-i-1] != T[n-j-1])
                S = S + T[n-i-1] * T[n-j-1];
    return S;
}
```

- Que fait cette fonction ?
- Quelle est sa complexité temporelle ?

Exercice 2 : (4 pts)

L'intersection de deux tableaux *A* et *B* est un tableau *C* qui contient tous les éléments qui sont à la fois dans *A* et dans *B*. Ecrire une fonction "*intersectionTrie*" qui prend en paramètre deux tableaux triés d'entiers et qui retourne un tableau trié contenant l'intersection des deux tableaux.

- Nous supposons que *A* et *B* sont déjà triés et ne contiennent pas de doublons.
- Nous supposons que le tableau *C* est créé à l'extérieur de la fonction avec une taille suffisante pour contenir les éléments de l'intersection.
- La fonction doit retourner la taille de *C*. L'entête suivante peut être utilisée :

```
void intersectionTrie(int A[], int na, int B[], int nb, int C[], int *nc);
```

- La fonction doit avoir une complexité temporelle linéaire.

Indication : Inspirez-vous de l'algorithme de fusion de deux tableaux triés (utilisé dans l'algorithme de tri fusion).

Exercice 3 : (6 pts)

Ecrire une fonction "*convertirAVL*" qui prend en paramètre un tableau *trié* d'entiers et qui retourne un arbre binaire de recherche équilibré contenant les éléments du tableau.

- Nous supposons que la structure de l'arbre est déjà définie tel que chaque nœud contient un entier et deux pointeurs vers le fils gauche et le fils droit.
- La fonction ne doit pas utiliser de rotation.

Indications :

- Ecrire une fonction récursive.
- Sur chaque appel récursif, pensez à équilibrer le nombre d'éléments dans les sous-arbres gauche et droit.