

Questionnaire examen intra

INF2010

Sigle du cours

<i>Identification de l'étudiant(e)</i>		
Nom :	Prénom :	
Signature :	Matricule :	Groupe :

<i>Sigle et titre du cours</i>		<i>Groupe</i>	<i>Trimestre</i>
INF2010 – Structures de données et algorithmes		Tous	20111
<i>Professeur</i>		<i>Local</i>	<i>Téléphone</i>
Ettore Merlo, responsable – Tarek Ould Bachir, chargé de cours		C-539.6	7128
<i>Jour</i>	<i>Date</i>	<i>Durée</i>	<i>Heure</i>
Mercredi	23 février 2009	2h00	15h15

<i>Documentation</i>	<i>Calculatrice</i>	
<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 ou téléavertisseurs sont interdits.

<i>Directives particulières</i>
<p style="text-align: right;"><i>Bonne chance à tous!</i></p>

Important	<p>Cet examen contient 5 questions sur un total de 14 pages (excluant cette page)</p> <p>La pondération de cet examen est de 30 %</p> <p>Vous devez répondre sur : <input checked="" type="checkbox"/> le questionnaire <input type="checkbox"/> le cahier <input type="checkbox"/> les deux</p> <p>Vous devez remettre le questionnaire : <input checked="" type="checkbox"/> oui <input type="checkbox"/> non</p>
------------------	--

Question 1 : Complexité algorithmique**(16 points)**

Considérez les deux fonctions suivantes qui retournent pour un N positif la valeur 2^N :

```
public static int DeuxPuisanceV1(int N)
{
    if( N<0 ) throw new IllegalArgumentException();
    if( N == 0 ) return 1;

    if ( N%2 == 0 )
        return DeuxPuisanceV1(N/2)*DeuxPuisanceV1(N/2);
    else
        return 2 * DeuxPuisanceV1((N-1)/2) * DeuxPuisanceV1((N-1)/2);
}
```

```
public static int DeuxPuisanceV2(int N)
{
    if( N<0 ) throw new IllegalArgumentException();
    if( N == 0 ) return 1;

    if ( N%2 == 0 )
    {
        int DeuxPuisanceNSur2 = DeuxPuisanceV2(N/2);
        return DeuxPuisanceNSur2 * DeuxPuisanceNSur2;
    }
    else
    {
        int DeuxPuisanceNMoins1Sur2 = DeuxPuisanceV2((N-1)/2);
        return 2* DeuxPuisanceNMoins1Sur2 * DeuxPuisanceNMoins1Sur2;
    }
}
```

1.1) **(4 points)** Donnez votre estimation de la complexité asymptotique de chacune de ces deux fonctions. Justifiez vos réponses par un calcul succinct.

1.2) **(8 points)** Proposez pour `DeuxPuisance` une implémentation non récursive de même complexité que l'une ou l'autre des deux fonctions précédentes. Énoncez clairement votre estimation de la complexité asymptotique de votre implémentation.

```
public static int DeuxPuisanceEquivalentNonRecuratif(int N)
{

}

}
```

Estimation de complexité asymptotique :

1.3) **(4 points)** Pensez-vous qu'il soit possible d'implémenter une fonction qui retourne en $O(1)$ la valeur 2^N pour tout N positif en paramètre en utilisant les opérateurs arithmétiques de base ? Expliquez brièvement.

Question 2 : Tables de dispersion**(20 points)**

Considérant une table de dispersion double où $\text{Hash}(X) = (H_1(X) + i \cdot H_2(X)) \% N$, et où $H_1(X) = (X^2) \% N$, et $H_2(X) = R - (X \% R)$. R et N sont des nombres premiers tels que $R < N$ et N est la taille de la table.

2.1) **(5 point)** Donnez la complexité asymptotique en cas moyen en insertion pour cette structure de données. Justifiez brièvement.

2.3) **(9 point)** Quel est le nombre de collisions obtenues si nous voulions insérer, dans l'ordre, les clés 2, 5, 6, 4, 7 et 9 dans une table de table $N = 11$ si $R = 7$? Donnez le détail de vos calculs.

Note : Si, lors de l'insertion d'une clé X dans la table, il aura fallu incrémenter i jusqu'à $i = 2$ avant de pouvoir trouver une espace libre pour l'insertion de la clé, alors cette insertion aura occasionné deux collisions.

2.4) **(6 points)** En vous servant du tableau ci-dessous, donnez l'état de la mémoire de la table de dispersion à la fin des insertions.

Indice	Données
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Question 3 : Tris en $n \log(n)$ **(15 points)**

Considérez le code Java du tri rapide (QuickSort) suivant :

```

private static final int CUTOFF = 4;

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

        // Begin partitioning
        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 );    // Restore pivot

        quicksort( a, left, i - 1 );    // Sort small elements
        quicksort( a, i + 1, right );    // Sort large elements
    }
    else // Do an insertion sort on the subarray
        insertionSort( a, left, right );
}

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 );

    // Place pivot at position right - 1
    swapReferences( a, center, right - 1 );
    return a[ right - 1 ];
}

```

3.1) (5 points) Calculez le nombre de fois que sera appelée la fonction

```
private static void quicksort(AnyType [ ] a, int left, int right)
```

suite à l'appel suivant :

```
Integer[ ] a = { 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 };  
quicksort( a );
```

Faites bien attention à la valeur du *CUTOFF* dans le code fourni.

3.2) (5 points) Donnez l'état du vecteur *a* après la fin de premier partitionnement exécuté par *quicksort* :

Indice	0	1	2	3	4	5	6	7	8	9
Données										

3.3) (5 points) Quelle est la plus petite taille du sous-vecteur de *a* sur lequel sera appelé *insertionSort*() ?

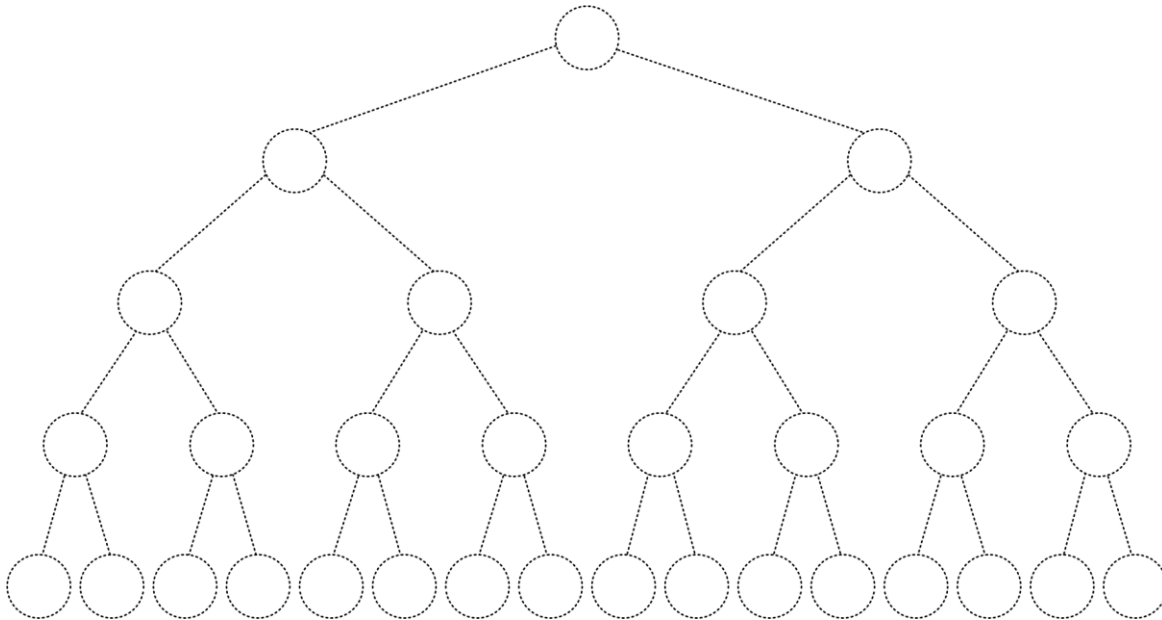
Question 4 : Arbres binaire de recherche**(25 points)**

Considérez les affichages des arbres binaires de recherche suivants. Pour chacun d'eux, donnez la représentation graphique de l'arbre. Dans chaque cas, dites si l'arbre binaire est un AVL.

4.1) **(5 points)** L'affichage post-ordre de l'arbre binaire de recherche donne :

3, 48, 47, 35, 58, 66, 55, 76, 99, 82, 71

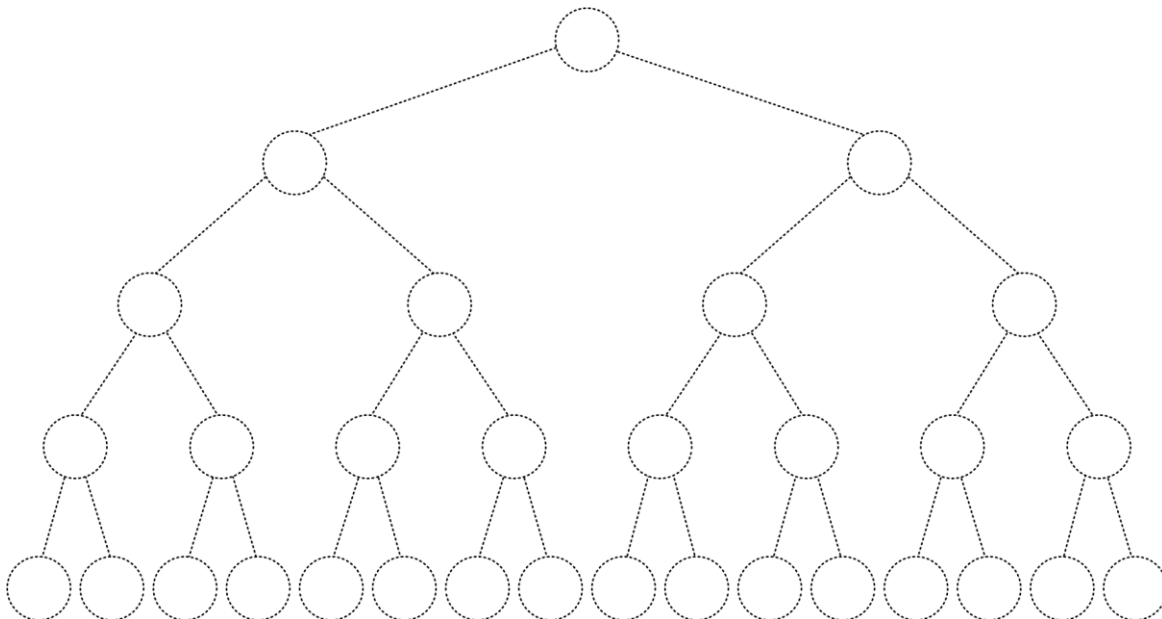
AVL? : _____



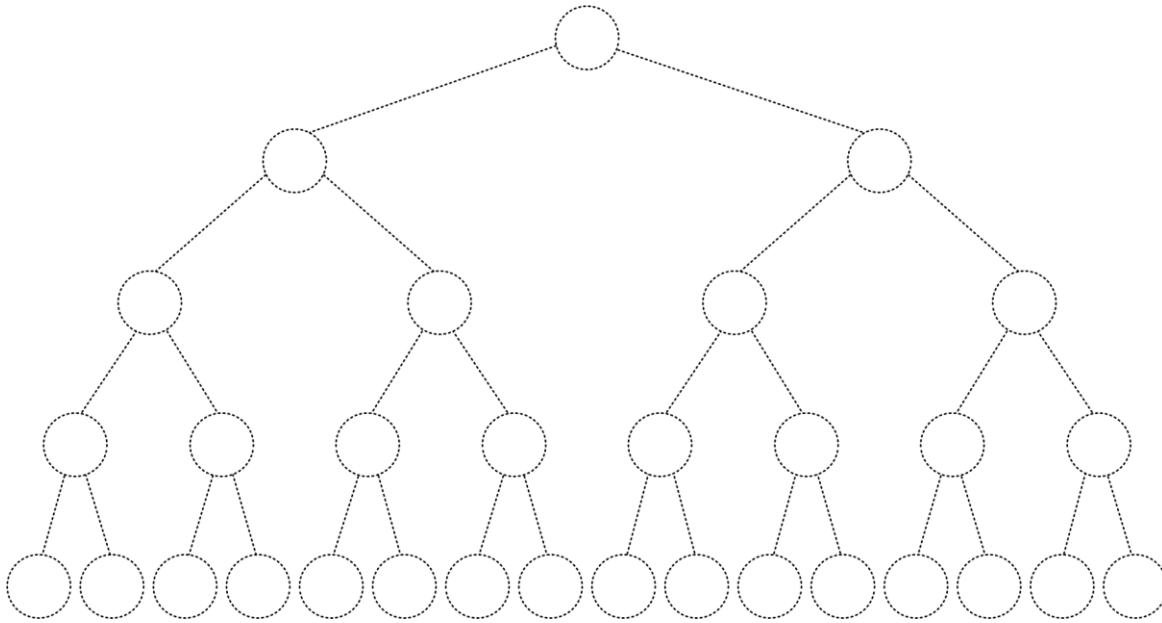
4.2) **(5 points)** L'affichage par niveaux de l'arbre binaire de recherche donne :

48, 35, 55, 3, 47, 71, 66, 82, 58, 76, 99

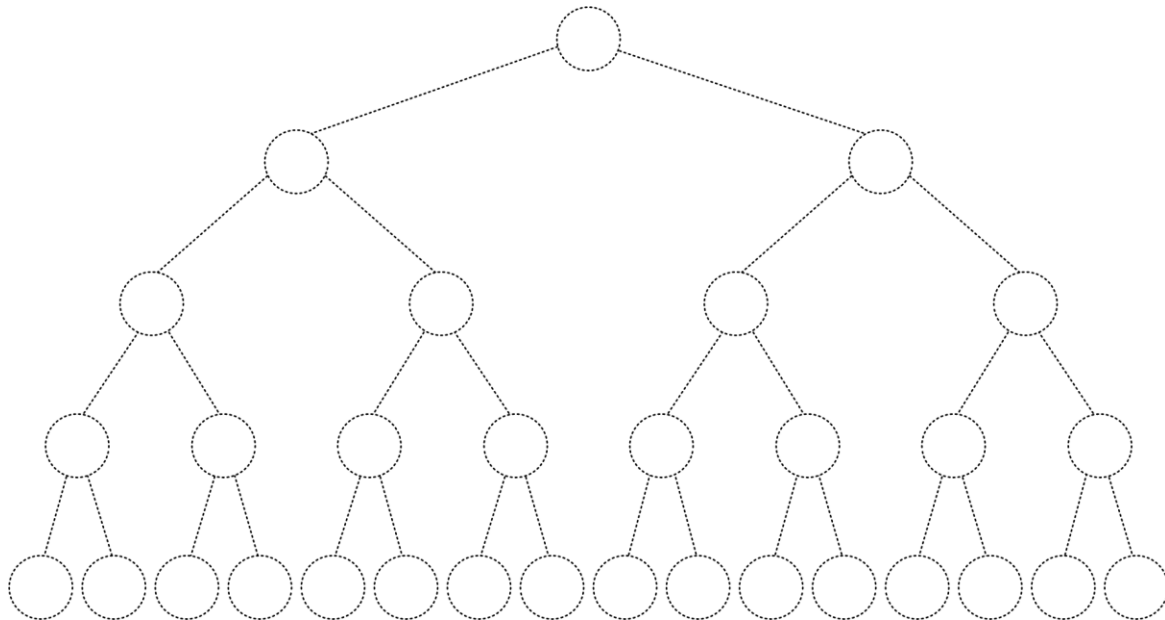
AVL? : _____



4.3) **(5 points)** Reprenez l'arbre de la question (4.1) et réalisez un *get()* de type Splay sur le nœud dont la clé est 48 :



4.4) **(5 points)** Reprenez l'arbre de la question (4.2) et réalisez un *delete()* de type Splay sur le nœud dont la clé est 71 :

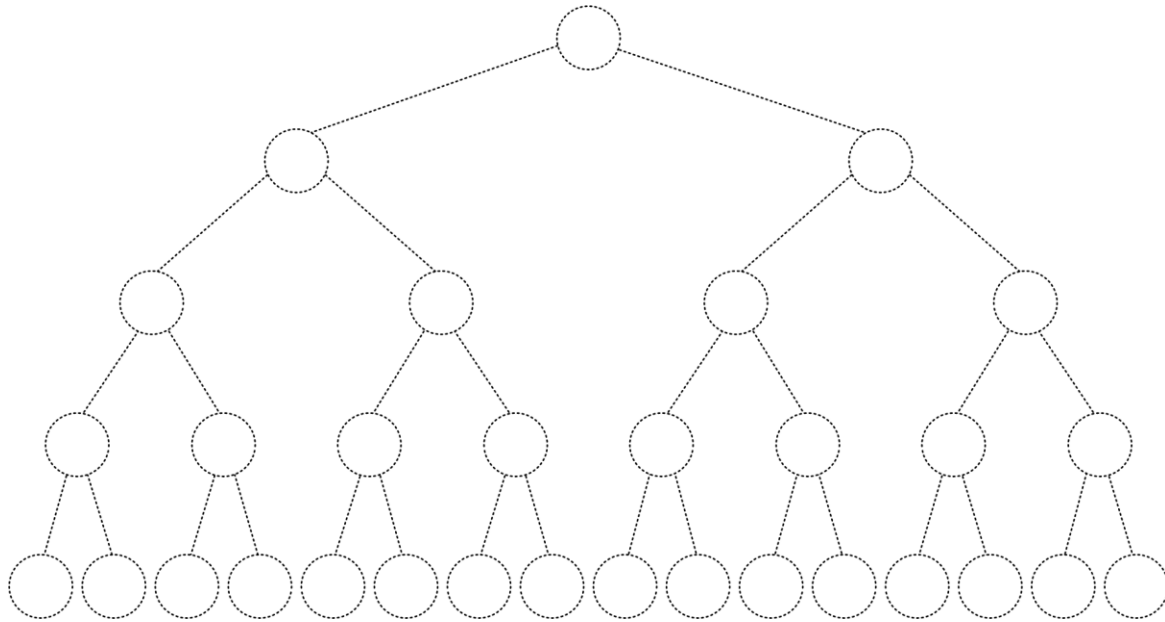


4.5) (**5 points**) L'affichage en ordre d'un arbre binaire de recherche ne permet pas d'en déduire la constitution. Néanmoins, sachant que :

- L'arbre est un arbre AVL ;
- La structure de l'arbre est symétrique par rapport au nœud racine ;
- La règle de symétrie s'applique récursivement à chaque nœud.

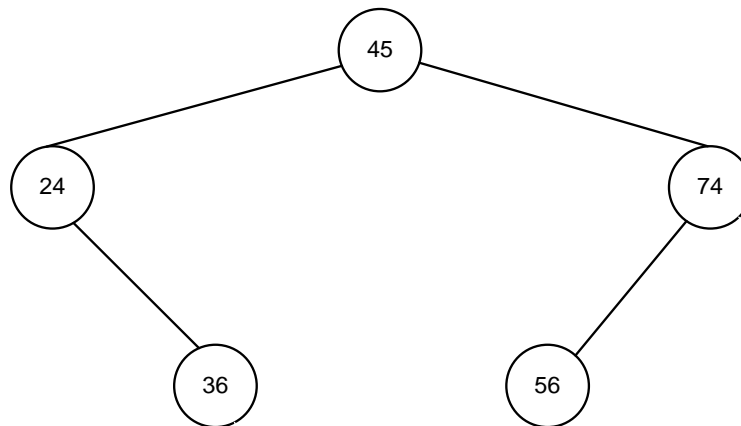
Donnez la représentation graphique de l'AVL dont l'affiche en-ordre est :

10, 21, 23, 41, 45, 46, 57, 61, 79, 80, 82, 90, 91, 95, 99



Question 5 : Arbre binaire de recherche de type AVL**(24 points)**

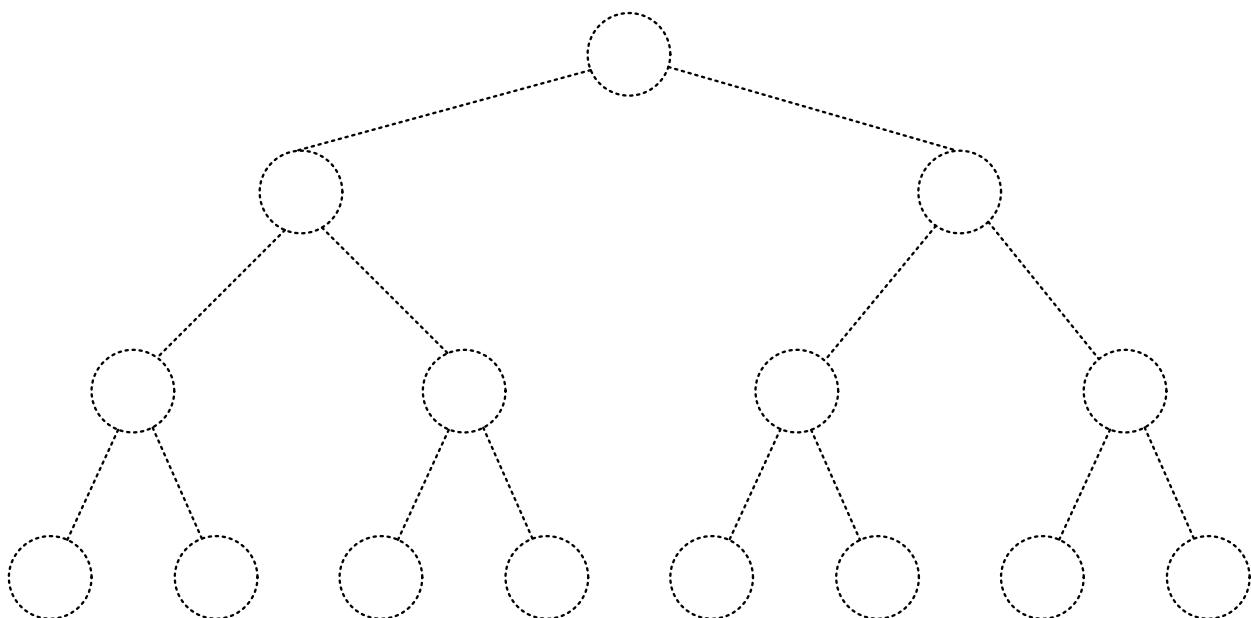
En considérant l'arbre AVL suivant :



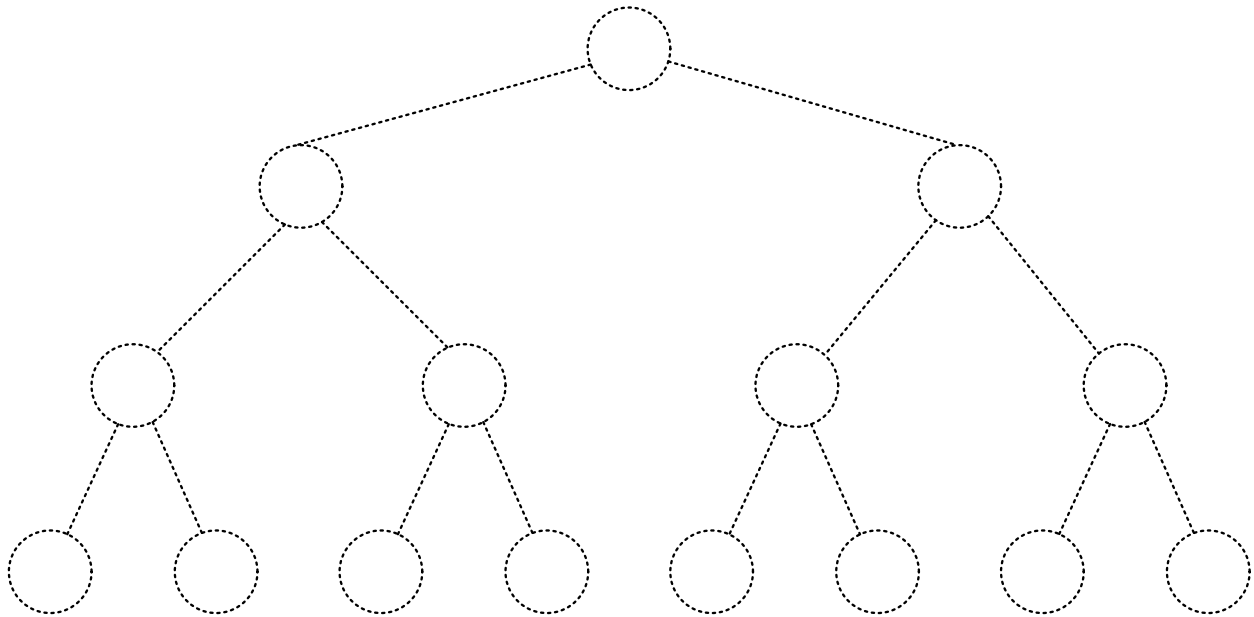
Effectuez l'ensemble des opérations suivantes dans l'ordre en vous servant des arbres ci-dessous. Les suppressions sont les mêmes que celles opérées sur un arbre binaire de recherche standard.

Insérez	40, 50
Supprimez	45
Insérez	45, 15, 38, 39
Supprimez	36

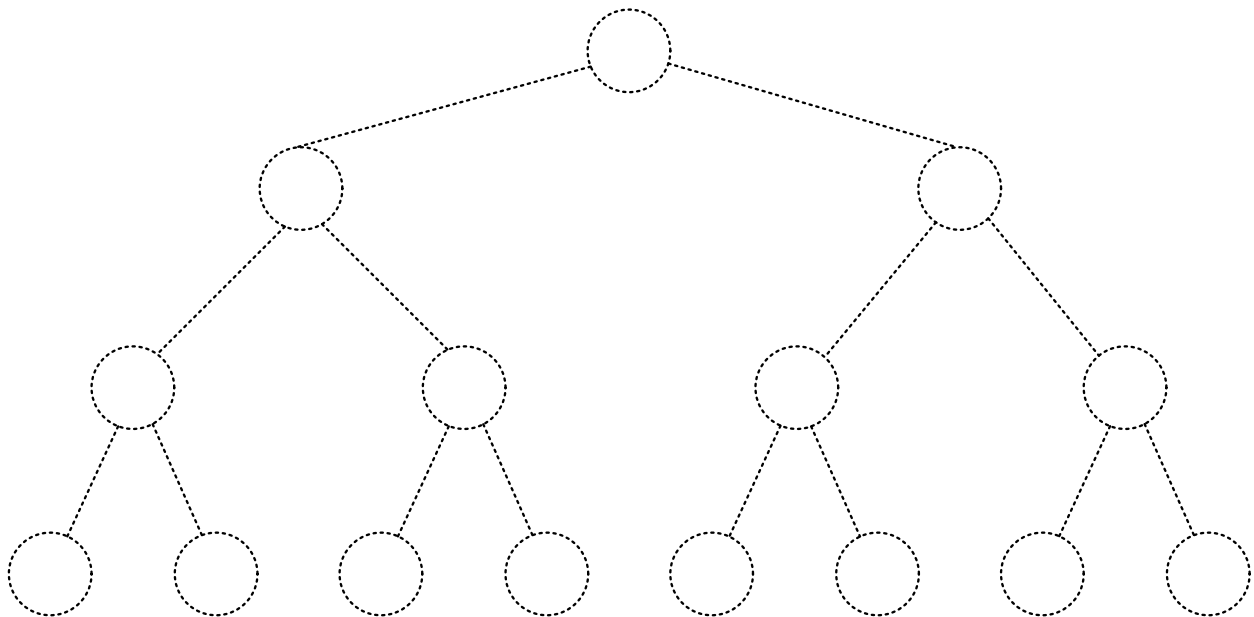
5.1) **(3 pnt)** Insérez 40.



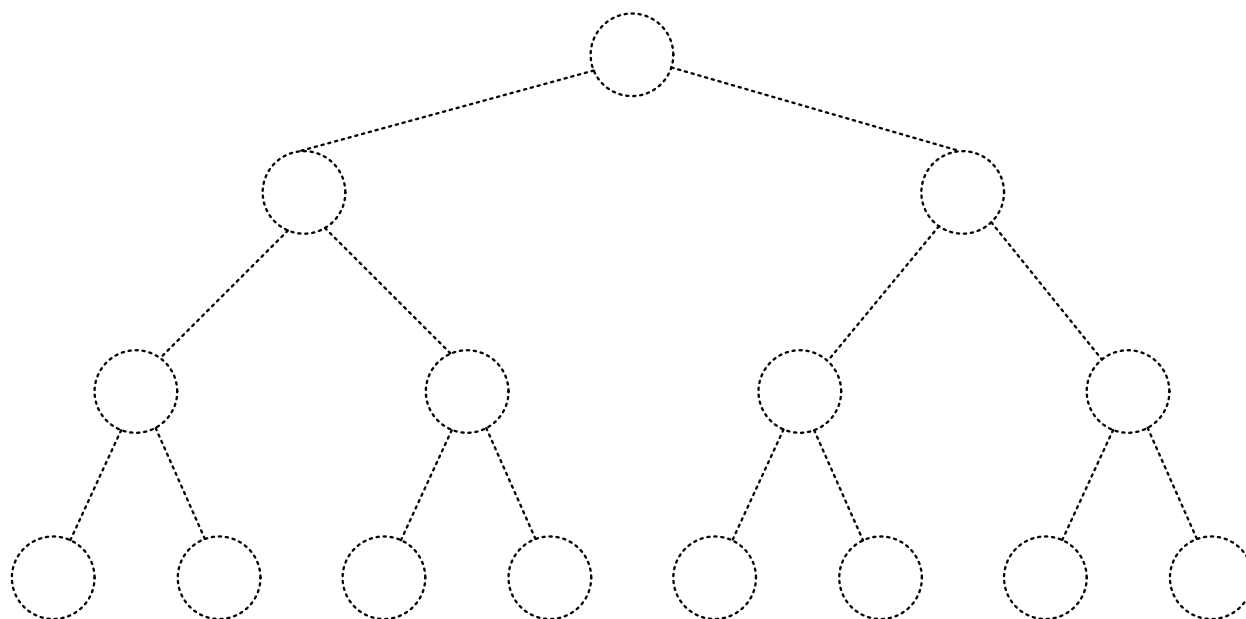
5.2) (3 **pnt**) Insérez 50.



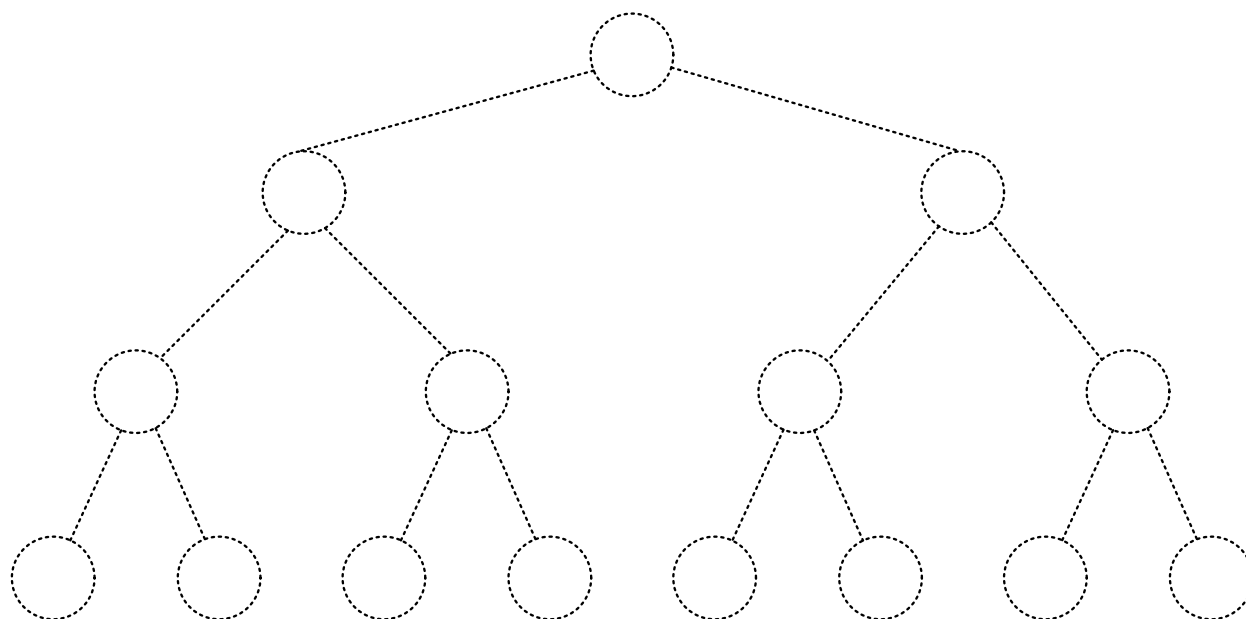
5.3) (3 **pnt**) Supprimez 45.



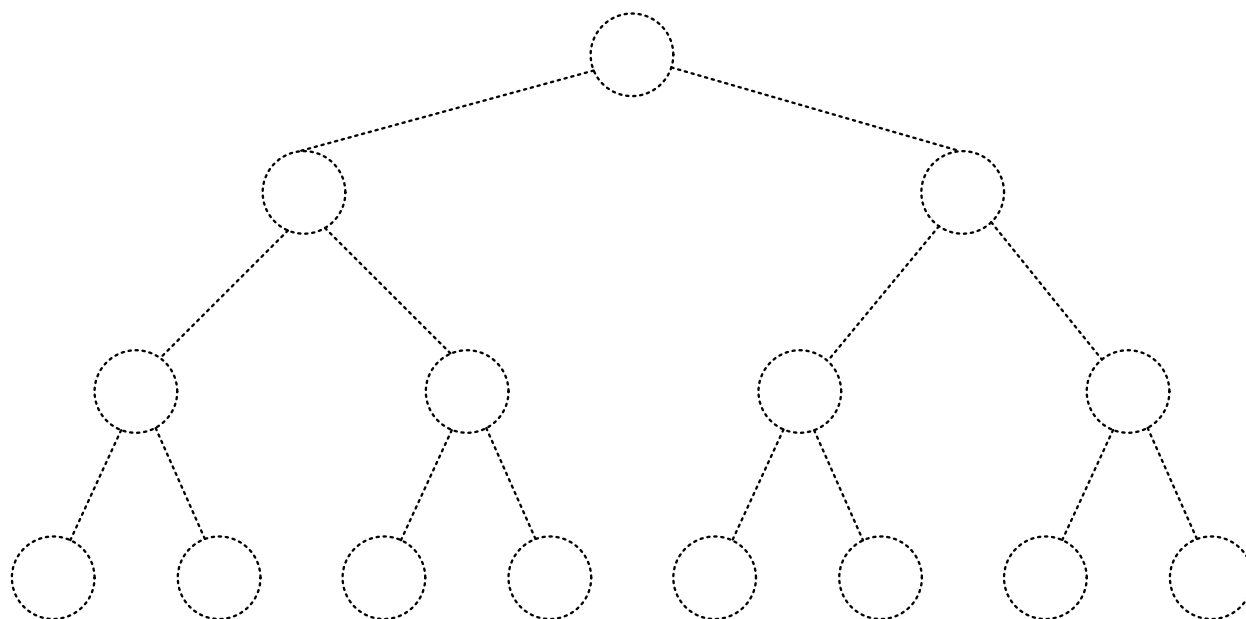
5.4) (3 **pnt**) Insérez 45.



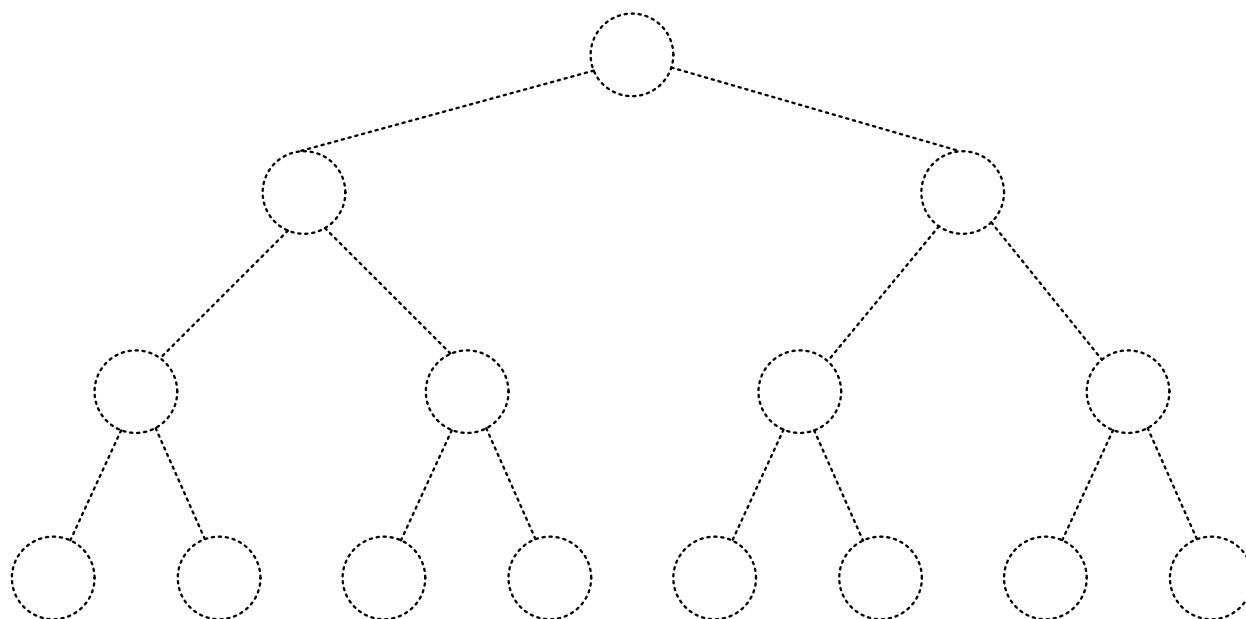
5.5) (3 **pnt**) Insérez 15.



5.6) (3 **pnt**) Insérez 38.



5.7) (3 **pnt**) Insérez 39.



5.8) (3 **pnt**) Supprimez 36.

