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**	**20091**
Professeur			***Local***	***Téléphone***
Ettore Merlo, responsable – Tarek Ould Bachir, chargé de cours			**B-418**	**5758**
Jour	***Date***	***Durée***		***Heure***
Mercredi	**18 février 2009**	**2h00**		**9h00**

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

Directives particulières

Bonne chance à tous!

Important

Cet examen contient **5** questions sur un total de **17** 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 (16 points)

Considérez une table de dispersement par débordement progressif de dimension 11 avec "*sondage*" à double dispersement (f(i) = i * $h_2(x)$), avec fonction de dispersement primaire $h_1(x)$ = x % 11 et avec fonctions de dispersement secondaire $h_2(x)$ = 7 - (x % 7).

1.1) **(12 pnt)** Quelles sont les valeurs i et $h_i(x)$ utilisées pour l'insertion dans l'ordre de chaque clef 36, 80, 16, 47. Détaillez les calculs.

i) 36
i = 0; 36%11 = 3

ii) 80
i = 0; 80%11=3
i = 1; 3 + 7 – (80%7) = 7

iii) 16
i = 0; 16%11 = 5

iv) 47
i = 0; 47 % 11 = 3
i = 1; 3 + 7 – (47%7) = 5
i = 2; 5 + 7 – (47%7) = 7
i = 3; 7 + 7 – (47%7) = 9

1.2) **(4 pnt)** Dessinez l'état de la table à la fin de l'insertion de toutes les clefs indiquées.

Index	0	1	2	3	4	5	6	7	8	9	10
Clé				**36**		**16**		**80**		**47**	

Question 2 : Tris en *n* log(*n*) (16 points)

Exécuter l'algorithme « MergeSort » et « QuickSort » pour trier le vecteur suivant avec un index de position qui commence à zéro. La valeur *cut-off* pour l'algorithme « QuickSort » est 10.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valeurs	47	99	83	49	12	63	19	59	69	31	12	3	51	47	9	18

Réponse :

2.1) **(6 pnt)** « MergeSort » :

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valeurs	47	99	83	49	12	63	19	59	69	31	12	3	51	47	9	18
	47	**99**	**49**	**83**	**12**	**63**	**19**	**59**	**31**	**69**	**3**	**12**	**47**	**51**	**9**	**18**
	47	**49**	**83**	**99**	**12**	**19**	**59**	**63**	**3**	**12**	**31**	**69**	**9**	**18**	**47**	**51**
	12	**19**	**47**	**49**	**59**	**63**	**83**	**99**	**3**	**9**	**12**	**18**	**31**	**47**	**51**	**69**
	3	**9**	**12**	**12**	**18**	**19**	**31**	**47**	**47**	**49**	**51**	**59**	**63**	**69**	**83**	**99**

2.2) **(10 pnt)** « QuickSort » :

a) **(6 pnt)** Partie « QuickSort »

Les trois valeurs de « Median3 » : **47, 59, 18** Valeur médiane : **47**

État du vecteur après l'exécution de Median 3

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valeur	**18**	99	83	49	12	63	19	**9**	69	31	12	3	51	47	**47**	**59**

État du vecteur après l'exécution du premier round du tri QuickSort (partitionnement)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valeur	**18**	**47**	**3**	**12**	**12**	**31**	**19**	**9**	**47**	**63**	**49**	**83**	**51**	**99**	**69**	**59**

b) **(3 pnt)** Recours à « InsertionSort » après cut-off

État du vecteur à la gauche de la médiane après InsertionSort :

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valeur	**3**	**9**	**12**	**12**	**18**	**19**	**31**	**47**								

État du vecteur à la droite de la médiane après InsertionSort :

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valeur	**49**	**51**	**59**	**63**	**69**	**83**	**99**									

c) **(1 pnt)** Résultat final

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valeur	**3**	**9**	**12**	**12**	**18**	**19**	**31**	**47**	**47**	**49**	**51**	**59**	**63**	**69**	**83**	**99**

Question 3 Tri linéaire (variation du *BubbleSort*) (20 points)

Considérer le code source Java donné à l'annexe 1 pour trier le vecteur qui suit :

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

3.1) **(10 pnt)** Si une boucle n'est pas exécutée, laisser la case vide

3.1.a)
```
While: st = 0 ; limit = 7
    For 1: j= 0 ... 6
```

État du vecteur après la boucle

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

```
    For 2: j= 6 ... 0
```

État du vecteur après la boucle

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

3.1.b)
```
While: st = 1 ; limit = 6
    For 1: j= 1 ... 5
```

État du vecteur après la boucle

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

```
    For 2: j= 5 ... 1
```

État du vecteur après la boucle

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

3.1.c)
```
While: st = 2 ; limit = 5
    For 1: j= 2 ... 4
```

État du vecteur après la boucle

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

```
For 2: j= 4 ... 2
```

État du vecteur après la boucle

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

3.1.d)
```
While: st = 3 ; limit = 4
For 1: j= 3 ... 3
```

État du vecteur après la boucle

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

```
For 2: j= 3 ... 3
```

État du vecteur après la boucle **(NON EXÉCUTÉ)**

Index	0	1	2	3	4	5	6	7
Valeurs								

3.2) **(4 pnt)** Donner une estimation de complexité algorithmique en pire et en meilleur cas. Justifier brièvement.

La boucle while s'exécute au plus n/2 fois. À chaque exécution de la boucle while, les deux boucles internes exécutent deux itérations de moins qu'à l'itération précédente. Nous aurons en pire cas :

$$\sum_{i=0}^{\frac{n}{2}-1} 2(n-1-2i) = \frac{n}{2}2(n-1) - 4\sum_{i=0}^{\frac{n}{2}-1} i \quad = n(n-1) - 2\left(\frac{n}{2}-1\right)\left(\frac{n}{2}+1\right) \quad = n(n-1) - 2\left(\frac{n}{2}-1\right)\left(\frac{n}{2}+1\right)$$

$$= n^2 - n - 2\left(\frac{1}{4}n^2 - 1\right) \quad = \frac{1}{2}n^2 - n + 2$$

Nous avons donc une complexité en $O(n^2)$ en pire cas.

Le meilleur cas survient lorsque le vecteur est déjé trié, auquel cas la boucle while ne s'exécute qu'une seule fois et la première boucle for une seule fois. On a donc une complexité en $O(n)$ en meilleur cas.

3.3) **(6 pnt)** Comparer le temps d'exécution de cet algorithme à celui de BubbleSort (plus lent, plus rapide). Argumenter la réponse.

Du fait qu'il effectue deux passes en sens inverse au lieu d'une, l'algorithme DoubleBubbleSort (également appelé Cocktail Sort) est plus rapide que le BubbleSort classique. Il est intéressant de noter que le nombre d'itérations de BubbleSort est de $0.5n^2$-$3n$+1 ; ceci signifie qu'en pire cas, BubbleSort peut être plus rapide que DoubleBubbleSort, en autant que la condition de sortie ne soit pas appliquée.

Question 4 : Arbre binaire de recherche (30 points)

Considérer le code source Java donné aux annexes 2 et 3 pour manipuler un arbre binaire de recherche :

4.1) **(2 pnt)** Donner le résultat du premier affichage (annexe 2) :

```
// Creation d'un arbre vide
BinarySearchTree<Integer> t = new BinarySearchTree<Integer>( );

/** Premier affichage: Etat de l'arbre*/
System.out.println( "Premier Arbre:" );
t.printTree( );
```

Utiliser la grille suivante pour ce faire ; chaque colonne représente un caractère :

P	r	e	m	i	e	r		A	r	b	r	e			
|	_	_	n	u	l	l									

4.2) **(4 pnt)** Donner dans l'ordre les valeurs **j** insérées dans l'arbre par l'appel (annexe 2) :

```
// Remplissage de l,arbre avec des valeurs
for(int i=0; i < PROFONDEUR_MAX; ++i)
        for(int j = ( 1 << (3-i) ); j < ( 1 << 4 ); j += ( 1 << (4-i) ) )
                t.insert( j );
```

Pour mémoire, l'opérateur << effectue un décalage à gauche, de sorte que :

```
int i=1;
i << 0;// 1
i << 1;// 2
i << 2;// 4
i << 3;// 8
// etc...
```

Dans l'ordre, les valeurs insérées sont :

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

4.3) **(3 pnt)** Donner dans l'ordre les valeurs **i** retirées de l'arbre par l'appel (annexe 2) :

```
// Retirer certaines valeurs
for(int i =  (1 << PROFONDEUR_MAX) - 1; i > (1 << 3); i -= 1)
        t.remove( i );
```

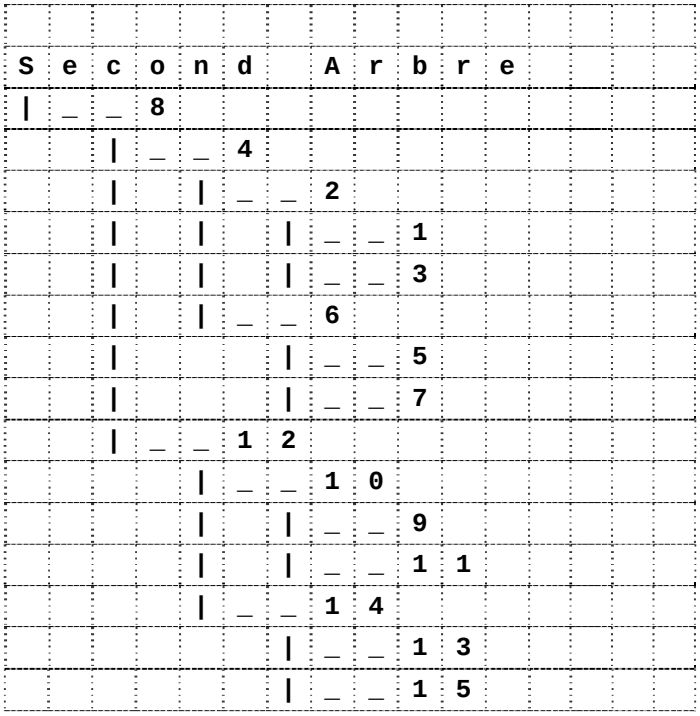
Dans l'ordre, les valeurs retirées sont :

15 ; 14 ; 13 ; 12 ; 11 ; 10 ; 9

4.4) **(6 pnt)** Donner le résultat du deuxième affichage (annexe 2) :

```
/** Second affichage: Etat de l'arbre*/
System.out.println( "\nSecond Arbre:" );
// Remplissage de l'arbre avec des valeurs
// ... code 4.2)
t.printTree( );
```

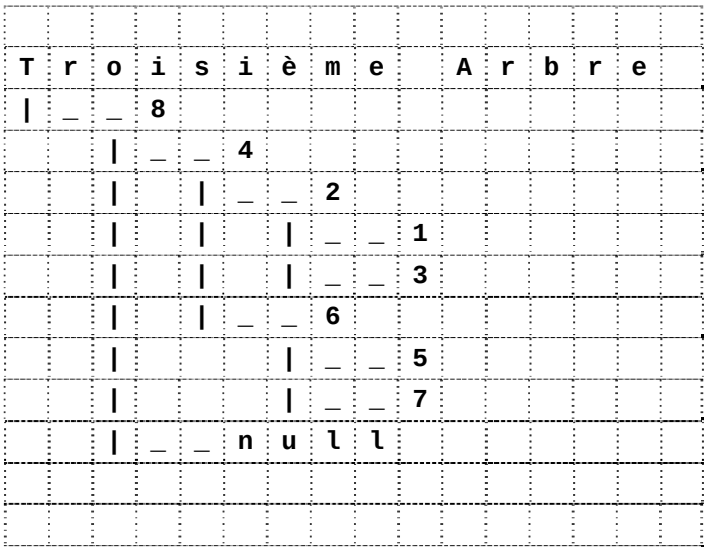
Utiliser la grille suivante pour ce faire ; chaque colonne représente un caractère :



4.5) **(6 pnt)** Donner le résultat du quatrième affichage (annexe 2) :

```
/** Quatrième affichage: Etat de l'arbre*/
System.out.println( "\nTroisième Arbre:" );
// Retirer certaines valeurs
// ... code 4.3)
t.printTree( );
```

Utiliser la grille suivante pour ce faire ; chaque colonne représente un caractère :



4.6) **(4 pnt)** Donner le résultat du troisième et cinquième affichage (annexe 2) :

Appel	Affichage	Réponse
a) `/** Troisième affichage: Est-ce un AVL?*/`	`Ceci est un arbre AVL.`	X
	`Ceci n'est pas un arbre AVL.`	
b) `/** Cinquième affichage: Est-ce un AVL?*/`	`Ceci est un arbre AVL.`	
	`Ceci n'est pas un arbre AVL.`	X

4.7) **(5 pnt)** Considérer le code de la méthode **`isAVL()`** de l'annexe 3 et proposez une amélioration permettant d'obtenir un gain de temps d'exécution de facteur 2. Justifier votre réponse.

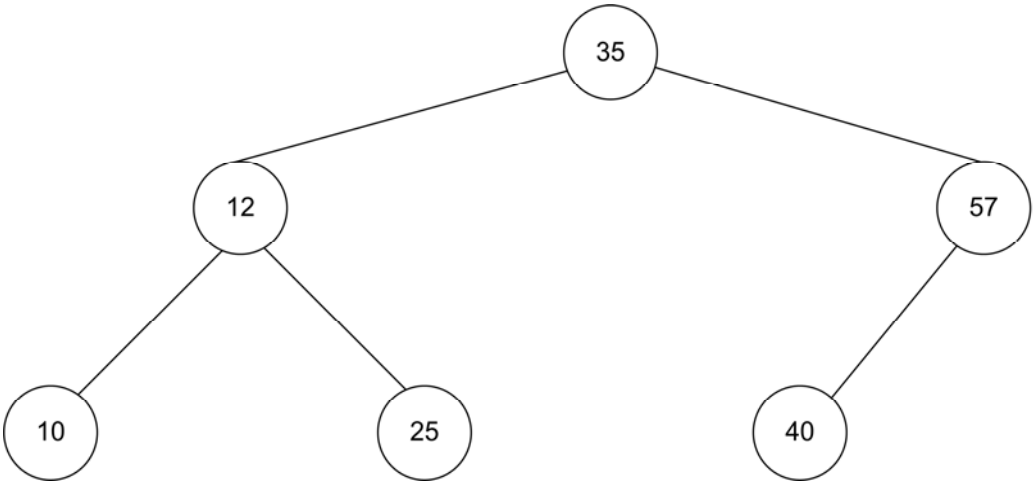
```
/**
 * Verifie si l'arbre est un AVL
 */
public boolean isAVL()
{
        if( !isBST(root) ) return false;
        return isAVL(root);
}

// …
private boolean isAVL(BinaryNode<AnyType> node)
{
        if (node==null) return(true);

        int h1 = height( node.left );
        int h2 = height( node.right );
        if( Math.abs( h1 - h2 ) >1 ) return false;

        if( !isAVL( node.left )) return false;
        return isAVL( node.right );
}
```

Puisque l'arbre est forcément un arbre binaire de recherche, il suffit de retirer la ligne :

```
if( !isBST(root) ) return false;
```

Question 5 : Arbre binaire de recherche de type AVL (18 points)

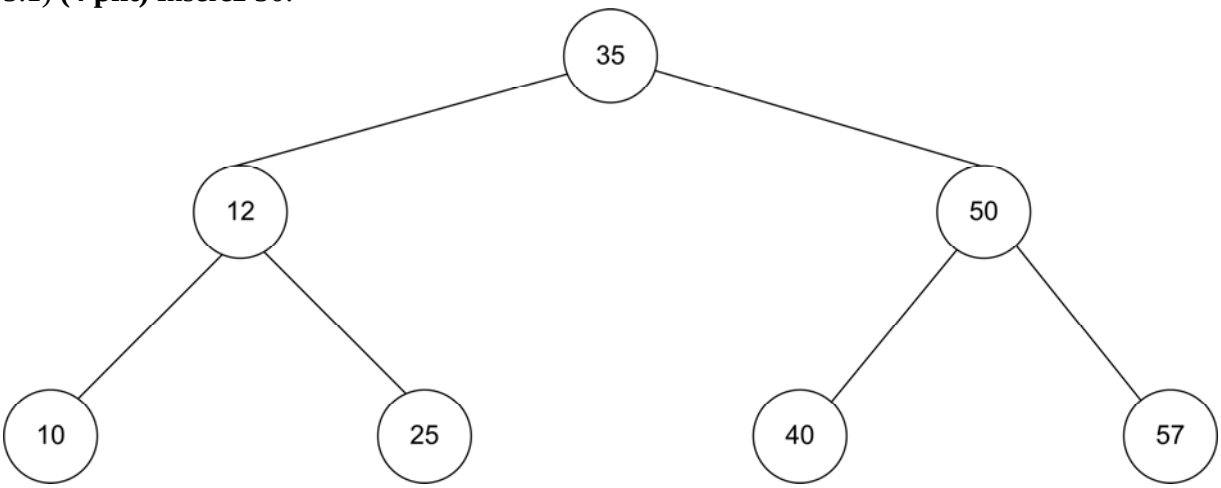
En considérant l'arbre AVL suivant :



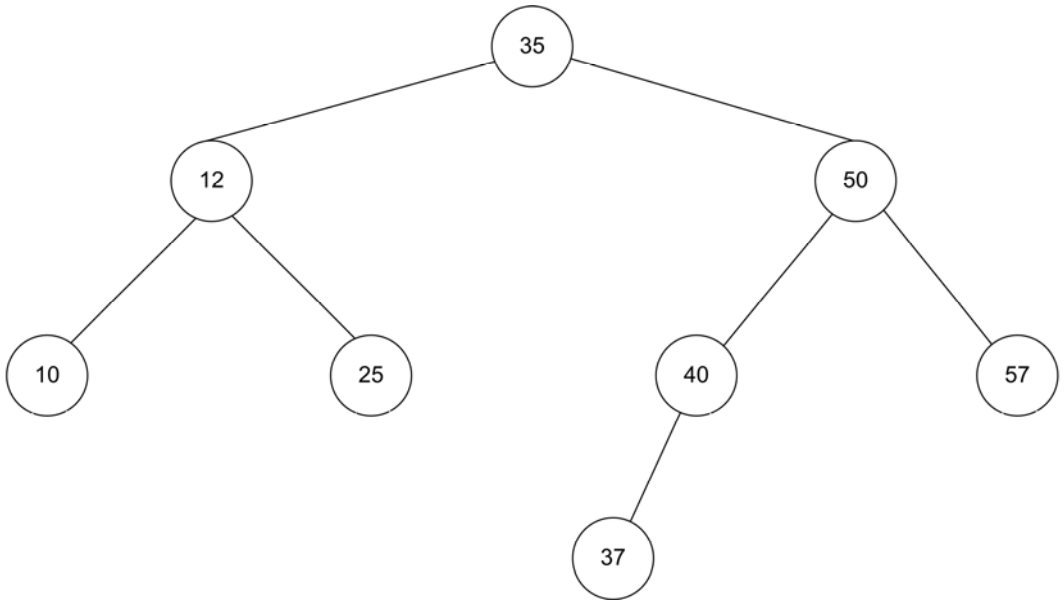
Effectuez l'ensemble des opérations suivantes dans l'ordre en vous servant des arbres ci-bas :

Insérez 50.
Insérez 37.
Insérez 45.
Insérez 52.
Insérez 47.

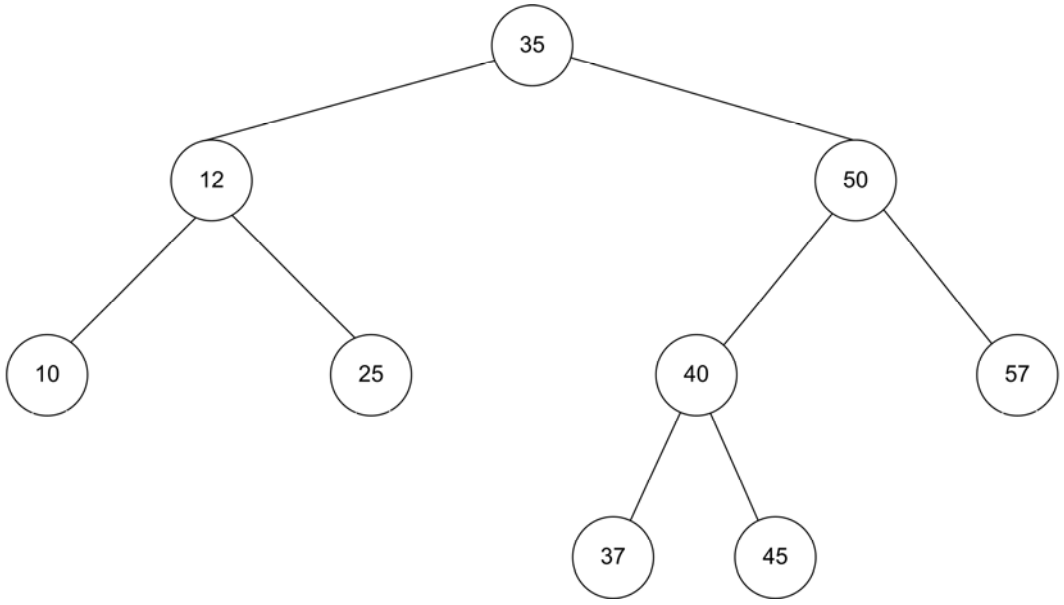
5.1) **(4 pnt)** Insérez 50.



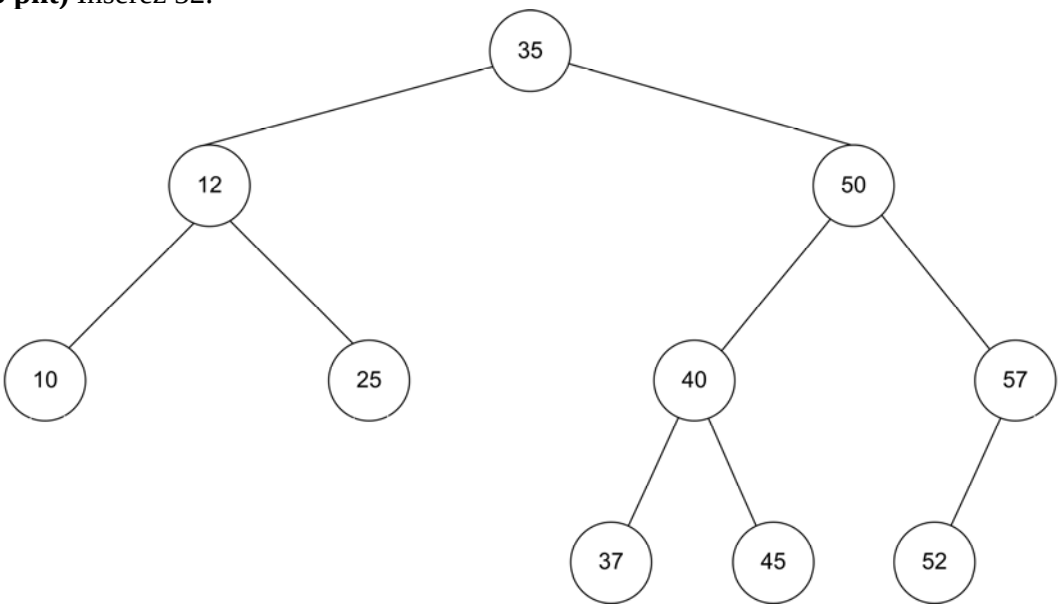
5.2) **(3 pnt)** Insérez 37.



5.3) **(3 pnt)** Insérez 45.



5.3) **(3 pnt)** Insérez 52.



5.4) **(5 pnt)** Insérez 47.