

Questionnaire examen intra

INF2010

Sigle du cours

<i>Identification de l'étudiant(e)</i>		
Nom :	Prénom :	
Signature :	Matricule :	Groupe :

<i>Sigle et titre du cours</i>		<i>Groupe</i>	<i>Trimestre</i>
INF2010 – Structures de données et algorithmes		Tous	20093
<i>Professeur</i>		<i>Local</i>	<i>Téléphone</i>
Ettore Merlo, responsable – T. Ould Bachir, T. Lavoie chargés de cours		B-415	5758
<i>Jour</i>	<i>Date</i>	<i>Durée</i>	<i>Heure</i>
Mardi	20 octobre 2009	2h00	18h30

<i>Documentation</i>	<i>Calculatrice</i>	
<input type="checkbox"/> Toute <input type="checkbox"/> Aucune <input type="checkbox"/> Voir directives particulières	<input type="checkbox"/> Aucune <input type="checkbox"/> Programmable <input type="checkbox"/> Non programmable	Les cellulaires, agendas électroniques ou téléavertisseurs sont interdits.

<i>Directives particulières</i>
<p style="text-align: right;"><i>Bonne chance à tous!</i></p>

Important	<p>Cet examen contient 6 questions sur un total de 12 pages (excluant cette page)</p> <p>La pondération de cet examen est de 30 %</p> <p>Vous devez répondre sur : <input type="checkbox"/> le questionnaire <input type="checkbox"/> le cahier <input type="checkbox"/> les deux</p> <p>Vous devez remettre le questionnaire : <input type="checkbox"/> oui <input type="checkbox"/> non</p>
------------------	--

L'étudiant doit honorer l'engagement pris lors de la signature du code de conduite.

Question 1 : Généralités**(10 points)**

Répondre par vrai ou par faux (en justifiant brièvement) aux assertions suivantes :

1.1) (1 point) La signature suivante est bonne pour implémenter un itérateur sur une liste.

```
public class MaListe<T> implements Iterable<T>
{
    private int theSize;
    private T [ ] theItems;
    ...
    public java.util.Iterator< T > iterator( )
    { return new MonIterateur< T >( this ); }

    public static class MonIterateur implements java.util.Iterator< T >
    {
        ...
    }
}
```

1.2) (1 point) La signature suivante est bonne pour implémenter un itérateur sur une liste.

```
public class MaListe<T> implements Iterable<T>
{
    private int theSize;
    private T [ ] theItems;
    ...
    public java.util.Iterator< T > iterator( )
    { return new MonIterateur< T >( this ); }

    private static class MonIterateur implements java.util.Iterator< T >
    {
        ...
    }
}
```

1.3) (1 point)) La signature suivante est bonne pour implémenter le nœud d'une liste chaînée.

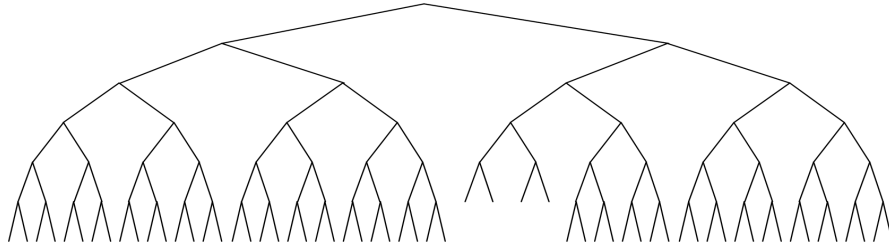
```
public class MaListeChaine< T > implements Iterable< T >
{
    private Noeud < T > queue;
    private Noeud < T > tete;
    ...
    private static class Noeud< T >
    {
        ...
    }
}
```

1.4) (1 point) L'algorithme QuickSort a une complexité $O(n \log(n))$ en tout temps.

1.5) (1 point) Un arbre AVL a un temps d'insertion $O(n)$ en tout temps.

1.6) (1 point) Un AVL ne peut pas être un arbre binaire complet :

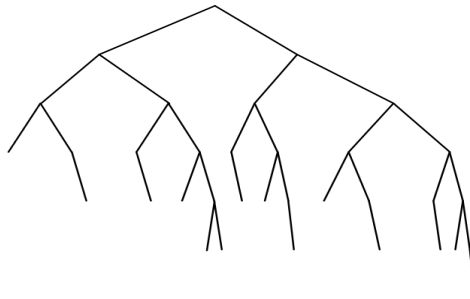
1.7) (2 point) Considérez la structure de l'arbre binaire suivant:



7.a. Cette structure d'arbre est bien celle d'un arbre binaire complet

7.b. Cette structure d'arbre est bien celle d'un arbre AVL

8) (2 point) Considérez la structure de l'arbre binaire suivant:



8.a. Cette structure d'arbre est bien celle d'un arbre AVL

8.b. Cette structure d'arbre est celle du plus petit AVL de cette hauteur (on ne peut retirer aucun élément sans réduire la hauteur de l'arbre)

Question 2: Tables de dispersion**(23 points)**

2.1) **(10 points)** Soit la table de dispersion suivante de taille $N = 13$:

26	40			17				8		62		
----	----	--	--	----	--	--	--	---	--	----	--	--

et $H(x) = x \% N + 3i$, la fonction de dispersion avec résolution linéaire des collisions que l'on vous demande d'utiliser. Insérer les éléments suivants dans la table: 4, 1, 33, 10, 19. Indiquez l'état de la table après chaque insertion.

2.2) **(5 points)** Un étudiant implémente une table de dispersion de la manière suivante:

- Si le nombre d'éléments contenus dans la table est supérieur à 75% de la taille de la table, il double la taille de la table ($N \rightarrow 2N$) et il redisperse (rehash) les éléments.
- La résolution des collisions dans la table se fait par dispersion quadratique.

Commentez cette implémentation sur les points suivants :

- Le choix du facteur de compression limite de 75%;
- La façon de redimensionner de la table;

2.3) **(8 points)** Deux étudiants doivent résoudre un problème qui nécessite de trier des éléments au fur et à mesure qu'ils sont insérés dans le programme. L'étudiant A traite un problème qui manipule des chaînes de caractères tandis que l'étudiant B traite un problème qui manipule des entiers. Les deux étudiants décident d'utiliser des tables de dispersion pour trier leurs éléments au lieu d'un arbre AVL à cause de la performance supérieure des lectures dans la table de dispersion. Commentez la stratégie de chacun des deux étudiants en vous appuyant sur les questions suivantes.

- Peut-on utiliser un AVL pour maintenir des entiers triés?
- Peut-on utiliser un AVL pour maintenir des *strings* triés?
- Peut-on utiliser une table de dispersion pour maintenir des entiers triés?
- Peut-on utiliser une table de dispersion pour maintenir des *strings* triés?

Question 3 : Utilisation de listes, piles et files**(12 points)**

Il a été vu en classe que MergeSort fonctionne très bien lorsqu'on lui passe en paramètre un tableau (vecteur) d'éléments à trier. Pour chacune des structures de données ci-dessous, indiquez si MergeSort est toujours applicable si on remplace le tableau par la structure proposée. Expliquez brièvement et indiquez si la variante de MergeSort sous cette structure de données est interne (i.e. N'utilise pas d'espace mémoire à l'extérieur de la structure de donnée qui est passée originalement en paramètre) ou externe (i.e. Utilise de la mémoire supplémentaire).

3.1) **(4 points)** Liste doublement chaînée

3.2) **(4 points)** File

3.3) **(4 points)** Pile

Question4 : k -ième élément**(20 points)**

Considérer le code source Java donné à l'annexe 1 :

4.1) **(5 points)** Donner le résultat de l'affichage quand la fonction principale (main) est appelée :

4.2) **(5 points)** Dans l'exécution de la fonction principale, combien de fois est appelée la méthode interne: `void Median(AnyType [] a, int left, int right)`. Détailler votre réponse.

4.3) **(10 points)** Donner l'état du vecteur `a[]` à la fin de l'exécution de la fonction principale:

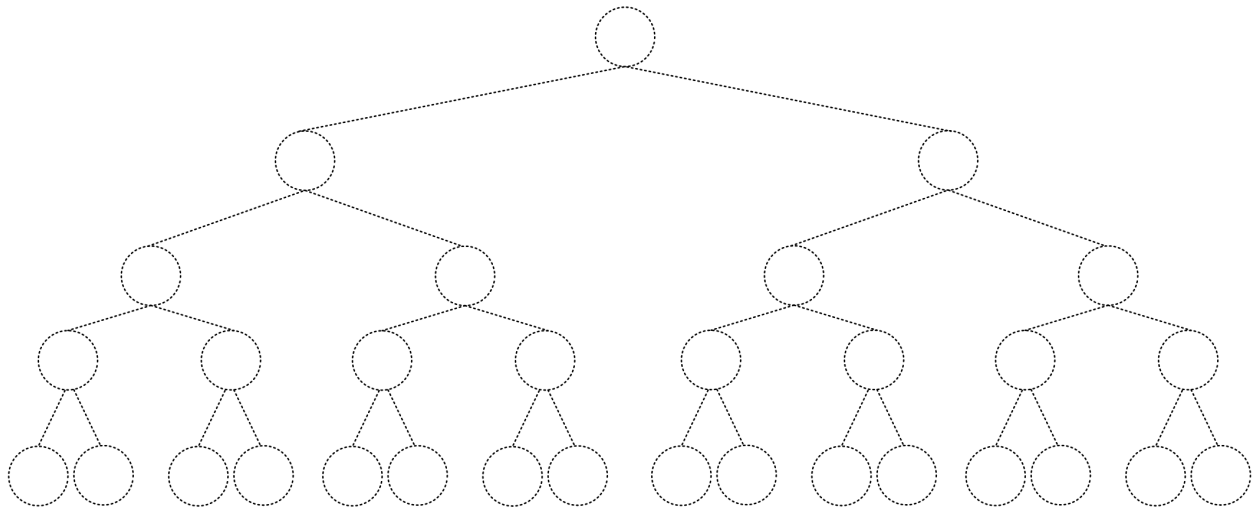
Question 5 : Arbre binaire de recherche de type AVL**(20 points)**

Vous disposez d'une structure de données implémentant un arbre binaire de recherche de type AVL contenant des entiers.

5.1) **(6 points)** Vous effectuer un affichage post-ordre sur l'arbre et vous obtenez l'affichage suivant.

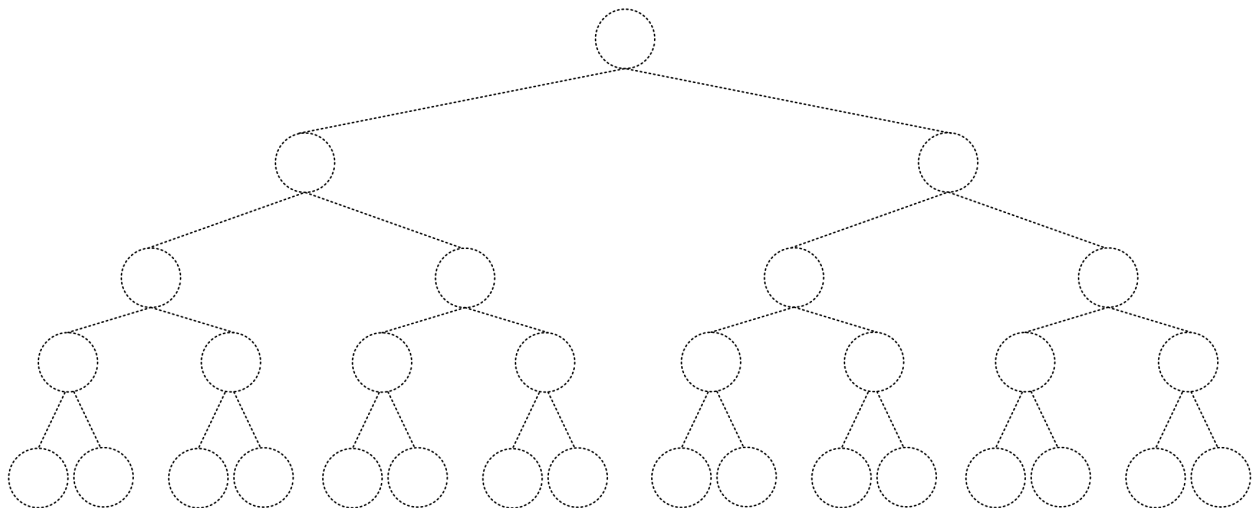
27, 36, 33, 30, 40, 45, 42, 51, 60, 57, 54, 48, 39

Dessiner l'état de l'arbre AVL en mémoire :

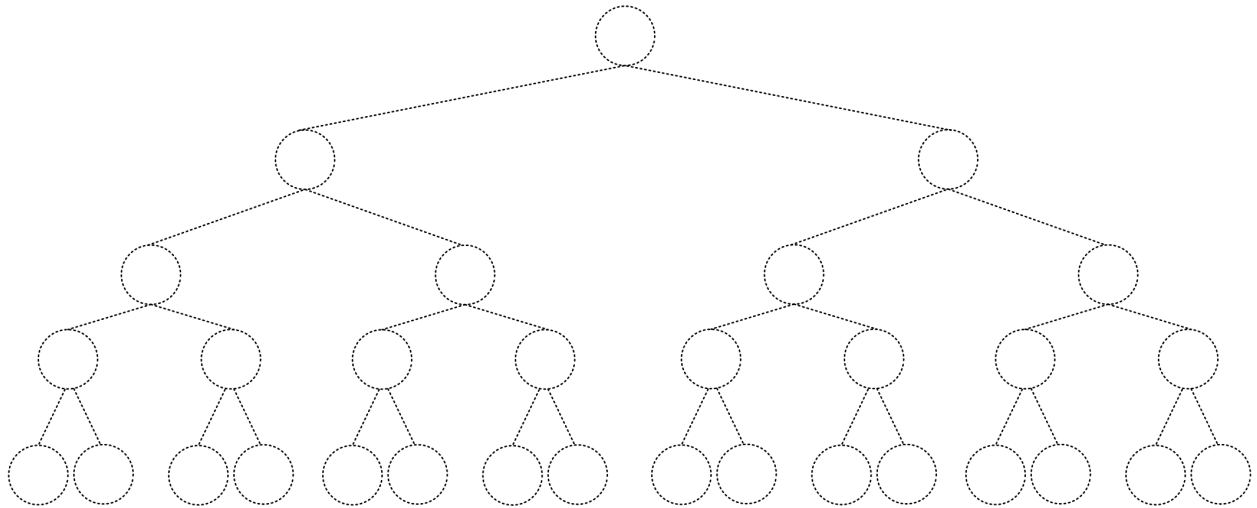


5.2) **(4 points)** On insère dans l'ordre les valeurs suivante : 43, 55, 46, 38, 41.

Dessiner l'état de l'arbre AVL à la fin des insertions :



Dessiner l'état de l'arbre AVL à la fin de l'opération :



Question 6: Analyse de complexité**(15 points)**

6.1) **(5 points)** Donnez la formule de récurrence permettant d'aboutir à la preuve de complexité de MergeSort.

6.2) **(5 points)** Soit N éléments triés suivis de M éléments placés de façon aléatoire. Si $M = \log(N)$, est-il possible de trier l'ensemble des éléments ($M+N$) en $O(N)$? Expliquez votre réponse.

6.3) **(5 points)** En considérant la liste décrite à la question précédente, donnez une borne supérieure à M pour que la liste soit triée en $O(N)$. Indice: Observez la différence de complexité pour le cas taille de $M = \sqrt{N}$ et $M = N$.

Annexe 1

```

public final class Sort
{
    /**
     * Méthode permettant d'interchanger de position deux objets
     * @param a un tableau d'objets.
     * @param index1 L'Indice du premier objet dans a.
     * @param index2 L'Indice du second objet dans a.
     */
    public static <AnyType> void swapReferences( AnyType [ ] a, int index1, int index2 )
    {
        AnyType tmp = a[ index1 ];
        a[ index1 ] = a[ index2 ];
        a[ index2 ] = tmp;
    }

    /**
     * Méthode interne effectuant le median3 pour QuickSort ;)
     */
    private static <AnyType extends Comparable<? super AnyType>>
    AnyType median3( AnyType [ ] a, int left, int right )
    {
        int center = ( left + right ) / 2;
        if( a[ center ].compareTo( a[ left ] ) < 0 )
            swapReferences( a, left, center );
        if( a[ right ].compareTo( a[ left ] ) < 0 )
            swapReferences( a, left, right );
        if( a[ right ].compareTo( a[ center ] ) < 0 )
            swapReferences( a, center, right );

        // Place pivot at position right - 1
        swapReferences( a, center, right - 1 );
        return a[ right - 1 ];
    }

    /**
     * Méthode interne effectuant un InsertionSort sur un sous-tableau
     * @param a Un tableau de Comparable items.
     * @param left La position à gauche du sous tableau à trier
     * @param right La position à droite du sous tableau à trier
     */
    private static <AnyType extends Comparable<? super AnyType>>
    void insertionSort( AnyType [ ] a, int left, int right )
    {
        for( int p = left + 1; p <= right; p++ )
        {
            AnyType tmp = a[ p ];
            int j;

            for( j = p; j > left && tmp.compareTo( a[ j - 1 ] ) < 0; j-- )
                a[ j ] = a[ j - 1 ];
            a[ j ] = tmp;
        }
    }

    /**
     * Median
     * @param a un tableau de Comparable.
     */
    public static <AnyType extends Comparable<? super AnyType>>
    void Median( AnyType [ ] a )
    {
        Median( a, 0, a.length - 1 );
    }
}

```

```

/**
 * Méthode interne effectuant des appels récurifs.
 * Appelle median3 et un cutoff de valeur CUTOFF (voir constante final).
 * Places la valeur médiane à sa position sans trier complètement le reste du vecteur.
 * @param a un tableau d'objets Comparable.
 * @param left la position à gauche du sous tableau.
 * @param right la position à droite du sous-tableau.
 */
private static <AnyType extends Comparable<? super AnyType>>
void Median( AnyType [ ] a, int left, int right )
{
    if( left + CUTOFF <= right )
    {
        AnyType pivot = median3( a, left, right );

        // Begin partitioning
        int i = left, j = right - 1;
        for( ; ; )
        {
            while( a[ ++i ].compareTo( pivot ) < 0 ) { }
            while( a[ --j ].compareTo( pivot ) > 0 ) { }
            if( i < j )
                swapReferences( a, i, j );
            else
                break;
        }

        swapReferences( a, i, right - 1 ); // Restore pivot

        if( a.length/2 <= i )
            Median( a, left, i - 1 );
        else if( a.length/2 > i + 1 )
            Median( a, i + 1, right );
    }
    else // Do an insertion sort on the subarray
        insertionSort( a, left, right );
}

private static final int CUTOFF = 5;
private static final int NUM_ITEMS = 16;

public static void main( String [ ] args )
{
    Integer [ ] a = {12, 59, 94, 3, 35, 26, 77, 45, 84, 25, 67, 5, 91, 36, 64, 74};

    Sort.Median( a );
    System.out.println( "La valeur médiane est " + a[ NUM_ITEMS/2-1 ] );
}

```