ÉCOLE POLYTECHNIQUE MONTRÉAL

Le génie sans frontières

Questionnaire examen intra

INF2010

Sigle du cours

Identification de l'étudiant(e)		
Nom :	**Prénom :**	
Signature :	**Matricule :**	**Groupe :**

Sigle et titre du cours		*Groupe*	*Trimestre*
INF2010 – Structures de données et algorithmes		**Tous**	**20083**
Professeur		***Local***	***Téléphone***
Ettore Merlo, responsable – Tarek Ould Bachi, chargé de cours		**M-4105**	**5758**
Jour	***Date***	***Durée***	***Heures***
Jeudi	**23 octobre 2008**	**2h00**	**18h00**

Documentation	*Calculatrice*	
☐ Toute	☐ Aucune	**Les cellulaires, agendas électroniques ou téléavertisseurs sont interdits.**
☒ Aucune	☐ Programmable	
☐ Voir directives particulières	☒ Non programmable	

Directives particulières

Bonne chance à tous!

Important

Cet examen contient **5** questions sur un total de **16** pages **(excluant cette page)**

La pondération de cet examen est de **30** **%**

Vous devez répondre sur : ☐ le questionnaire ☐ le cahier ☒ les deux

Vous devez remettre le questionnaire : ☒ oui ☐ non

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

Question 1 (10 points)

En vous inspirant de l'implantation de la fonction « hash » du livre pour les chaînes de caractères, écrivez le code Java pour une fonction hash des vecteurs de Integer dont la signature est la suivante :

public static int hash_2010 (Integer [] v, int tableSize)

Définissez vous-mêmes les constantes, les variables, les méthodes et les classes nécessaires. Justifiez brièvement vos définitions.

Question 2 (15 points)

Considérez :

- la fonction de dispersement h(x) = x % 11
- différentes tables de dispersement de dimension 11
- les clefs :
  i) 47 57 35 71
  ii) 34 56 12 54
  iii) 25 24 73 66

2.1 Considérez une table de dispersement par débordement progressif avec "*sondage*" linéaire (f(i) = i) **(7 pts)**

2.1.1 Quelles sont les valeurs i et $h_i(x)$ utilisées pour l'insertion de chaque clef indiquée ? Détaillez les calculs sur le cahier.

2.1.2 Dessinez l'état de la table à la fin de l'insertion de toutes les clefs indiquées.

2.2 Considérez une table de dispersement par débordement progressif avec "*sondage*" à double dispersement (f(i) = i * $h_2(x)$), avec fonction de dispersement primaire $h_1(x) = h(x)$ et avec fonctions de dispersement secondaire ($h_2(x) = 7 - (x \% 7)$). **(8 pts)**

2.2.1 Quelles sont les valeurs i et $h_i(x)$ utilisées pour l'insertion de chaque clef indiquée ? Détaillez les calculs sur le cahier.

2.2.2 Dessinez l'état de la table à la fin de l'insertion de toutes les clefs indiquées.

Question 3 (15 points)

Considérez l'algorithme « Quicksort » pour le tri des vecteurs.

Considérez les vecteurs suivants avec un index de position qui commence à zéro :

a) 47 94 6 99 2 23 19 59 83 31
b) 73 85 51 71
c) 79 3 65 81 4 41 30 64 69 18 2 6 90 74

Remplissez l'annexe 1 aux pages 6, 7 et 8 avec le calcul, pour chaque vecteur indiqué, des partitions gauche et droite pour le pivot calculé comme « élément central » et pour le pivot calculé comme « mediane3 ».

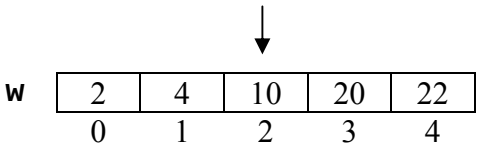
Question 4 (30 points)

L'annexe 2 aux pages 9 à 13 vous donne le listing des fonctions utiles pour ce programme. La fonction `mediane( w )`retourne la médiane d'un vecteur **`w.`** La valeur médiane est définie comme la valeur qui permet de partager une série numérique ordonnée (triée) en deux parties de même nombre d'éléments. Considérant une liste de valeurs réelles **`w`** de taille k, la fonction `mediane( w )` retourne la valeur de rang moyen dans la liste. La valeur médiane est obtenue en triant le vecteur puis en donnant la valeur à la position $(k-1)/2$.

Par exemple, sur la liste suivante de taille impaire k=5:

w	22	4	20	10	2
	0	1	2	3	4

la valeur médiane est 10 :



4.1 En justifiant clairement vos réponses et en vous référant à l'implémentation donnée à l'annexe 2, donnez la complexité algorithmique d'une implémentation possible de la fonction `mediane( w )` (fonction de k). **(15 pts)**

4.1.1 Pire cas,

4.1.2 Moyen cas,

4.1.3 Meilleur cas.

Justifiez vos réponses.

4.2 On vous demande d'analyser l'algorithme de la fonction `filtre` (page 11) en agissant sur une liste **V** que l'on peut exprimer ainsi : 1) Traverser la liste **V** de taille *n* de la position 1 à la position *n* – 2, et 2) remplacer chaque cellule *i* par la valeur médiane des trois valeurs centrées autour de *i* : **w** = { **V**[*i* – 1] , **V**[*i*] , **V**[*i* + 1] }.

Pour mieux comprendre, considérons l'exemple suivant.

a. À l'état initial, **V** est donné comme suit :

V	22	4	10	20	2	6	8	18	14	24	16	12	26
	0	1	2	3	4	5	6	7	8	9	10	11	12

b. À la première itération, **w** = {22 , 4 , 10}, `mediane( w ) = 10` :

Avant :

V	**22**	**4**	**10**	20	2	6	8	18	14	24	16	12	26
	0	**1**	2	3	4	5	6	7	8	9	10	11	12

Après :

V	**22**	10	**10**	20	2	6	8	18	14	24	16	12	26
	0	**1**	2	3	4	5	6	7	8	9	10	11	12

c. À la seconde itération, **w** = {10, 10, 20}, `mediane( w ) = 10` :

Avant :

V	22	**10**	**10**	**20**	2	6	8	18	14	24	16	12	26
	0	1	**2**	3	4	5	6	7	8	9	10	11	12

Après :

V	22	**10**	10	**20**	2	6	8	18	14	24	16	12	26
	0	1	**2**	3	4	5	6	7	8	9	10	11	12

d. À la dernière itération, **w** = {16, 16, 26}, `mediane( w ) = 16`, on trouve :

Avant :

V	22	10	10	10	6	6	8	14	14	16	**16**	**16**	**26**
	0	1	2	3	4	5	6	7	8	9	10	**11**	12

Après :

V	22	10	10	10	6	6	8	14	14	16	**16**	16	**26**
	0	1	2	3	4	5	6	7	8	9	10	**11**	12

Donnez la complexité algorithmique de cet algorithme en fonction de *n,* la taille de **V**. Justifiez. **(15 pts)**

Question 5 (30 points)

Vous être responsable d'un petit site web appelé *Locatif.ca*. Ce site permet de consulter des petites annonces en ligne pour se trouver un logement à louer. Les annonces sont stockées dans une liste. Les nouvelles annonces sont ajoutées à la fin de la liste. Les recherches effectuées par les internautes sont des requêtes qui utilisent le prix comme critère de base : l'internaute spécifie un montant maximal et votre moteur de recherche renvoie une liste d'appartements correspondant aux critères de prix spécifié.

Illustrons la requête par un exemple. Prenons l'ensemble des annonces suivantes :

Annonce	Prix
(1)	600$
(2)	750$
(3)	1200$
(4)	615$
(5)	720$
(6)	550$
(7)	780$
(8)	510$
(9)	580$

La requête « Prix ≤ 650$ » (`LogementsPrixInferieurEgal( 650)`) renvoie :

Annonce	Prix
(1)	600$
(4)	615$
(6)	550$
(8)	510$
(9)	580$

Jusqu'alors, votre algorithme consistait à traverser séquentiellement l'ensemble des annonces jusqu'à trouver celles respectant la requête et les renvoyer.

5.1 Quelle est la complexité algorithmique (fonction de la taille *n* de votre base de données) de votre approche actuelle ? Justifiez brièvement. **(5 pts)**

On vous propose d'utiliser un arbre de recherche utilisant comme clé le prix de location, tel que présenté par la figure **Arbre.1** pour l'ensemble des annonces de l'exemple précédent :

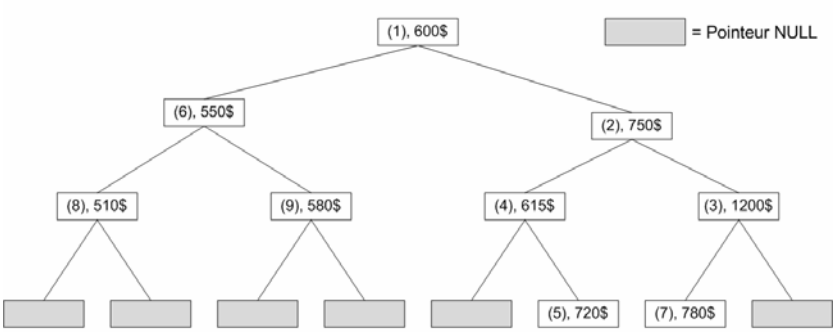


Figure Arbre.1.

5.2 Quelle est la complexité algorithmique d'insertion d'une nouvelle annonce dans la base de données ? Justifiez brièvement et comparez avec la complexité algorithmique d'insertion de l'approche précédemment utilisée. **(6 pts)**

5.3 En utilisant les schémas de l'annexe 3 aux pages 14 et 15 que l'on vous demande de joindre à votre cahier d'examen, illustrez l'état de l'arbre de recherche de la figure **Arbre.1** après les opérations suivantes **(7 pts)** :

5.3.1 Insertion « (10), 610 » ;

5.3.2 Suite à l'ajout de « (10), 610 », retrait de « (2), 750 » ;

5.3.3 Suite au retrait de « (2), 750 », retrait de « (1), 600 ».

5.4 En vous servant des schémas de l'annexe 4 à la page 16 que l'on vous demande de joindre à votre cahier d'examen, encerclez les sous-arbres devant être retounés pour chacune des requêtes suivantes **(7 pts)** :

5.4.1 La requête « Prix ≤ 600$ » (`LogementsPrixInferieurEgal( 600)`) ;

5.4.2 La requête « Prix ≤ 750$ » (`LogementsPrixInferieurEgal( 750)`).

5.5 L'utilisation d'un arbre de recherche pour votre site web *Locatif.ca* présente des limitations. Donnez au moins deux raisons claires et importantes pour étayer cette affirmation. **(5 pts)**

Annexe 1

Document à joindre à votre cahier d'examen

a)

a.1)

Pivot (élément central)	

a.2)

Partition gauche															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

a.3)

Partition droite															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

a.4)

Pivot (mediane3)	

a.5)

Partition gauche															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

a.6)

Partition droite															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Nom de l'étudiant : _____ Matricule : _____

b) Document à joindre à votre cahier d'examen

b.1)

Pivot (élément central)	

b.2)

Partition gauche															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

b.3)

Partition droite															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

b.4)

Pivot (mediane3)	

b.5)

Partition gauche															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

b.6)

Partition droite															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Nom de l'étudiant : _____ Matricule : _____

c) Document à joindre à votre cahier d'examen

c.1)

Pivot (élément central)	

c.2)

Partition gauche															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

c.3)

Partition droite															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

c.4)

Pivot (mediane3)	

c.5)

Partition gauche															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

c.6)

Partition droite															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Nom de l'étudiant : _____ Matricule : _____

Annexe 2

```
/**
  * Classe ComparableArrayList implémentant un array liste
  */
public class ComparableArrayList<AnyType extends Comparable<? super AnyType>> implements
Iterable<AnyType>
{
    /**
     * Création d'une liste vide
     */
    public ComparableArrayList( ) { clear( ); }

    /**
     * Renvoie le nombre d'éléments dans la liste
     */
    public int size( ) { return theSize; }

    /**
     * Renvoie vrai si la liste est vide
     */
    public boolean isEmpty( ) { return size( ) == 0; }

    /**
     * Renvoie l'élément de la liste à la position idx
     */
    public AnyType get( int idx )
    {
        if( idx < 0 || idx >= size( ) )
            throw new ArrayIndexOutOfBoundsException( "Index " + idx + "; size " + size( ) );
        return theItems[ idx ];
    }

    /**
     * Modifie l'élément à la position idx en lui donnant la valeur newVal
     */
    public AnyType set( int idx, AnyType newVal )
    {
        if( idx < 0 || idx >= size( ) )
            throw new ArrayIndexOutOfBoundsException( "Index " + idx + "; size " + size( ) );
        AnyType old = theItems[ idx ];
        theItems[ idx ] = newVal;

        return old;
    }

    /**
     * S'Occupe de grossir le vecteur au besoin
     */
    @SuppressWarnings("unchecked") // cette commande évite des warnings lors de la compilation
    public void ensureCapacity( int newCapacity )
    {
        if( newCapacity < theSize )
            return;

        AnyType [ ] old = theItems;
        theItems = (AnyType []) new Comparable[ newCapacity ];
        for( int i = 0; i < size( ); i++ )
            theItems[ i ] = old[ i ];
    }

    /**
     * Ajoute l'élément x à la fin de la liste
     */
    public boolean add( AnyType x )
    {
         add( size( ), x );
        return true;
    }
```

```
/**
 * Ajoute l'élément x à la position idx de la liste
 */
public void add( int idx, AnyType x )
{
    if( theItems.length == size( ) )
                ensureCapacity( size( ) * 2 + 1 );

           for( int i = theSize; i > idx; i-- )
                theItems[ i ] = theItems[ i - 1 ];

    theItems[ idx ] = x;
           theSize++;
}

/**
 * Retire l'élément à la position idx
 */
public AnyType remove( int idx )
{
    AnyType removedItem = theItems[ idx ];

    for( int i = idx; i < size( ) - 1; i++ )
        theItems[ i ] = theItems[ i + 1 ];
    theSize--;

    return removedItem;
}

/**
 * Vide la liste
 */
public void clear( )
{
    theSize = 0;
    ensureCapacity( DEFAULT_CAPACITY );
}

/**
 * Permet d'obtenir un itérateur sur la liste
 */
public java.util.Iterator<AnyType> iterator( ) {  return new ArrayListIterator( ); }

/**
  * Permet d'affiche la liste
  */
public String toString( )
{
    StringBuilder sb = new StringBuilder( "[ " );

    for( AnyType x : this )
           sb.append( x + " " );
    sb.append( "]" );

    return new String( sb );
}

/**
  * Classe interne implémentation les itérateurs sur la liste
  */
private class ArrayListIterator implements java.util.Iterator<AnyType>
{
   private int current = 0;
    private boolean okToRemove = false;

    public boolean hasNext( )
    {
        return current < size( );
    }
```

```
        public AnyType next( )
        {
            if( !hasNext( ) )
                throw new java.util.NoSuchElementException( );

            okToRemove = true;
            return theItems[ current++ ];
        }

        public void remove( )
        {
            if( !okToRemove )
                throw new IllegalStateException( );

            ComparableArrayList.this.remove( --current );
            okToRemove = false;
        }
    }

    /**
     * Algorithme median.
     */
    private AnyType median( AnyType [] w)
    {
       // Verifier que le vecteur contient des elements
       if(w == null)
              return null;

       // Verifier que le vecteur contient des elements
       if(w.length <1)
              return null;

       // trier le vecteur w
       IntraSort(w);

       // retourner la valeur mediane
       int med = ( (w.length - 1)/2 );

       return w[ med ];
    }

    /**
     * Algorithme de filtrage
     */
    @SuppressWarnings("unchecked")
    public void filtre()
    {
       // Verifier que le vecteur contient au moins 3 elements
       if(theSize<3)
              return;

       for(int i=1; i<theSize-1; i++)
       {
              // creer le vecteur de 3 elements
              AnyType [ ] w = (AnyType[]) new Comparable[ 3 ];

              // inserer les elements
              w[ 0 ] = theItems[ i-1 ];
              w[ 1 ] = theItems[ i   ];
              w[ 2 ] = theItems[ i+1 ];

              // remplacer V[i] par la valeur mediane
              theItems[ i ] = median( w );
       }
    }
```

```java
/**
 *  Algorithm de tri
 */
@SuppressWarnings("unchecked") // évite les warning lors de la compilation
private AnyType [ ] IntraSort( AnyType [] w )
{
    // Copie le vecteur w
    AnyType [ ] tw = (AnyType[]) new Comparable[ w.length ];

    // Appelle un algorithme de tri sur le vecteur w
     IntraSort( w, tw, 0, w.length - 1 );

     return tw;
}

/**
 * Fonction récursive pour le tri
 * @param w    : vecteur d'entrée
 * @param tw   : vecteur copie
 * @param left : position la plus à gauche de la sous-liste à trier
 * @param right: position la plus à droite de la sous-liste à trier
 */
private void IntraSort( AnyType [ ] w, AnyType [ ] tw, int left, int right )
{
    if( left < right )
    {
        // diviser la sous-liste en deux
           int center = ( left + right ) / 2;

           // Appliquer récurivement sur la moitié de gauche
        IntraSort( w, tw, left, center );

        // Appliquer récurivement sur la moitié de droite
        IntraSort( w, tw, center + 1, right );

        // Unir les résultats
        merge( w, tw, left, center + 1, right );
    }
}

/**
 * Unit deux sous-listes triées
 * @param w    : vecteur d'entrée
 * @param tw   : vecteur copie
 * @param leftPos : index de l'élément le plus à gauche
 * @param rightPos: index du premier élément de la seconde sous-liste
 * @param rightEnd : index de l'élément le plus à droite
 */
void merge( AnyType [ ] w, AnyType [ ] tw, int leftPos, int rightPos, int rightEnd )
{
    int leftEnd = rightPos - 1; int tmpPos = leftPos;
    int numElements = rightEnd - leftPos + 1;

    // Boucle principale
    while( leftPos <= leftEnd && rightPos <= rightEnd )
        if( w[ leftPos ].compareTo( w[ rightPos ] ) <= 0 )
            tw[ tmpPos++ ] = w[ leftPos++ ];
        else
            tw[ tmpPos++ ] = w[ rightPos++ ];

   // Recopier les éléments de la première sous-liste si il en reste
    while( leftPos <= leftEnd )
        tw[ tmpPos++ ] = w[ leftPos++ ];
    // Recopier les éléments de la seconde sous-liste si il en reste
    while( rightPos <= rightEnd )
        tw[ tmpPos++ ] = w[ rightPos++ ];

 // Recopier tw dans w
    for( int i = 0; i < numElements; i++, rightEnd-- )
        w[ rightEnd ] = tw[ rightEnd ];
}
```

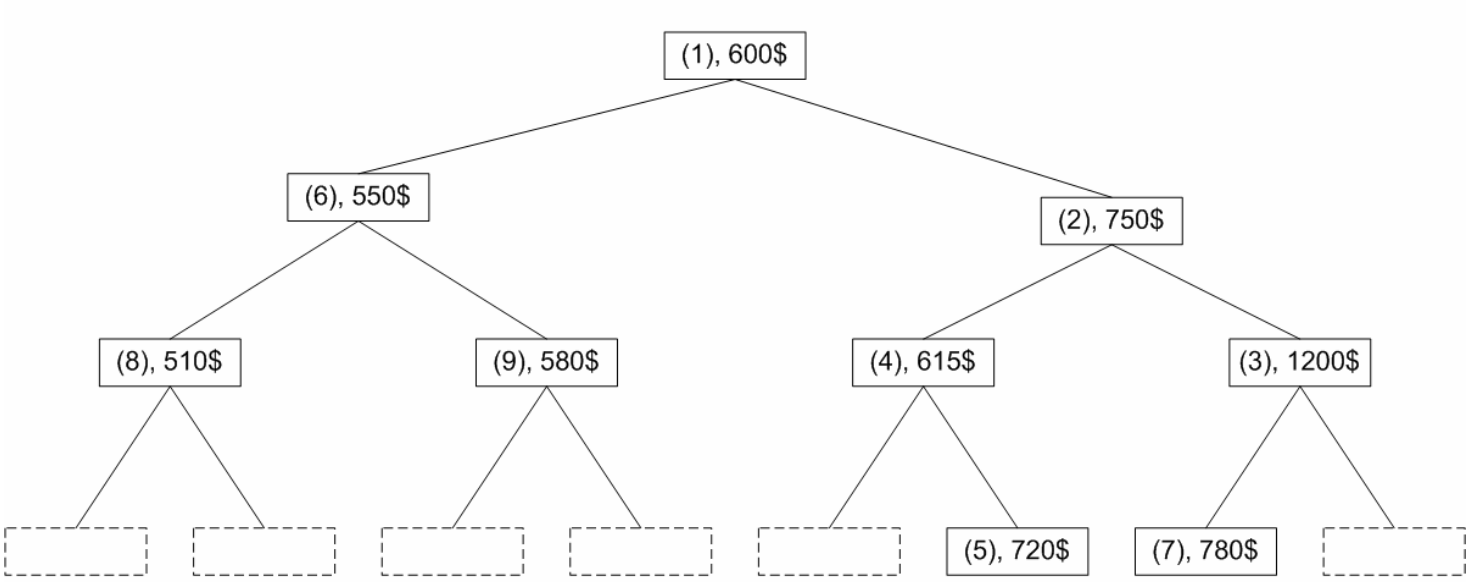
```
    private static final int DEFAULT_CAPACITY = 10;// taille min du vecteur
    private AnyType [ ] theItems; // vecteur des données dans la liste
    private int theSize; // taille de la liste
}

/**
 * Fonction principale de l'exemple
 */
class IntraArrayList
{
        public static void main( String [ ] args )
        {
                // Créer V
                ComparableArrayList<Integer> V = new ComparableArrayList<Integer>( );

                // Remplir V avec les valeurs
                V.add(22);V.add(4);V.add(10);V.add(20);V.add(2);V.add(6);
                V.add(8);V.add(18);V.add(14);V.add(24);V.add(16);V.add(12);V.add(26);

                // Afficher V
                System.out.println( V );
                // L'appel donne
                // [ 22 4 10 20 2 6 8 18 14 24 16 12 26 ]

                // Appliquer l'algorithme de filtre sur V
                V.filtre();

                // Afficher V
                System.out.println( V );
                // L'appel donne
                // [ 22 10 10 10 6 6 8 14 14 16 16 16 26 ]
        }
}
```

Annexe 3

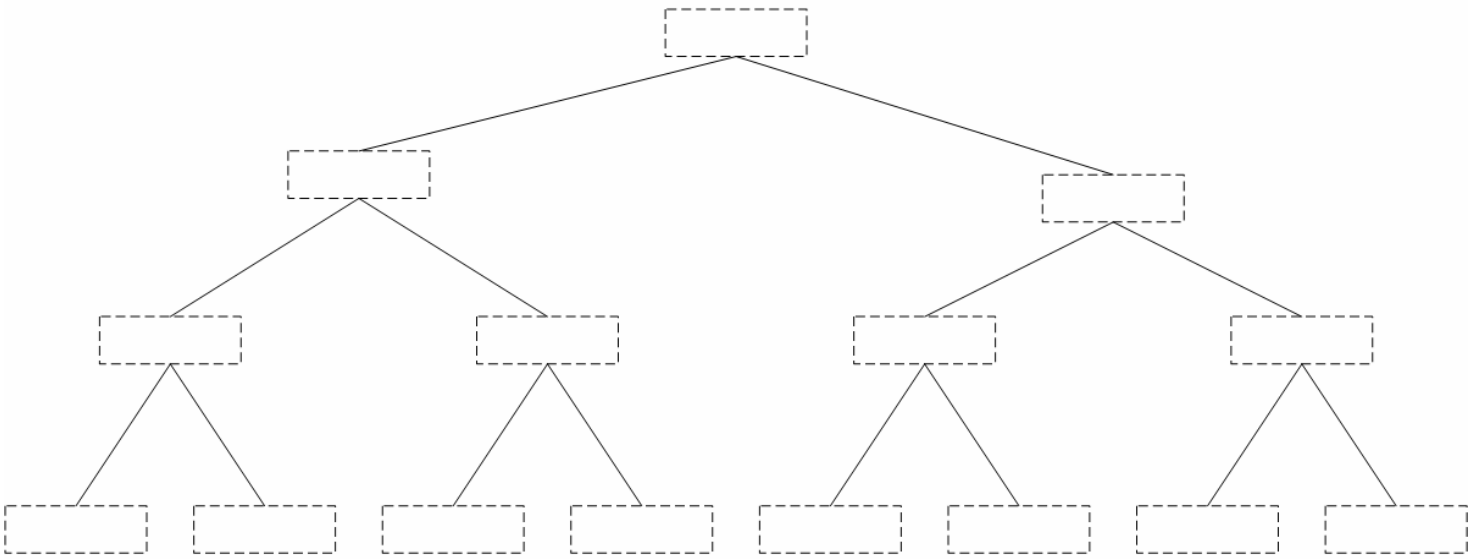
Document à joindre à votre cahier d'examen

Illustrez l'état de l'arbre de recherche de la figure **Arbre.1** après les opérations suivantes :

5.3.1 Insertion « (10), 610 » :



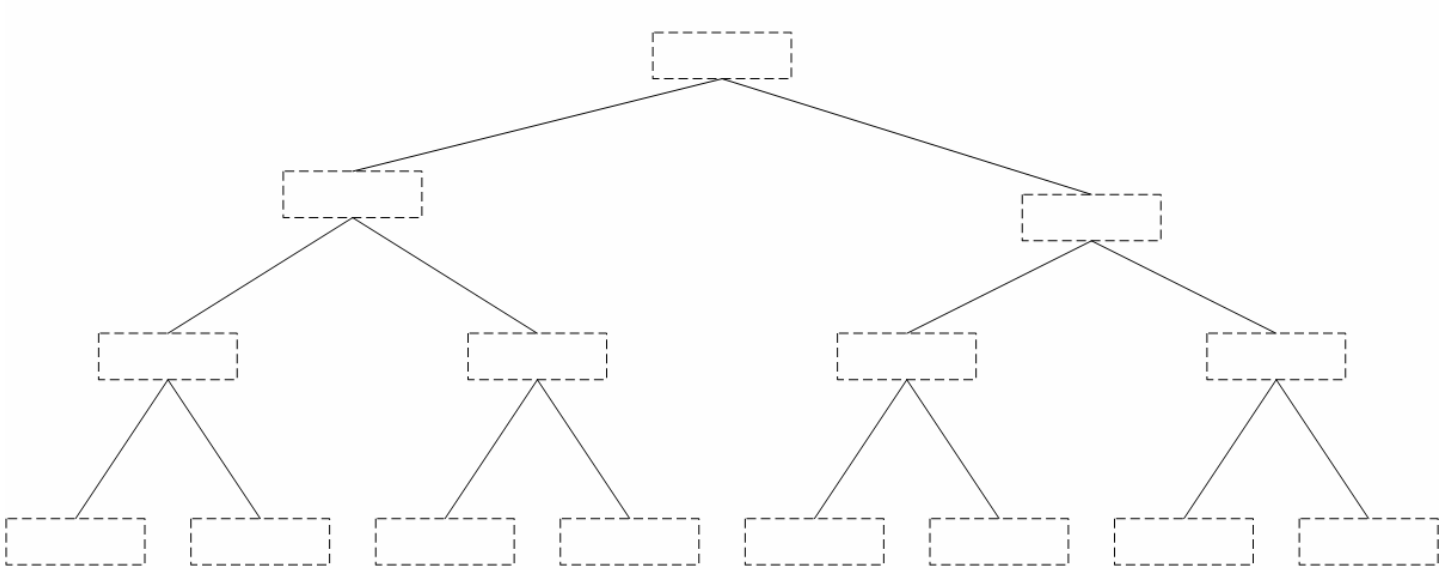
5.3.2 Suite à l'ajout de « (10), 610 », retrait de « (2), 750 » :



Nom de l'étudiant : _____ Matricule : _____

Document à joindre à votre cahier d'examen

5.3.3 Suite au retrait de « (2), 750 », retrait de « (1), 600 » :



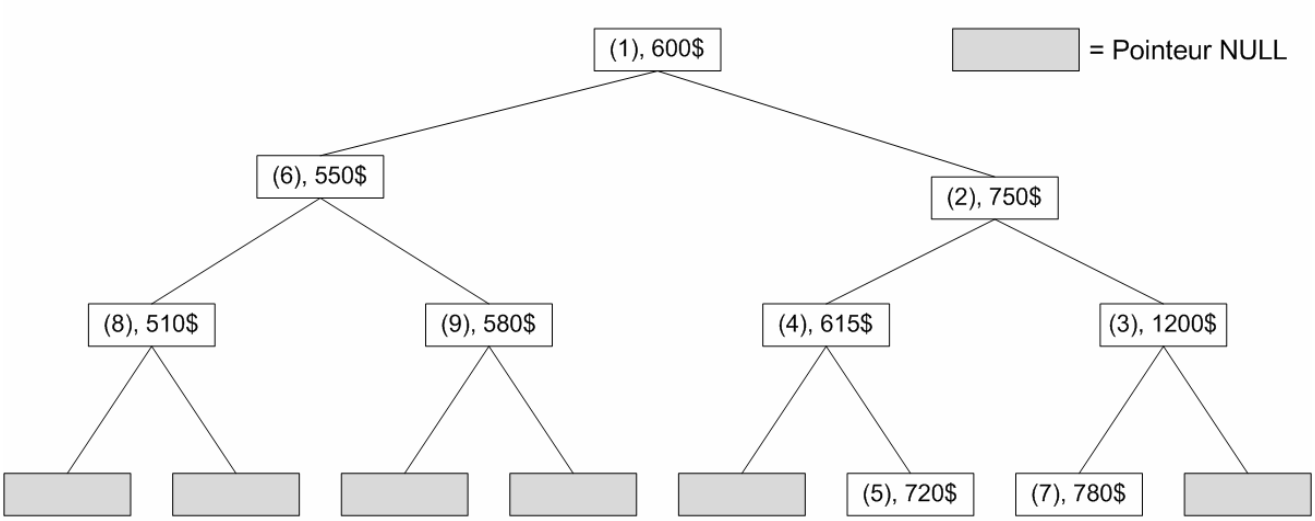
Nom de l'étudiant : _____ Matricule : _____

Annexe 4

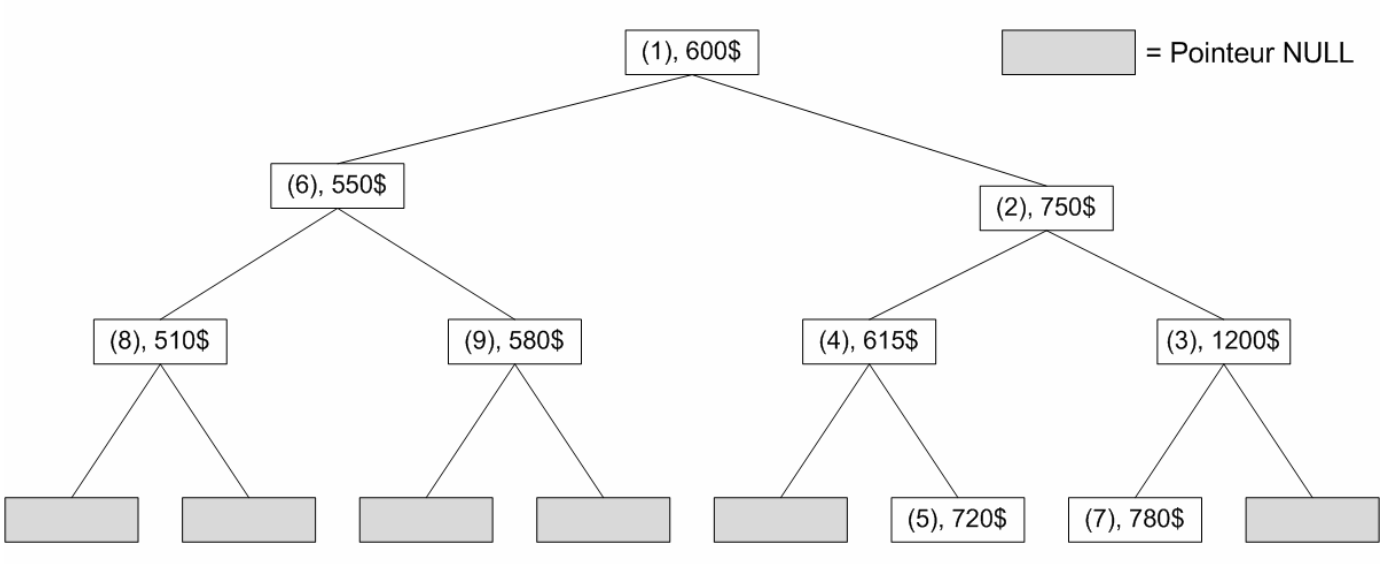
Document à joindre à votre cahier d'examen

Encerclez les sous-arbres devant être retounés pour chacune des requêtes suivantes :

5.4.1 La requête « Prix ≤ 600$ » (**`LogementsPrixInferieurEgal`**(`600`)) renvoie :



5.4.2 La requête « Prix ≤ 750$ » (**`LogementsPrixInferieurEgal`**(`750`)) renvoie :



Nom de l'étudiant : _____ Matricule : _____