Département d'informatique
2ème année Ingénieur en Informatique
2023/2024
Dr. Chemseddine Chohra

Examen : Algorithmique et Complexité (Durée : 2h00)

Questions : (6 pts)

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

1. La complexité spaciale d'un algorithme est :
   A. Le nombre d'opérations élémentaires effectuées par l'algorithme en fonction de la taille de l'entrée.
   B. La quantité de mémoire utilisée par l'algorithme en fonction de la taille de l'entrée.
   C. Le nombre d'opérations élémentaires effectuées par l'algorithme en fonction de la taille de la mémoire.

2. Une complexité temporelle de ***O(n)*** signifie que :
   A. Un nombre d'opérations proportionnel à la taille de l'entrée.
   B. L'implémentation de l'algorithme prend exactement n unités de temps.
   C. L'algorithme effectue un nombre d'opérations constant indépendant de la taille de l'entrée.

3. Quelle est la complexité temporelle de l'algorithme de tri par fusion (merge sort) dans le pire des cas ?
   A. O(n)
   B. O(n log n)
   C. O(n^2)

4. L'algorithme de tri à bulles (bubble 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. Un arbre équilibré est un arbre :
   A. La différence de hauteur entre les sous-arbres gauche et droit de chaque nœud est au plus 1.
   B. La différence de hauteur entre les sous-arbres gauche et droit de chaque nœud est au moins 1.
   C. La différence de hauteur entre les sous-arbres gauche et droit de chaque nœud est exactement 1.

6. La racine d'un arbre est :
   A. Le nœud qui n'a pas de parent.
   B. Le nœud qui a le plus de fils.
   C. Le nœud qui n'a pas de fils.

Exercice 1 : (4 pts)

Soit la fonction suivante :

```
void doingSomething(int *T, int n) {
    int i = 0;
    int j = n - 1;
    while (i < j) {
        while (T[i] > 0) i++;
        while (T[j] < 0) j--;
        if (i < j) {
            int temp = T[i];
            T[i] = T[j];
            T[j] = temp;
        }
    }
}
```

- Que fait cette fonction ?
  Elle réorganise les éléments du tableau de telle sorte que tous les entiers positifs soient placés à gauche du tableau et tous les entiers négatifs à droite
- Quelle est sa complexité temporelle ?
  O(n).

Exercice 2 : (4 pts)

Écrire une fonction qui prend en paramètre un tableau d'entiers trié par ordre croissant et sa taille, et qui supprime les doublons du tableau. La fonction doit retourner la nouvelle taille du tableau après suppression des doublons.

- La fonction pourrait avoir la signature suivante

```
void supDoublonsTrie(int T[], int size, int *newSize);
```

- La fonction doit avoir une complexité temporelle linéaire.

```
void supDoublonsTrie(int T[], int size, int *newSize) {
    int index = 0;
    for (int i = 1; i < size; i++) {
        if (T[index] != T[i]) {
            T[index++] = T[i];
        }
    }
    if (size == 0) *newSize = 0;
    else *newSize = index + 1;
}
```

Exercice 3 : (6 pts)

Ecrire une fonction ***« insertAllNoRepeat »*** qui prend comme paramètre un arbre binaire de recherche ***A*** et un arbre binaire ***B.*** La fonction doit insérer tous les éléments de ***B*** dans ***A*** sans répétition puis retourner le résultat.

- Nous supposons que l'arbre ***A*** ne contient pas des doublons.
- Nous supposons que 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 pourrait avoir la signature suivante :

```
Arbre insertAllNoRepeat(Arbre A, Arbre B);
```

Indication :

- Il faut parcourir l'arbre ***B*** et insérer ses éléments dans ***A*** en utilisant une version légèrement modifiée de la fonction ***« insertBST »*** vue en cours.

```
Arbre insertBSTNoRep(Arbre root, int value) {
    if (root == NULL) Node *newNode = creerNoeud(value);
    if (value < root->data)
        root->left = insertBSTNoRep(root->left, value);
    else if (value > root->data)
        root->right = insertBSTNoRep(root->right, value);
    return root;
}

Arbre insertAllNoRepeat(Arbre A, Arbre B) {
    if (B == NULL) return A;
    A = insertBST(A, B->data);
    A = insertAllNoRepeat(A, B->left);
    A = insertAllNoRepeat(A, B->right);
    return A;
}
```