



POLYTECHNIQUE
MONTREAL

Corrigé 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	20173
Professeur		Local	Téléphone
Ettore Merlo et Tarek Ould Bachir			
Jour	Date	Durée	Heure
Lundi	16 octobre 2017	2h00	18h00
Documentation		Calculatrice	
<input type="checkbox"/> Toute <input checked="" type="checkbox"/> Aucune <input type="checkbox"/> Voir directives particulières		<input type="checkbox"/> Aucune <input type="checkbox"/> Programmable <input checked="" type="checkbox"/> Non programmable	Les cellulaires, agendas électroniques et téléavertisseurs sont interdits.

Directives particulières

Bonne chance à tous!

Important

Cet examen contient **6** 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 dispersion**(18 points)**

Soit une table de dispersion avec sondage quadratique $\text{Hash}(\text{ clé }) = (\text{ clé } + i^2) \% N$ dont l'implémentation est fournie à l'Annexe 1 à titre de référence. Sachant que les cinq (5) clés suivantes ont été insérées dans cet ordre :

81, 29, 56, 68, 69.

1.1) **(5 points)** Trouvez les deux clés qui ne sont pas à leur place dans la table de taille $N=13$ présentée ci-après :

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Entrées				81	29				69		68		56

Clés mal positionnées : **56, 68**

1.2) **(2 points)** Remplacez à leur bonne position les clés identifiées à la question 1.1) et donnez ci-après l'état de la table après correction:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Entrées				81	29	56		68	69				

1.3) **(2 points)** Quelle a été la plus grande valeur prise par i (le i de $\text{Hash}(\text{ clé }) = (\text{ clé } + i^2) \% N$) lors de l'insertion des cinq clés précédentes ? Vous pouvez vous aider du code source fourni à l'Annexe 1.

Plus grande valeur prise par i : **2**

1.4) **(2 points)** Après l'insertion des cinq clés précédentes, on effectue un appel à `remove(29)`. Donnez le détail de cet appel. Soyez bref mais précis. Vous pouvez vous aider du code source fourni à l'Annexe 1.

29%13 = 3, collision

3+1%13 = 4, valeur trouvée

29 retirée

1.5) **(1 points)** Quelle sera la plus grande valeur prise par i (le i de $\text{Hash}(\text{clé}) = (\text{clé} + i^2) \% N$) lors de l'appel `remove(29)`.

Plus grande valeur prise par i : **1**

1.6) **(2 points)** Après l'appel à `remove(29)` à la question 1.4), on effectue un appel à `remove(81)`. Donnez le détail de cet appel. Soyez bref mais précis. Vous pouvez vous aider du code source fourni à l'Annexe 1.

81%13 = 3, valeur trouvée

81 retirée

1.7) **(2 points)** Donnez l'état de la table après l'appel à `remove(81)` de la question 1.6):

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Entrées						56		68	69				

1.8) **(2 points)** Après l'appel à `remove(81)` à la question 1.6), on effectue un appel à `insert(42)`. Donnez l'état de la table après cette insertion. Vous pouvez vous aider du code source fourni à l'Annexe 1.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Entrées						56		68	69				42

Question 2 : Tri fusion**(25 points)**

Partie I : On désire exécuter l'algorithme `mergeSort` pour trier le vecteur ci-après. 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
Valeurs	3	2	4	15	12	10	7	13	19	21	12	17	12	5	3

2.1) **(5 points)** Donnez, dans l'ordre d'appel, l'ensemble des dix (10) premières valeurs que prennent les paramètres `left` et `right` lors des appels successifs à la méthode privée `mergeSort`. Aidez-vous du code de l'Annexe 2.

Pour précision, il est bien question de la méthode dont la signature est :

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

Appels	left	right
Appel 1	0	14
Appel 2	0	7
Appel 3	0	3
Appel 4	0	1
Appel 5	0	0
Appel 6	1	1
Appel 7	2	3
Appel 8	2	2
Appel 9	3	3
Appel 10	4	7

2.2) (4 points) À la fin de l'exécution de l'algorithme, quelle aura été la seconde plus grande taille de vecteur sur laquelle la méthode privée mergeSort aura été appelée, la plus grande étant 15. Aidez-vous du code de l'Annexe 2.

Pour précision, il est bien question de la méthode dont la signature est :

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

Taille du second plus grand vecteur sur lequel mergeSort est appelé : **8**

2.3) (4 points) Positionnez dans le tableau fourni ci-après les éléments du vecteur identifié à la question 2.2) entre les positions left et right (inclusivement) au moment d'entrer dans la méthode privée mergeSort.

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

2.4) (4 points) Positionnez dans le tableau fourni ci-après les éléments du vecteur identifié à la question 2.2) entre les positions left et right (inclusivement) au moment de sortir de la méthode privée mergeSort.

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

Partie II : On vous propose une variation sur l'implémentation de l'algorithme mergeSort où la méthode privée est donnée comme suit :

```
private static <AnyType extends Comparable<? super AnyType>>
void mergeSort( AnyType[] a, AnyType[] tmpArray, int left, int right ){
    if( left < right ){
        int center = reverse(a, left, right);
        mergeSort( a, tmpArray, left, center );
        mergeSort( a, tmpArray, center + 1, right );
        merge( a, tmpArray, left, center + 1, right );
    }
}
```

où :

```
private static <AnyType extends Comparable<? super AnyType>>
int reverse( AnyType[] a, int left, int right ){
    int center = ( left + right ) / 2;

    for(int i=left, j=right; i<=center; i++, j--){
        swapReferences( a, i, j );
    }

    return center;
}

private static <AnyType> void swapReferences(AnyType[] a, int idx1, int idx2 ){
    AnyType tmp = a[ idx1 ];
    a[ idx1 ]    = a[ idx2 ];
    a[ idx2 ]    = tmp;
}
```

2.5) (2 points) Quelle est la complexité en pire cas de cette nouvelle implémentation ? Justifiez brièvement.

$$T(n) = 2T(n/2) + 2n, T(1) = 1 \rightarrow T(n) = 2n \log(n) + n$$

$$\rightarrow O(n \log(n))$$

2.6) (3 points) Positionnez dans le tableau fourni ci-après les éléments du vecteur a au moment de sortir du premier appel à la méthode reverse(...).

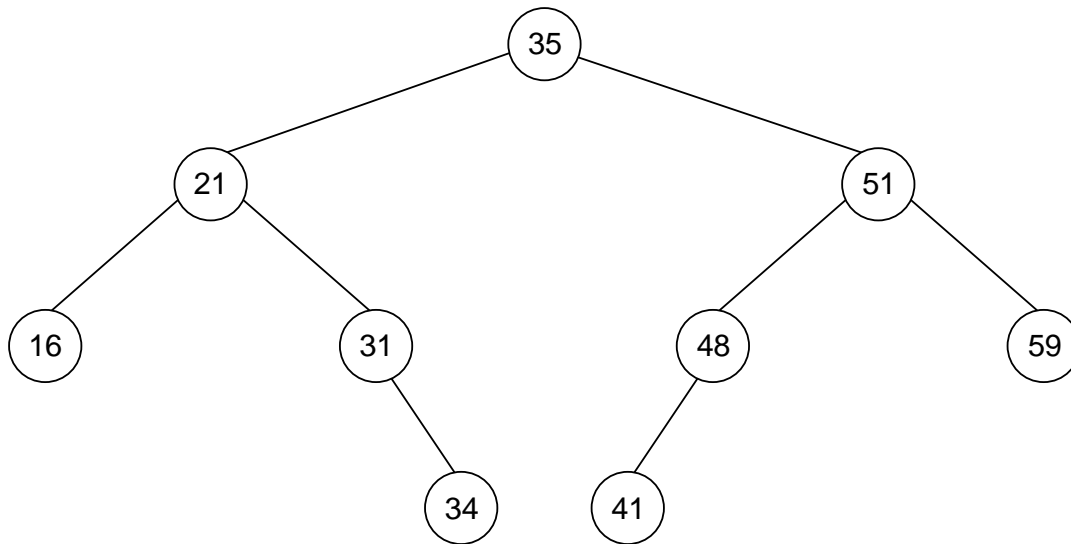
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Valeurs	3	5	12	17	12	21	19	13	7	10	12	15	4	2	3

2.7) (3 points) Positionnez dans le tableau fourni ci-après les éléments du vecteur a au moment de sortir du second appel à la méthode reverse(...).

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Valeurs	13	19	21	12	17	12	5	3	7	10	12	15	4	2	3

Question 3 : Parcours d'arbres**(12 points)**

Considérez le code de l'Annexe 3 ainsi que l'arbre binaire suivant. La méthode `toString()` est appelée sur cet arbre.



3.1) (4 points) Donnez la valeur du booléen `startWithLeft` passé en paramètre à la méthode `printTree(...)` en même temps que chacun des nœuds de l'arbre lors de l'appel à la méthode `toString()`. Par exemple, la racine (nœud 35) est passée en paramètre avec la valeur `true`.

Répondez en remplissant le tableau suivant.

Nœud	<code>startWithLeft</code>
35	<code>true</code>
21	<code>true</code>
51	<code>false</code>
16	<code>true</code>
31	<code>false</code>
48	<code>false</code>
59	<code>true</code>
34	<code>true</code>
41	<code>false</code>

3.2) **(2.5 points)** Si la variable `BinaryNode<Integer> t` contient une référence vers le nœud 21 de l'arbre précédent, donnez le contenu de la chaîne de caractères générée à la fin de l'exécution du code suivant :

```
StringBuilder s = new StringBuilder();  
printTree(t, s, false);  
s.toString();
```

Réponse:

31, 34, 21, 16

3.3) **(2.5 points)** Si la variable `BinaryNode<Integer> t` contient une référence vers le nœud 51 de l'arbre précédent, donnez le contenu de la chaîne de caractères générée à la fin de l'exécution du code suivant :

```
StringBuilder s = new StringBuilder();  
printTree(t, s, false);  
s.toString();
```

Réponse:

59, 51, 48, 41

3.4) **(3 points)** Donnez le résultat de l'affichage de l'arbre binaire précédent lors de l'appel à la méthode `toString()`. Référez-vous au code de l'Annexe 3.

Réponse:

16, 21, 34, 31, 35, 59, 51, 48, 41

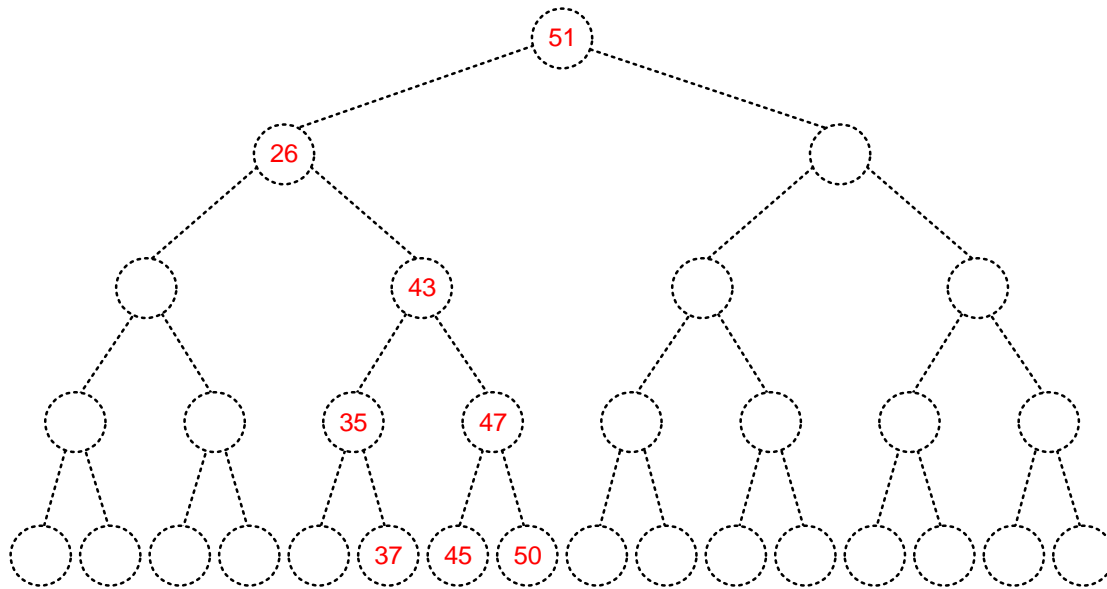
Question 4 : Arbres binaire de recherche**(13 points)**

Note : Tel que vu en classe, l'ordre de traversement des enfants d'un nœud se fait de gauche à droite selon la figure.

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

51, 26, 43, 35, 47, 37, 45, 50

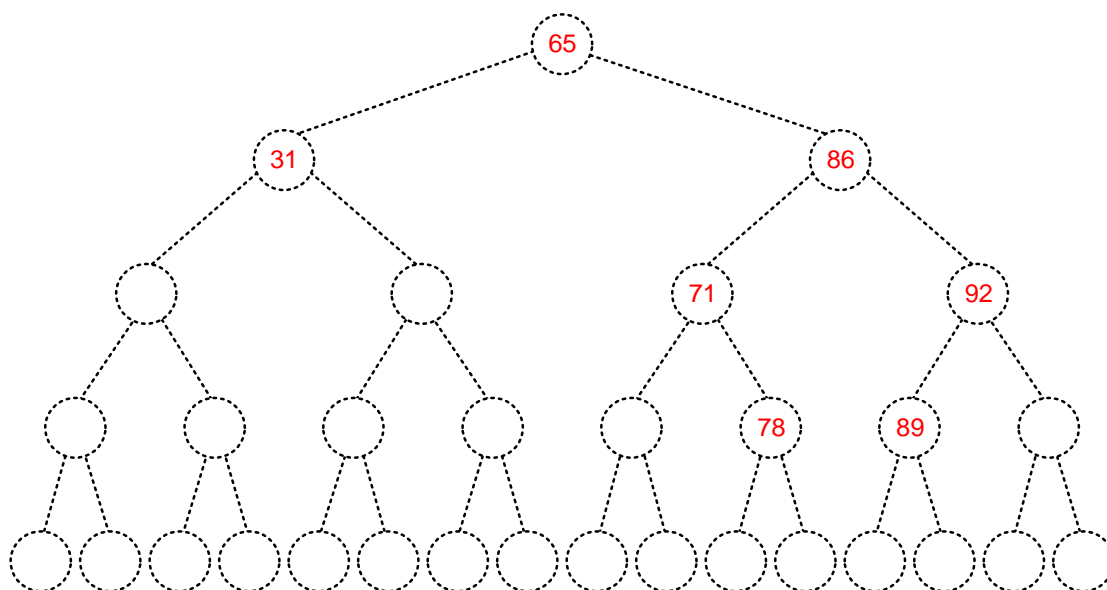
Donnez sa représentation graphique :



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

65, 31, 86, 71, 78, 92, 89

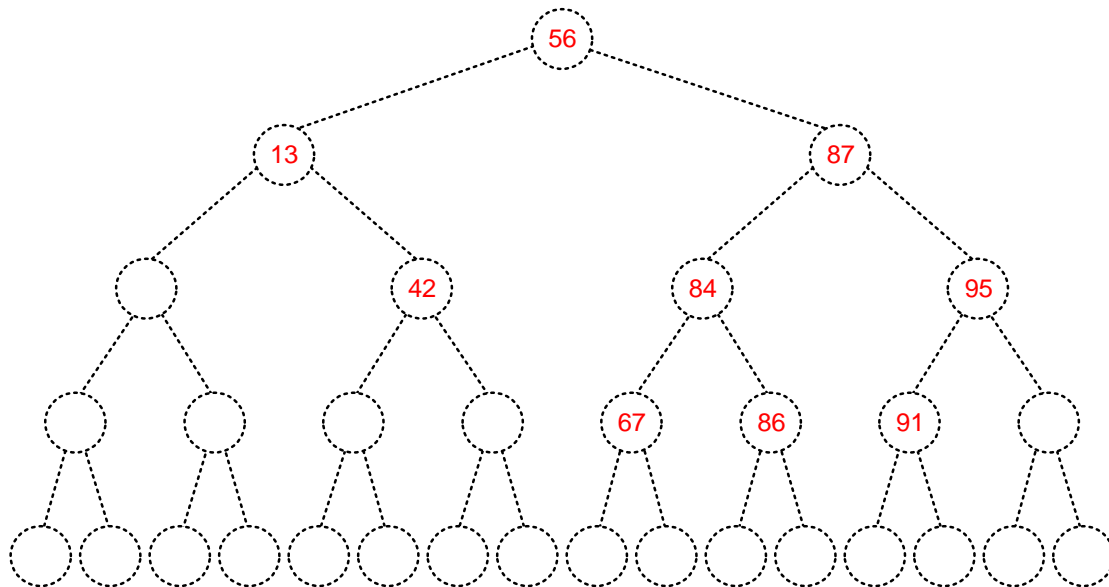
Donnez sa représentation graphique :



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

42, 13, 67, 86, 84, 91, 95, 87, 56

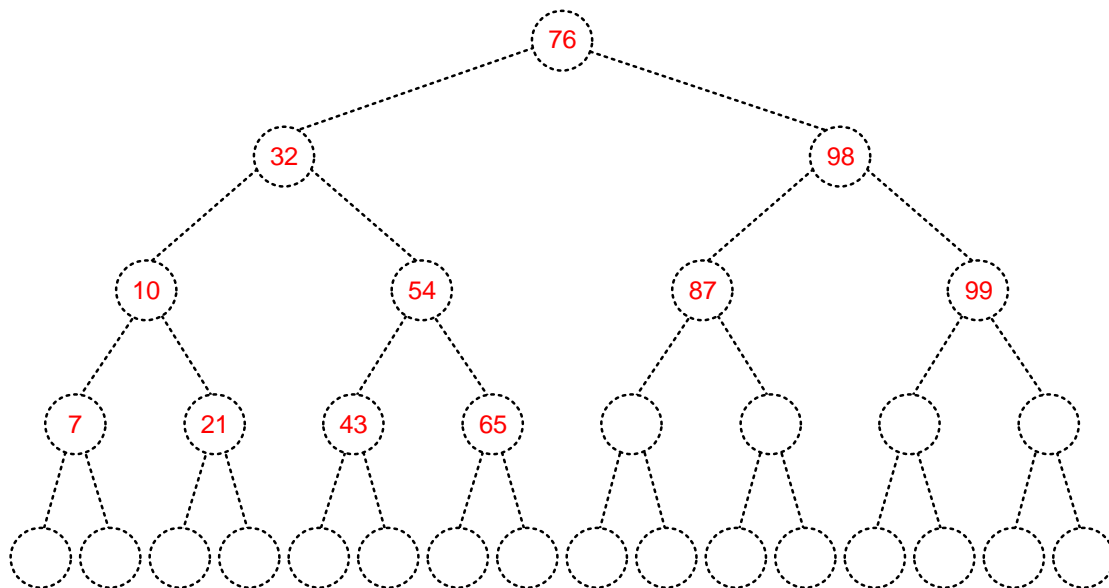
Donnez sa représentation graphique :



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

7, 10, 21, 32, 43, 54, 65, 76, 87, 98, 99

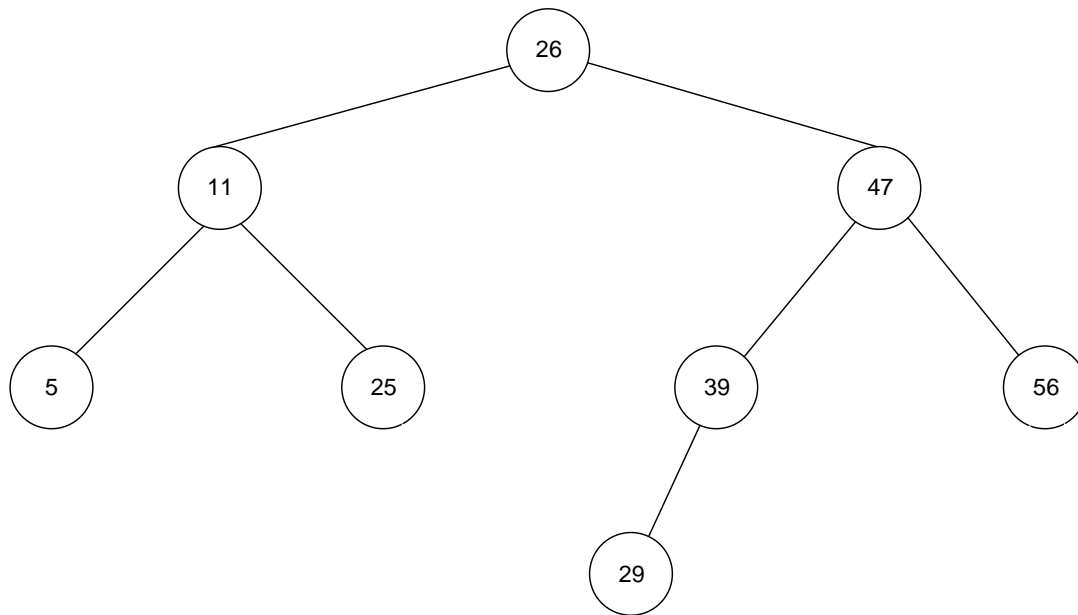
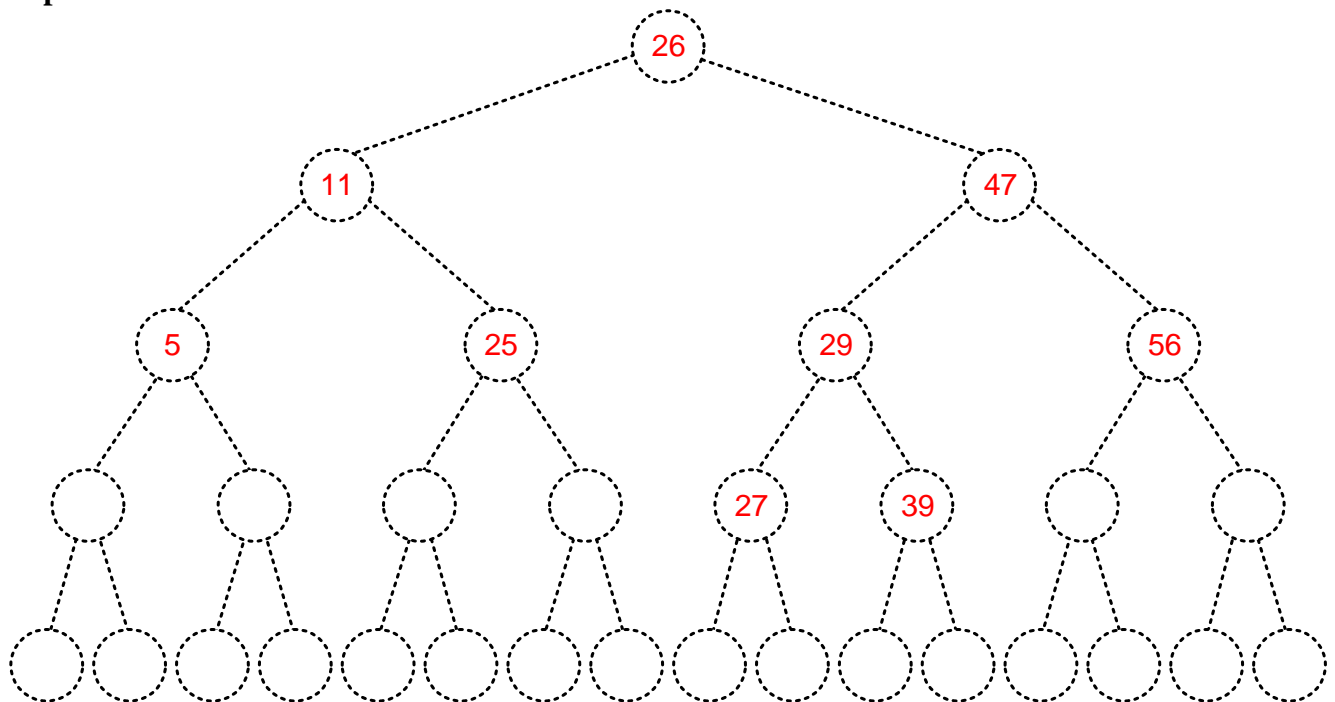
Donnez sa représentation graphique sachant qu'il s'agit d'un AVL dont la racine est 76.



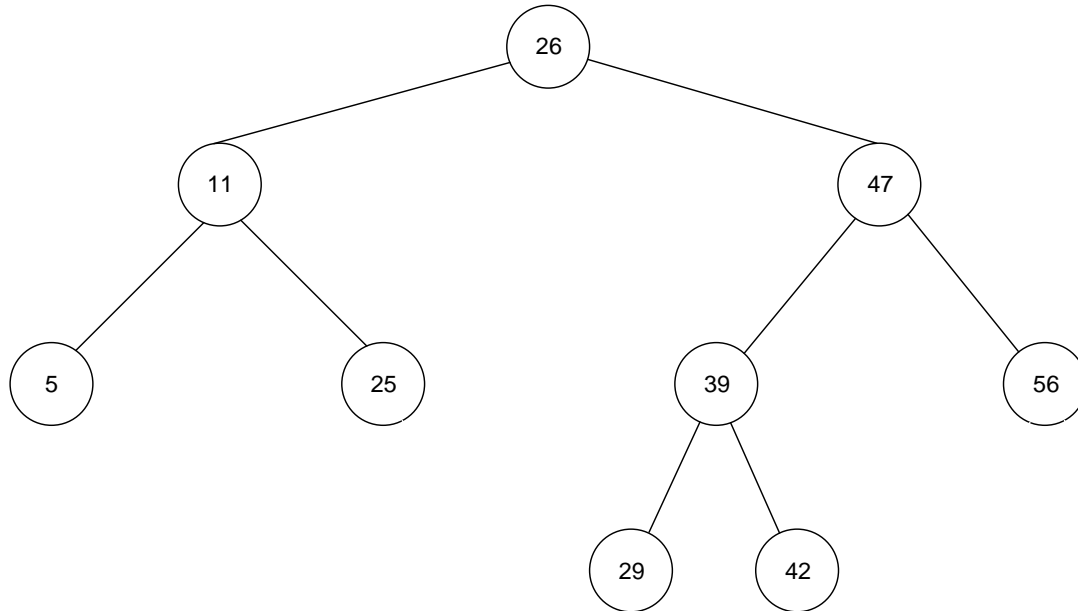
Question 5 : Arbre binaire de recherche de type AVL**(19 points)**

En partant de chacun des arbres AVL suivants, effectuer les opérations demandées.

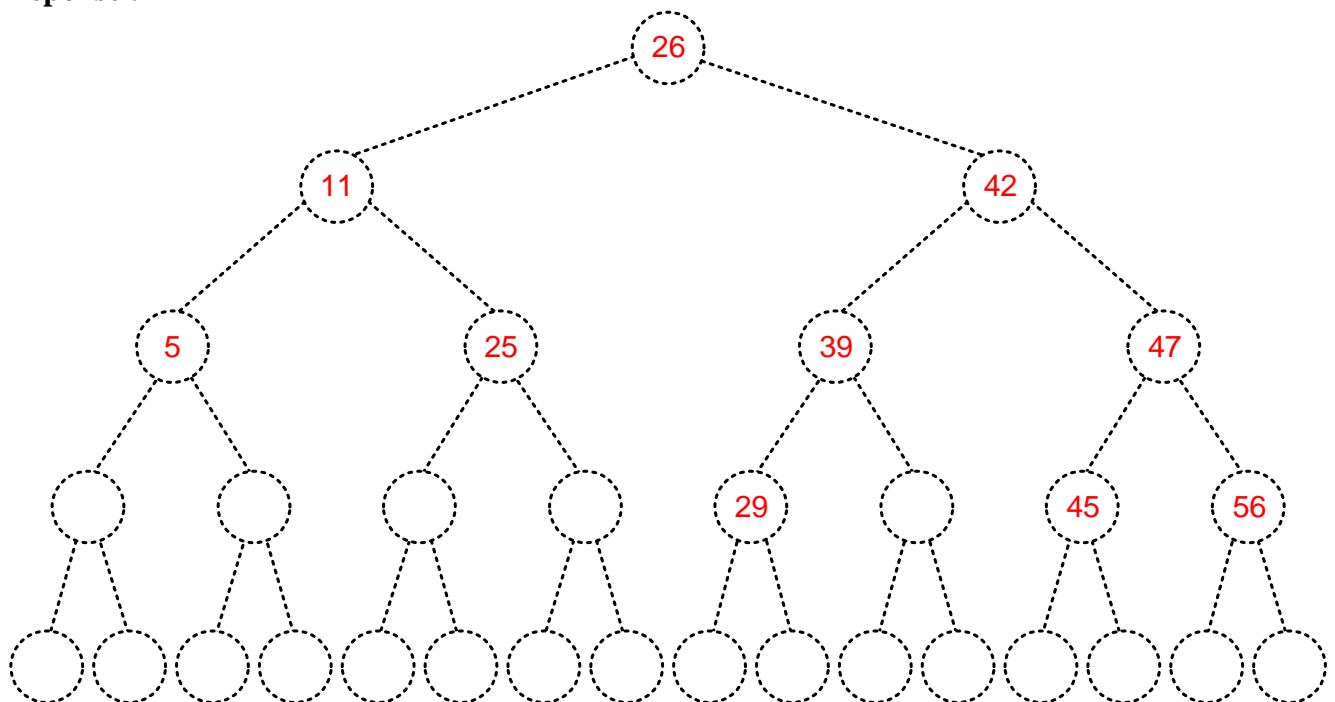
5.1) (2 points) Insérez 27.

**Réponse :**

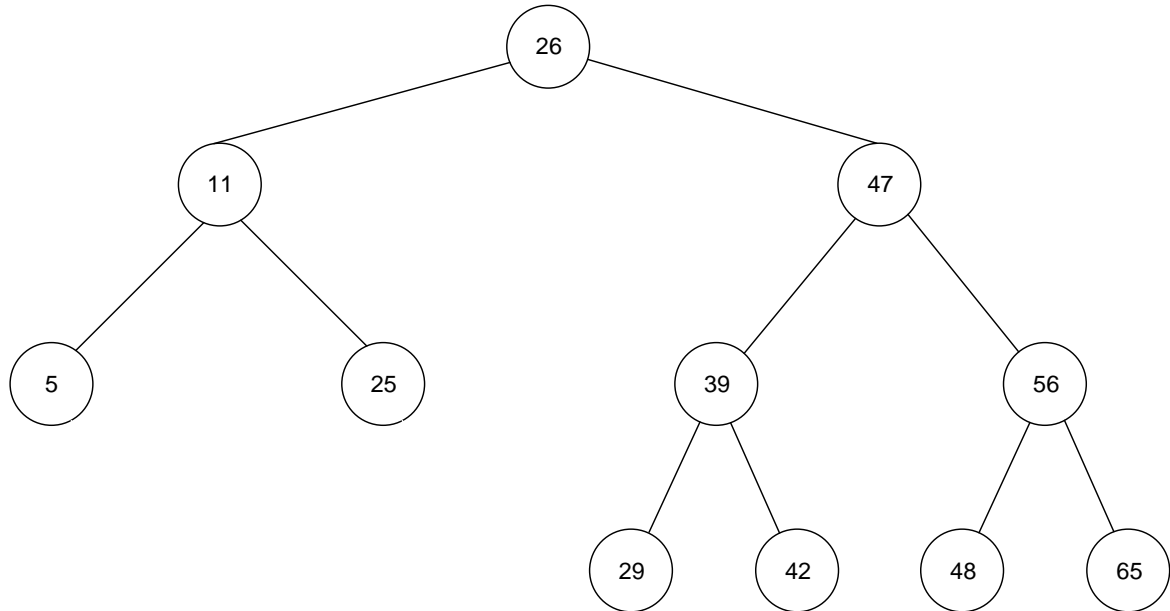
5.2) (3 points) Insérez 45.



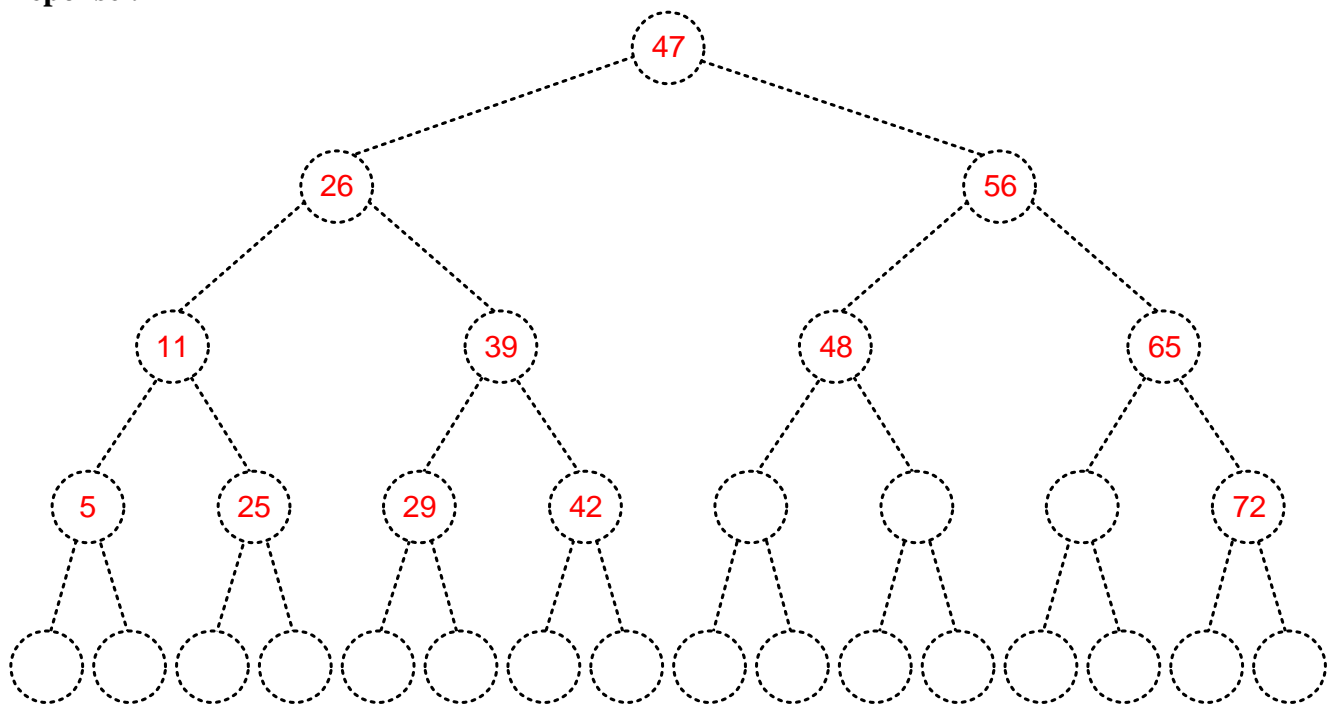
Réponse :



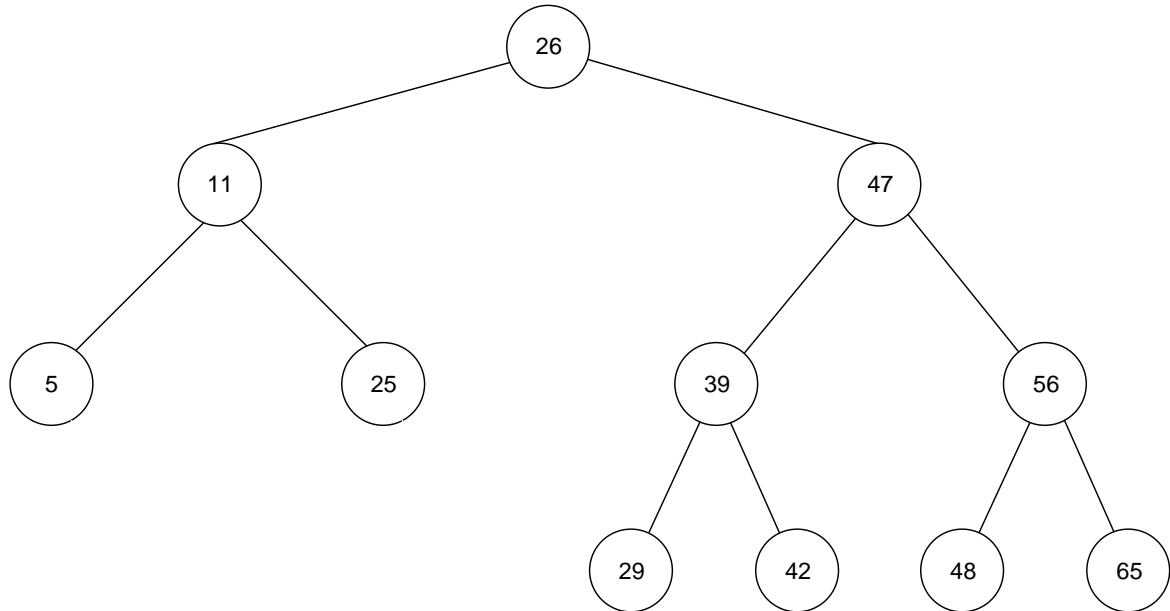
5.3) (3 points) Insérez 72.



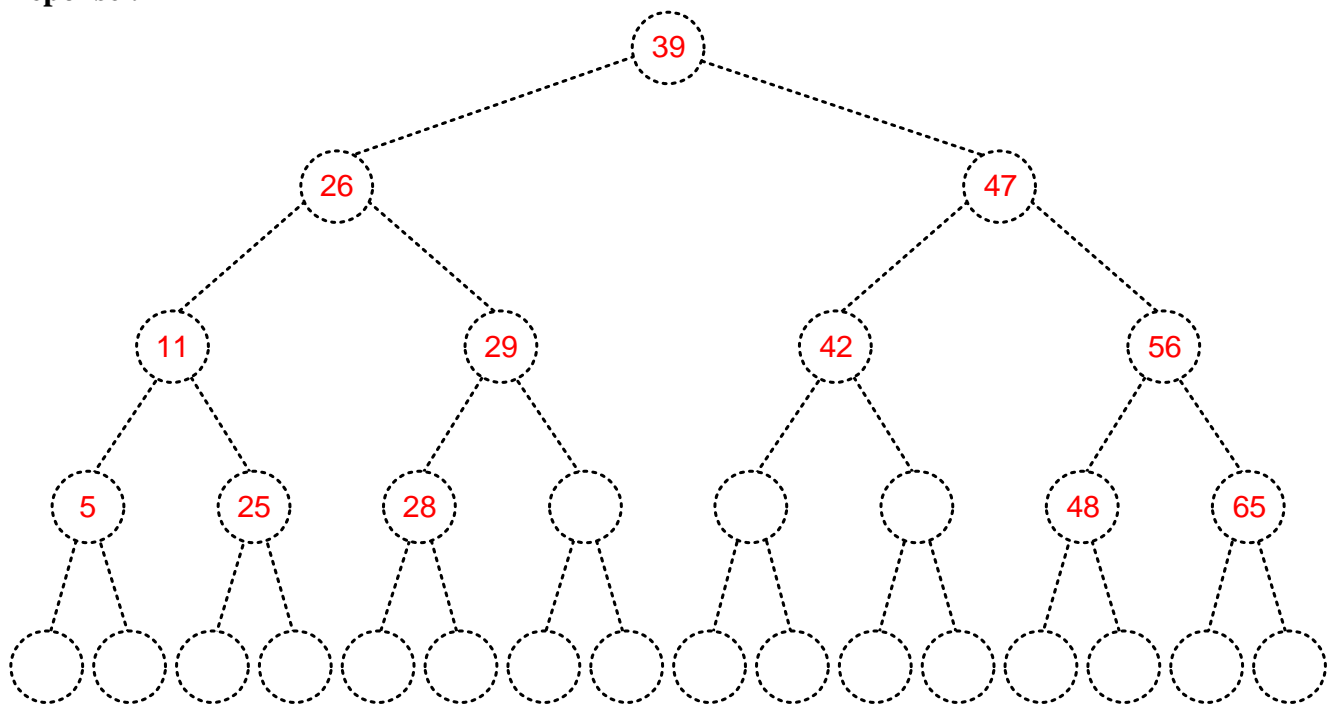
Réponse :



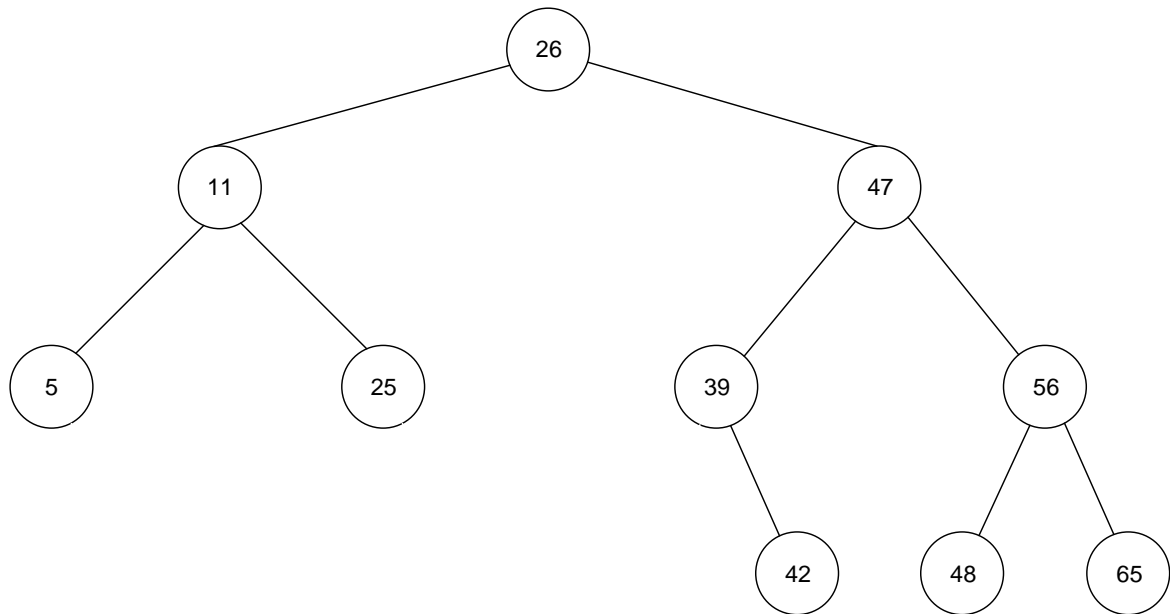
5.4) (3 points) Insérez 28.



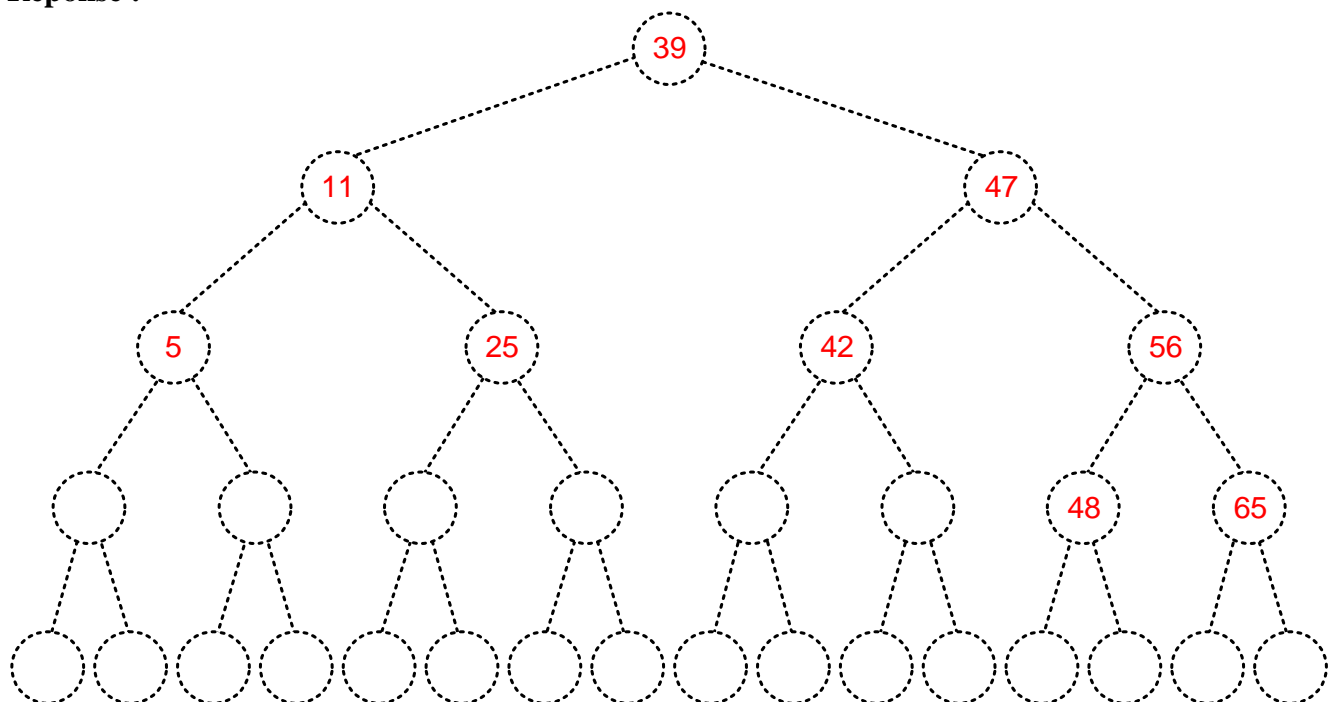
Réponse :



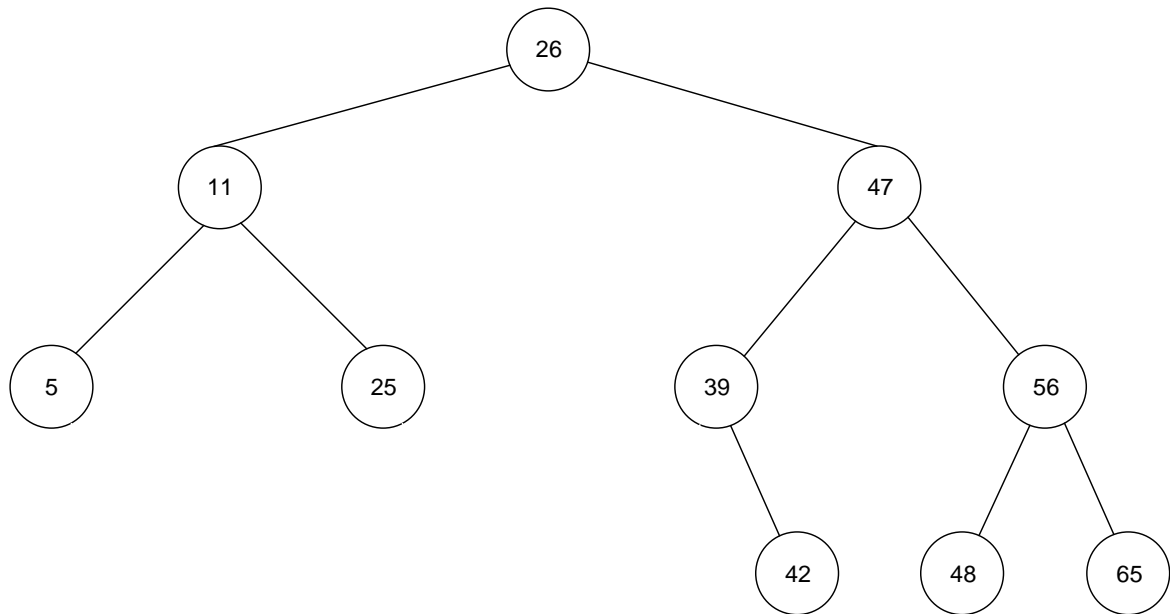
5.5) (2.5 points) Retirez la racine (utilisez la méthode `remove` d'un arbre binaire de recherche standard telle que vue en classe).



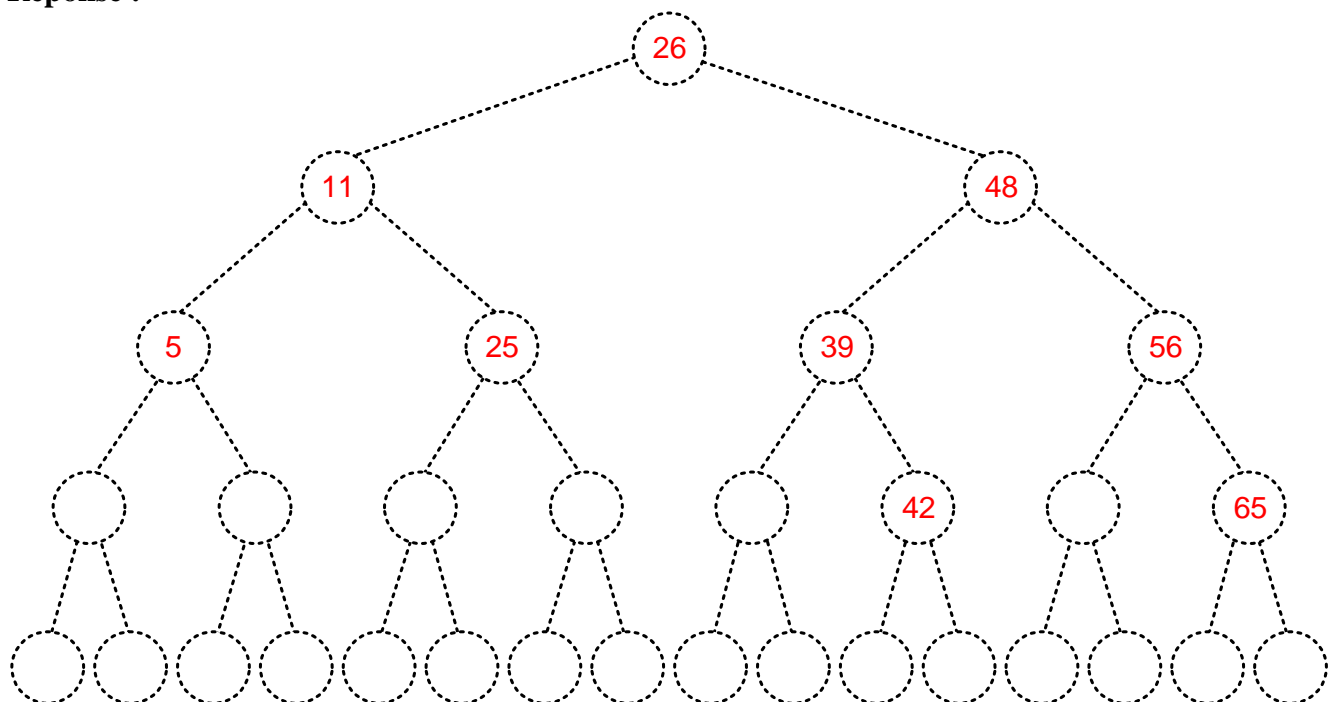
Réponse :



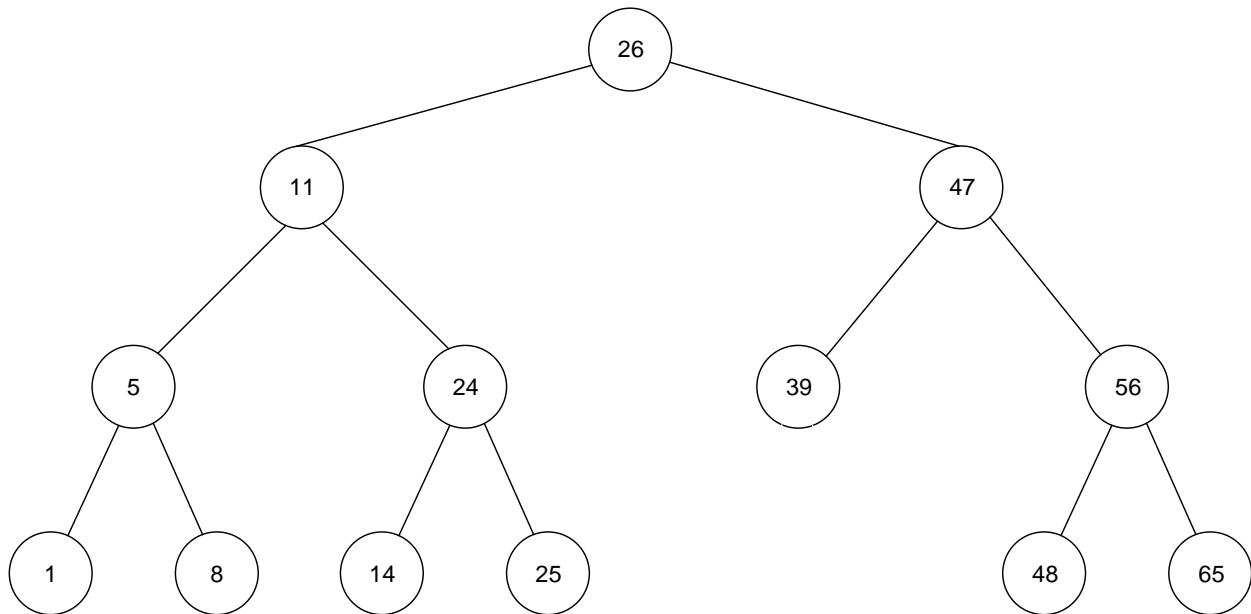
5.6) (2.5 points) Retirez le nœud 47 (utilisez la méthode `remove` d'un arbre binaire de recherche standard telle que vue en classe).



Réponse :



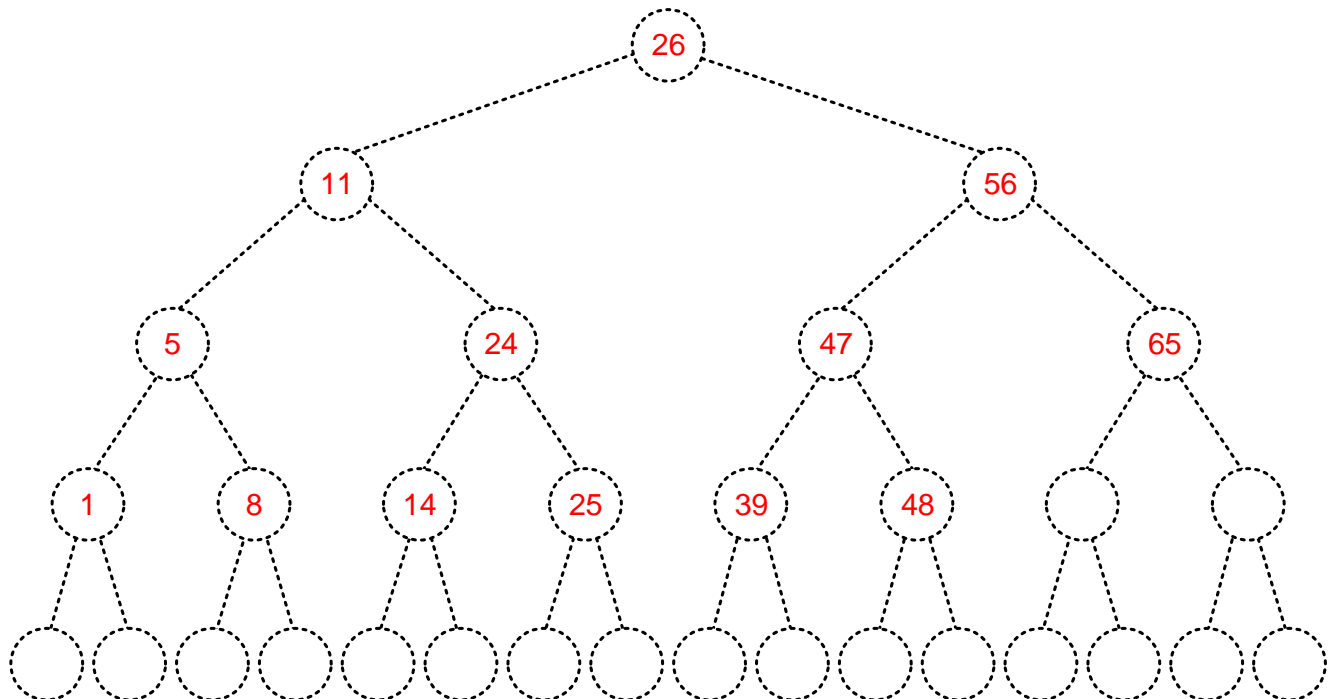
5.7) (3 points) Effectuer une rotation simple à un des nœuds de l'arbre pour obtenir un arbre complet.



Réponse :

Nœud choisi : 47, Direction de la rotation : gauche

Résultat :



Question 6 : Généralités**(13 points)**

Répondez aux assertions suivantes par « vrai » ou par « faux ». Justifier votre réponse brièvement. Les réponses non justifiées ne seront pas considérées.

6.1) (2 points) La fusion de deux listes triées en ordre inverse en une liste unique triée en ordre peut s'effectuer en $O(n)$.

Vrai. Il suffit de parcourir les deux listes depuis la fin et de construire la nouvelle liste au fur et à mesure qu'on rencontre le plus petit élément à date.

6.2) (2.5 points) Si la liste `liste` est une `ArrayList`, le code suivant aura une complexité $O(n^2)$ en pire cas.

```
public static void dupliquerCles (List<Integer> liste, int cle)
{
    int i = 0;
    while( i < liste.size() ){
        if( liste.get( i ).hashCode() == cle )
            liste.add( i++, cle );
        i++;
    }
}
```

Vrai. Le `while` $O(n)$, `get` $O(1)$, `add` $O(n)$, ce dernier s'imbrique dans `while` pour donner $O(n^2)$.

6.3) (2.5 points) Si la liste `list` est une `ArrayList`, le code suivant aura une complexité $O(n^2)$ en pire cas.

```
public static void dupliquerCles (List<Integer> liste, int cle )
{
    ListIterator<Integer> it = liste.listIterator();
    while( it.hasNext() )
        if( it.next().hashCode() == cle )
            it.add(cle);
}
```

Vrai. Le `while` $O(n)$, `next` $O(1)$, `add` $O(n)$, le tout donne $O(n^2)$.

6.4) (2 points) En meilleur cas, l'algorithme mergeSort a une complexité $O(n \log(n))$.

Vrai. L'algorithme mergeSort est $O(n \log(n))$ en tout cas.

6.5) (2 points) Un arbre binaire de recherche de type Splay a nécessairement une hauteur proportionnelle à $\log(n)$, où n est le nombre de nœuds dans l'arbre.

Faux. Lors des insertions dans l'ordre de 1, 2, 3, ..., n , il formera une liste chaînée et sa hauteur sera proportionnelle à n .

6.6) (2 points) Un arbre AVL de hauteur $h=7$ possède au moins 55 nœuds.

Faux, au moins 54 nœuds.