POLYTECHNIQUE MONTRÉAL

Questionnaire examen intra

INF2010

Sigle du cours

Q1	
Q2	
Q3	
Q4	
Q5	
Q6	
Q7	
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**	**20171**
Professeur			***Local***	***Téléphone***
Ettore Merlo, responsable – Tarek Ould Bachir, chargé de cours			**M-1020**	
Jour	***Date***	***Durée***		***Heure***
Lundi	**20 février 2017**	**2h00**		**10h00**

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 **18** 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 (14 points)

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

1.1) **(8 points)** En vous servant du tableau ci-dessous, donnez l'état de la mémoire d'une table de taille N=12 après l'insertion, dans l'ordre, des clés suivantes:

26, 61, 41, 37.

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

Donnez le détail de vos calculs ci-après:

x = 26:
x = 61:
x = 41:
x = 37:

1.2) **(3 points)** Sachant que $(i + 6)^2 \% 12 = i^2 \% 12$, expliquez ce qui se passe lorsqu'on essaie d'insérer la clé 13 à la table obtenue à la question 1.1). Soyez bref et clair dans vos explications.

1.3) **(3 points)** Quelle est la plus grande valeur que prendra *i* dans l'exécution de l'appel de la question 1.2 ?

Question 2 : Tris en $n \log(n)$ (16 points)

On désire exécuter l'algorithme *Quick Sort* pour trier le vecteur ci-après. On considère une valeur *cut-off* de 3. Le code source vous est fourni à l'Annexe 1.

Indices	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Valeurs	5	1	2	2	3	2	1	5	6

2.1) (**2 points**) Donnez l'état du vecteur après la mise à l'écart du pivot lors de la première récursion de QuickSort :

Indices	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Valeurs									

2.2) (**3 points**) Donnez l'état du vecteur après l'exécution du partitionnement de la première récursion :

Indices	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Valeurs									

2.3) **(3 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 grande 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 au moment d'entrer dans `insertionSort` (entre les indices `left` et `right` inclusivement) :

Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Valeurs									

2.6) (**2 points**) Quelles sont les valeurs de `left` et `right` lors du dernier appel à la fonction récursive `quicksort`, c'est-à-dire celle dont la signature est

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

Votre réponse :

`left`: _____, `right`: _____

2.7) (**2 points**) Quelles sont les valeurs de `left` et `right` lors du 3e appel à la fonction récursive `quicksort`, c'est-à-dire celle dont la signature est

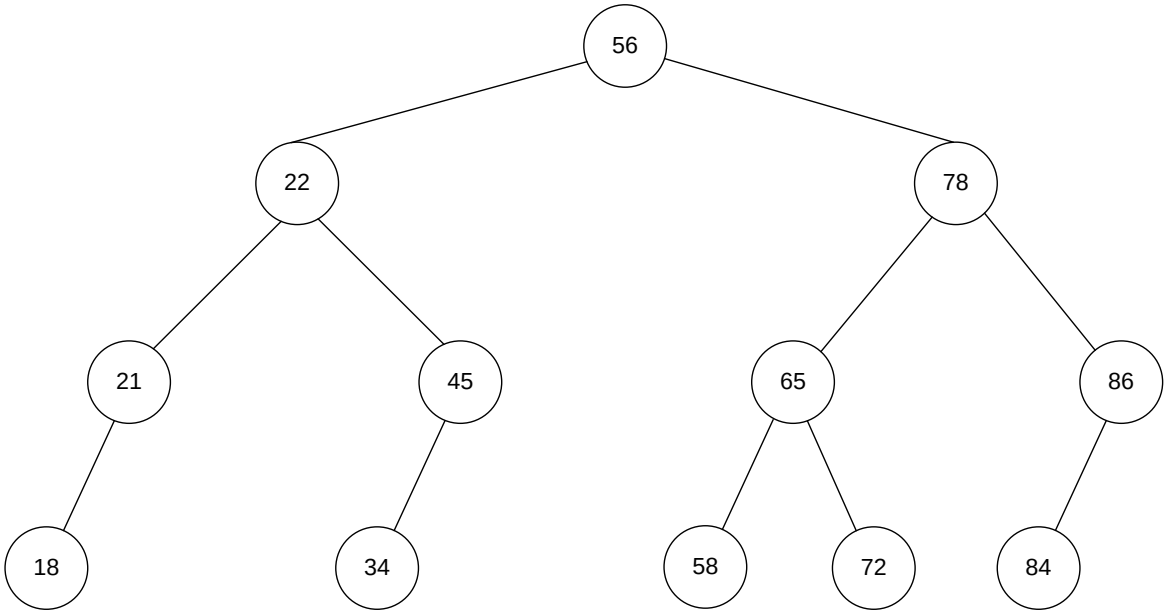
```
private static <AnyType extends Comparable<? super AnyType>>
void quicksort( AnyType [ ] a, int left, int right )
```

Votre réponse :

`left`: _____, `right`: _____

Question 3 : Parcours d'arbres (10 points)

Considérez l'arbre binaire suivant:



3.1) (**2.5 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) (**2.5 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) (**2.5 points**) Donner le résultat de l'affichage de l'arbre binaire si il est parcouru en ordre. Séparez les éléments par des virgules.

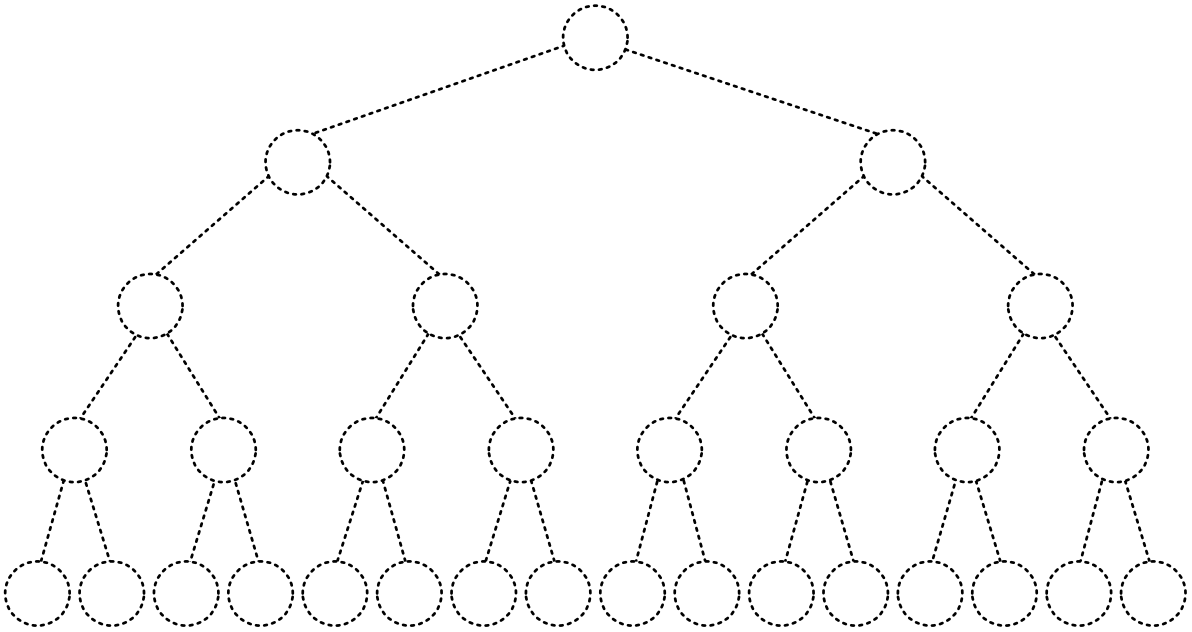
3.4) (**2.5 points**) Donner le résultat de l'affichage de l'arbre binaire si il est parcouru par niveaux. Séparez les éléments par des virgules.

Question 4 : Arbres binaire de recherche (14 points)

4.1) **(3 points)** Si l'affichage par niveaux de l'arbre binaire de recherche donne :

24, 18, 36, 12, 30, 48, 42

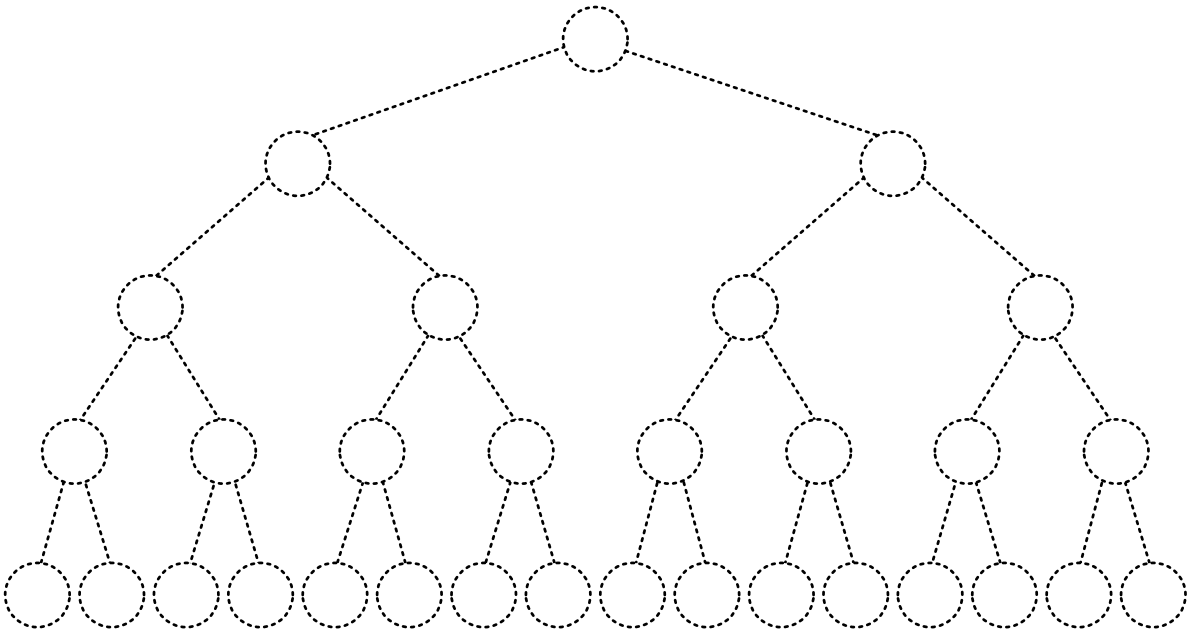
Donnez la représentation graphique de l'arbre.



4.2) **(3 points)** Si l'affichage pré-ordre de l'arbre binaire de recherche donne :

42, 21, 14, 35, 28, 56, 49

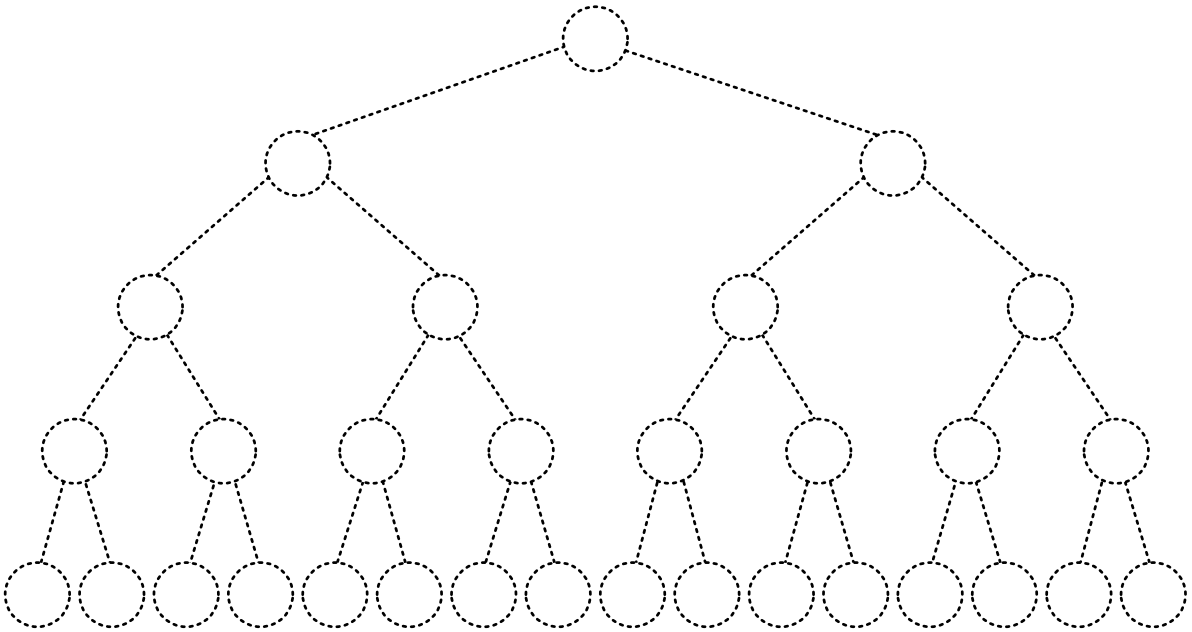
Donnez la représentation graphique de l'arbre.



4.3) **(3 points)** Si l'affichage post-ordre de l'arbre binaire de recherche donne :

27, 18, 45, 72, 63, 54, 36

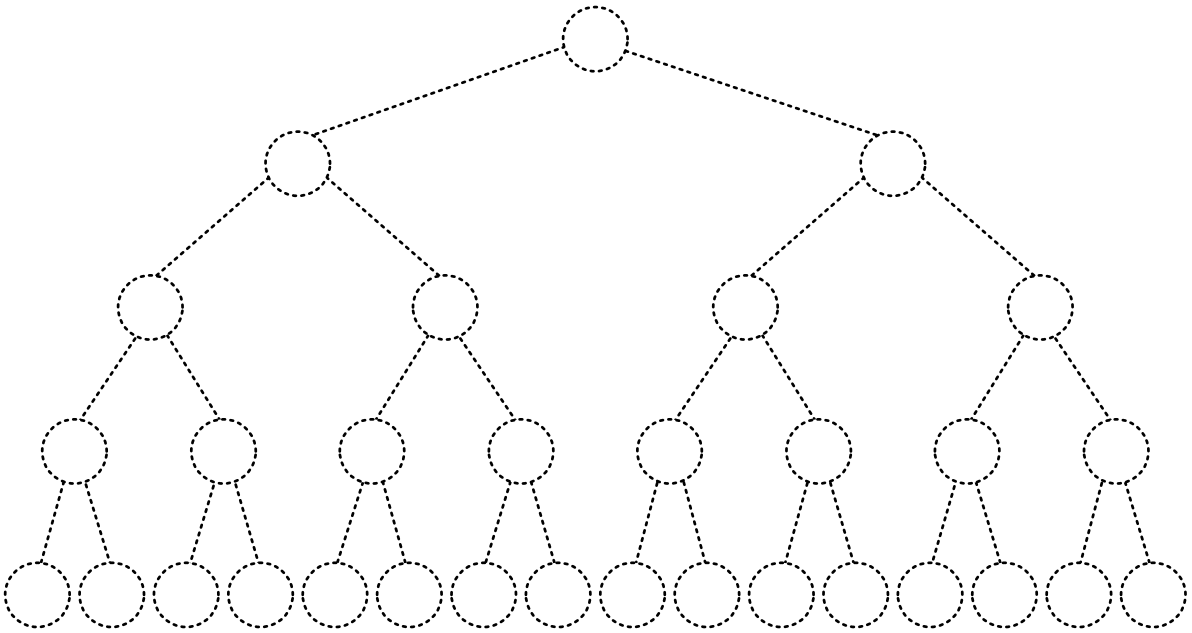
Donnez la représentation graphique de l'arbre.



4.4) **(5 points)** Si l'affichage en-ordre de l'arbre binaire de recherche donne :

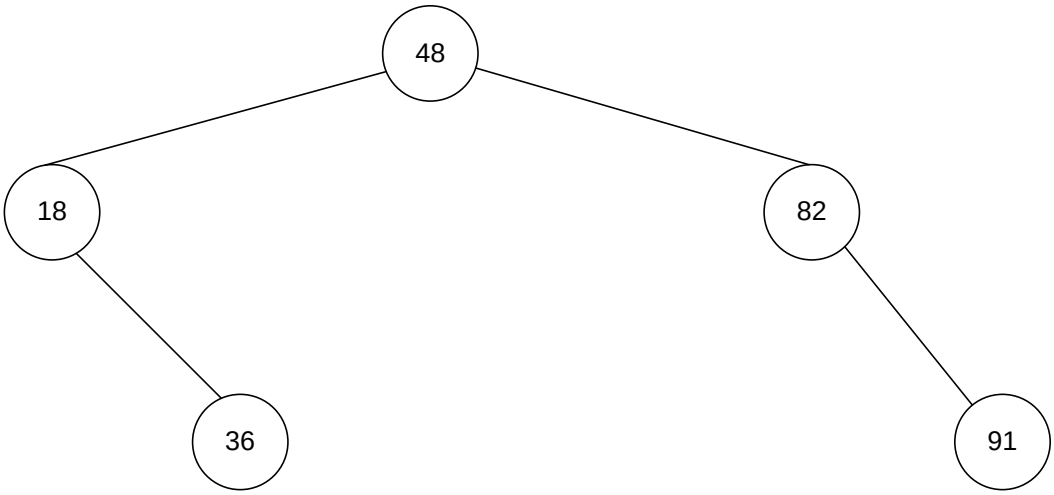
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

Donnez la représentation graphique de l'arbre, sachant que les sous-arbres à la gauche et à la droite de la racine sont des AVL de hauteur h = 2.



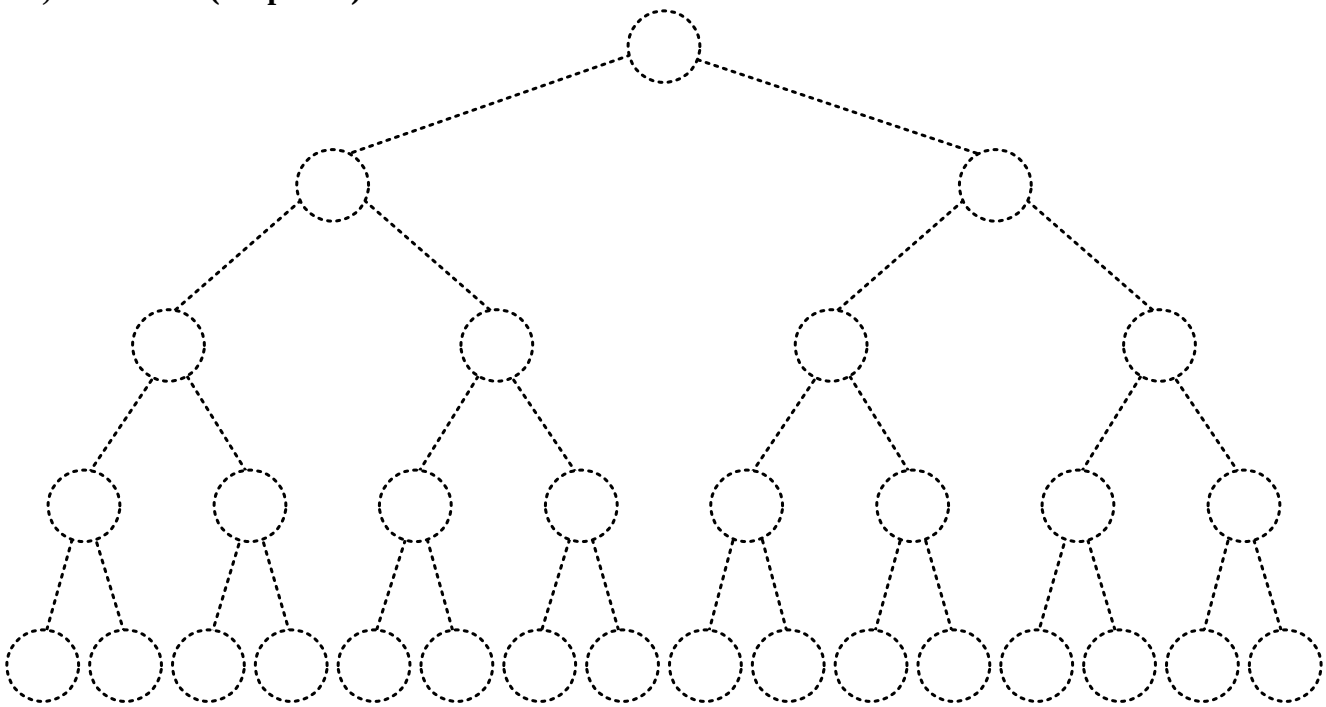
Question 5 : Arbre binaire de recherche de type AVL (23 points)

En partant de l'arbre AVL suivant :

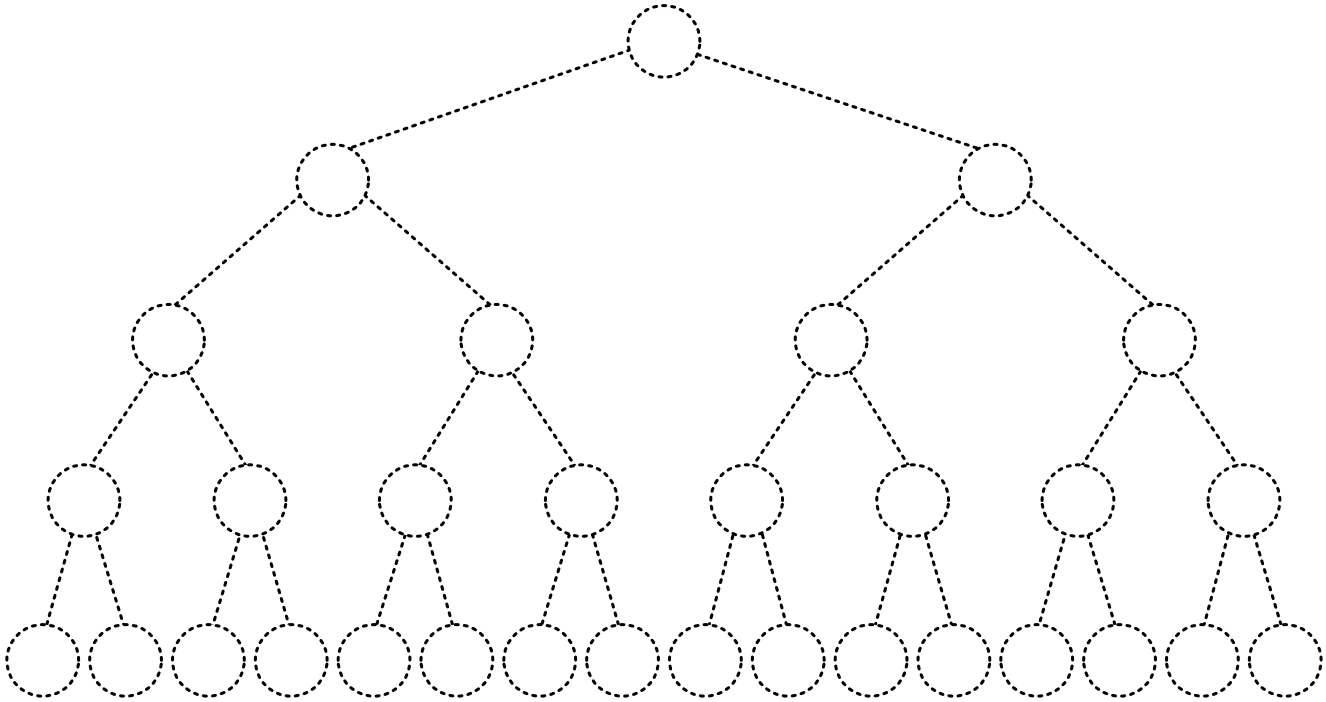


Insérez dans l'ordre les clés suivantes : 42, 90

5.1) Insérez 42. **(2.5 points)**

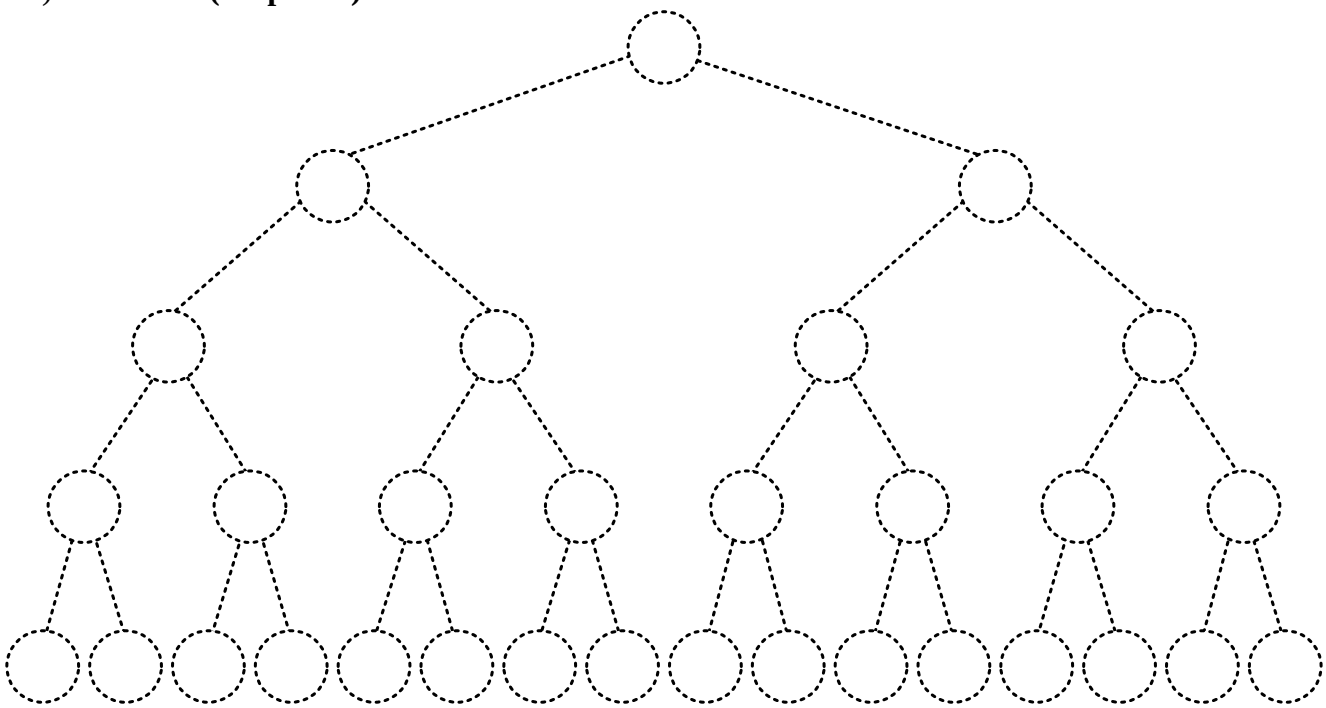


5.2) Insérez 90. **(2.5 points)**

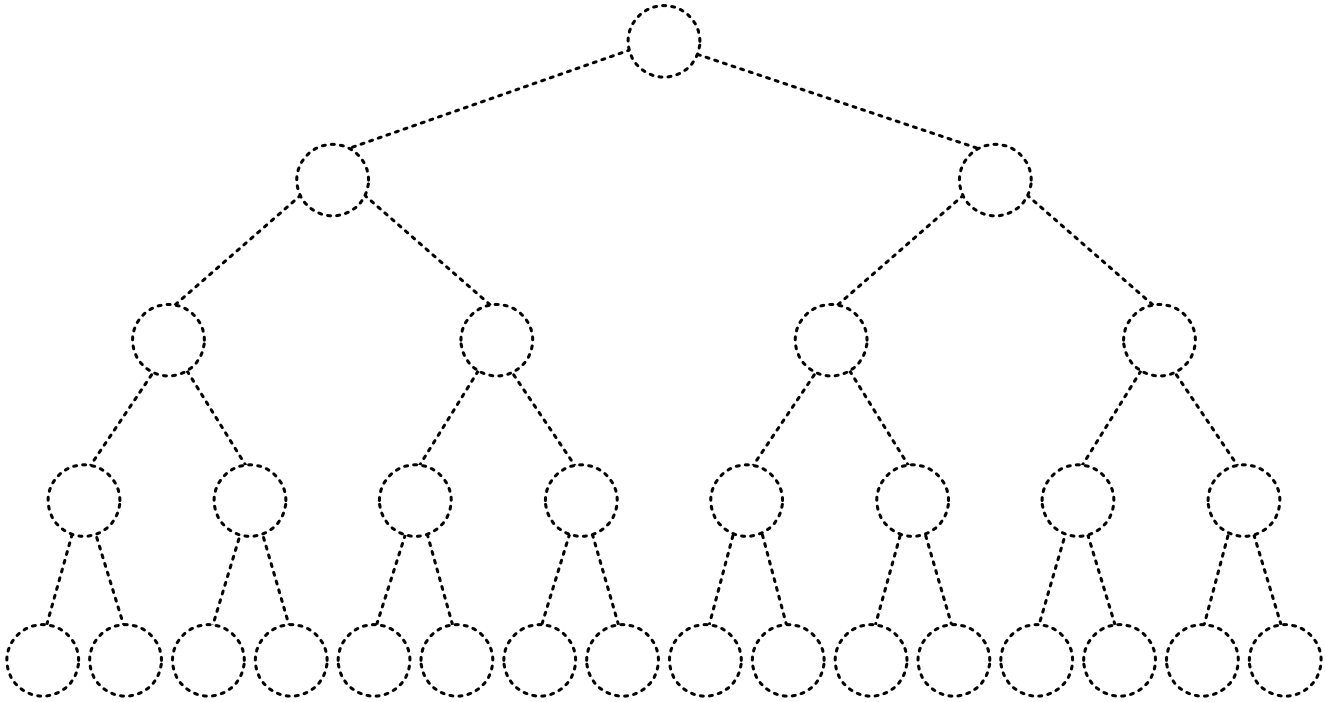


En partant de l'AVL obtenu en 5.2), insérez dans l'ordre 83, 84, et 85.

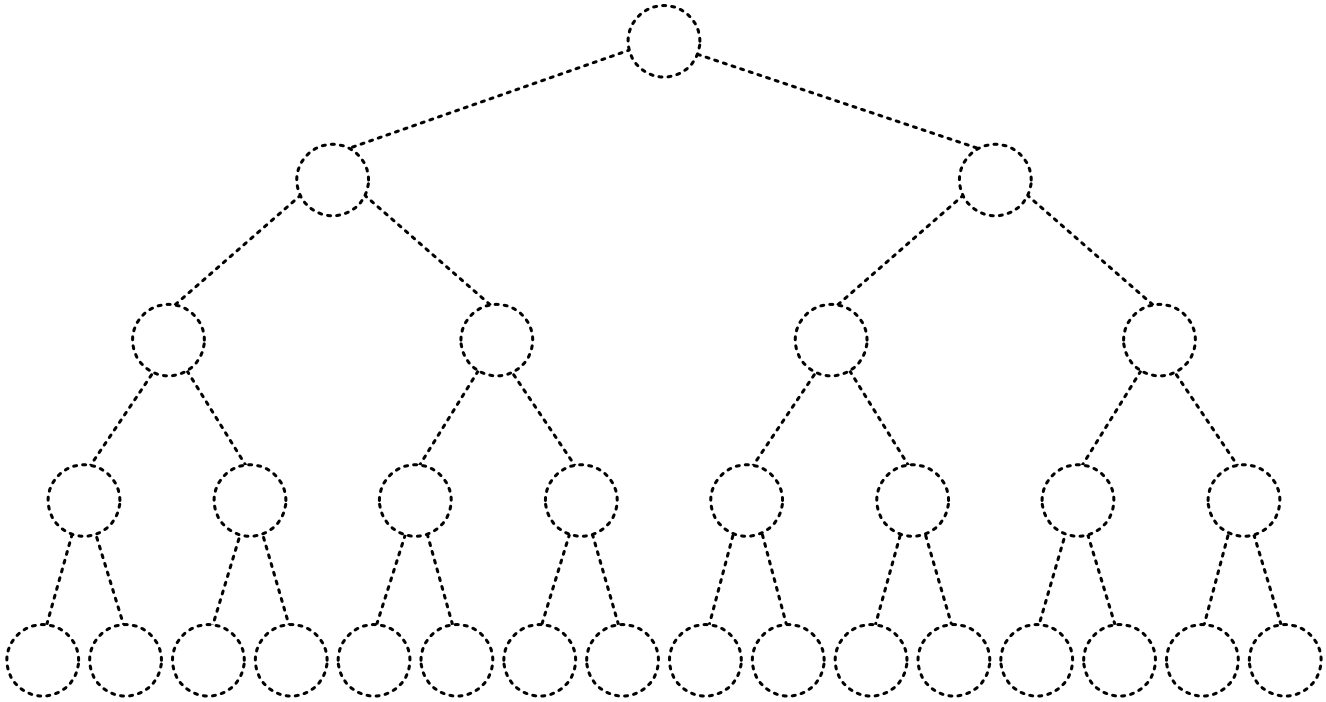
5.3) Insérez 83. **(2.5 points)**



5.4) Insérez 84. **(2.5 points)**

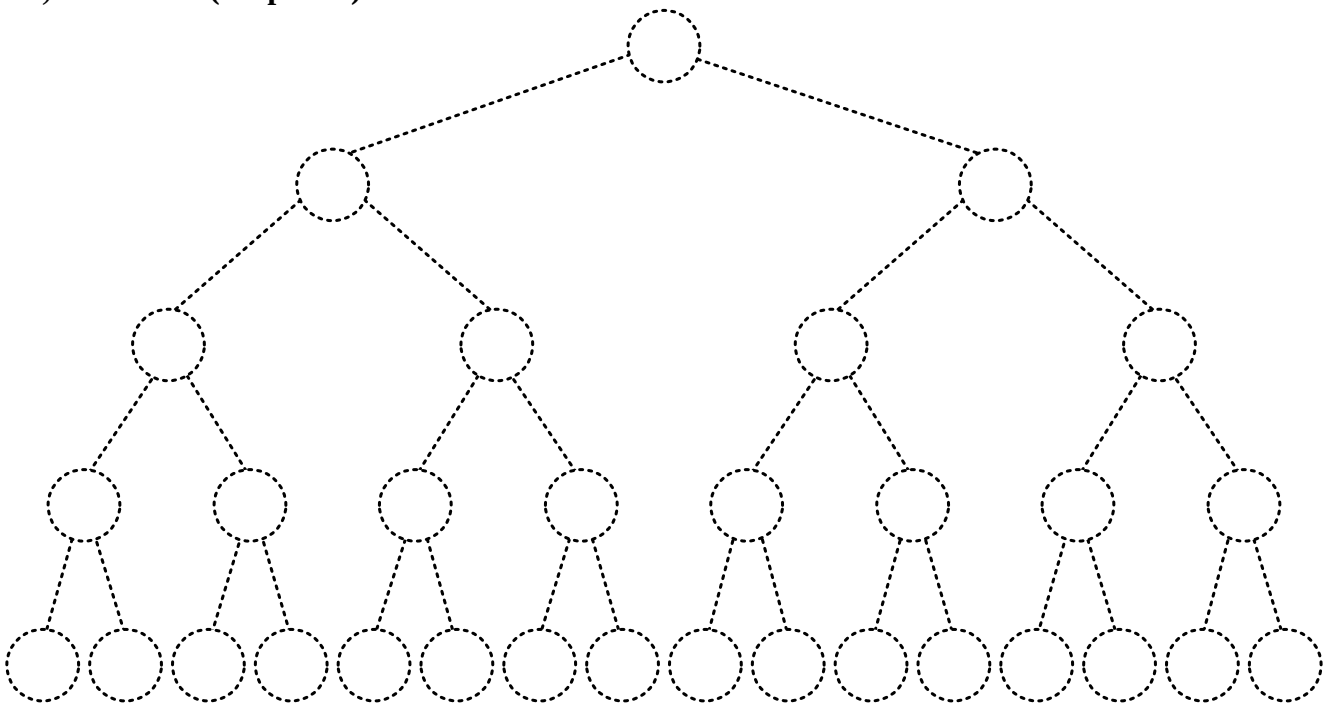


5.5) Insérez 85. **(2.5 points)**

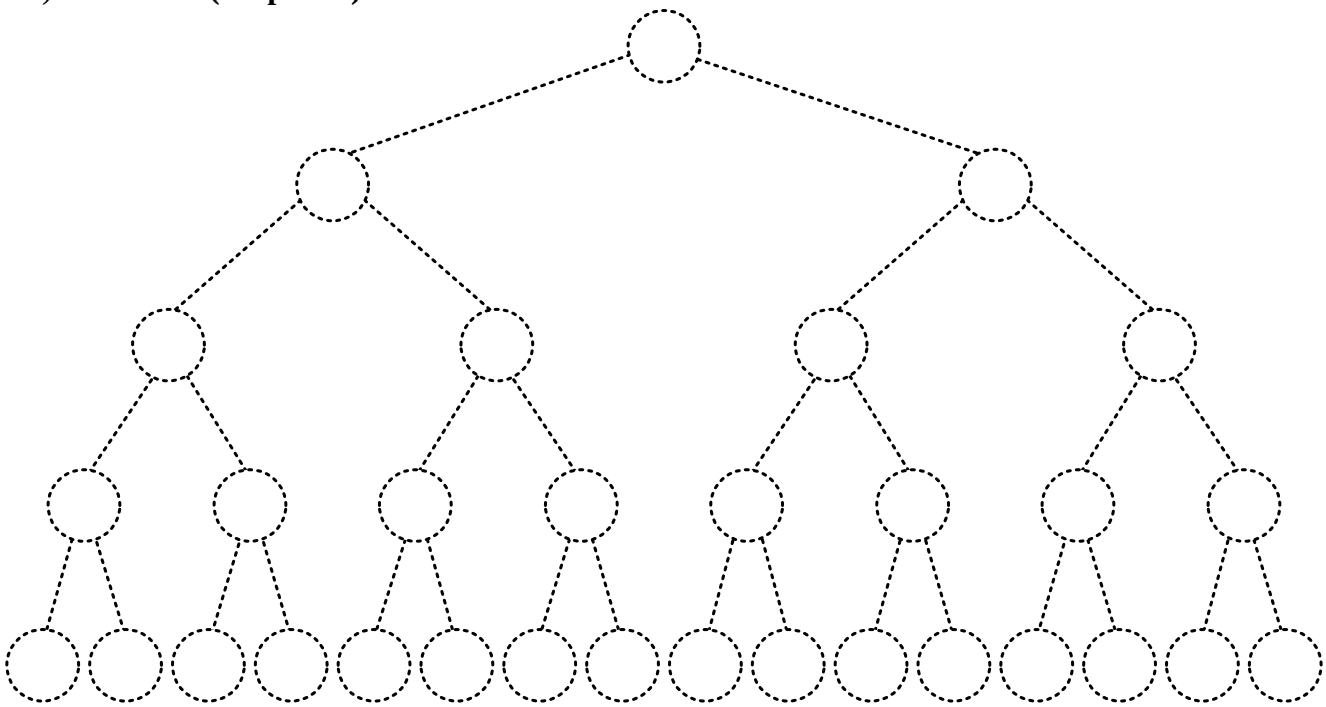


En partant de l'AVL obtenu en 5.2), insérez dans l'ordre 44, 45, et 43.

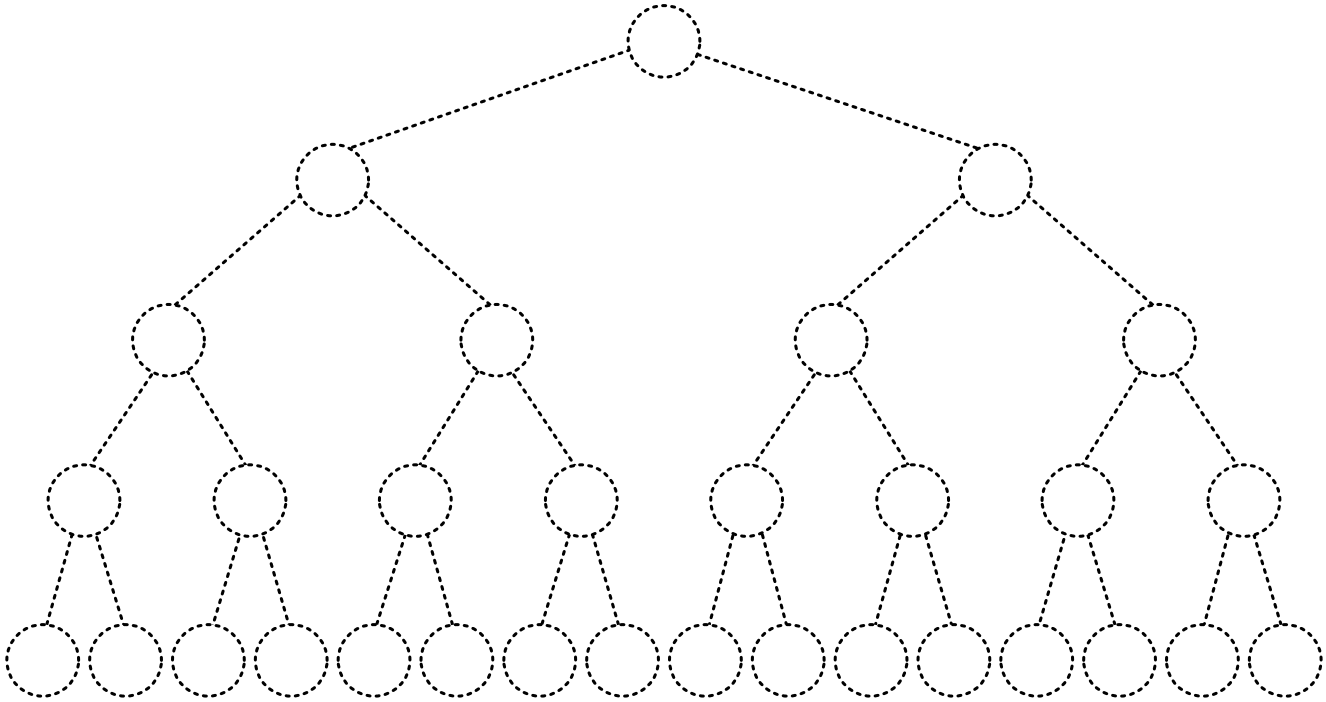
5.6) Insérez 44. **(2.5 points)**



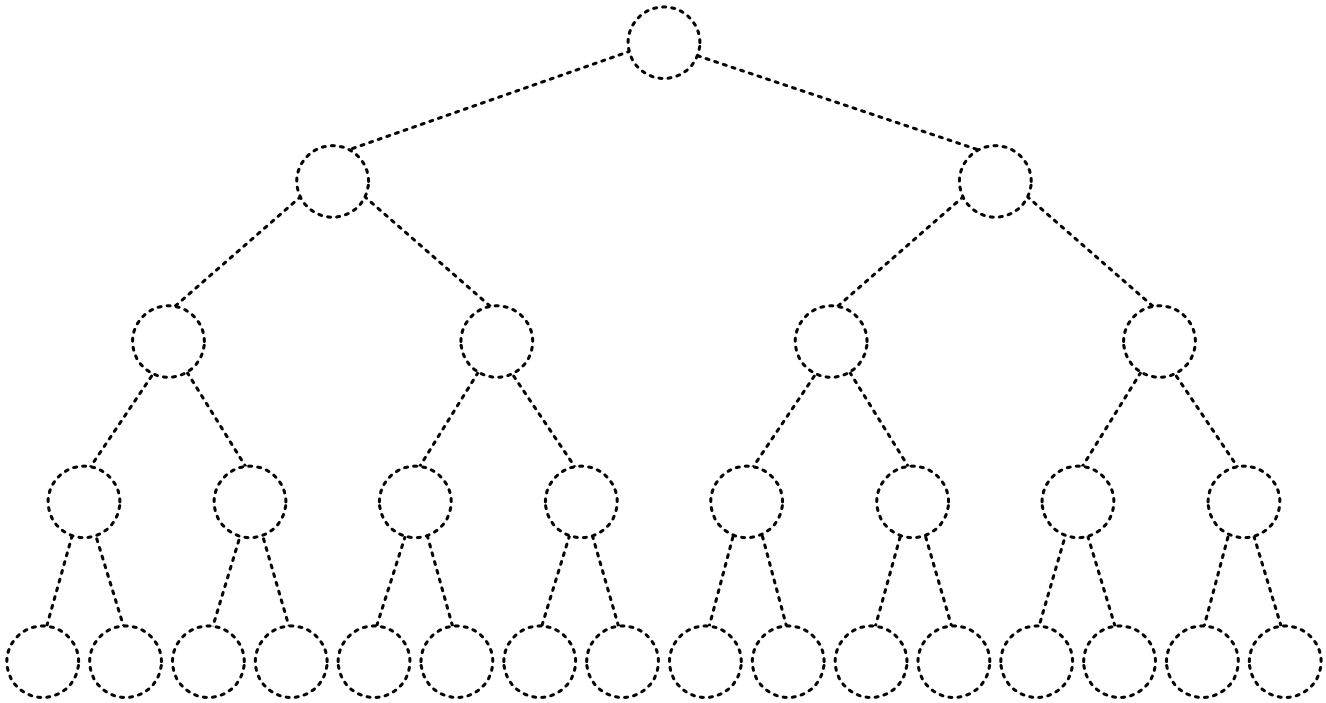
5.7) Insérez 45. **(2.5 points)**



5.8) Insérez 43. **(2.5 points)**

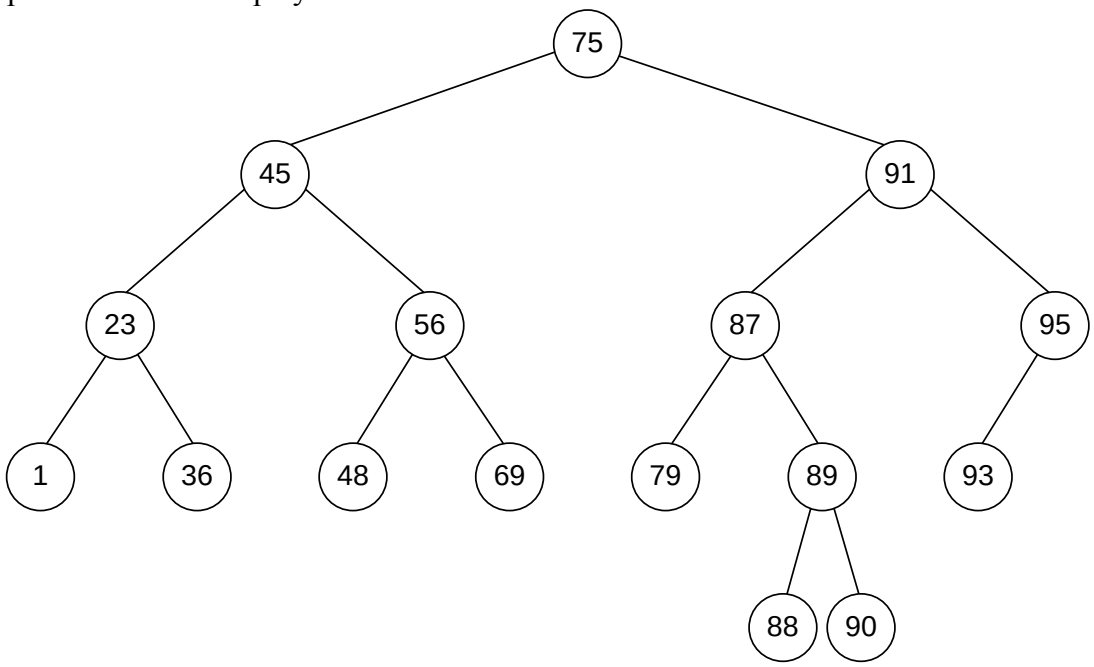


5.9) En partant de l'AVL obtenu en 5.5), retirez la racine. **(3 points)**

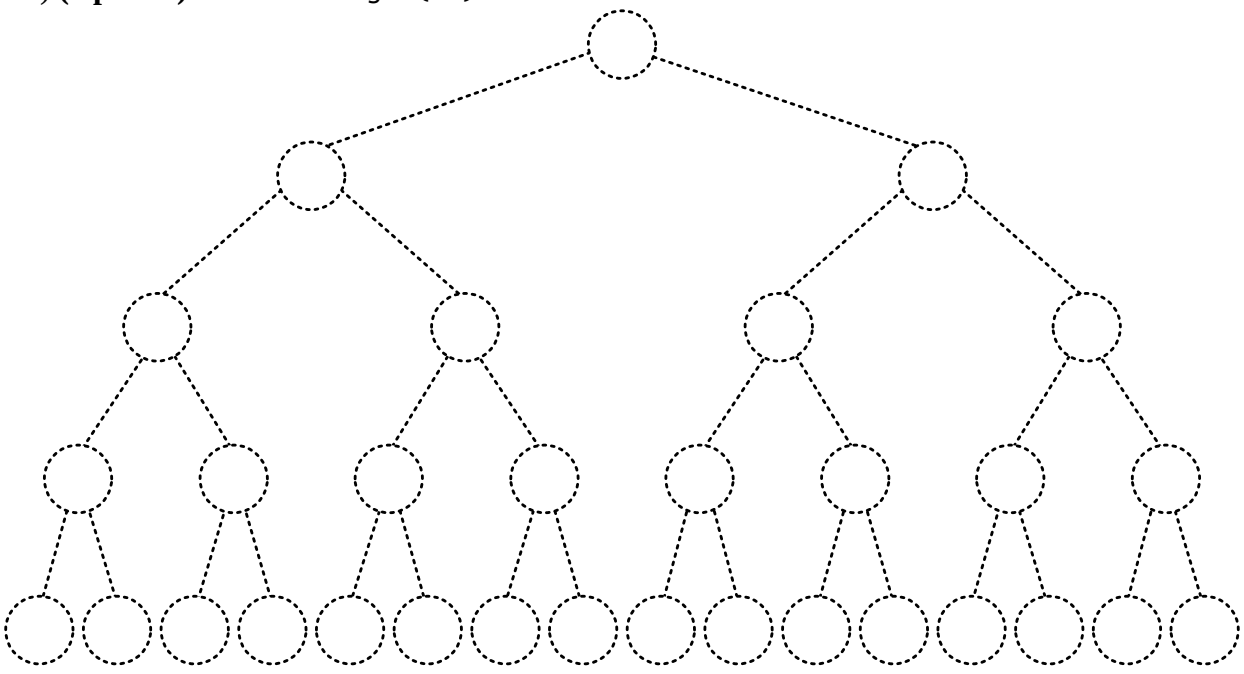


Question 6 : Arbre binaire de recherche de type Splay (8 points)

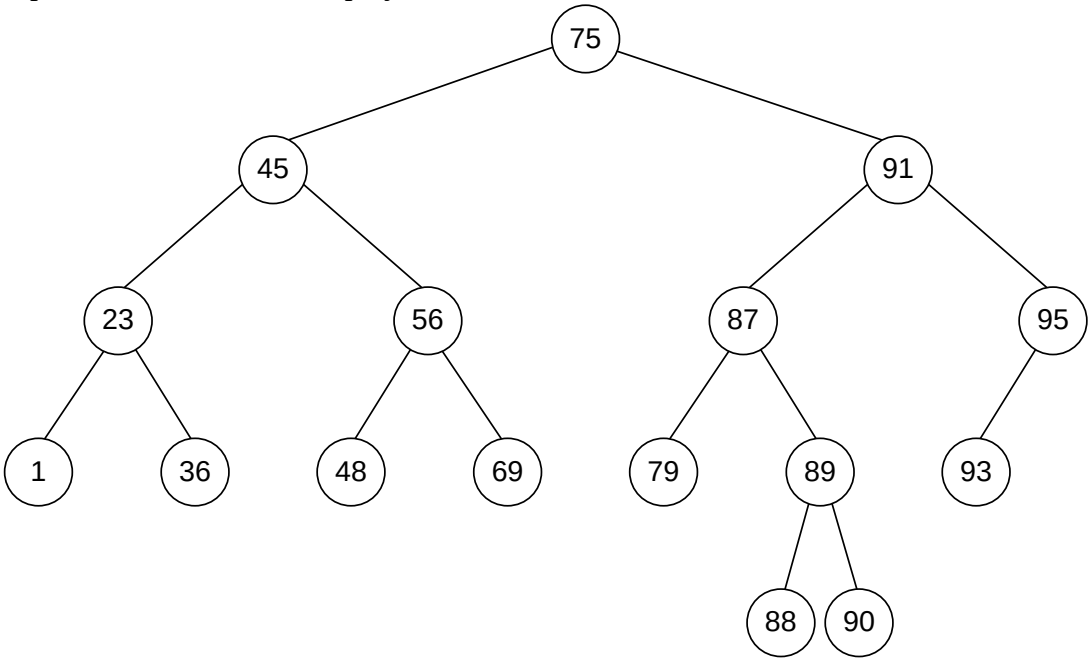
En partant de l'arbre Splay suivant :



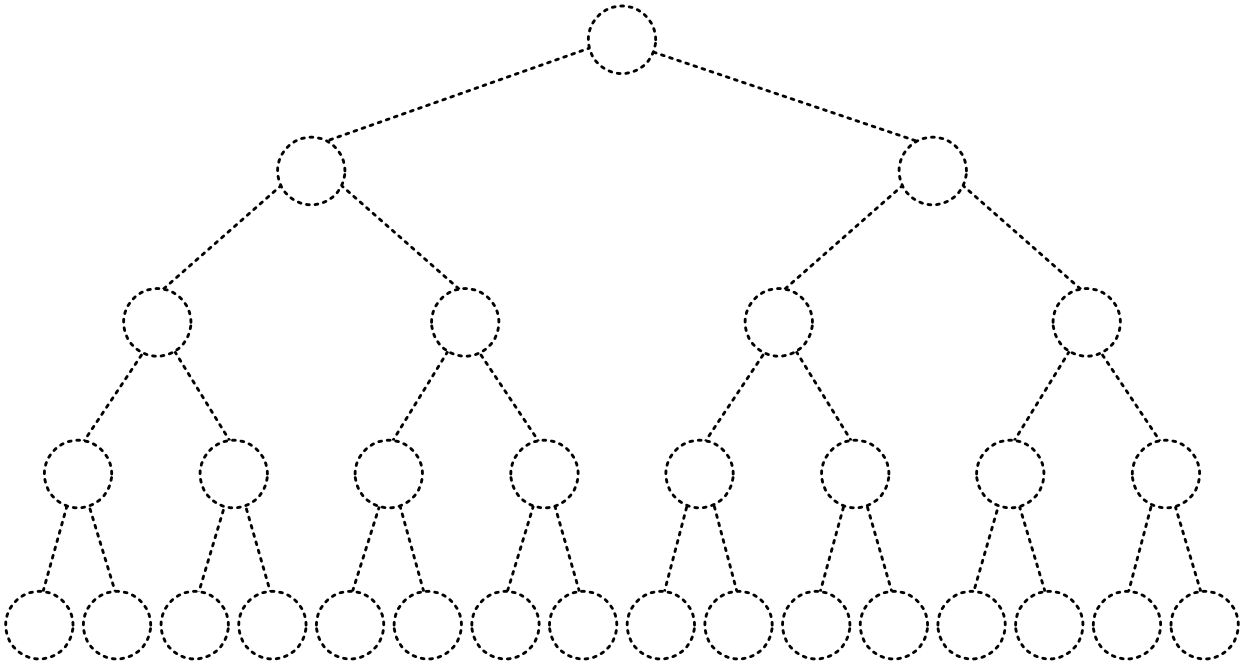
6.1) **(4 points)** Effectuez un `get(90)`.



En repartant du même arbre Splay :



6.2) **(4 points)** Effectuez un `delete(88)`.



Question 7 : Généralités (15 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 !

7.1) Une table de dispersement utilisant une résolution de collision par chaînage doit garantir un facteur de compression d'au plus 50% pour assurer une complexité asymptotique de O(1). **(2 pts)**

Répondez en cochant le tableau suivant :

VRAI		FAUX	

7.2) L'algorithme `MergeSort` prend environs deux fois plus de temps à s'exécuter sur un tableau de taille 2*n* que sur un tableau de taille *n*. **(2 pts)**

Répondez en cochant le tableau suivant :

VRAI		FAUX	

7.3) 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 premier et dont le facteur de compression est de 45% **(2 pts)**

Répondez en cochant le tableau suivant :

VRAI		FAUX	

7.4) L'opération de retrait usuelle, telle qu'effectuée sur un arbre binaire de recherche standard, effectuée sur la racine d'un arbre AVL peut produire un arbre complet **(2 pts)**.

Répondez en cochant le tableau suivant :

VRAI		FAUX	

7.5) Si la liste `liste` est une `LinkedList`, le code suivant aura une complexité O(n^2). **(2 pts)**

```
public static void retirerCle (List<Integer> liste, int cle)
{
   int i = 0;
   Iterator<Integer> it = liste.iterator();

    while( i < liste.size() )
        if( it.next().hashCode() == cle )
           it.remove();
        else
           i++;
}
```

Répondez en cochant le tableau suivant :

VRAI		FAUX	

7.6) Un arbre AVL de hauteur h=8 possède au plus 512 nœuds. **(2 points)**.

Répondez en cochant le tableau suivant :

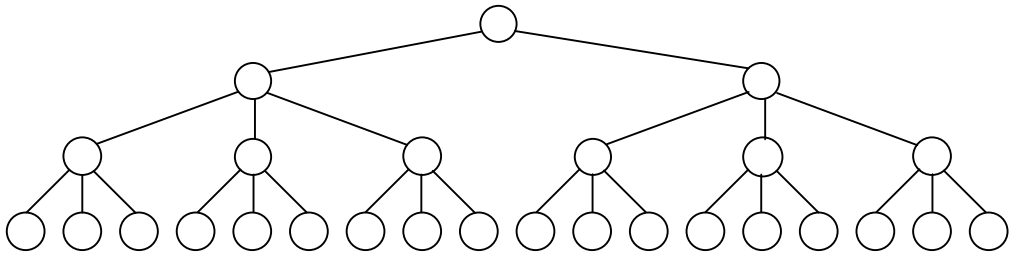
VRAI		FAUX	

7.7) Un arbre AVL de hauteur h=8 possède au moins 256 nœuds. **(2 points)**.

Répondez en cochant le tableau suivant :

VRAI		FAUX	

7.8) L'arbre suivant est un arbre complet d'ordre 3. **(1 points)**.



Répondez en cochant le tableau suivant :

VRAI		FAUX	

Annexe 1

```
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 = 3;

   public static <AnyType> void swapRefs(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 )
         swapRefs ( a, left, center );
      if( a[ right ].compareTo( a[ left ] ) < 0 )
         swapRefs ( a, left, right );
      if( a[ right ].compareTo( a[ center ] ) < 0 )
         swapRefs ( a, center, right );

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

   private static <AnyType extends Comparable<? super AnyType>>
   void quicksort( AnyType [ ] a, int left, int right ){
      if( left + CUTOFF <= right ){
         AnyType pivot = median3( 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 ) swapRefs ( a, i, j );
            else break;
         }

         swapRefs ( a, i, right - 1 );

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

```
   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;
      }
   }
}
```