POLYTECHNIQUE MONTRÉAL

Questionnaire examen intra

INF2010

Sigle du cours

Q1	
Q2	
Q3	
Q4	
Q5	
Q6	
Total	

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

Sigle et titre du cours		*Groupe*	*Trimestre*
INF2010 – Structures de données et algorithmes		**Tous**	**20161**
Professeur		***Local***	***Téléphone***
Ettore Merlo, responsable – Tarek Ould Bachir, chargé de cours		**M-1420**	
Jour	***Date***	***Durée***	***Heure***
Lundi	**22 février 2016**	**2h00**	**9h00**

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

Directives particulières

Bonne chance à tous!

Important

Cet examen contient **7** questions sur un total de **23** 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 : Tables de dispersement (10 points)

Soit une table de dispersement double Hash(clé) = (clé + i^2) % N.

1.1) **(5 points)** En vous servant du tableau ci-dessous, donnez l'état de la mémoire d'une table de taille N=13 après l'insertion, dans l'ordre, des clés suivantes:

13, 92, 78, 56, 31.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Entrées													

Donnez le détail de vos calculs ci-après:

x = 13:
x = 92:
x = 78:
x = 56:
x = 31:

1.2) **(2 points)** Après l'insertion des clés données à la question 1.1), on effectue un appel à `remove(69)` ? Donnez le détail de cet appel. Soyez bref mais précis. Vous pouvez vous aider du code source fourni à l'Annexe 1.

1.3) **(2 points)** Quelle sera la plus grande valeur prise par *i* (le *i* de Hash(clé) = (clé + i^2) % N) lors de l'appel `remove(69)`.

1.4) **(1 point)** Combien de clés supplémentaires faut-il ajouter à la table de dispersement pour que la fonction `rehash()` soit appelée?

Question 2 : Tris en $n \log(n)$ (20 points)

On désire exécuter l'algorithme *Quick Sort* pour trier le vecteur ci-après. On considère une valeur *cut-off* de 5. Le code source vous est fourni à l'Annexe 2.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Valeurs	16	15	2	9	4	3	6	13	8	7	10	1	12	11	14	5

2.1) (**3 points**) Donnez l'état du vecteur après la mise à l'écart du pivot lors de la première récursion de QuickSort :

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Valeurs																

2.2) (**5 points**) Donnez l'état du vecteur après l'exécution du partitionnement de la première récursion :

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Valeurs																

2.3) **(4 points)** Au total, quel est le nombre de fois que la fonction récursive `quicksort` aura été appelée pour exécuter le tri ? Pour éviter toute ambiguïté, la signature de la fonction est reproduite ci-après.

Signature de la fonction considérée :

```
private static <AnyType extends Comparable<? super AnyType>>
void quicksort( AnyType [ ] a, int left, int right )
```

Votre réponse: _____

2.4) **(2 points)** Quelle est la plus petite taille de vecteur sur laquelle la fonction `insertionSort` aura été appelée ?

Votre réponse: _____

2.5) (**2 points**) Reproduisez le vecteur correspondant à la question (2.4) en positionnant ses éléments à leur place (entre les indices `left` et `right` inclusivement) :

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Valeurs																

2.6) **(2 points)** Quelle est la plus grande taille de vecteur sur laquelle la fonction `insertionSort` aura été appelée ?

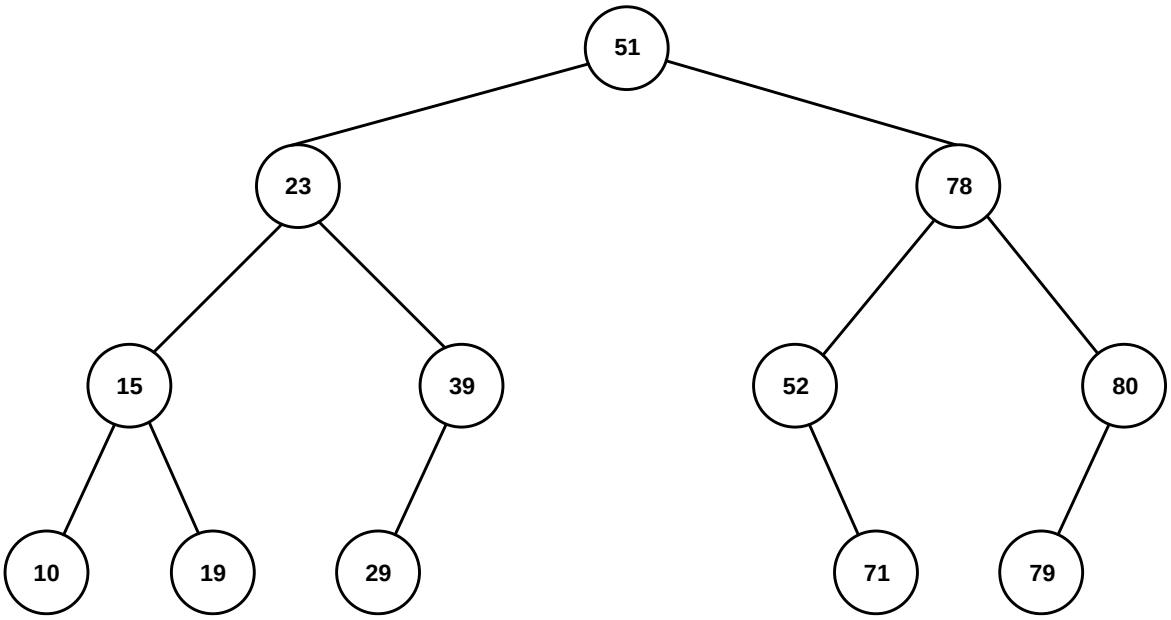
Votre réponse: _____

2.7) (**2 points**) Reproduisez le vecteur correspondant à la question (2.6) en positionnant ses éléments à leur place (entre les indices `left` et `right` inclusivement) :

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Valeurs																

Question 3 : Parcours d'arbres (8 points)

Considérez l'arbre binaire suivant:



3.1) (**2 points**) Donner le résultat de l'affichage de l'arbre binaire si il est parcouru en pré-ordre. Séparez les éléments par des virgules.

3.2) (**2 points**) Donner le résultat de l'affichage de l'arbre binaire si il est parcouru en post-ordre. Séparez les éléments par des virgules.

3.3) (**2 points**) Donner le résultat de l'affichage de l'arbre binaire si il est parcouru en ordre. Séparez les éléments par des virgules.

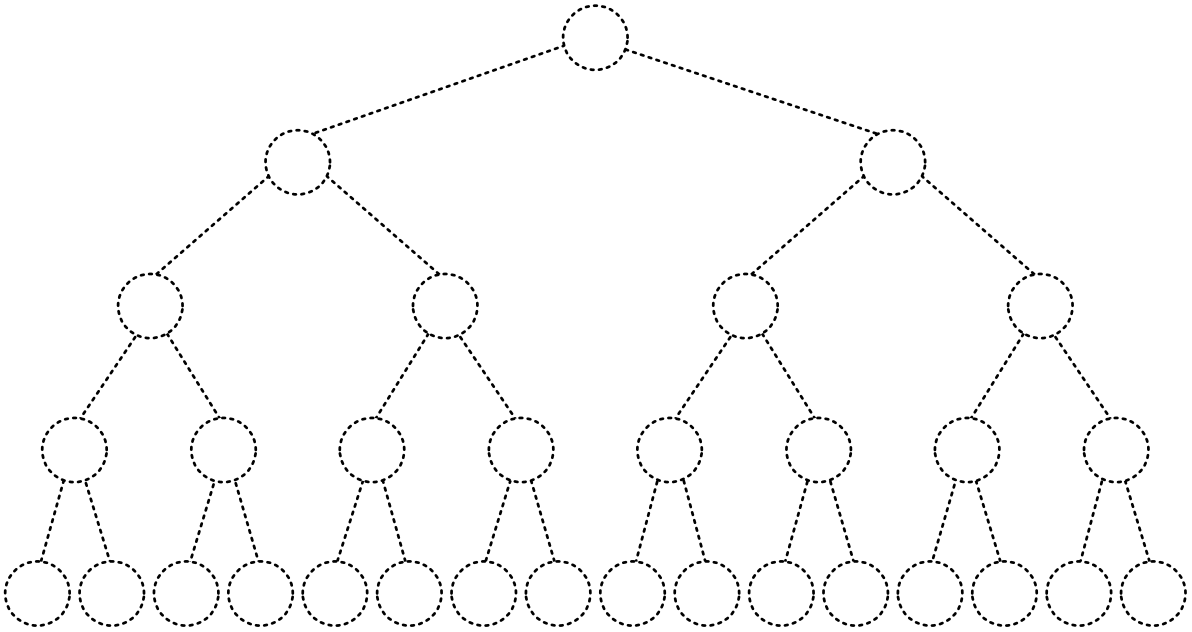
3.4) (**2 points**) Donner le résultat de l'affichage de l'arbre binaire si il est parcouru par niveaux. Séparez les éléments par des virgules.

Question 4 : Arbres binaire de recherche (16 points)

4.1) **(4 points)** Si l'affichage par niveaux de l'arbre binaire de recherche donne :

21, 10, 37, 18, 31, 12, 20, 25, 36

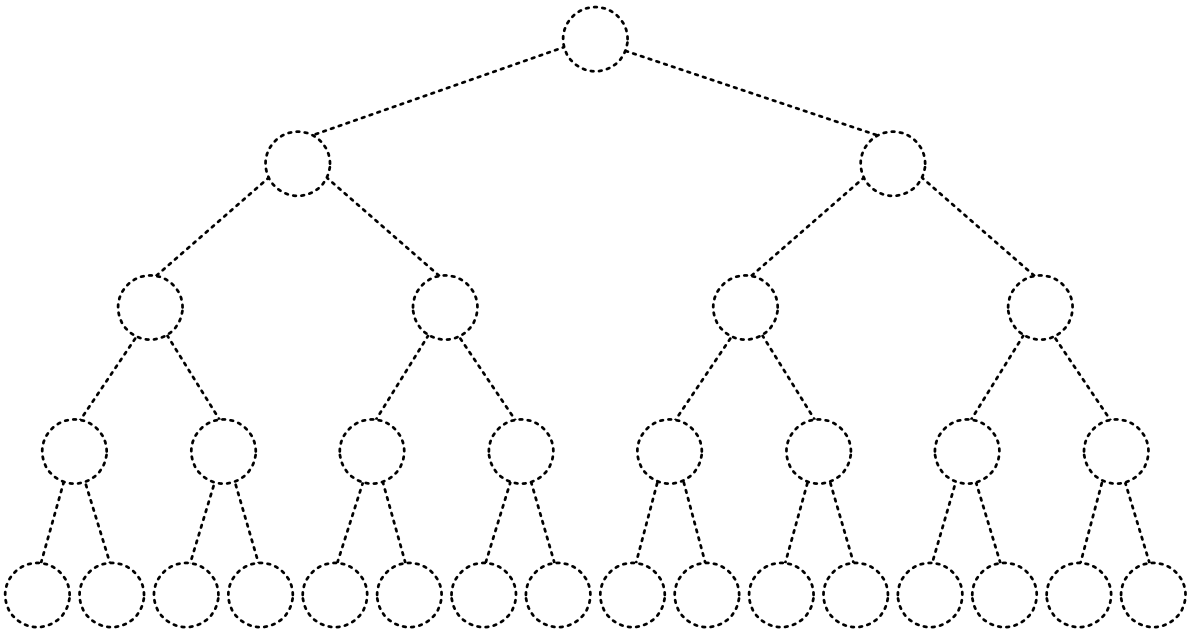
Donnez la représentation graphique de l'arbre.



4.2) **(4 points)** Si l'affichage pré-ordre de l'arbre binaire de recherche donne :

15, 26, 16, 21, 17, 36, 31, 35

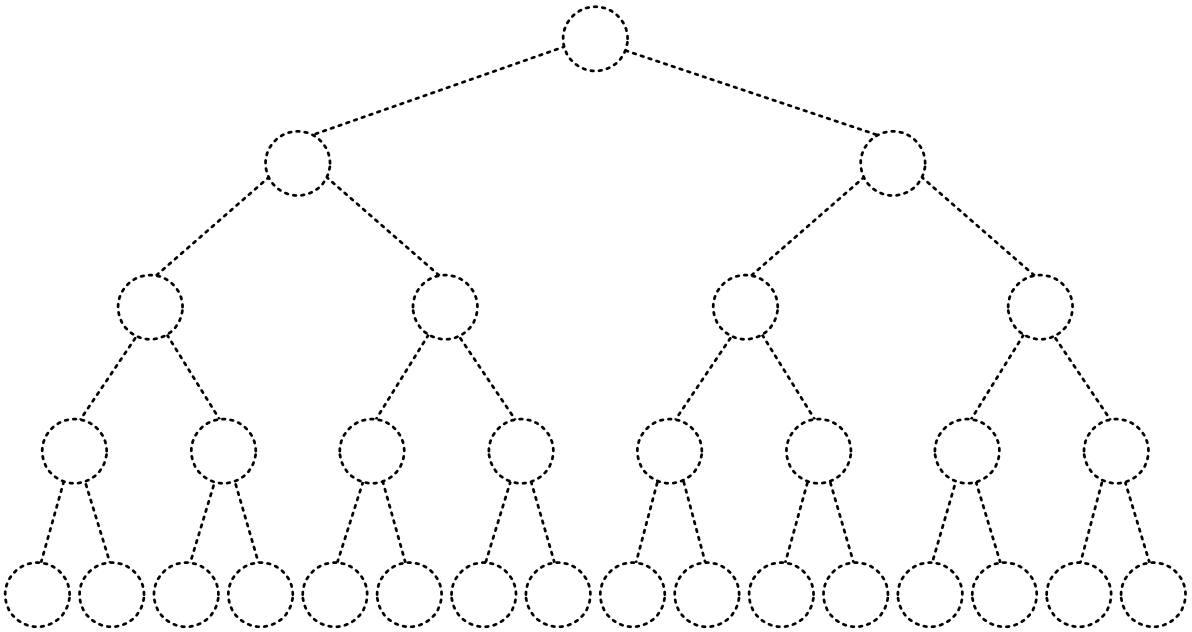
Donnez la représentation graphique de l'arbre.



4.3) **(4 points)** Si l'affichage post-ordre de l'arbre binaire de recherche donne :

17, 21, 16, 34, 31, 35, 26, 36

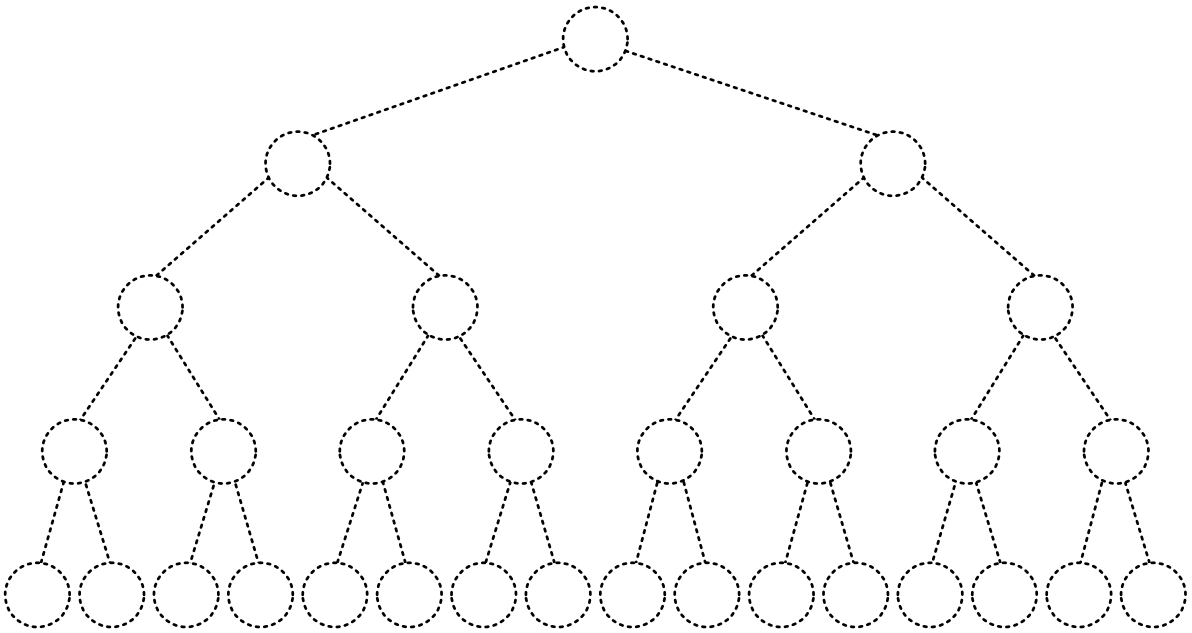
Donnez la représentation graphique de l'arbre.



4.4) **(4 points)** Si l'affichage en-ordre de l'arbre binaire de recherche donne :

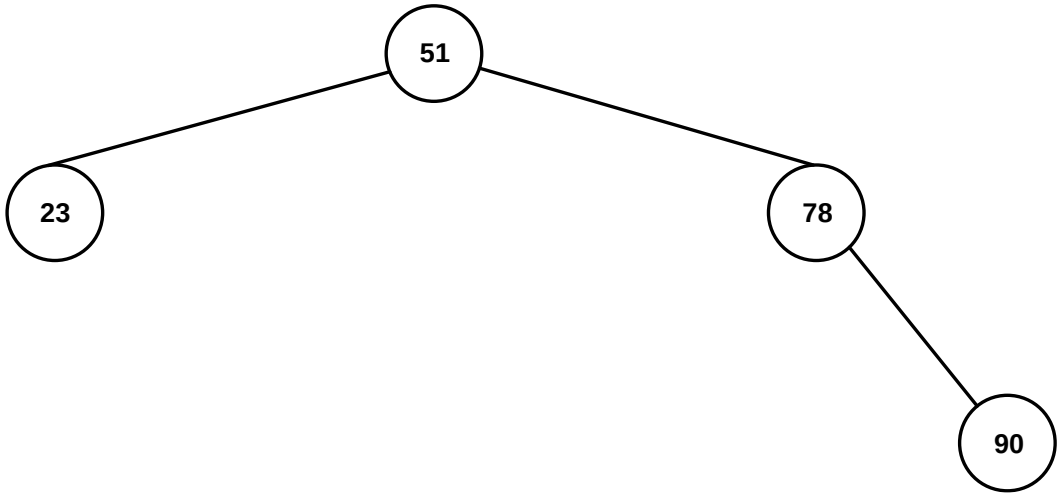
1, 10, 13, 14, 25, 31, 50, 61, 78, 80, 81

Donnez la représentation graphique de l'arbre, sachant que les sous-arbres à la gauche et à la droite de la racine sont des arbres complets comportant le même nombre de nœuds.



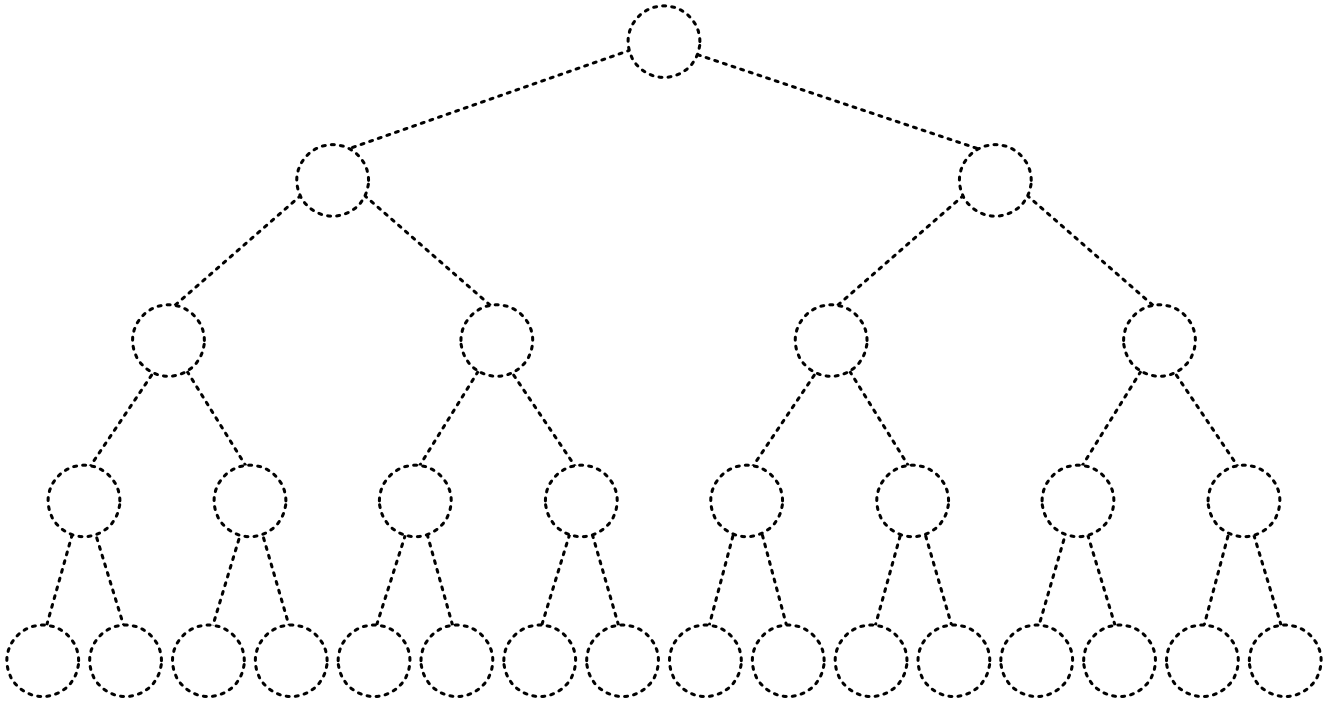
Question 5 : Arbre binaire de recherche de type AVL (18 points)

En partant de l'arbre AVL suivant :

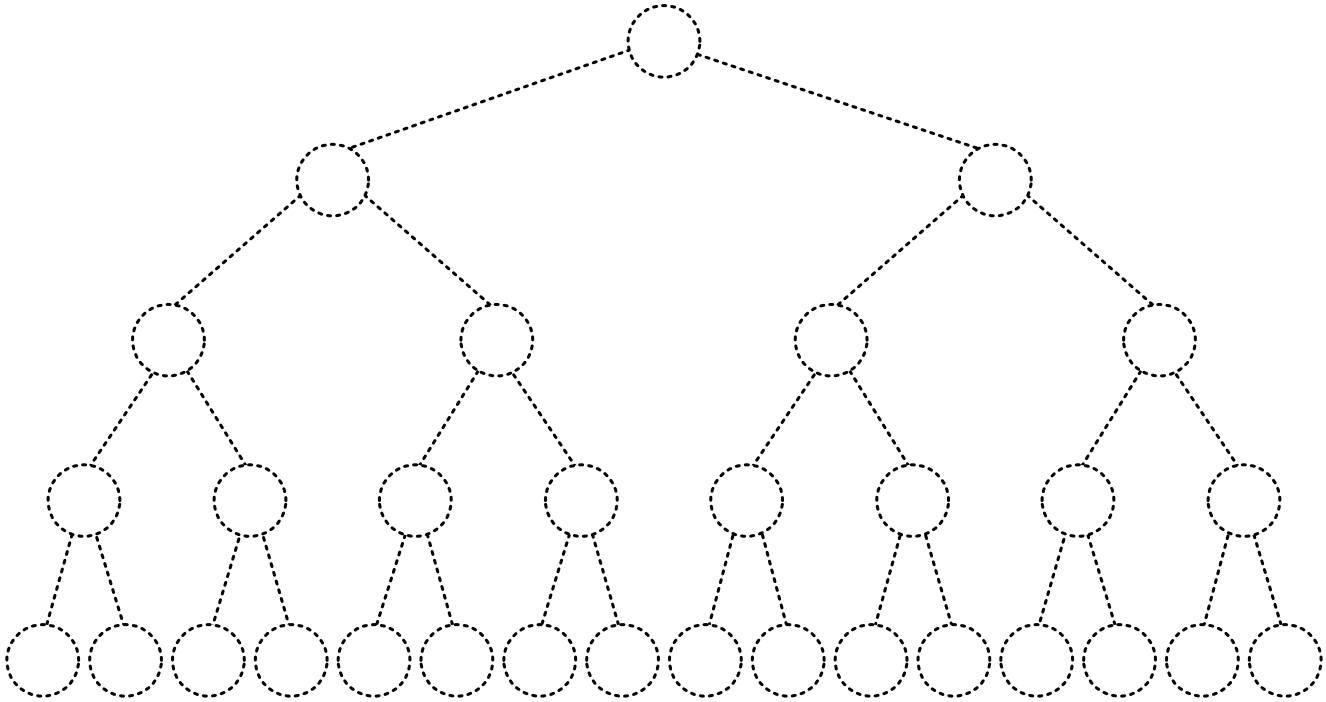


Insérez dans l'ordre les clés suivantes : 95, 85, 89, 81, 82

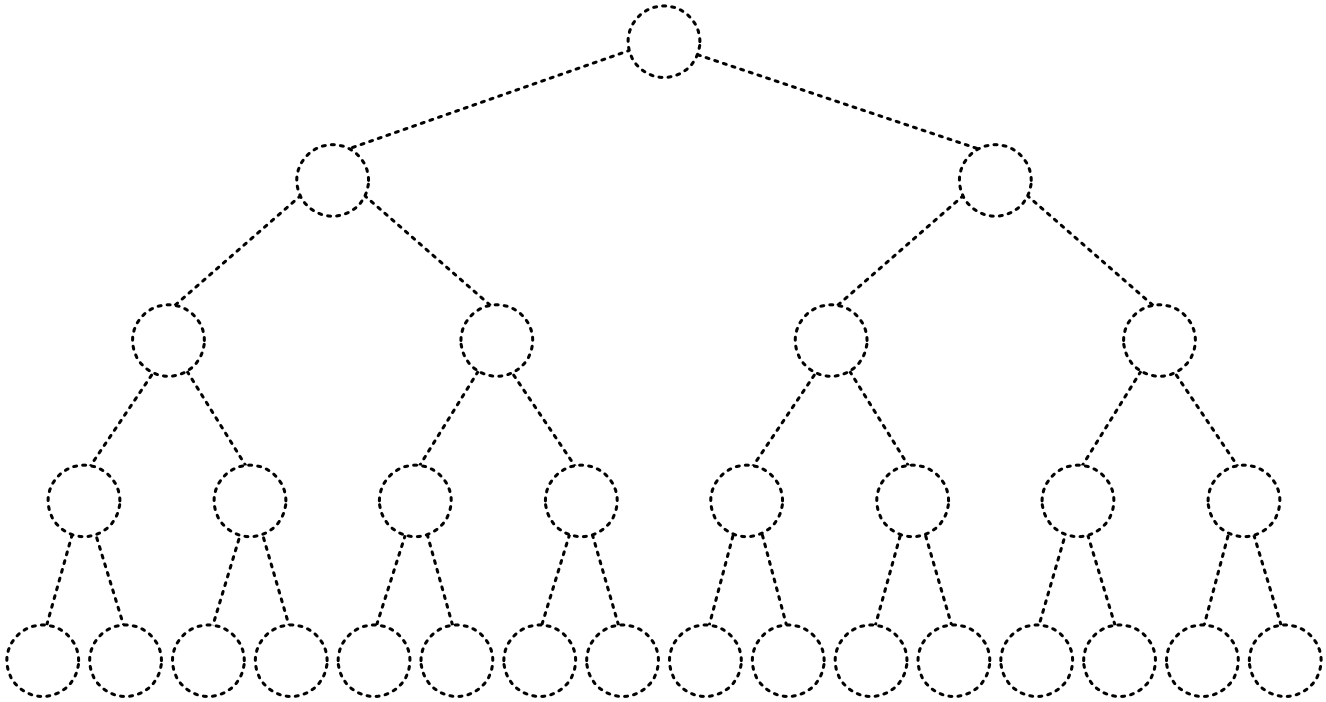
5.1) Insérez 95. **(3 points)**



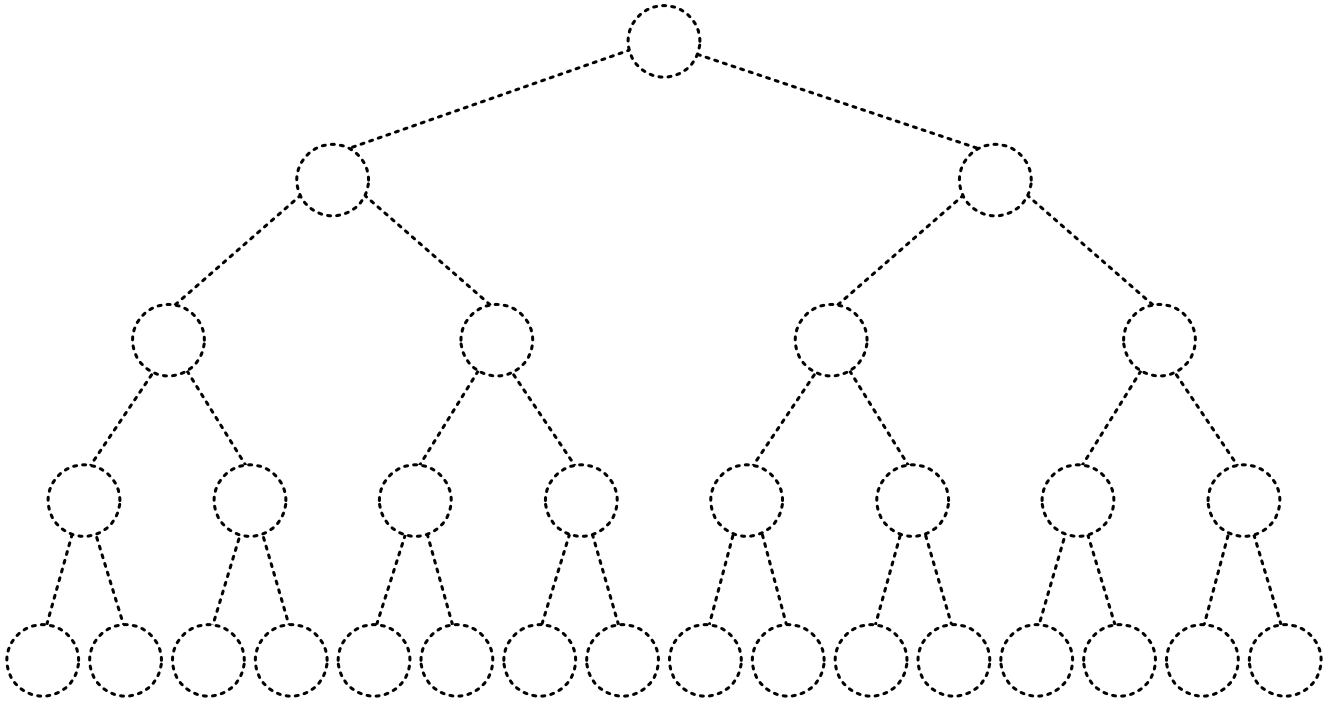
5.2) Insérez 85. **(3 points)**



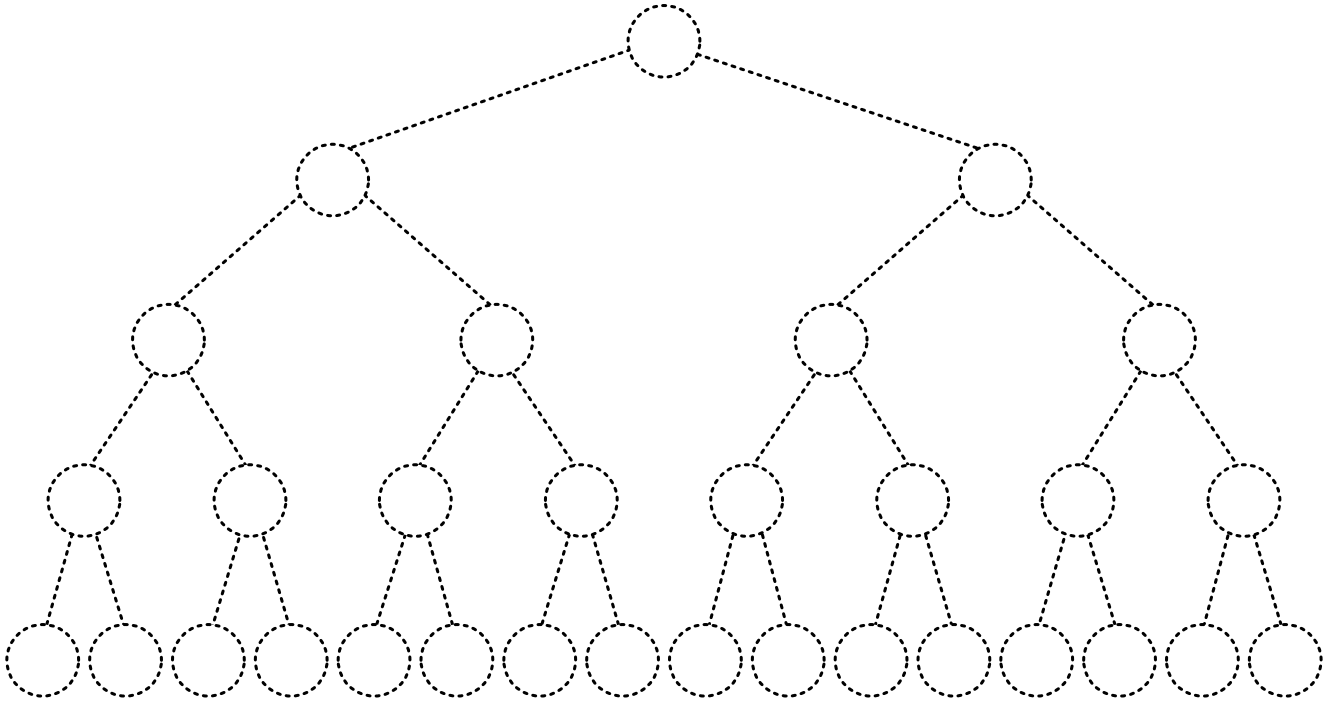
5.3) Insérez 89. **(3 points)**



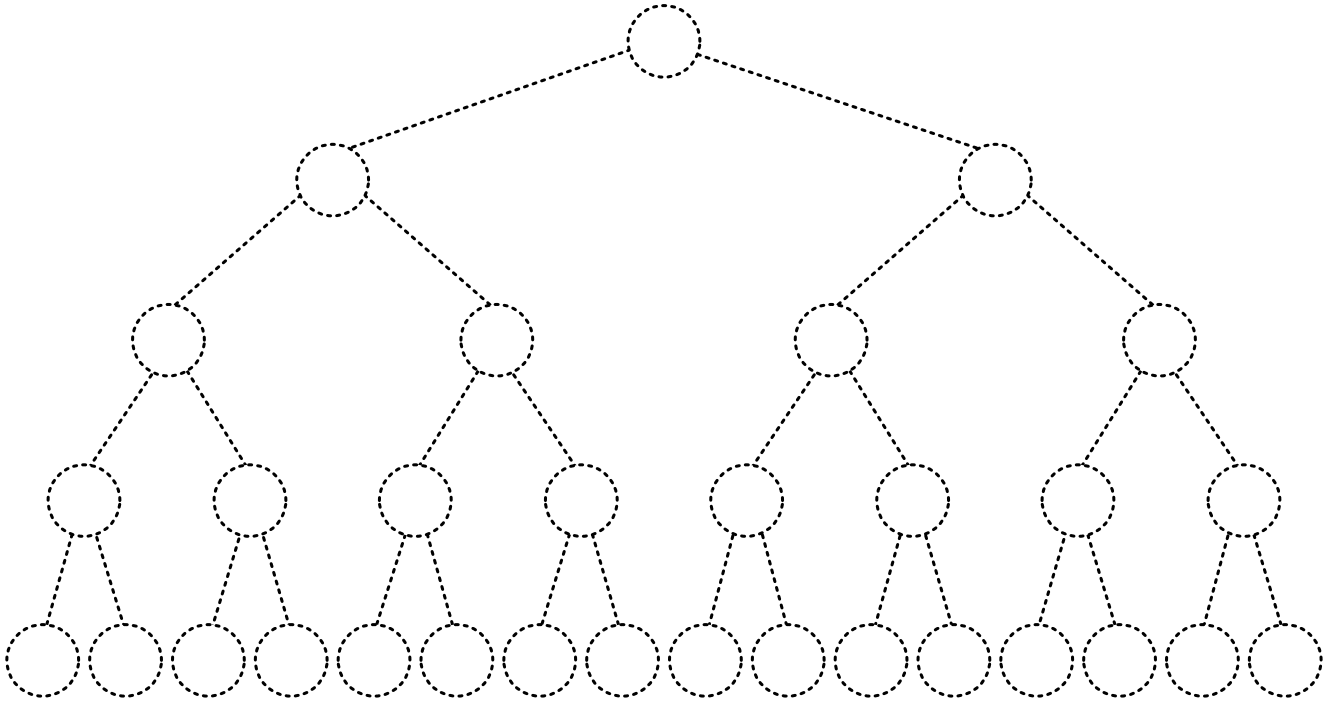
5.4) Insérez 81. **(3 points)**



5.5) Insérez 82. **(3 points)**

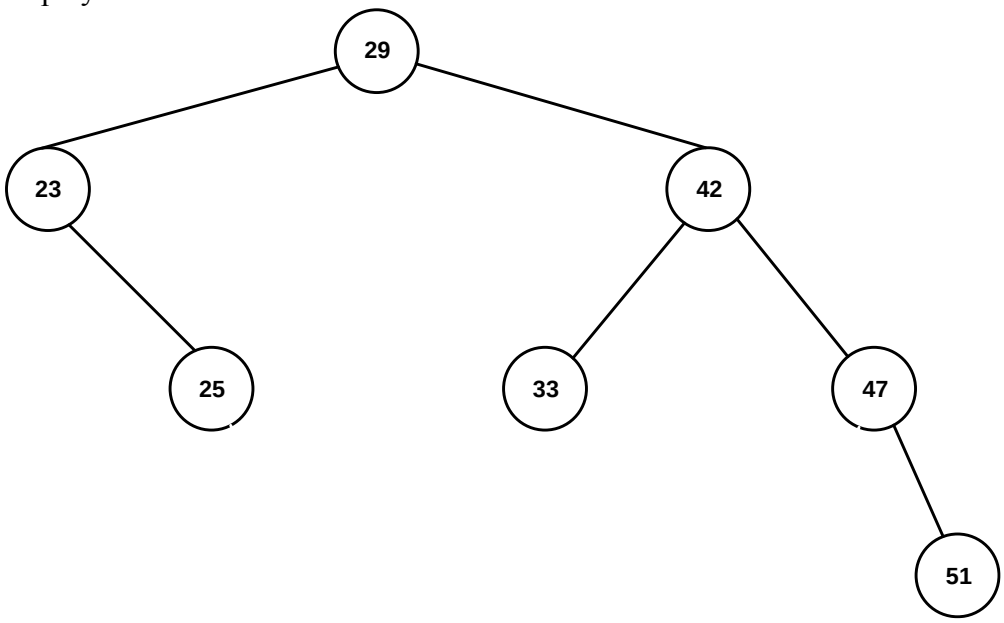


5.6) En partant du résultat de la question 5.5), retirez la racine. **(3 points)**

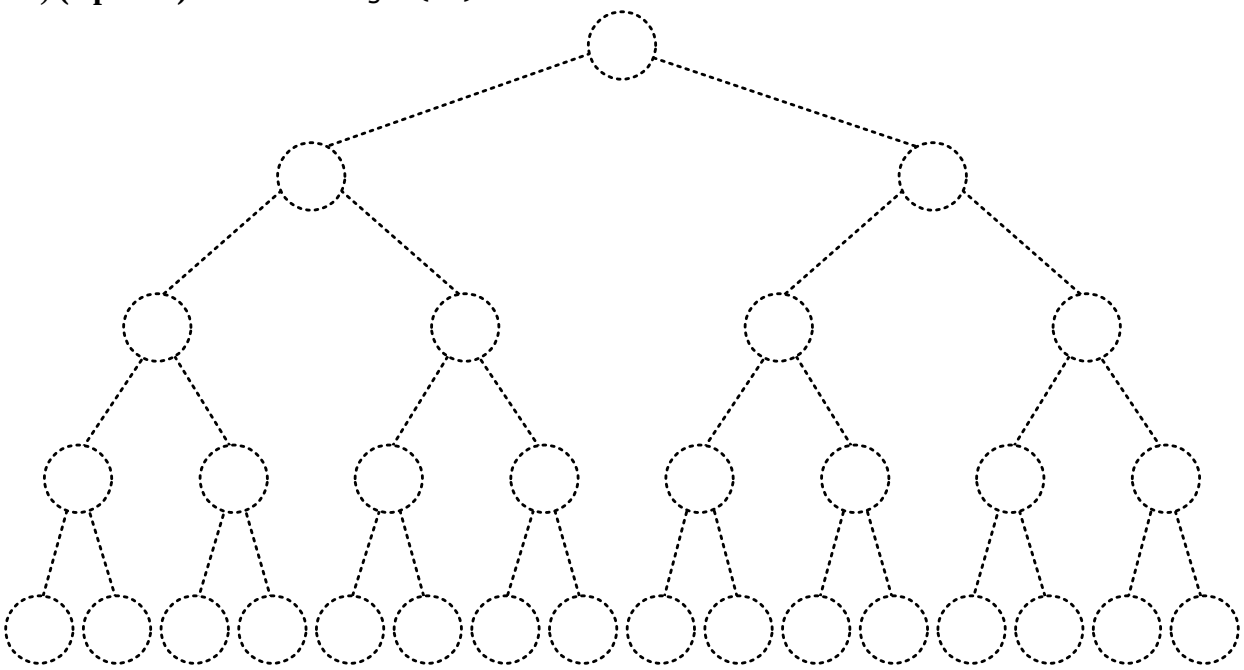


Question 6 : Arbre binaire de recherche de type Splay (12 points)

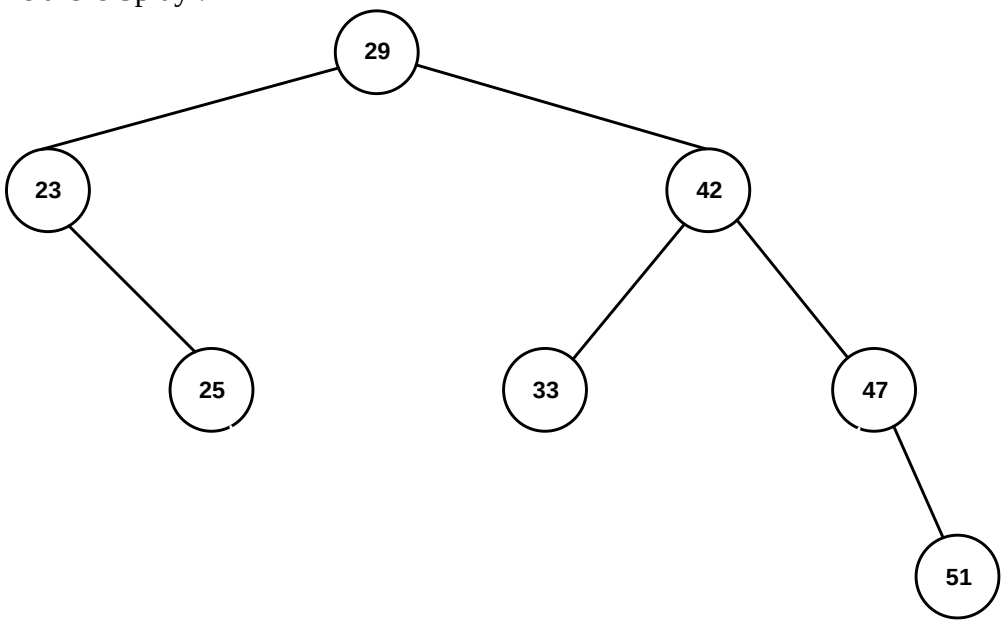
En partant de l'arbre Splay suivant :



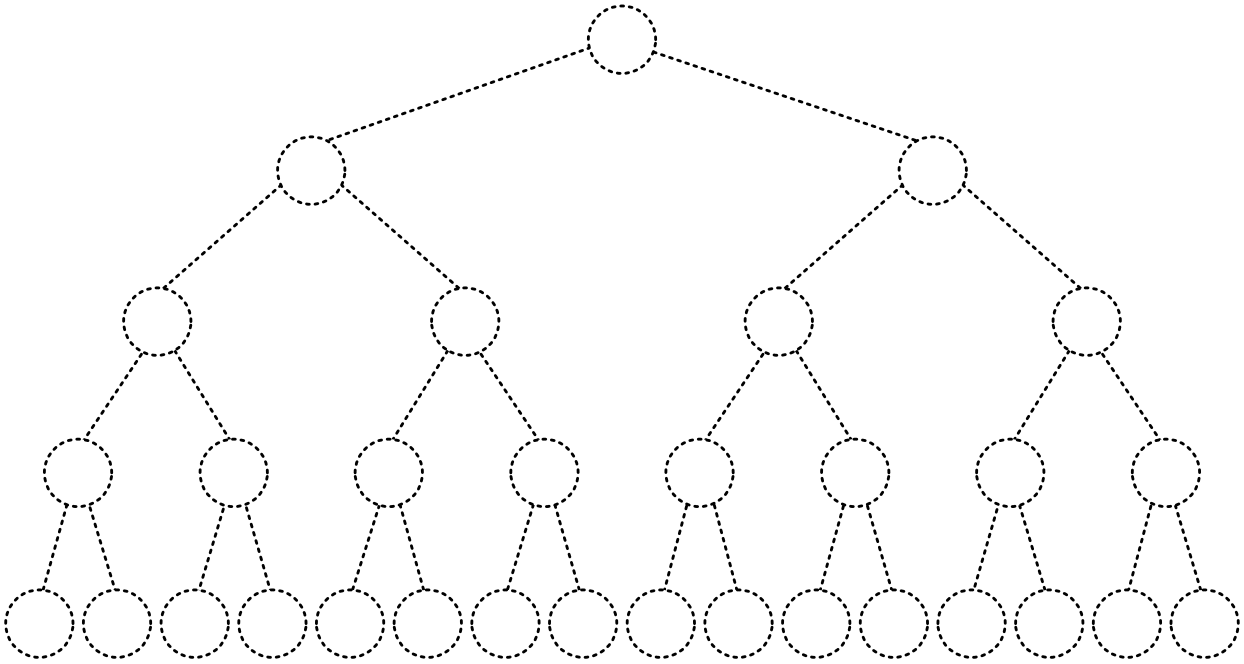
6.1) **(3 points)** Effectuez un `get(51)`.



En repartant du même arbre Splay :

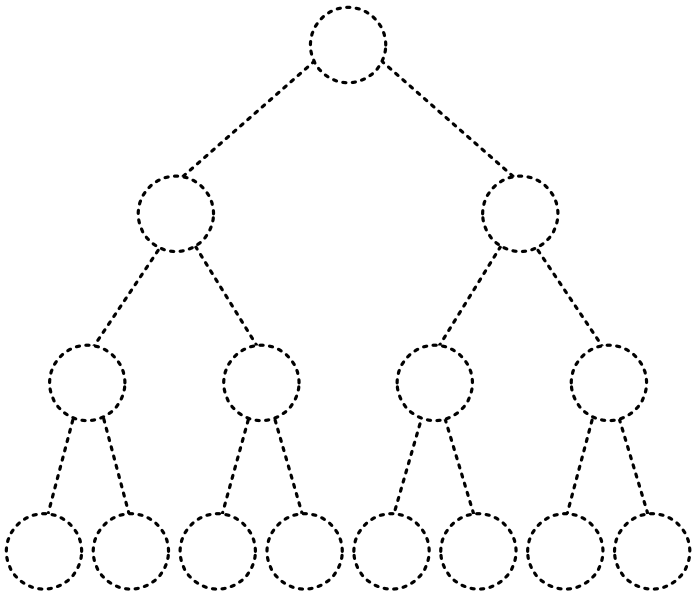


6.2) **(3 points)** Effectuez un `delete(33)`.

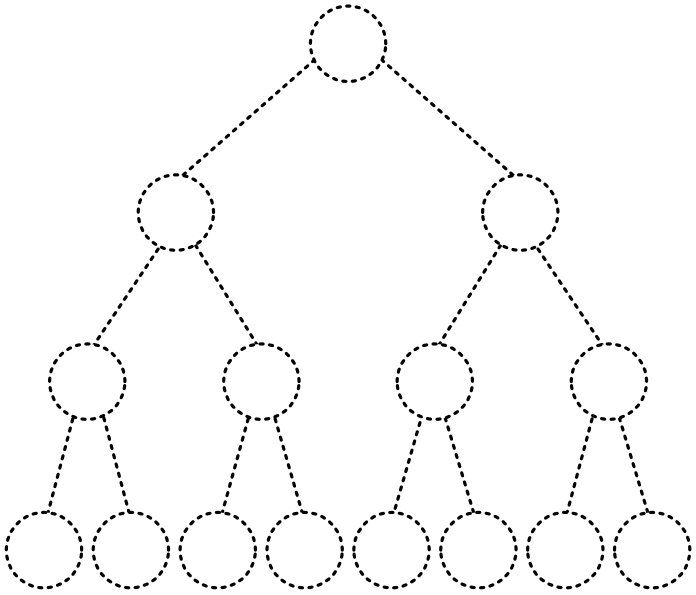


6.3) **(3 points)** Si le `get(51)` de la question 6.1) avait été exécuté par une implémentation de type top-down, quels auraient été les sous-arbres R et L juste avant que 51 ne soit placé à la racine ? **Aidez-vous des données de l'Annexe 3.**

Sous-arbre L:

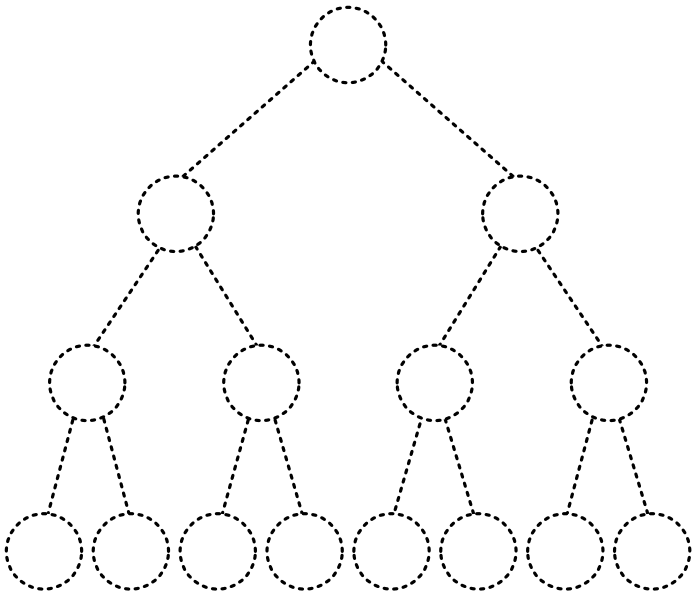


Sous-arbre R:

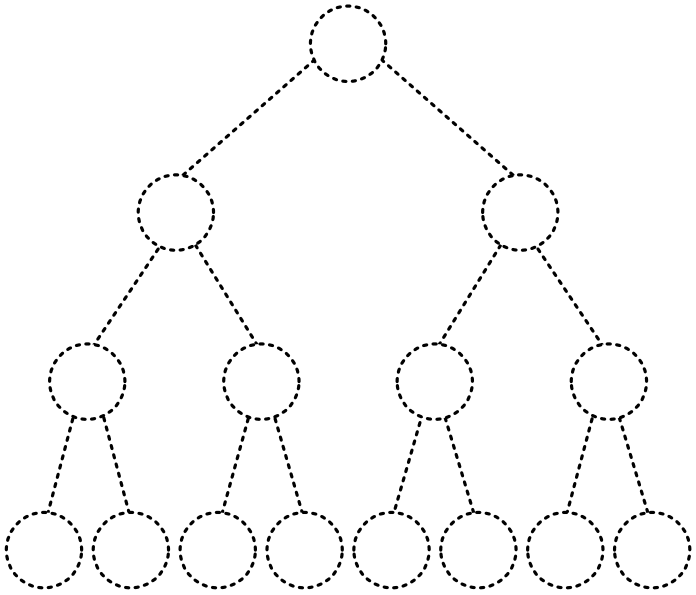


6.4) **(3 points)** Si le `delete(33)` de la question 6.2) avait été exécuté par une implémentation de type top-down, quels auraient été les sous-arbres R et L juste avant que 33 ne soit placé à la racine ? **Aidez-vous des données de l'Annexe 3.**

Sous-arbre L:



Sous-arbre R:



Question 7 : Généralités (16 points)

Répondez aux assertions suivantes par « vrai » ou par « faux ». Il ne vous est pas demandé de justifier votre réponse. Les mauvaises réponses sont cependant sanctionnées (pointage négatif) ! Par conséquent, si vous ne connaissez pas la réponse à une question, ne répondez pas !

a) La signature suivante est à recommander pour l'implémentation d'un itérateur sur la liste `MyList`. **(2 pts)**

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

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

Répondez en cochant le tableau suivant :

VRAI	
FAUX	

b) En pire cas, l'algorithme MergeSorte a une complexité O($n \log(n)$) **(2 pts)**.

Répondez en cochant le tableau suivant :

VRAI	
FAUX	

c) Il est toujours possible d'insérer un nouvel élément dans une table de dispersement utilisant une résolution de collision par sondage quadratique dont la taille est un nombre paire et dont le facteur de compression est de 40% **(2 pts)**.

Répondez en cochant le tableau suivant :

VRAI	
FAUX	

d) L'opération de retrait usuelle, telle qu'effectuée sur un arbre binaire de recherche standard, effectuée sur un arbre AVL ne produit pas nécessairement un arbre AVL **(2 pts)**.

Répondez en cochant le tableau suivant :

VRAI	
FAUX	

e) Si la liste `lst` est un `ArrayList`, le code suivant aura une complexité O(*n*). **(2 points)**.

```
public static void retireNombresPaires (List<Integer> lst)
{
   int i = 0;
   while( i < lst.size() )
      if( lst.get( i ) % 2 == 0 )
         lst.remove( i );
      else
         i++;
}
```

Répondez en cochant le tableau suivant :

VRAI	
FAUX	

f) Si la liste `lst` est un `ArrayList`, le code suivant aura une complexité O(*n*). **(2 points)**.

```
public static void retireNombresPaires (List<Integer> lst )
{
   Iterator<Integer> itr = lst.iterator();
   while( itr.hasNext() )
      if( itr.next() % 2 == 0 )
         itr.remove();
}
```

Répondez en cochant le tableau suivant :

VRAI	
FAUX	

g) Un arbre complet de hauteur h=4 possède au plus 31 nœuds. **(2 points)**.

Répondez en cochant le tableau suivant :

VRAI	
FAUX	

h) Un arbre complet de hauteur h=6 possède au moins 63 nœuds. **(2 points)**.

Répondez en cochant le tableau suivant :

VRAI	
FAUX	

Annexe 1

```
public class QuadraticProbingHashTable<AnyType>{

   public QuadraticProbingHashTable(){
      this( DEFAULT_TABLE_SIZE );
   }

   public QuadraticProbingHashTable( int size ){
      allocateArray( size );
      makeEmpty( );
   }

   public void insert( AnyType x ){
      int currentPos = findPos( x );
      if( isActive( currentPos ) ) return;
      array[ currentPos ] = new HashEntry<AnyType>( x, true );
      // Rehash if need be
      if( ++currentSize > array.length / 2 ) rehash( );
   }

   private void rehash( ){
      System.out.println("DEBUG: rehash");
      HashEntry<AnyType> [ ] oldArray = array;

      // Create a new double-sized, empty table
      allocateArray( nextPrime( 2 * oldArray.length ) );
      currentSize = 0;

      // Copy table over
      for( int i = 0; i < oldArray.length; i++ )
      if( oldArray[ i ] != null && oldArray[ i ].isActive )
      insert( oldArray[ i ].element );
   }

   private int findPos( AnyType x ){

      int offset = 1;
      int currentPos = myhash( x );

      while( array[ currentPos ] != null &&
      !array[ currentPos ].element.equals( x ) ){
         currentPos += offset;
         offset += 2;
         if( currentPos >= array.length )
         currentPos -= array.length;
      }

      return currentPos;
   }
```

```
public void remove( AnyType x ){
   int currentPos = findPos( x );
   if( isActive( currentPos ) )
      array[ currentPos ].isActive = false;
   else{
      System.out.println("DEBUG: attempted removing " + x
         + ", stopped at " + currentPos);
   }
}

public boolean contains( AnyType x ){
   int currentPos = findPos( x );
   return isActive( currentPos );
}

private boolean isActive( int currentPos ){
   return array[ currentPos ] != null && array[ currentPos ].isActive;
}

public void makeEmpty( ){
   currentSize = 0;
   for( int i = 0; i < array.length; i++ )
   array[ i ] = null;
}

private int myhash( AnyType x ){
   int hashVal = x.hashCode( );

   hashVal %= array.length;
   if( hashVal < 0 )
   hashVal += array.length;

   return hashVal;
}

private static class HashEntry<AnyType>{
   public AnyType  element;
   public boolean isActive;

   public HashEntry( AnyType e, boolean i ){
      element  = e;
      isActive = i;
   }
}

private static final int DEFAULT_TABLE_SIZE = 13;

private HashEntry<AnyType> [ ] array;
private int currentSize;

@SuppressWarnings("unchecked")
private void allocateArray( int arraySize ){
   array = new HashEntry[ nextPrime( arraySize ) ];
}
```

```
   private static int nextPrime( int n ){
      if( n <= 0 ) n = 3;
      if( n % 2 == 0 ) n++;
      for( ; !isPrime( n ); n += 2 ) ;
      return n;
   }

   private static boolean isPrime( int n ){
      if( n == 2 || n == 3 ) return true;
      if( n == 1 || n % 2 == 0 ) return false;

      for( int i = 3; i * i <= n; i += 2 )
         if( n % i == 0 ) return false;

      return true;
   }

   public static void main( String [ ] args ){
      QuadraticProbingHashTable<Integer> H = new QuadraticProbingHashTable<Integer>(
);
      Integer[] values = {13, 92, 78, 56, 31};

      for( int item : values ) H.insert( item );

      H.remove(69);
      System.out.println("Insert 2"); H.insert(2);
      System.out.println("Insert 3"); H.insert(3);
      System.out.println("Insert 4"); H.insert(4);
   }

}
```

Annexe 2

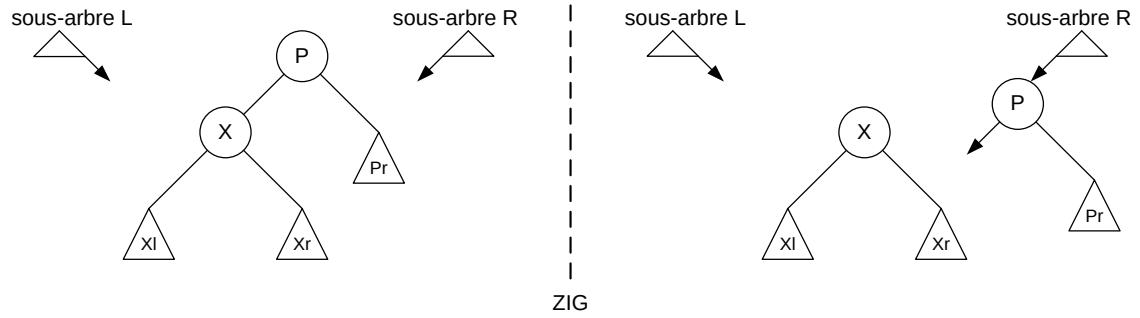
```
public final class Sort{

   public static <AnyType extends Comparable<? super AnyType>>
   void quicksort( AnyType [ ] a ){
      quicksort( a, 0, a.length - 1 );
   }

   private static final int CUTOFF = 5;

   public static <AnyType> void swapReferences( AnyType [ ] a, int index1, int index2
){
      AnyType tmp = a[ index1 ];
      a[ index1 ] = a[ index2 ];
      a[ index2 ] = tmp;
   }

   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 );

      swapReferences( a, center, right - 1 );
      return a[ right - 1 ];
   }

   private static <AnyType extends Comparable<? super AnyType>>
   void quicksort( AnyType [ ] a, int left, int right ){
      System.out.println("DEBUG: call to quicksort");

      if( left + CUTOFF <= right ){
         AnyType pivot = median3( a, left, right );
         System.out.println("DEBUG: Subarray after call to median3");
         printArray(a, left, right);

         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 );

         System.out.println("DEBUG: Subarray after repositionning pivot " + a[i]);
         printArray(a, left, right);
```

```
            quicksort( a, left, i - 1 );
            quicksort( a, i + 1, right );
        }
        else
            insertionSort( a, left, right );
    }

    public static <AnyType extends Comparable<? super AnyType>>
    void printArray(AnyType[] a){
        printArray(a, 0, a.length-1);
    }

    private static <AnyType extends Comparable<? super AnyType>>
    void printArray(AnyType[] a, int left, int right){
        for(int k = left; k<= right; k++ ) System.out.print(a[k] + " ");
        System.out.println();
    }

    private static <AnyType extends Comparable<? super AnyType>>
    void insertionSort( AnyType [ ] a, int left, int right )
    {
        System.out.println("DEBUG: call to insertion sort");
        printArray(a, left, 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;
        }
    }

    public static void main( String [ ] args )
    {
        Integer [ ] a = new Integer[ 16 ];
        for( int i = 0; i < a.length; i++ ){
            int val = ( 2*i*i - 3*i + 15 )% 16 + 1;
            a[ i ] = (val < 0) ? -val : val;
        }

        printArray( a );
        quicksort( a );
        printArray( a );
    }
}
```

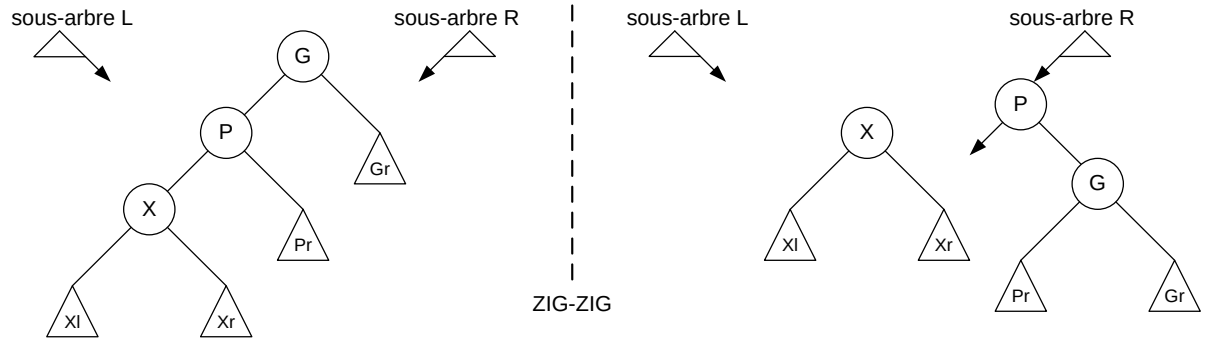
Annexe 3

Transformations TOP/DOWN utilisées dans les arbre Splay

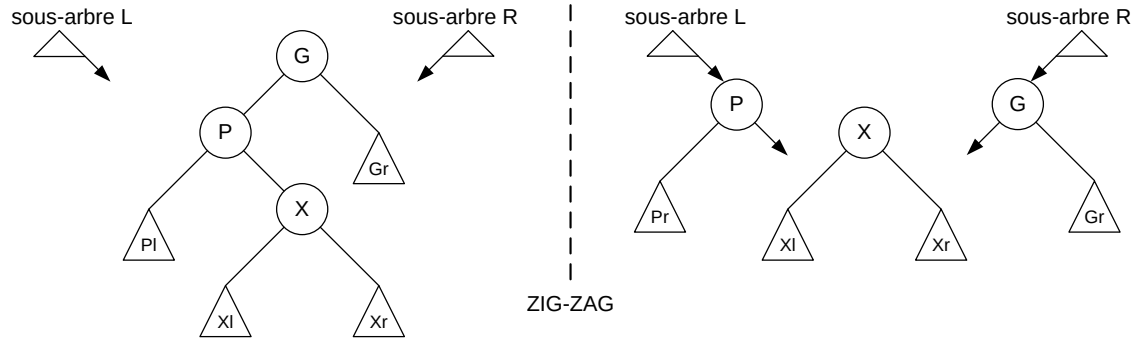
ZIG:



ZIG-ZIG



ZIG-ZAG



FIN: