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**	**20141**
Professeur		***Local***	***Téléphone***
Ettore Merlo, responsable – Tarek Ould Bachir, chargé de cours		**L-1710**	**7128**
Jour	***Date***	***Durée***	***Heure***
Jeudi	**27 février 2013**	**2h00**	**14h00**

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 **6** questions sur un total de **19** 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 (20 points)

Soit une table de dispersement double Hash(x) = (H_1(x) + i H_2(x)) % N où

$x \geq 0$
$H_1(x) = \lfloor x/N \rfloor$: résultat de la division entière de x par N
$H_2(x) = R - (x \% R)$

R et N sont des nombres premiers tels que $0 < R < N$ et N est la taille de la table.

1.1) **(10 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 (on prendra R = 7):

91, 56, 53, 133, 94.

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

Donnez le détail de vos calculs ci-après:

x = 91:
x = 56:
x = 53:
x = 133:
x = 94:

1.2) **(6 points)** Pour chacun des cas énumérés ci-après, discutez brièvement, mais de manière argumentée, la qualité d'uniformité de dispersion de la fonction de hachage. Pour chaque cas, on supposera x uniformément réparti sur l'intervalle de référence.

a) $0 \leq x < N$

b) $0 \leq x < N(N-1)/2$

c) $0 \leq x < N^2$

d) $0 \leq x < N^3$

e) $0 \leq x < 2^{31}$, $N = 3^{10}$

f) $0 \leq x < 2^{31}$, $N = 3^9$

1.3) **(4 points)** De façon générale, que peut-on dire du choix de la fonction $H_1(x)$ comme fonction de dispersement primaire de ce hachage double ?

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

On désire étudier **la version de l'algorithme *Quick Sort* donnée à l'Annexe 1** pour trier le vecteur ci-après. On considère une valeur *cut-off* de 8. Attention, cette version diffère quelque peu de celle vue en cours. Notamment: Elle fait appel à *Merge Sort* plutôt qu'à *Insertion Sort* pour trier les petits vecteurs; Elle n'utilise pas *median3* pour déterminer le pivot.

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

2.1) (**2 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) **(2 points)** Au total, quel est le nombre de fois que la fonction récursive `quicksort` aura été appelée pour exécuter le tri ? Pour éviter tout 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, AnyType tmpArray, int left, int right )
```

Votre réponse: _____

2.3) **(2 points)** Au total, quel est le nombre de fois que la fonction récursive `qsmergesort` aura été appelée pour exécuter le tri ? Pour éviter tout ambiguïté, la signature de la fonction est reproduite ci-après.

Signature de la fonction considérée :

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

Votre réponse: _____

2.4) **(2 points)** Quelle est la plus petite taille de vecteur sur laquelle la fonction récursive `qsmergesort` aura été appelée pour exécuter le tri ?

Votre réponse: _____

2.5) **(2 points)** Quelle est la plus grande taille de vecteur sur laquelle la fonction récursive `qsmergesort` aura été appelée pour exécuter le tri ?

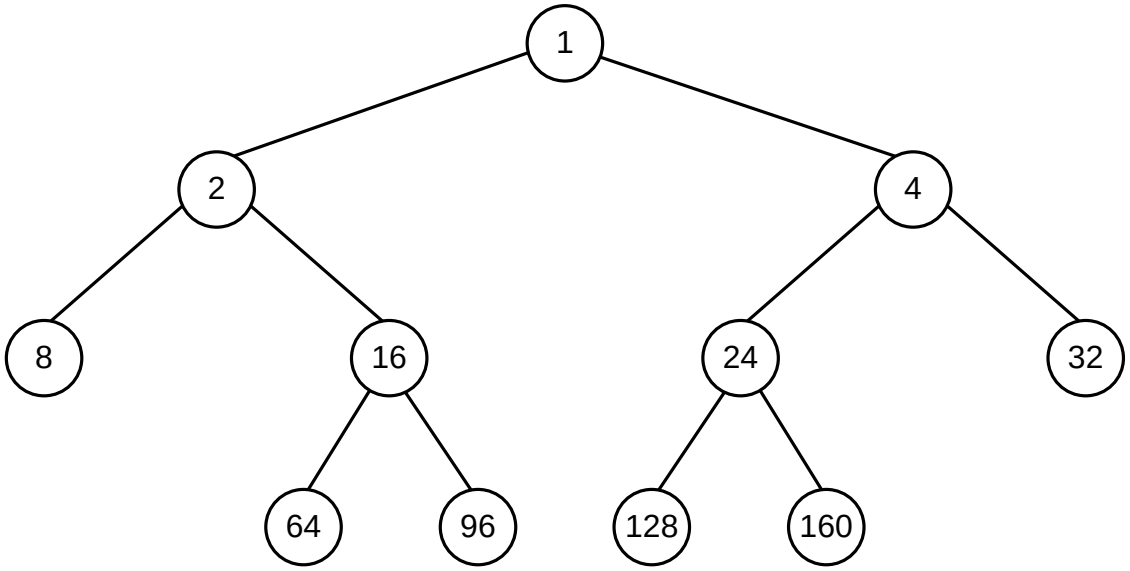
Votre réponse: _____

2.6) (**5 points**) Reproduisez le vecteur correspondant à la question (2.5) 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 (12 points)

Considérez l'arbre binaire suivant:



3.1) (**3 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) (**3 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) (**3 points**) Donner le résultat de l'affichage de l'arbre binaire si il est parcouru en en ordre. Séparez les éléments par des virgules.

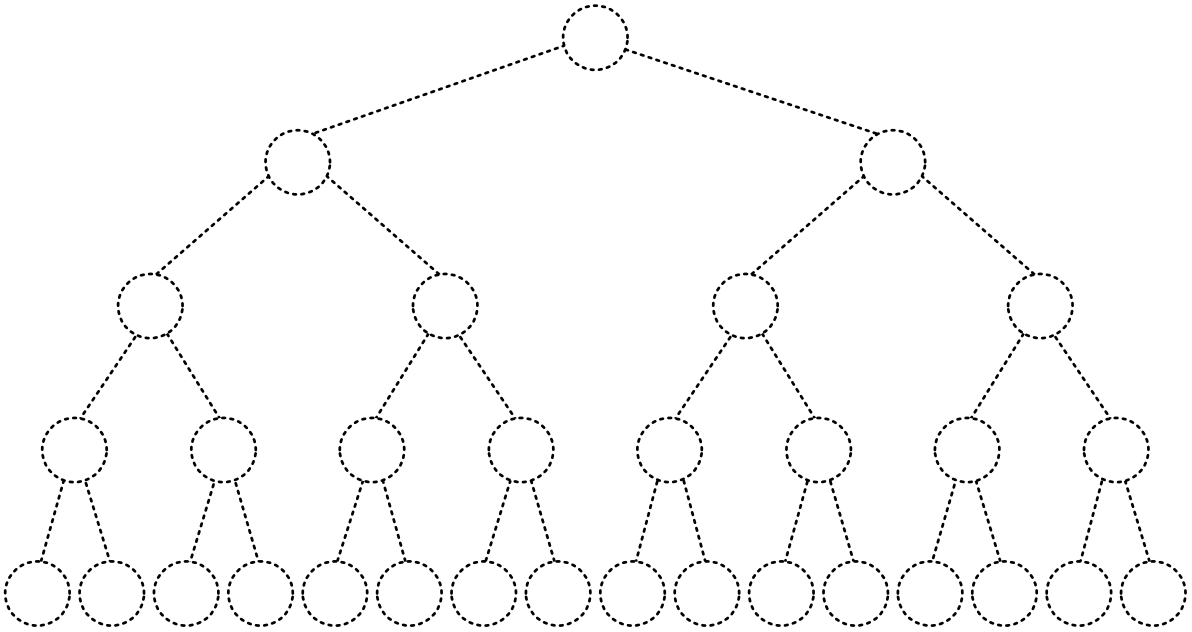
3.4) (**3 points**) Donner le résultat de l'affichage de l'arbre binaire si il est parcouru par niveaux. Séparez les éléments par des virgules.

Question 4 : Arbres binaire de recherche (12 points)

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

59, 28, 45, 38, 48, 35, 41, 47, 58

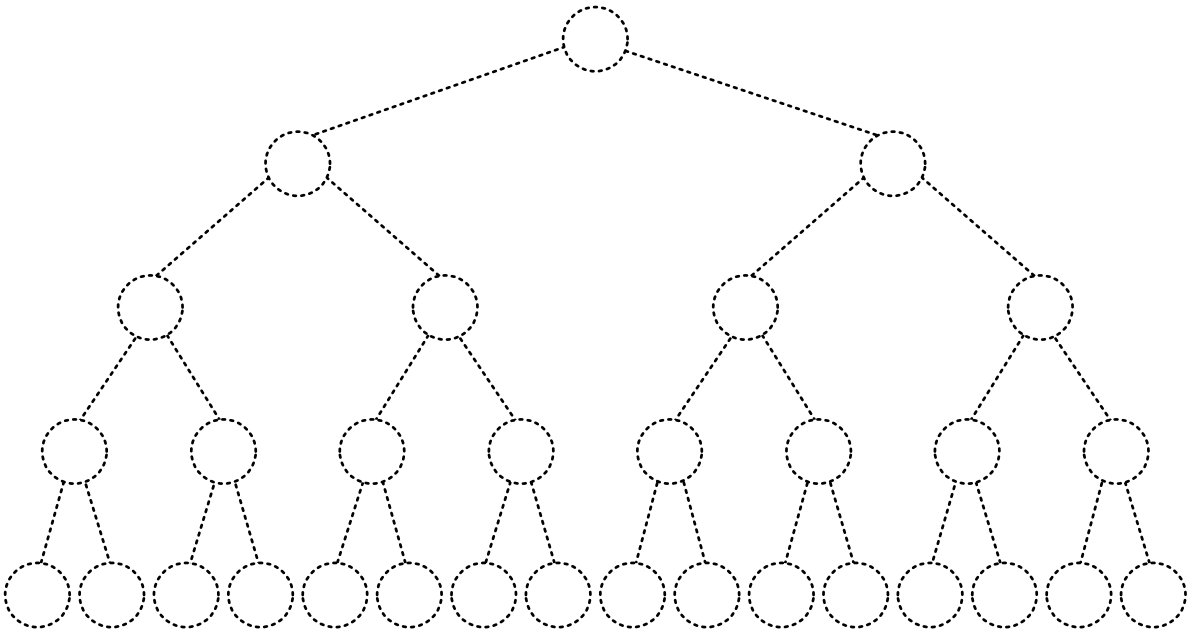
Donnez la représentation graphique de l'arbre.



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

59, 49, 29, 15, 35, 54, 78, 62, 71

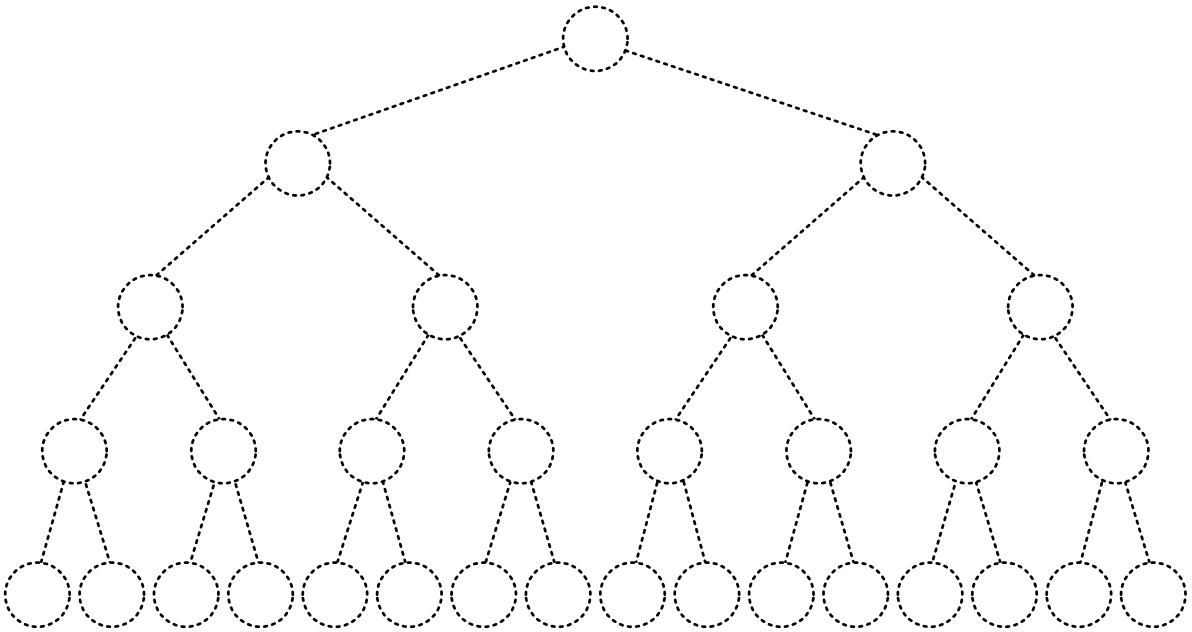
Donnez la représentation graphique de l'arbre.



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

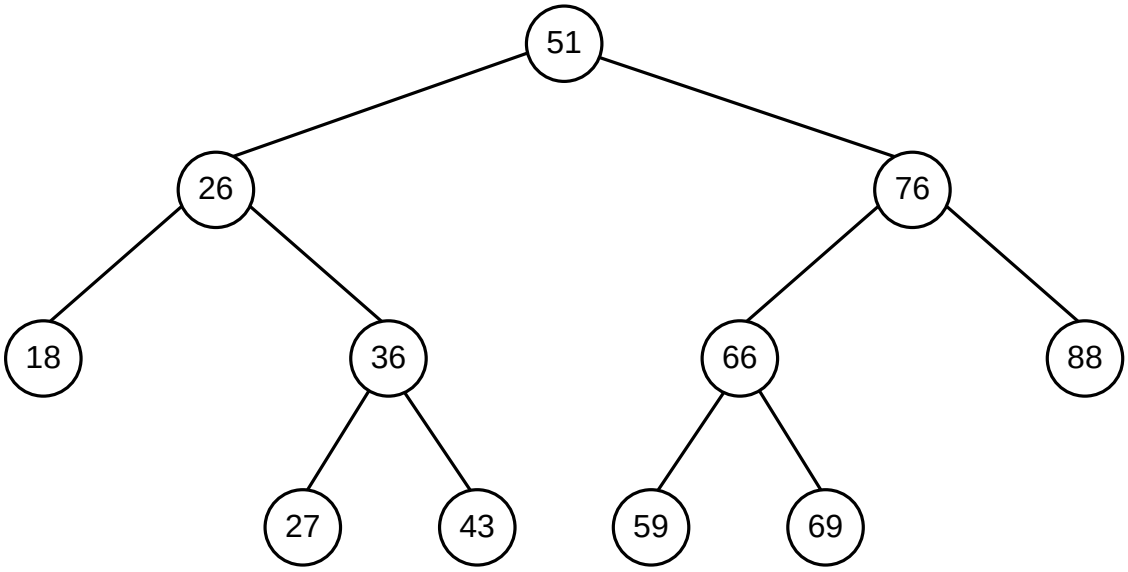
35, 41, 38, 47, 58, 48, 45, 28, 71, 62, 78, 59

Donnez la représentation graphique de l'arbre.

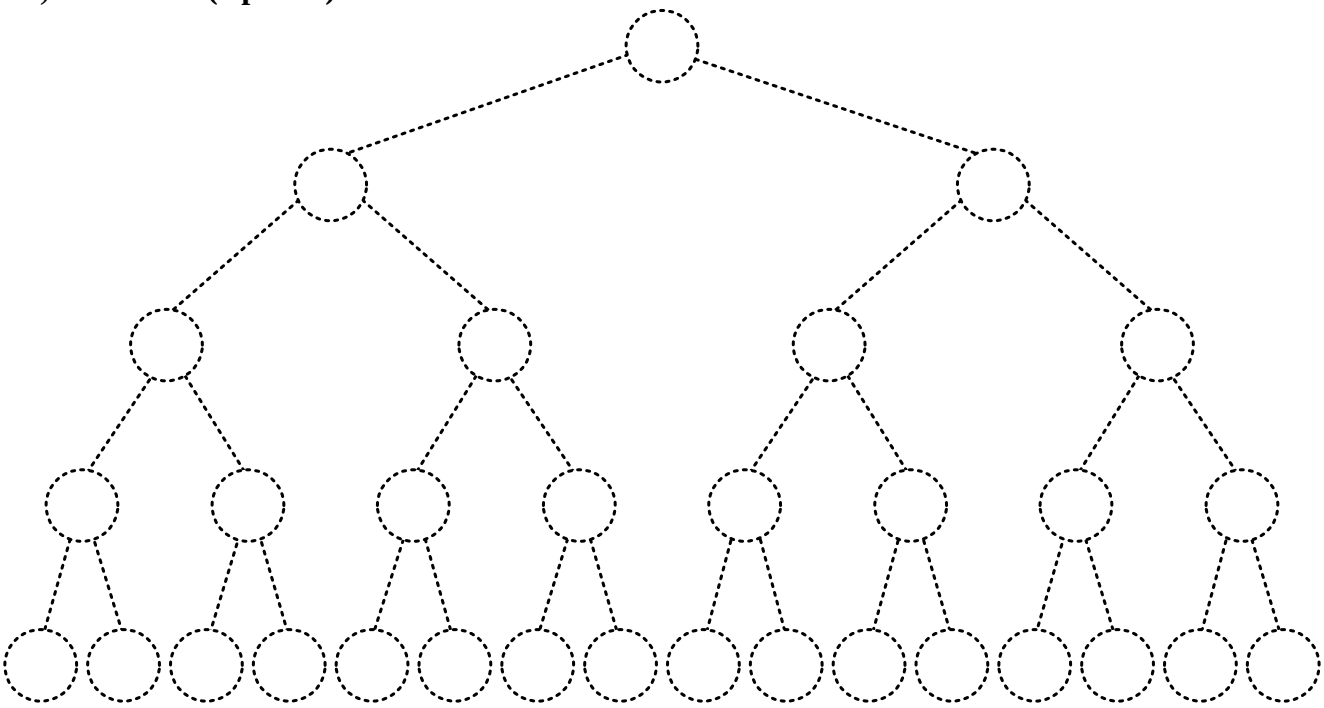


Question 5 : Arbre binaire de recherche de type AVL (15 points)

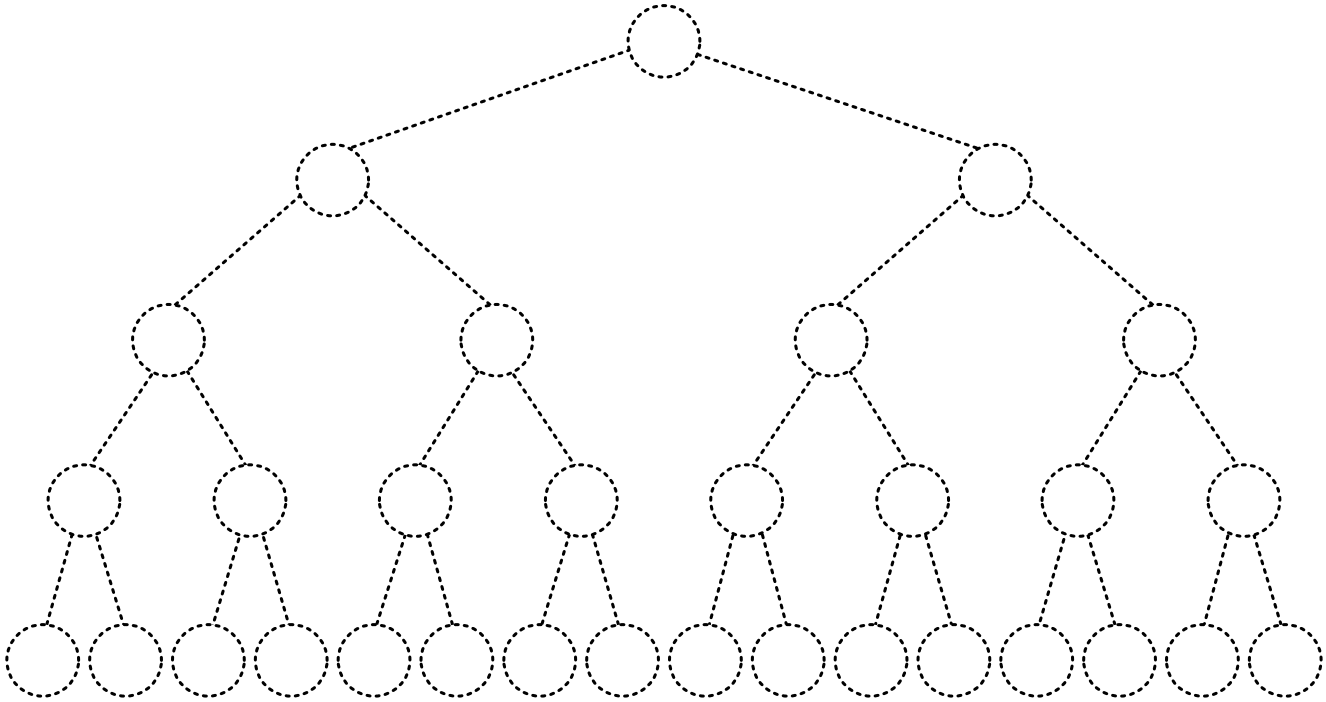
En partant de l'arbre AVL suivant :



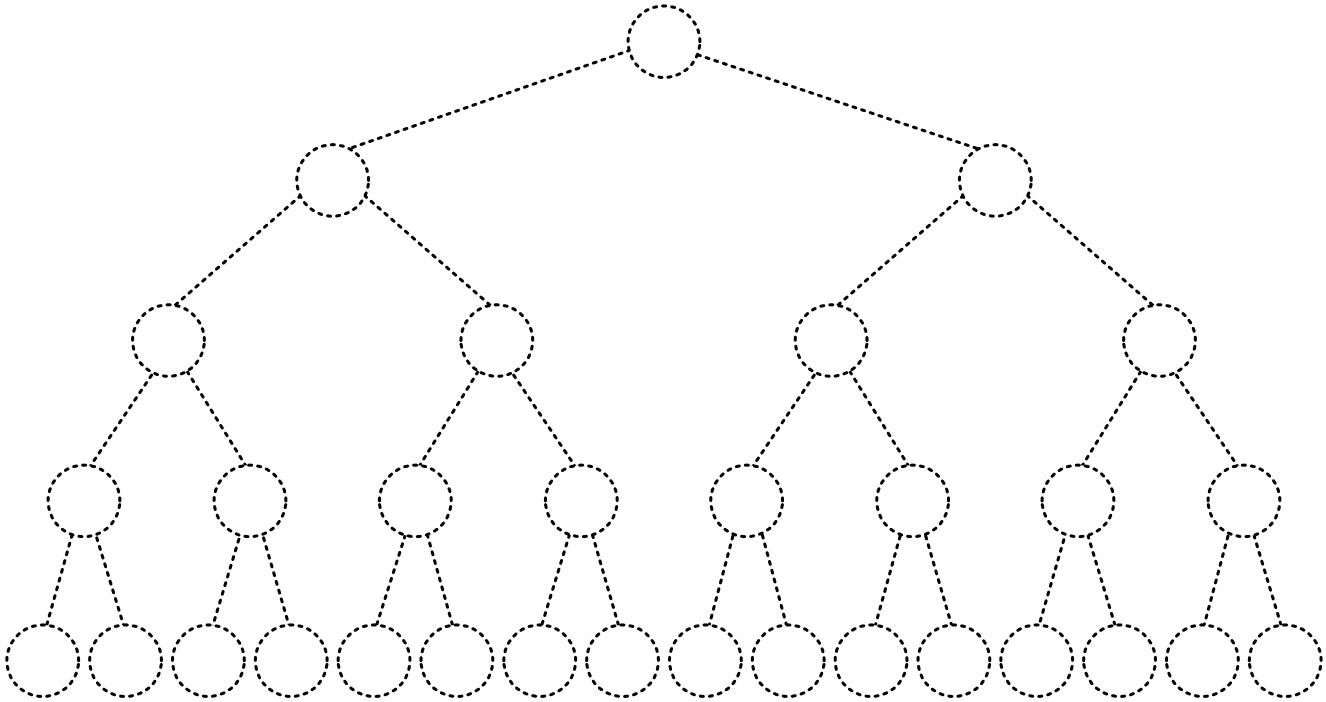
5.1) Insérez 21. **(3 points)**



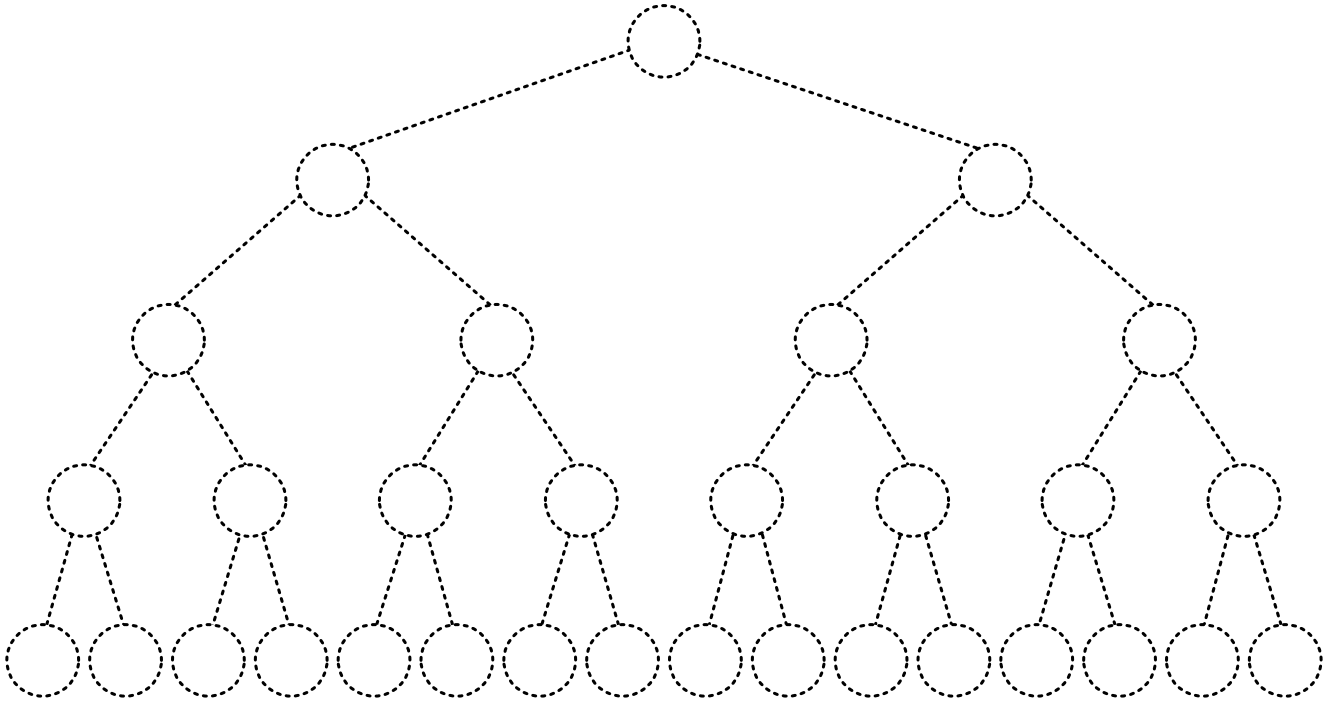
5.2) En partant du résultat de la question 5.1), insérez 48. **(3 points)**



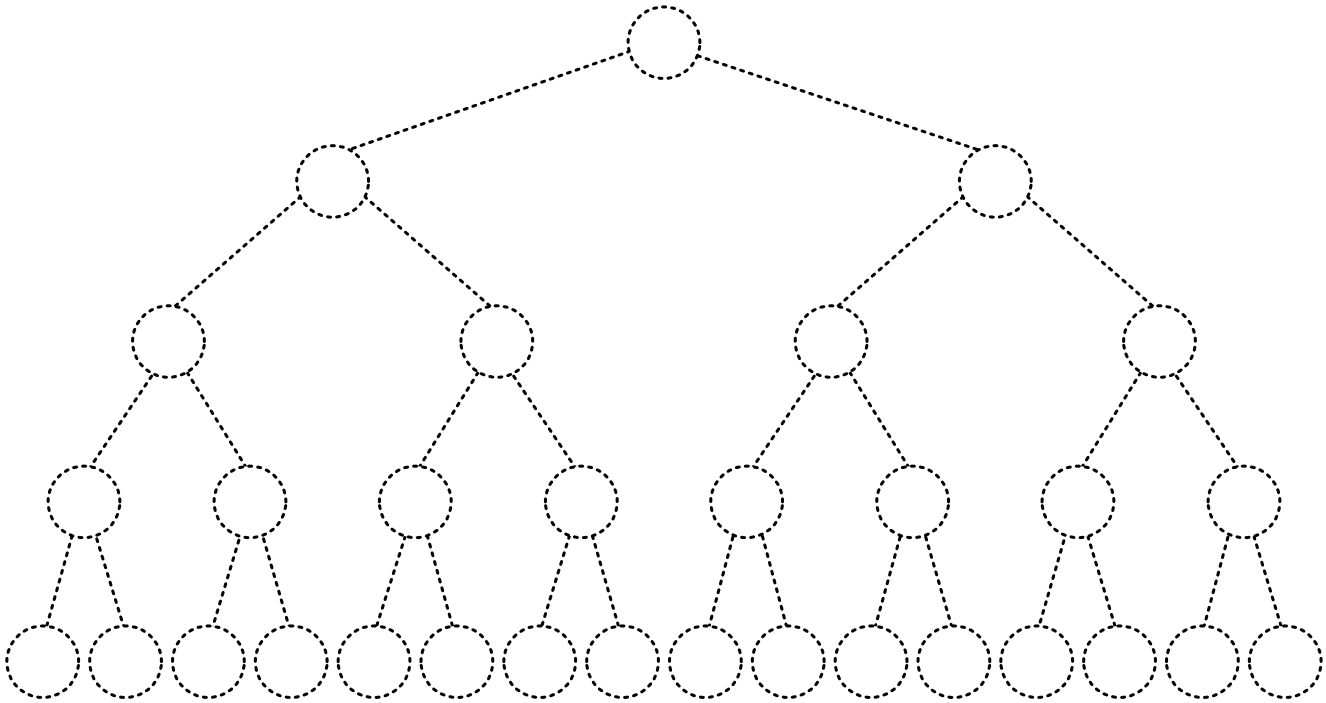
5.3) En partant du résultat de la question 5.2), insérez 31. **(3 points)**



5.4) En partant du résultat de la question 5.3), insérez 34. **(3 points)**

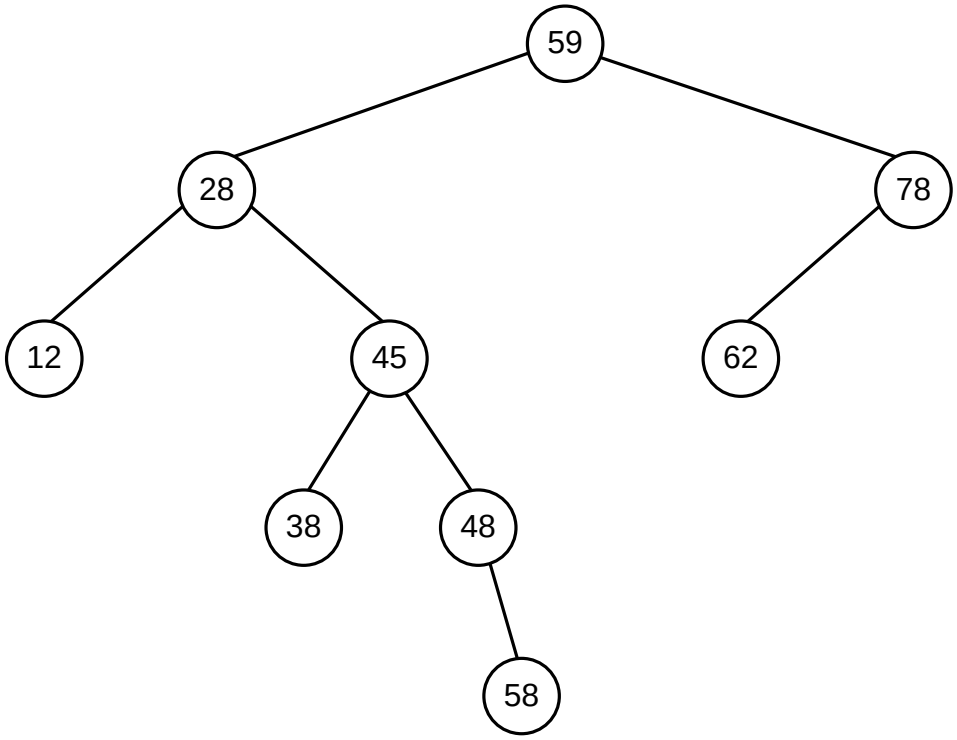


5.5) En partant du résultat de la question 5.4), insérez 35. **(3 points)**

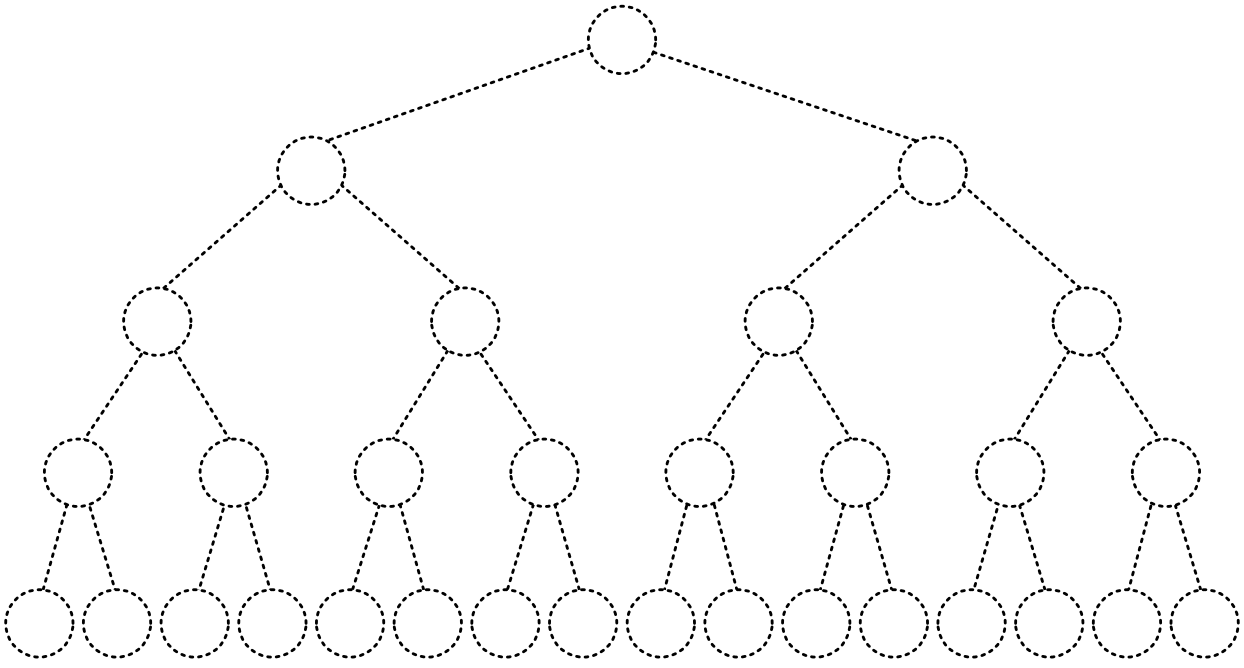


Question 6 : Arbre binaire de recherche de type Splay (21 points)

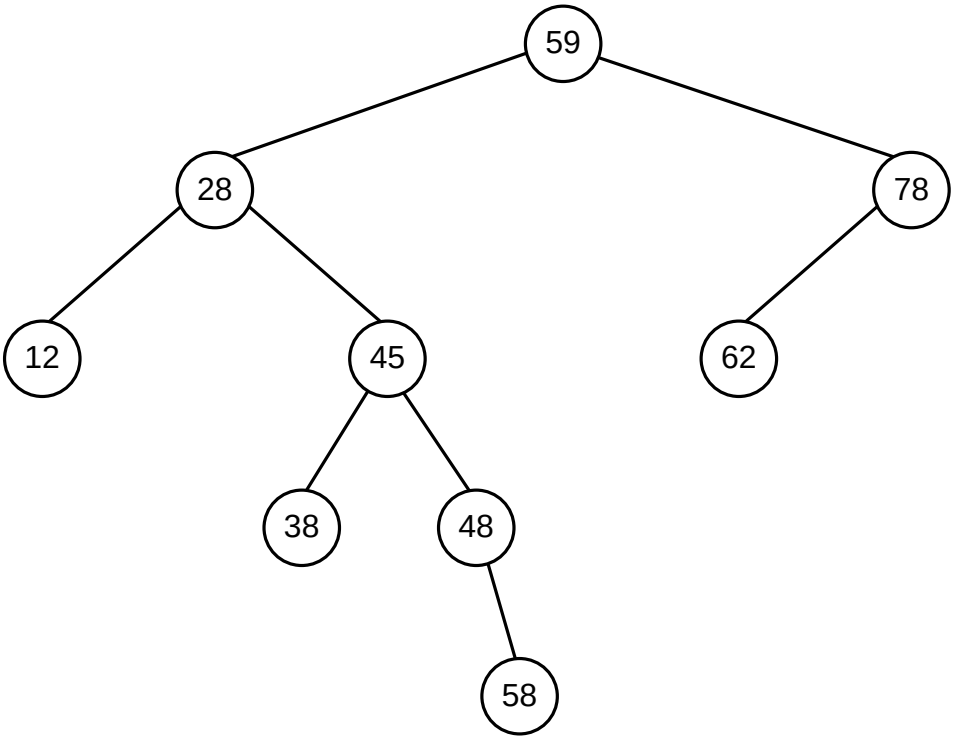
En partant de l'arbre Splay suivant :



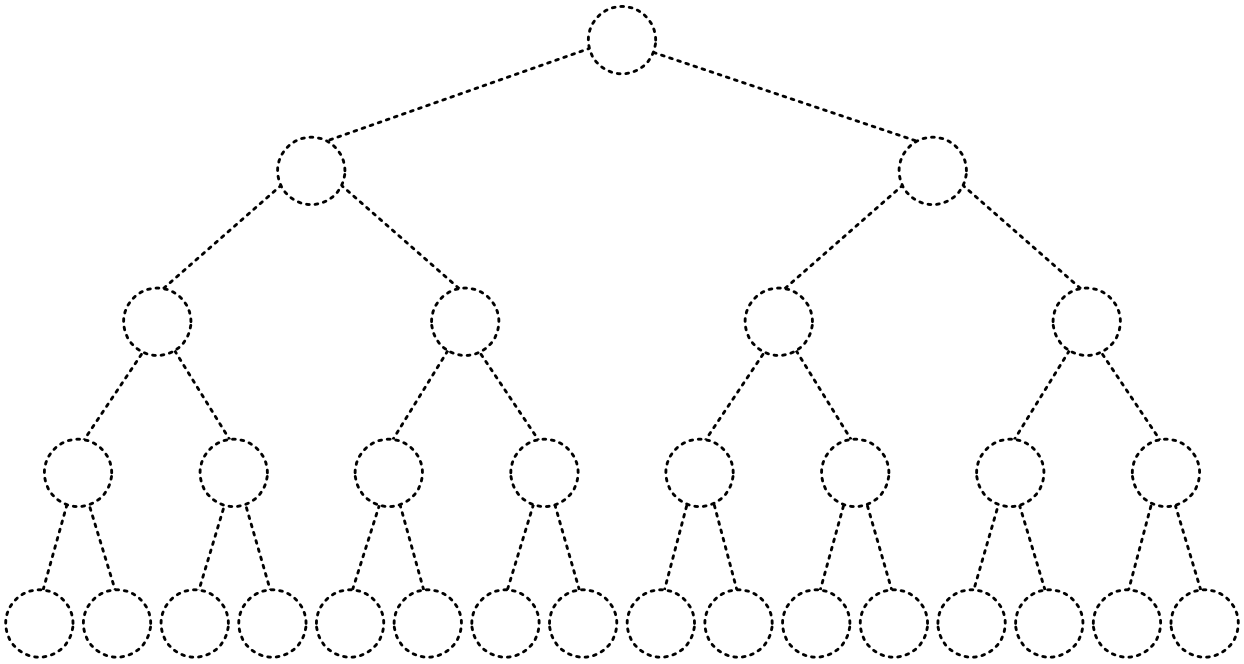
6.1) **(4 points)** Effectuez un `get(48)`.



En repartant du même arbre Splay :

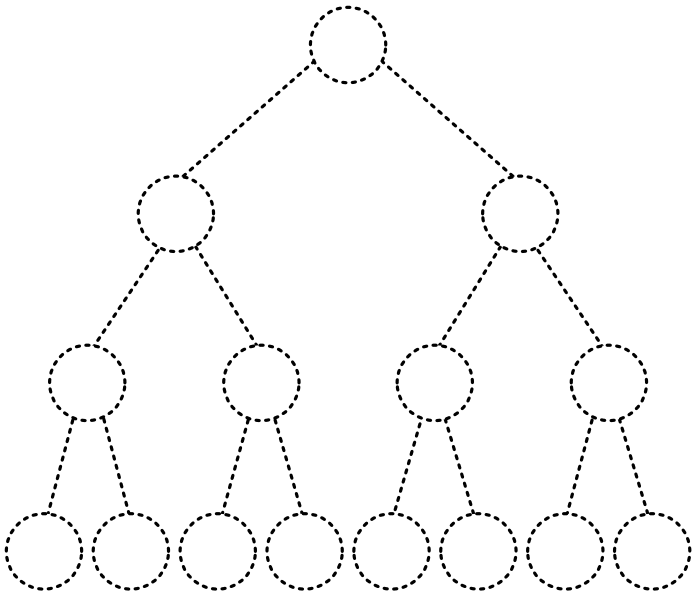


6.2) **(5 points)** Effectuez un `delete(45)`.

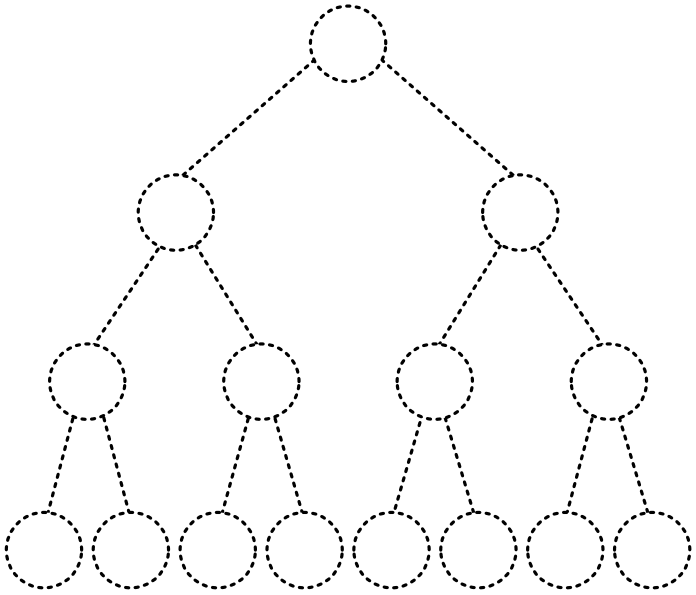


6.3) **(6 points)** Si le `get(48)` de la question 6.1) avait été exécuté par une implémentation de type top-down, quels auraient été les sous-arbres R et L juste avant que 48 ne soit placé à la racine ? **Aidez-vous des données de l'Annexe 2.**

Sous-arbre L:

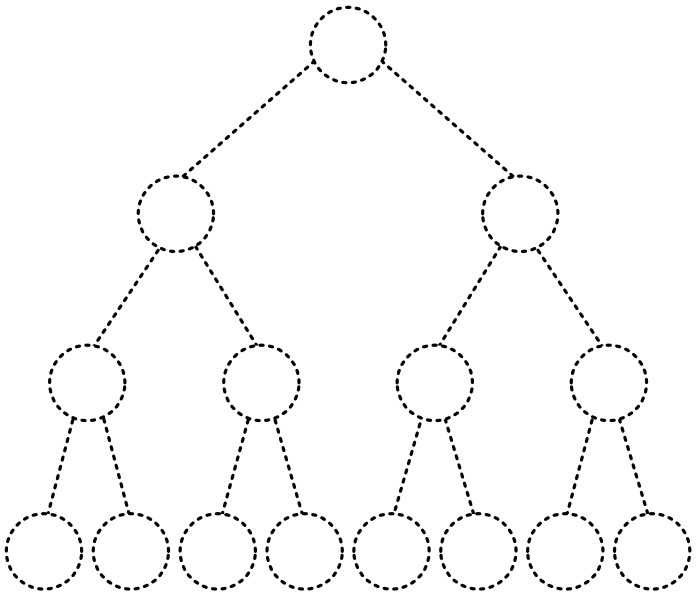


Sous-arbre R:

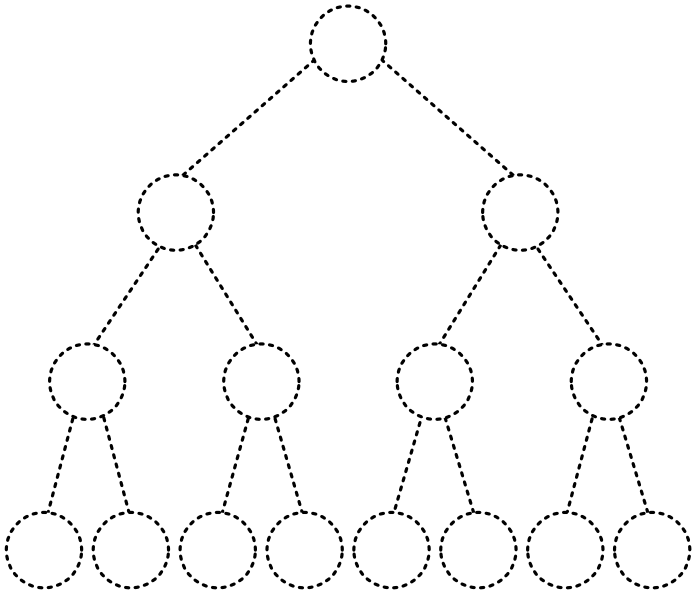


6.4) **(6 points)** Si le `delete(45)` de la question 6.2) avait été exécuté par une implémentation de type top-down, quels auraient été les sous-arbres R et L juste avant que 45 ne soit placé à la racine ? **Aidez-vous des données de l'Annexe 2.**

Sous-arbre L:



Sous-arbre R:



Annexe 1

```
public final class IntraSort
{
   /**
   * Appel interne à mergesort depuis QuickSort
   */
   private static <AnyType extends Comparable<? super AnyType>>
   void qsmergesort( AnyType [ ] a, AnyType [ ] tmpArray,
                     int left, int right )
   {
      if( left < right )
      {
         int center = ( left + right ) / 2;
         qsmergesort( a, tmpArray, left, center );
         qsmergesort( a, tmpArray, center + 1, right );
         qsmerge( a, tmpArray, left, center + 1, right );
      }
   }

   /**
   * Fusionne deux vecteurs
   */
   private static <AnyType extends Comparable<? super AnyType>>
   void qsmerge( AnyType [ ] a, AnyType [ ] tmpArray,
                 int leftPos, int rightPos, int rightEnd )
   {
      int leftEnd = rightPos - 1;
      int tmpPos = 0;
      int numElements = rightEnd - leftPos + 1;

      // Main loop
      while( leftPos <= leftEnd && rightPos <= rightEnd )
         if( a[ leftPos ].compareTo( a[ rightPos ] ) <= 0 )
            tmpArray[ tmpPos++ ] = a[ leftPos++ ];
         else
            tmpArray[ tmpPos++ ] = a[ rightPos++ ];

      while( leftPos <= leftEnd )    // Copy rest of first half
         tmpArray[ tmpPos++ ] = a[ leftPos++ ];

      while( rightPos <= rightEnd )  // Copy rest of right half
         tmpArray[ tmpPos++ ] = a[ rightPos++ ];

      // Copy tmpArray back
      for( int i = 0; i < numElements; i++, rightEnd-- )
         a[ rightEnd ] = tmpArray[ numElements-i-1 ];
   }

   private static final int CUTOFF = 8;

   /**
   * Quicksort
   */
```

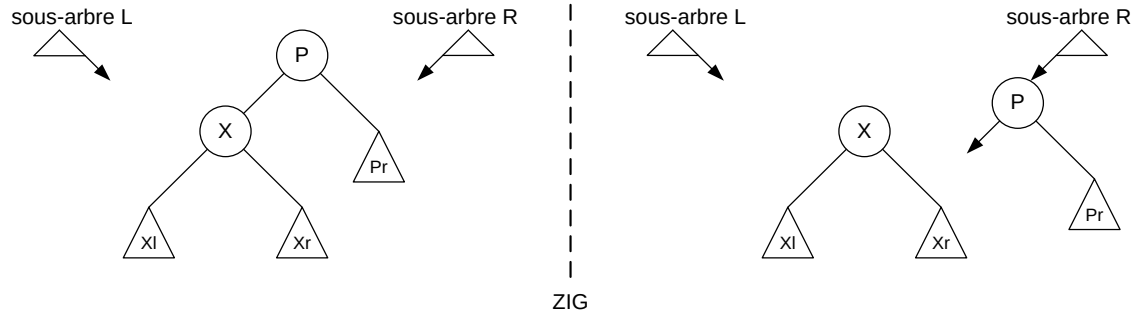
```
public static <AnyType extends Comparable<? super AnyType>>
void quicksort( AnyType [ ] a )
{
   AnyType [ ] tmpArray = (AnyType[]) new Comparable[ CUTOFF ];
   quicksort( a, tmpArray, 0, a.length - 1 );
}

/**
* Appel interne à quicksort, utilise élément central comme pivot,
* une valeur cutoff de 8, ainsi que MergeSort pour les petits vecteurs
*/
private static <AnyType extends Comparable<? super AnyType>>
void quicksort( AnyType [ ] a, AnyType [ ] tmpArray, int left, int right )
{
   if( left + CUTOFF <= right )
   {
      int center = ( left + right ) / 2;

      AnyType pivot = a[center];
      swapReferences( a, center, right );

      // partitionnement
      int i = left-1, j = right;
      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 );
      // fin du partitionnement

      // recursion
      quicksort( a, tmpArray, left, i - 1 );
      quicksort( a, tmpArray, i + 1, right );
   }
   else
      qsmergesort( a, tmpArray, left, right );
}

/**
* Interchange (swap) deux valeurs
*/
public static <AnyType> void
swapReferences( AnyType [ ] a, int index1, int index2 )
{
   AnyType tmp = a[ index1 ];
   a[ index1 ] = a[ index2 ];
   a[ index2 ] = tmp;
}
```

```
   public static void main( String [ ] args )
   {
      Integer [ ] a = {16, 14, 12, 10,  2, 6, 4, 8,
                       7, 13, 11,  9, 15, 5, 3, 1};

      quicksort( a );

      for(Integer valeur : a)
         System.out.print(valeur + " ");

   }
}
```

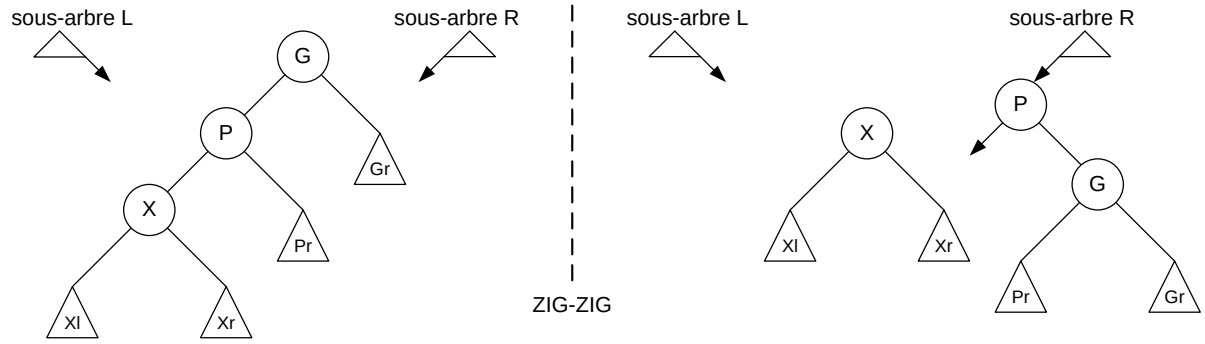
Annexe 2

Transformations TOP/DOWN utilisées dans les arbre Splay

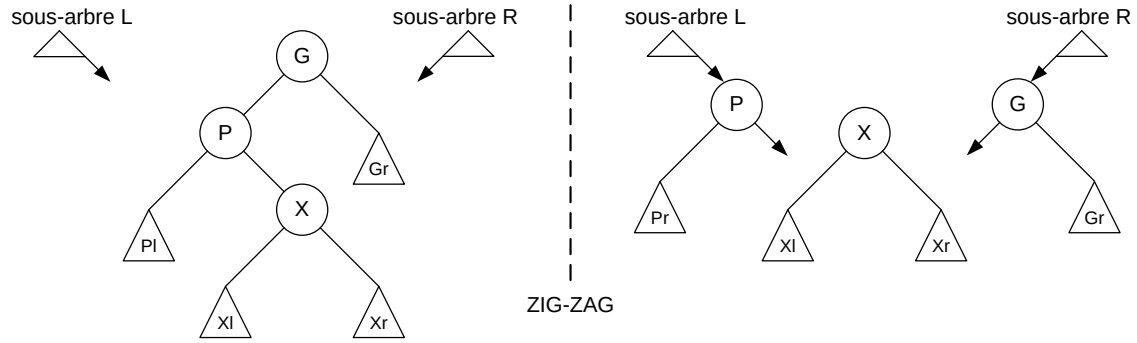
ZIG:



ZIG-ZIG



ZIG-ZAG



FIN: