ÉCOLE POLYTECHNIQUE MONTRÉAL

Le génie sans frontières

Corrigé examen intra

INF2010

Sigle du cours

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

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

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

Directives particulières

Bonne chance à tous!

Important

Cet examen contient **5** questions sur un total de **14** 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 : 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.

Posons $N=2^k$ (les autres cas se généralisent facilement)

Dans le cas de `DeuxPuisanceV1` :

Le premier niveau de récursion ($N=2^k$) produira 2 appels ;
Le second niveau de récursion ($N/2=2^{k-1}$) produira 4 appels ;
Le troisième niveau de récursion ($N/4=2^{k-2}$) produira 8 appels ;
…
Le $k\text{-}1^e$ niveau de récursion ($N/2^{k-2}=4$) produira 2^{k-1} appels ;
Le k^e niveau de récursion ($N/2^{k-1}=2$) produira 2^k appels ;
Le $k+1^e$ niveau de récursion ($N/2^k=1$) produira 2^{k+1} appels ;
Le $k+2^e$ niveau de récursion ($N/2^{k+1}$) sortira;

On a donc $T(N) = 2+4+8+\ldots+2^{k-1}+2^k+2^{k+1} = 2(1+2+4+\ldots+2^{k-2}+2^{k-1}+2^k)$
$= 2(2^{k+1}-1) = 4\cdot 2^k-2=4N-2.$

Dans le cas de `DeuxPuisanceV1`, on a donc O(N).

Dans le cas de `DeuxPuisanceV2` :

Le premier niveau de récursion ($N=2^k$) produira 1 appel ;
Le second niveau de récursion ($N/2=2^{k-1}$) produira 1 appel ;
Le troisième niveau de récursion ($N/4=2^{k-2}$) produira 1 appel ;
…
Le $k-1^e$ niveau de récursion ($N/2^{k-2}=4$) produira 1 appel ;
Le k^e niveau de récursion ($N/2^{k-1}=2$) produira 1 appel ;
Le $k+1^e$ niveau de récursion ($N/2^k=1$) produira 1 appel ;
Le $k+2^e$ niveau de récursion ($N/2^{k+1}$) sortira ;

On a donc T(N) = 1+1+1+…+1+1+1 = k+1 = log(N) + 1.

Dans le cas de `DeuxPuisanceV2`, on a donc O(log(N)).

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 DeuxPuisanceEquivalentNonRecursif(int N)
{

   if( N<0 ) throw new IllegalArgumentException();

   int pow = 1;

   for(int p = 1; p <= N; p++)
      pow *= 2;

   return pow;
}
```

Estimation de complexité asymptotique : O(N)

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.

```
public static int DeuxPuisanceOde1(int N)
{
   if( N<0 ) throw new IllegalArgumentException();

   int pow = 1 << N;

   return pow;
}
```

Question 2 : Tables de dispersement (20 points)

Considérant une table de dispersement double où Hash(X) = (H_1(X) + i·H_2(X)) % N, et où H_1(X) = (X^2) % N, et H2(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.

O(1) puisqu'il s'agit d'un hachage typique.

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 : $H_1(2) = 4$, $H_2(2) = 5$, Hash(2) = (4+0·5) % 11 = 4 → pas de collision
5 : $H_1(5) = 3$, $H_2(5) = 2$, Hash(2) = (3+0·2) % 11 = 3 → pas de collision
6 : $H_1(6) = 3$, $H_2(6) = 1$, Hash(2) = (3+2·1) % 11 = 5 → deux collisions
4 : $H_1(4) = 5$, $H_2(4) = 3$, Hash(2) = (5+1·3) % 11 = 8 → une collision
7 : $H_1(7) = 5$, $H_2(7) = 7$, Hash(7) = (5+1·7) % 11 = 1 → une collision
9 : $H_1(9) = 4$, $H_2(9) = 5$, Hash(9) = (4+1·5) % 11 = 9 → une collision

Il y a donc un total de 5 collisions.

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

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

Il y a un total de cinq (5) appels.

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

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

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

insertionSort() est appelée sur les vecteurs

0
4, 2, 3 et
7, 8, 6, 9

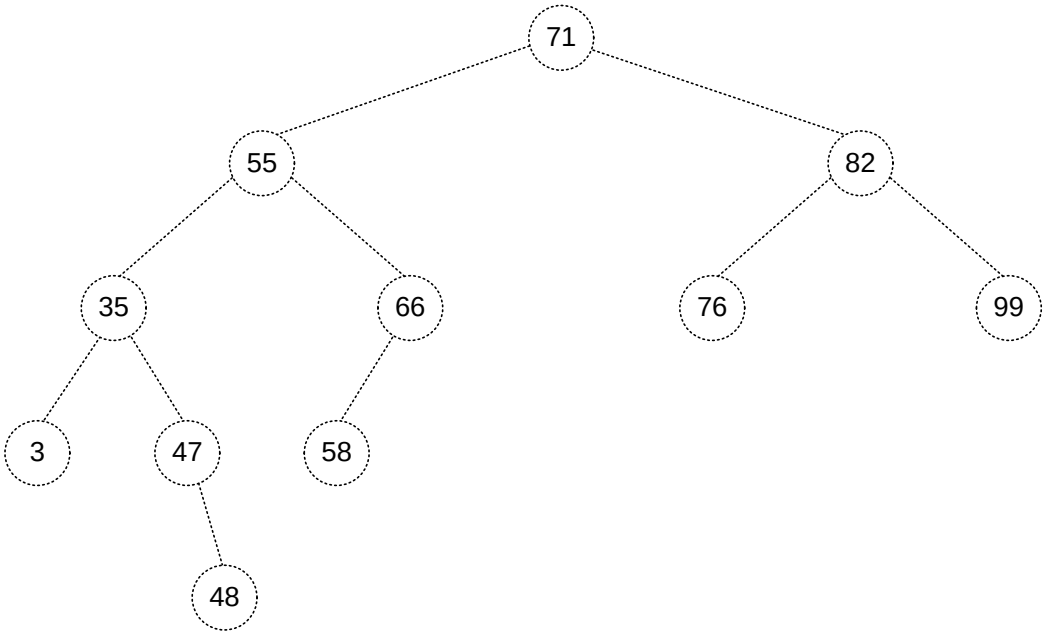
La plus petite taile du sous-vecteur sur lequel *insertionSort*() est appelé est donc 1.

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, dîtes si l'arbre binaire est un AVL.

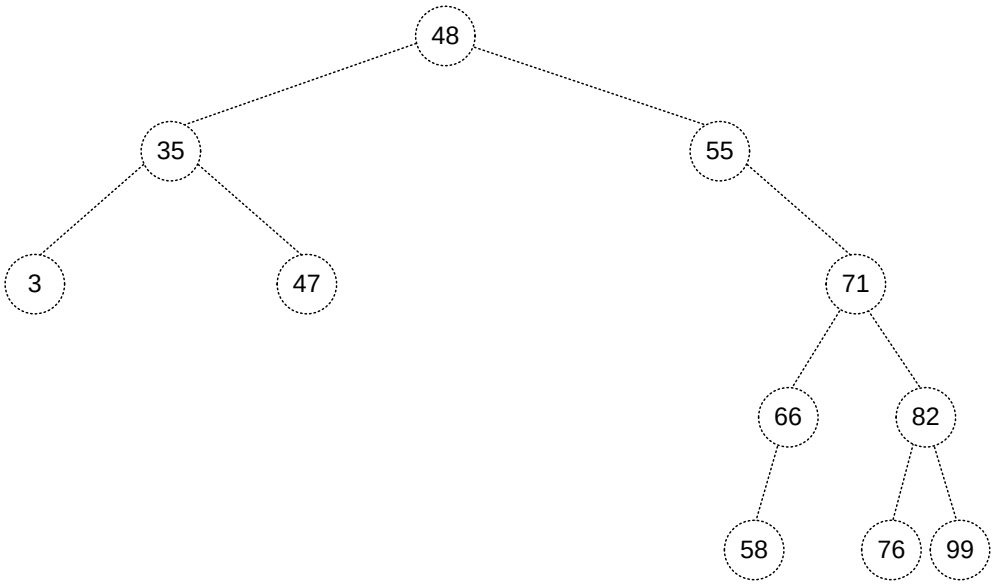
4.1) **(5 points)** L'affichage post-ordre de l'arbre binaire de recherche donne :

3, 48,47, 35, 58, 66, 55, 76, 99, 82, 71 AVL? : NON

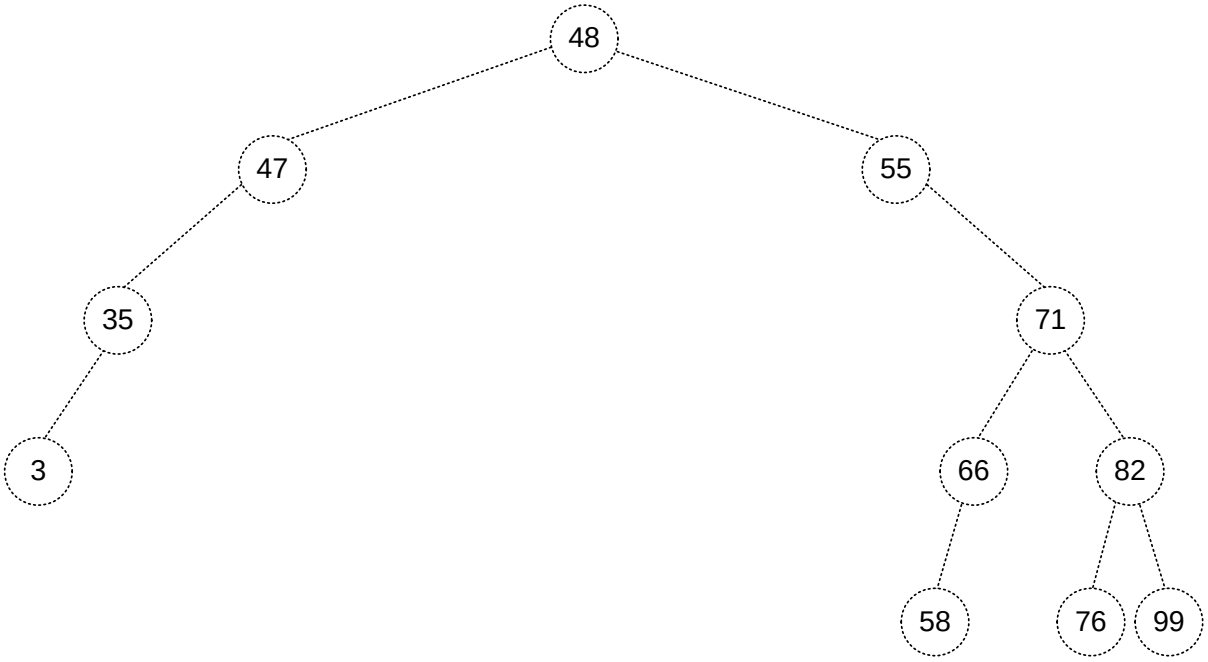


4.2) **(5 points)** L'affichage par niveaux de l'arbre binaire de recherche donne :

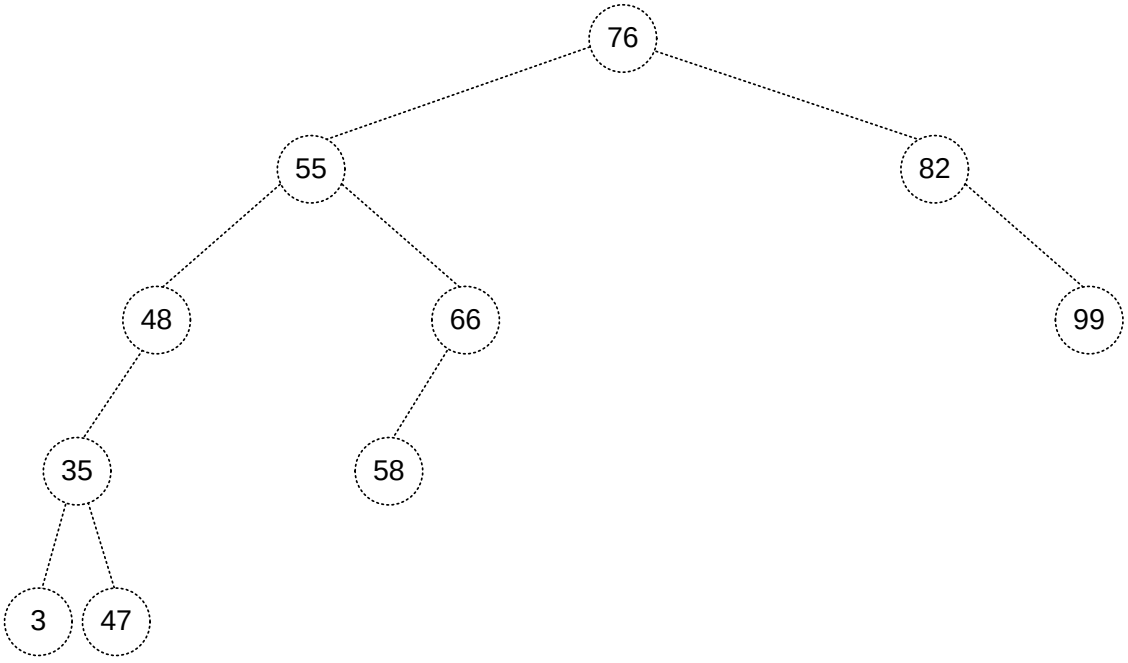
48, 35, 55, 3, 47, 71, 66, 82, 58, 76, 99 AVL? : NON



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 :

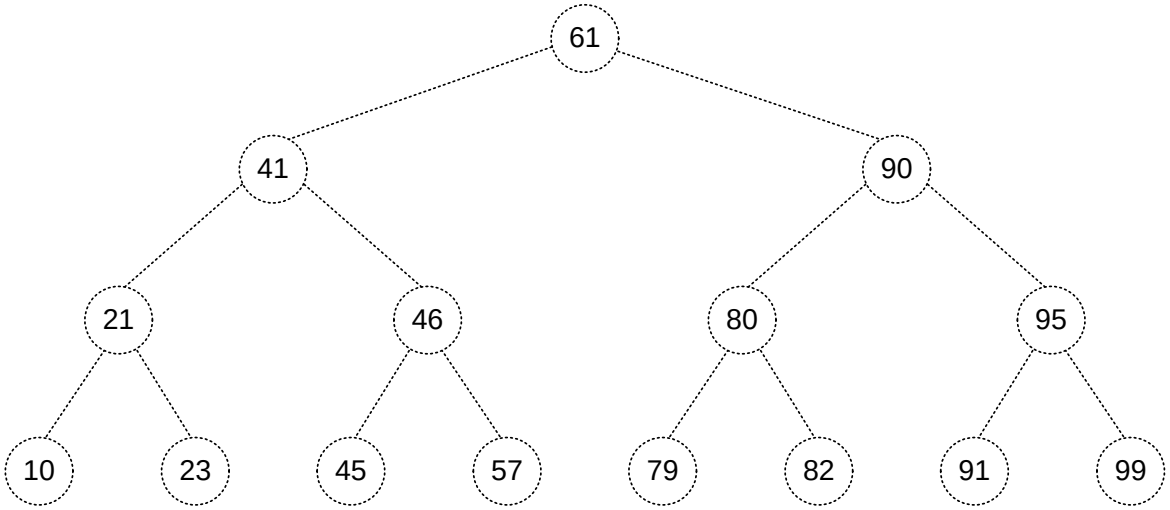


4.5) **(5 points)** L'affichage en ordre d'un arbre binaire de recherche ne permet pas d'en déduire la constitution. Néanmoins, sachant que :

- 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.

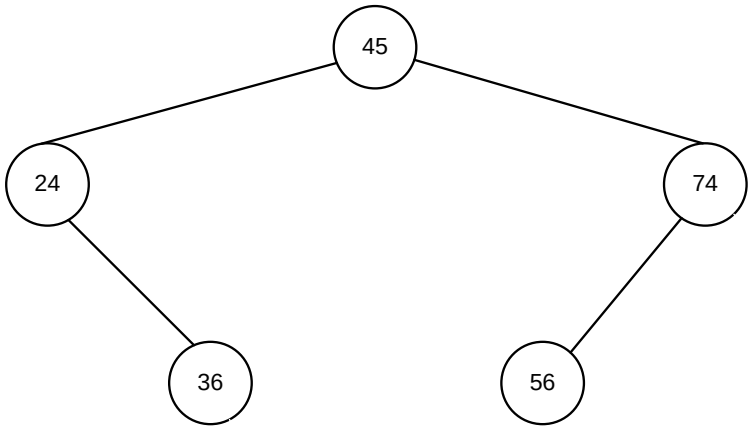
Donnez la représentation graphique de l'AVL dont l'affiche en-ordre est :

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)

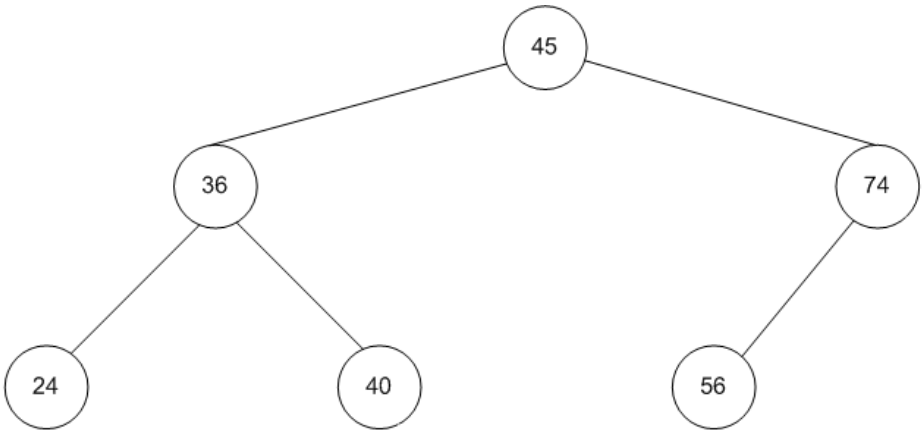
En considérant l'arbre AVL suivant :



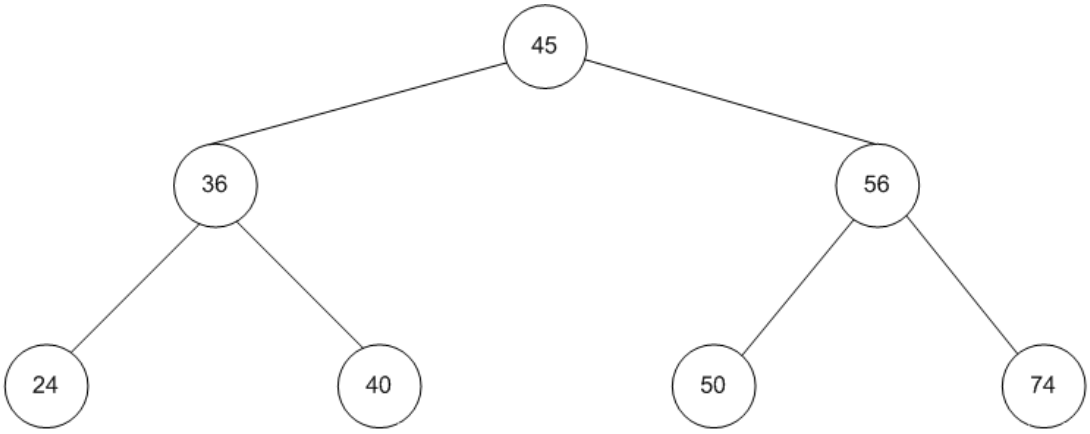
Effectuez l'ensemble des opérations suivantes dans l'ordre en vous servant des arbres ci-dessous. Les suppressions sont les mêmes que celles opérées sur un arbre binaire de recherche standard.

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

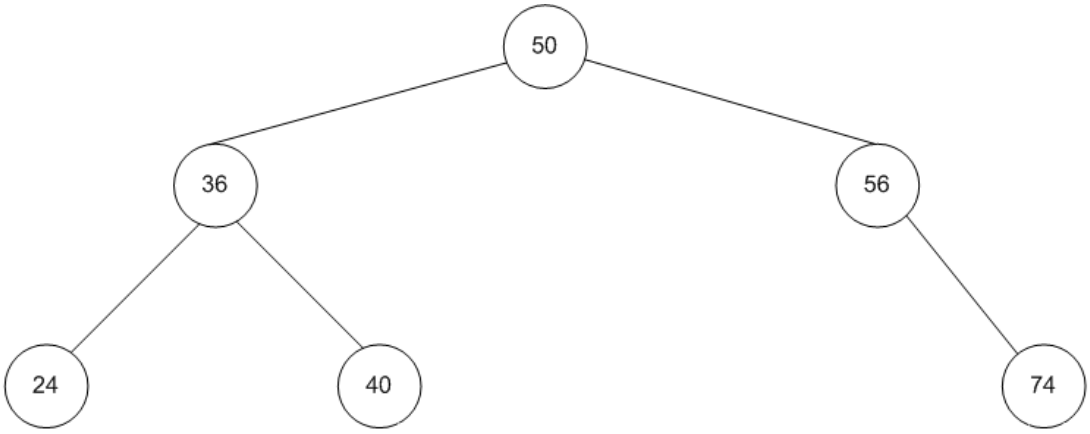
5.1) **(3 pnt)** Insérez 40.



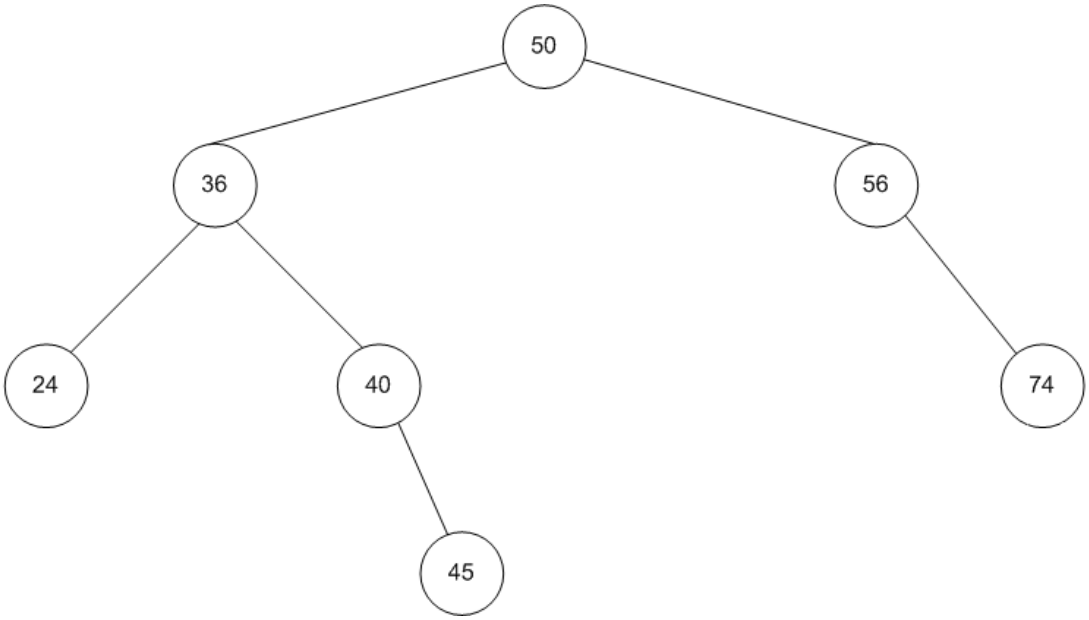
5.2) **(3 pnt)** Insérez 50.



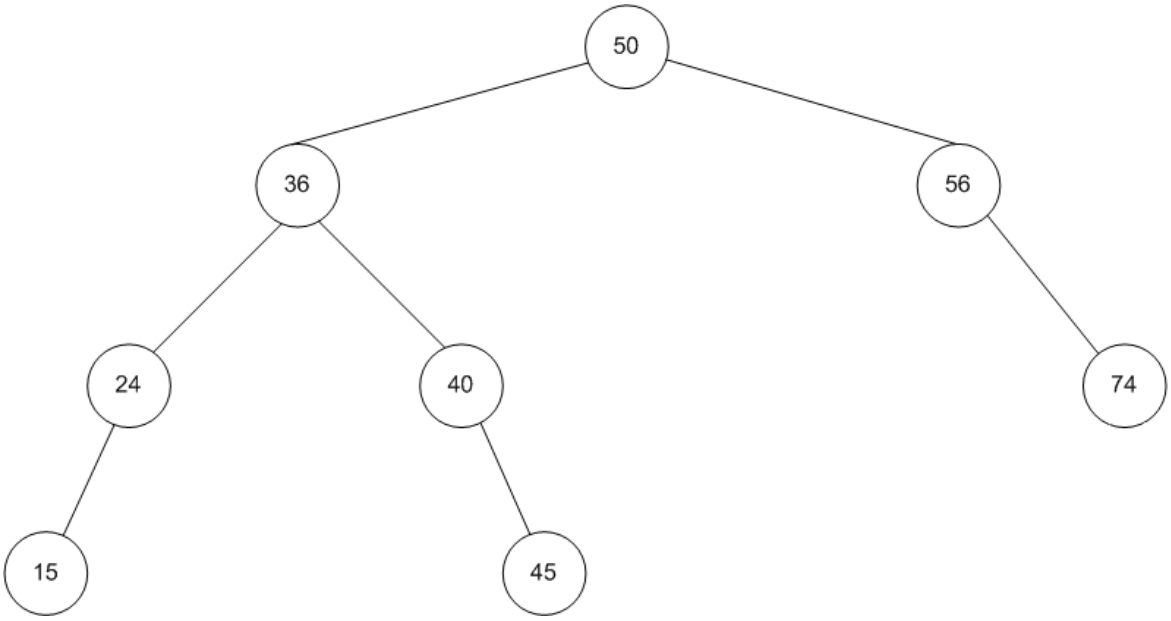
5.3) **(3 pnt)** Supprimez 45.



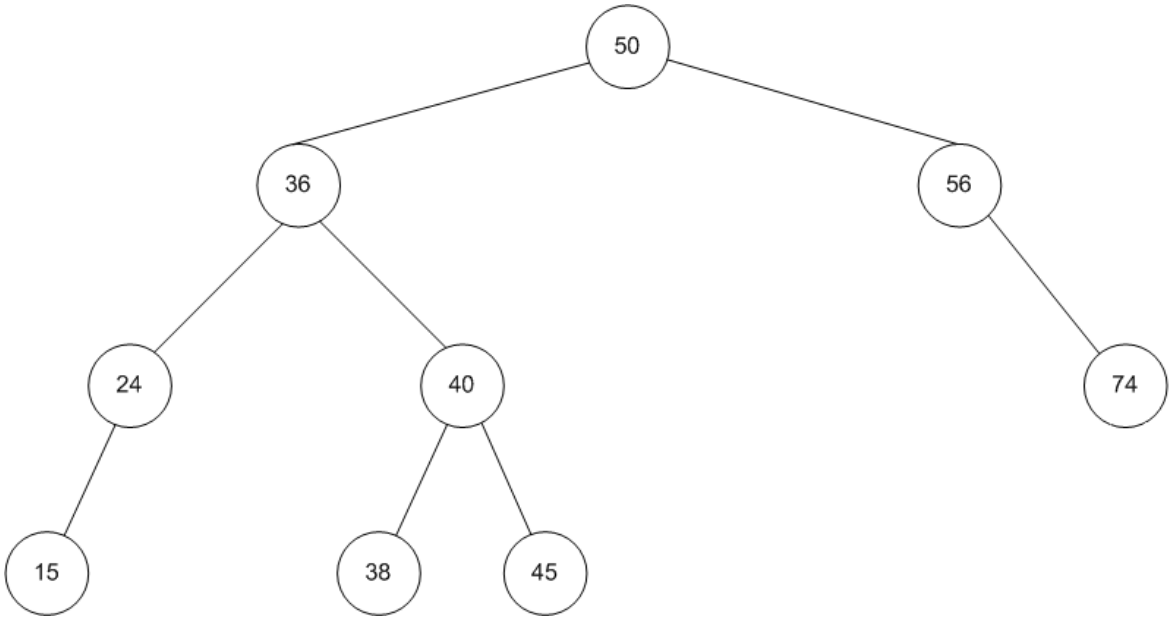
5.4) **(3 pnt)** Insérez 45.



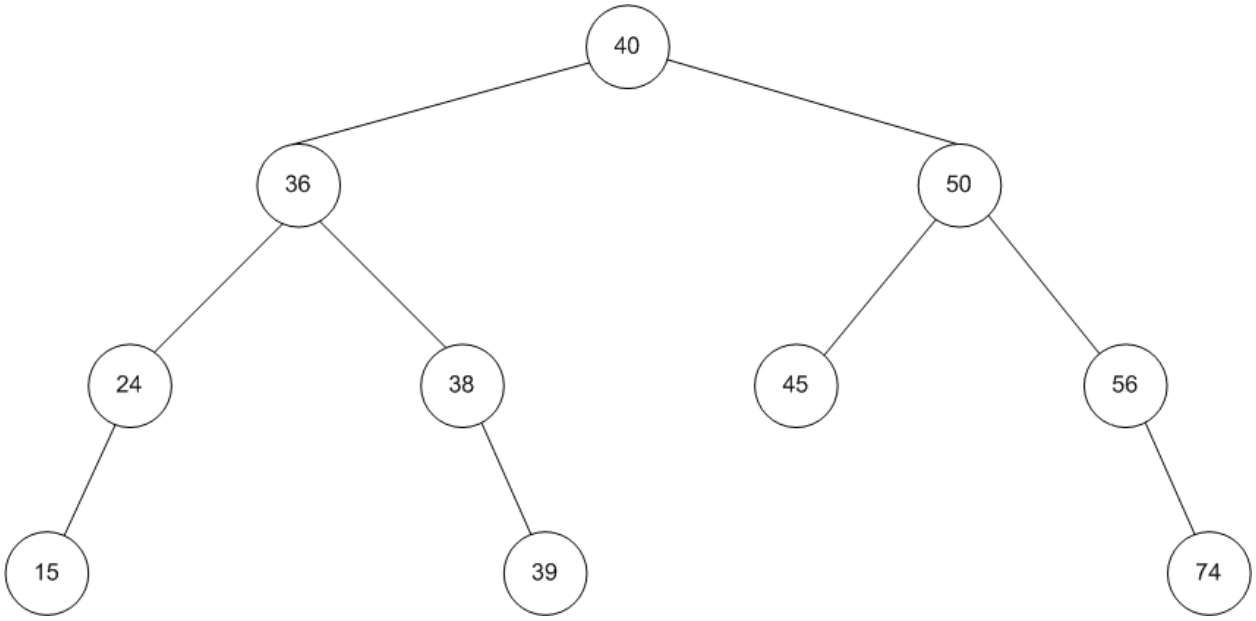
5.5) **(3 pnt)** Insérez 15.



5.6) **(3 pnt)** Insérez 38.



5.7) **(3 pnt)** Insérez 39.



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