POLYTECHNIQUE MONTRÉAL

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**	**20193**
Professeur	***Local***	***Téléphone***
Ettore Merlo, responsable – Tarek Ould Bachir, chargé de cours		

Jour	*Date*	*Durée*	*Heure*
Mardi	**22 octobre 2019**	**2h00**	**18h30**

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

Directives particulières

ATTENTION: Ne posez pas de questions ni aux surveillants ni aux professeurs concernant l'examen. En cas de doute sur le sens d'une question, faites des suppositions raisonnables, énoncez-les clairement dans votre réponse et poursuivez.

Un cahier d'annexe est fourni. Ne remettez pas le cahier d'annexes.

Bonne chance à tous!

Important

Cet examen contient **6** questions sur un total de **16** pages **(incluant 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 : Listes (3/20 points)

Considérez le code Java de l'Annexe 1 définissant la classe `OrderedList` implémentant l'interface `Iterable`.

1.1) **(1 point)** Donnez l'affichage résultant de l'exécution de la fonction `main` définie dans `OrderedList`.

```
[ 1 2 3 3 4 5 6 6 7 8 9 9 ]
[ 1 4 7 ]
```

1.2) **(1 point)** Donnez la complexité asymptotique en pire cas de la méthode publique `addAll(AnyType[] items)`. On supposera que la liste `OredredList` est initialement vide et que le paramètre `items` contient *n* éléments. Justifiez votre réponse.

En pire cas, la méthode s'exécutera en $O(n^2)$. Cela survient par exemple si items est trié.

1.3) **(1 point)** Considérez l'extrait de code Java suivant présent sur la fonction main. Donnez la complexité asymptotique en pire cas de cet extrait si la liste `list` (`OrderedList`) contient *n* éléments. Justifiez votre réponse.

```
Iterator<Integer> it = list.iterator();

while( it.hasNext() )
   if( (it.next() % 3) != 1 )
      it.remove();
```

En pire cas, la méthode s'exécutera en $O(n)$ car la méthode appelle la méthode `remove` sur un itérateur, ce qui est fait en temps constant.

Question 2 : Tables de dispersement (4/20 points)

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

2.1) **(2.5 points)** En vous servant du tableau ci-dessous, donnez l'état d'une table de taille N=11 après l'insertion, dans l'ordre, des clés suivantes :

22, 56, 66, 26, 27.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Entrées	22	56			66	26	27				

Donnez le détail de vos calculs ci-après:

x = 22: 22%11 = 0.
x = 56: 56%11 = 1.
x = 66: 66%11 = 0, collision; (0 + 1^2)%11 = 1, collision; (0 + 2^2)%11 = 4.
x = 26: 26%11 = 4, collision; (4 + 1^2)%11 = 5.
x = 27: 27%11 = 5, collision; (5 + 1^2)%11 = 6.

2.2) **(0.25 point)** Combien de clés supplémentaires faut-il ajouter à la suite des 5 clés insérées à la table de dispersement en 2.1) avant que la fonction `rehash()` ne soit appelée? Référez-vous au code source fourni à l'Annexe 2.

La méthode `rehash` sera appelé à l'insertion de la prochaine clé.

2.3) **(0.25 point)** Quelle sera la nouvelle taille de la table de dispersion après l'appel à la fonction `rehash()` ? Justifiez votre réponse.

11x2 = 22. `nextPrime`(22) = 23. 23 sera la nouvelle taille de la table.

2.4) **(0.5 point)** On effectue un appel à `remove(59)` sur la table de dispersement quadratique suivante. Donnez le détail de cet appel. Soyez bref mais précis. Référez-vous au code source fourni à l'Annexe 2.

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

59%11 = 4, 59 est absent de cette case ;
$(4 + 1^2)$%11 = 5, 59 est absent de cette case ;
$(4 + 2^2)$%11 = 8, 59 est absent de cette case ;
$(4 + 3^2)$%11 = 2, 59 est absent de cette case ;
$(4 + 4^2)$%11 = 9, 59 est absent de cette case ;
$(4 + 5^2)$%11 = 7, la case est vide. 59 est absent de la table.

2.5) **(0.5 point)** On effectue un appel à `remove(9)` puis `insert(37)` sur la table de dispersement quadratique suivante. Donnez l'état de la table de taille N=11. Référez-vous au code source fourni à l'Annexe 2.

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

`remove`(9) : 9%11 = 9, 9 est désactivée.
`insert`(37) : 37%11 = 4, collision.
$(4 + 1^2)$%11 = 5, collision ;
$(4 + 2^2)$%11 = 8, collision ;
$(4 + 3^2)$%11 = 2, collision ;
$(4 + 4^2)$%11 = 9, collision (case désactivée ne contenant pas 37) ;
$(4 + 5^2)$%11 = 7, la case est vide. 37 sera inséré là.

Question 3 : Tris en $n \log(n)$ (4/20 points)

On désire exécuter l'algorithme *Quick Sort* pour trier le vecteur ci-après. On considère une valeur *cut-off* de 4. Le code source vous est fourni à l'Annexe 3.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Valeurs	32	30	2	4	28	26	6	8	24	22	10	12	20	18

3.1) (**0.5 point**) Donnez l'état du vecteur après la mise à l'écart du pivot lors de la première récursion de QuickSort :

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Valeurs	**6**	30	2	4	28	26	**20**	8	24	22	10	12	**18**	**32**

3.2) (**0.5 point**) Donnez l'état du vecteur après l'exécution du partitionnement de la première récursion :

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Valeurs	6	12	2	4	10	8	18	26	24	22	28	30	20	32

3.3) (**1 points**) Donnez, dans l'ordre d'appel, l'ensemble des cinq (5) premières valeurs que prennent les paramètres `left` et `right` lors des appels successifs à la méthode privée `quicksort( AnyType [ ] a, int left, int right )`. Aidez-vous du code de l'Annexe 3.

Appels	**`left`**	**`right`**
Appel 1	0	13
Appel 2	0	5
Appel 3	0	1
Appel 4	3	5
Appel 5	7	13

3.4) **(1 point)** Au total, quel est le nombre de fois que la fonction récursive `quicksort` aura été appelée pour exécuter le tri ? Prenez en compte la valeur *cut-off* de 4 dans votre réponse.

Votre réponse: 7

3.5) **(1 point)** Au total, quel est le nombre de fois que la fonction récursive `quicksort` aurait été appelée si la valeur *cut-off* avait été 6 au lieu de 4.

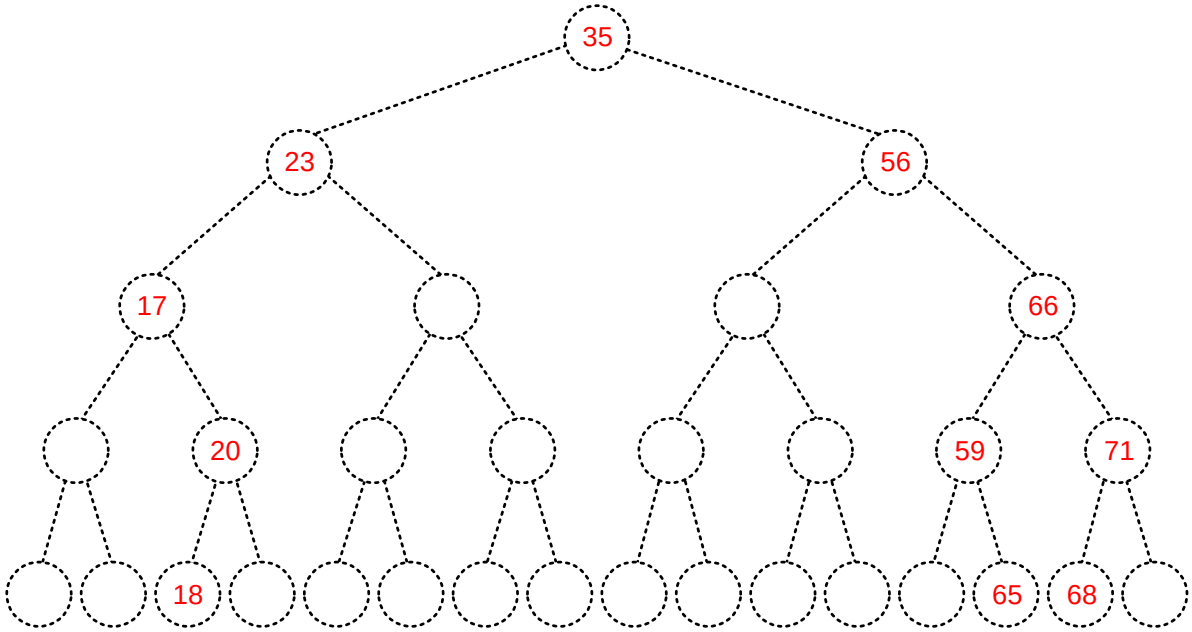
Votre réponse: 5

Question 4 : Parcours d'arbres binaire de recherche (2.5/20 points)

4.1) **(0.5 point)** Si l'affichage par niveaux de l'arbre binaire de recherche donne :

35, 23, 56, 17, 66, 20, 59, 71, 18, 65, 68

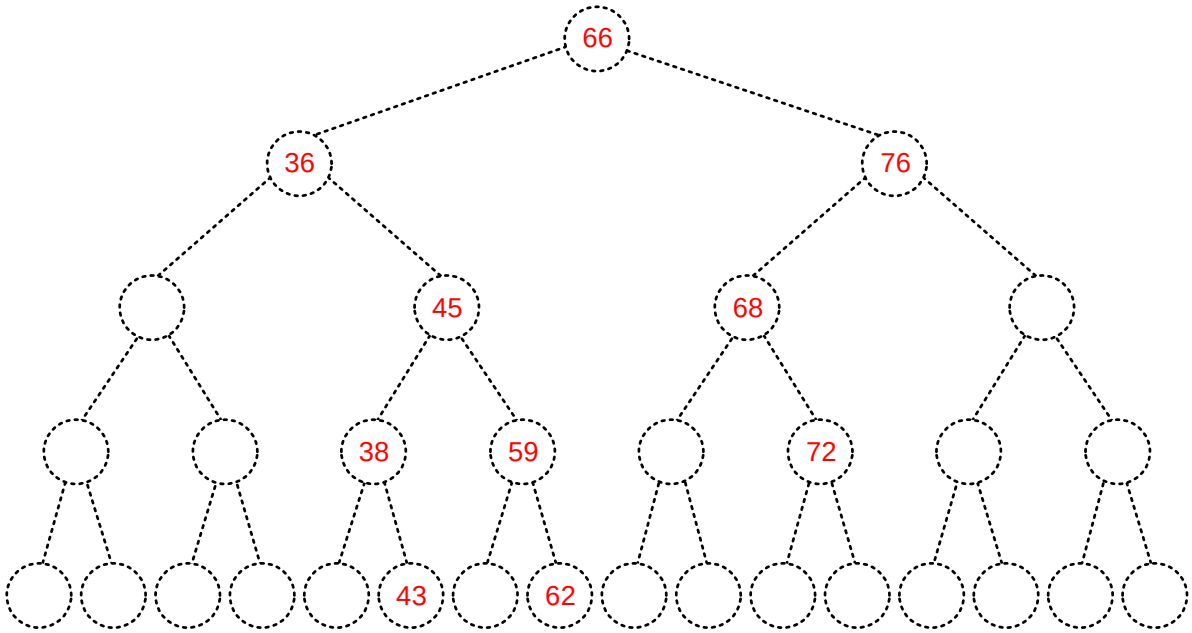
Donnez la représentation graphique de l'arbre.



4.2) **(0.5 point)** Si l'affichage pré-ordre de l'arbre binaire de recherche donne :

66, 36, 45, 38, 43, 59, 62, 76, 68, 72

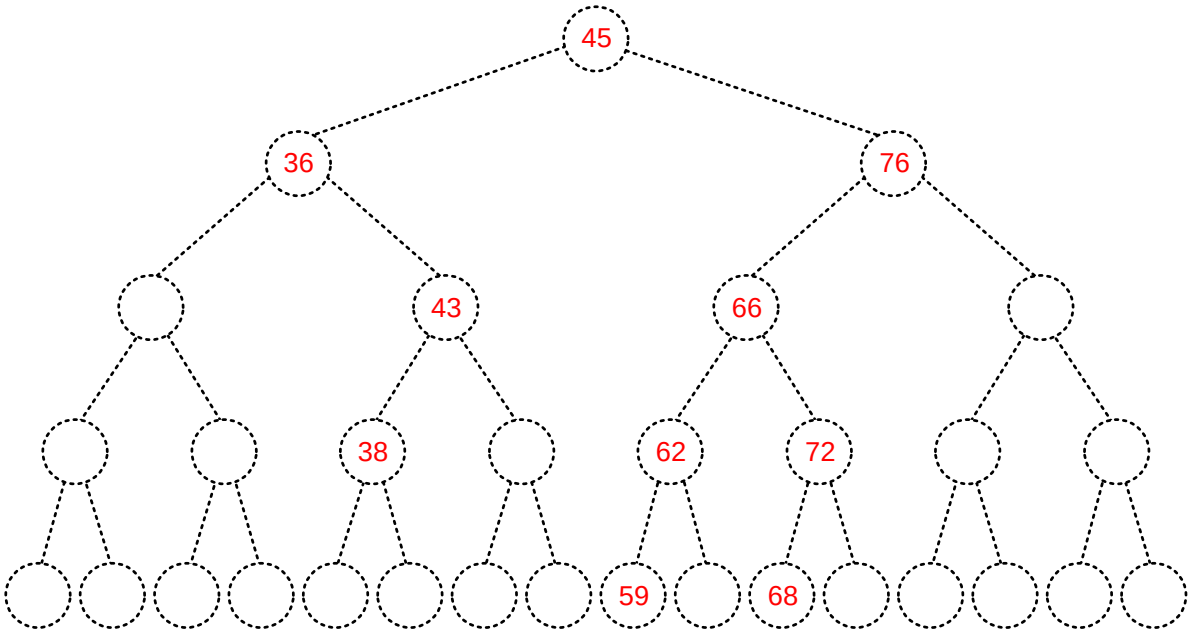
Donnez la représentation graphique de l'arbre.



4.3) **(0.5 point)** Si l'affichage post-ordre de l'arbre binaire de recherche donne :

38, 43, 36, 59, 62, 68, 72, 66, 76, 45

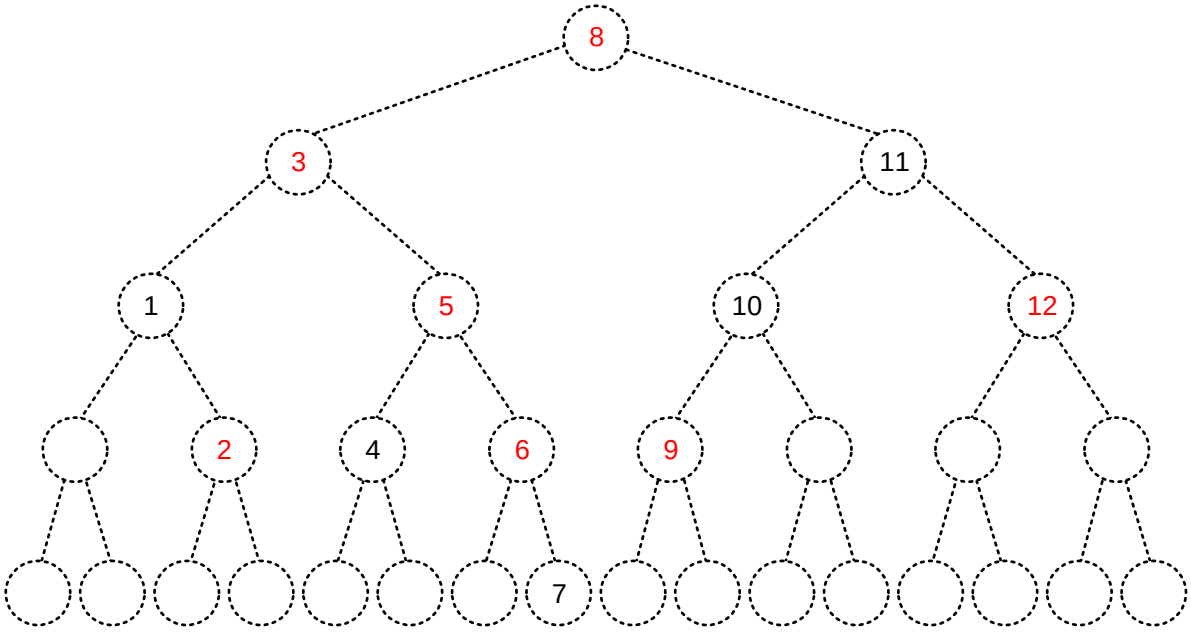
Donnez la représentation graphique de l'arbre.



4.4) **(1 point)** Si l'affichage en-ordre de l'arbre binaire de recherche donne :

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

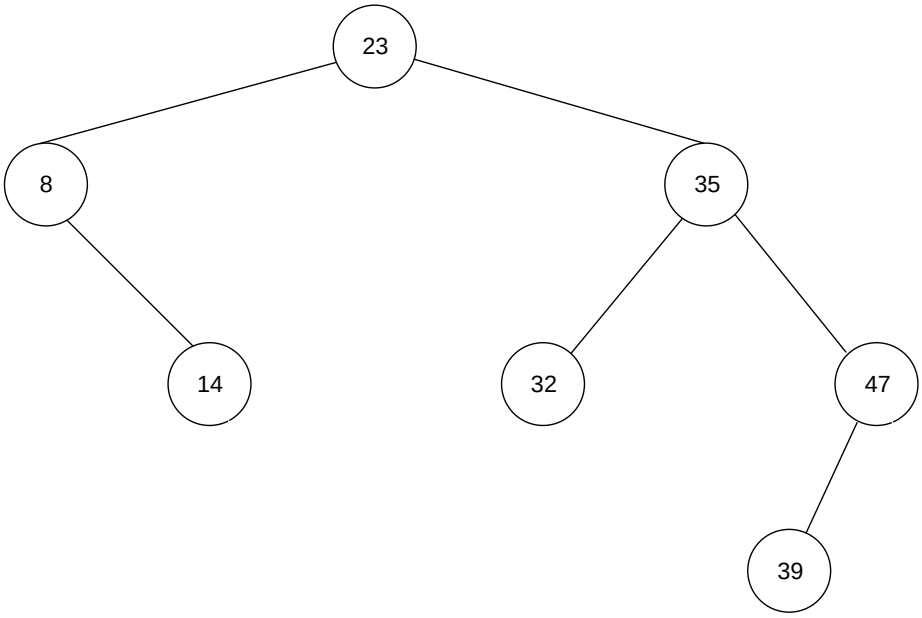
Complétez la représentation graphique de l'arbre qui suit sachant qu'il s'agit d'un arbre AVL de hauteur *h*=4.



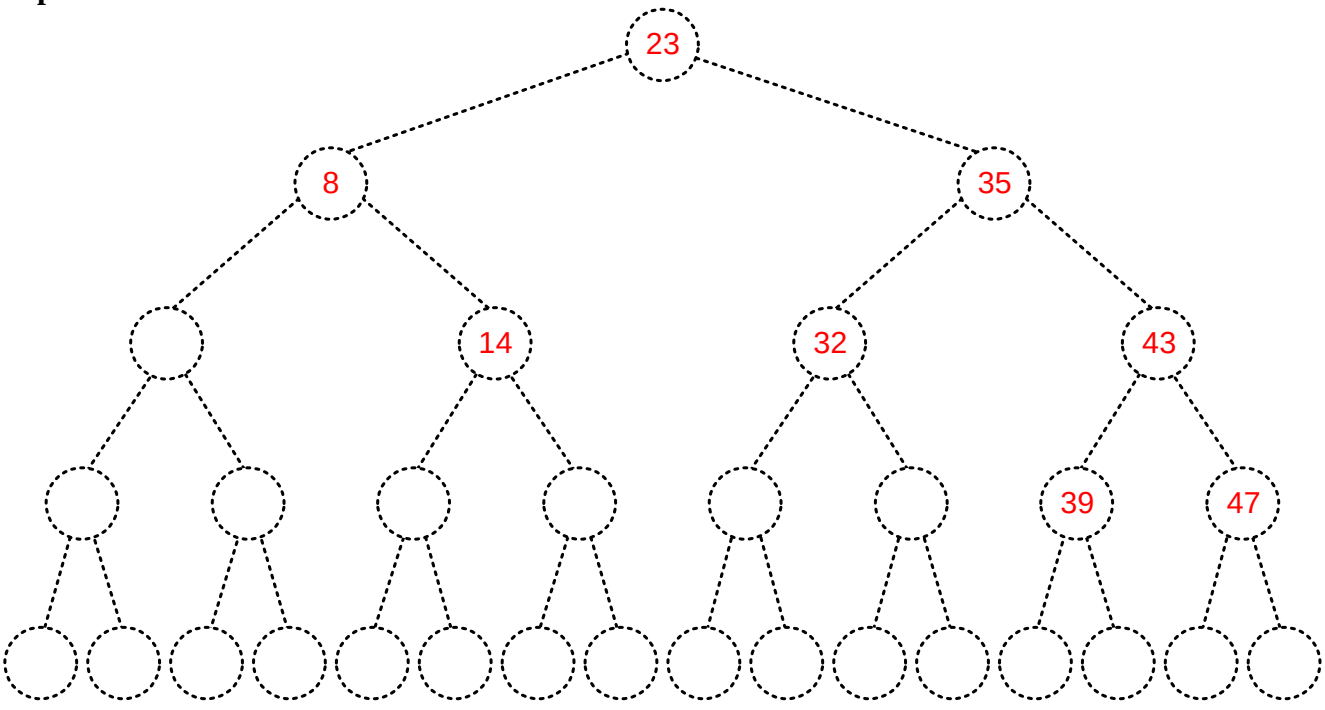
Question 5 : Arbre binaire de recherche de type AVL (4/20 points)

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

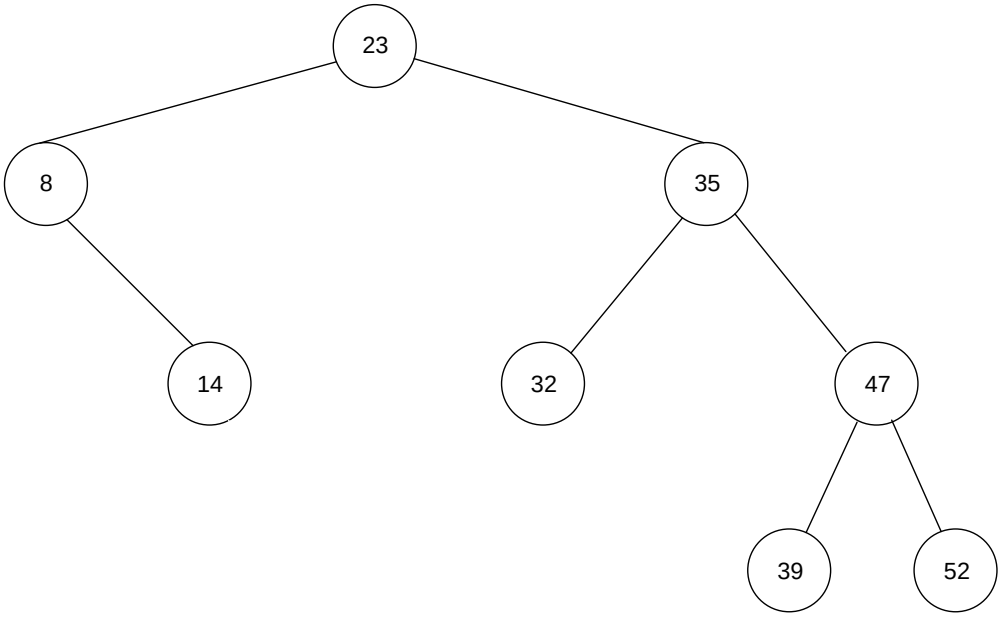
5.1) **(0.5 point)** Insérez 43.



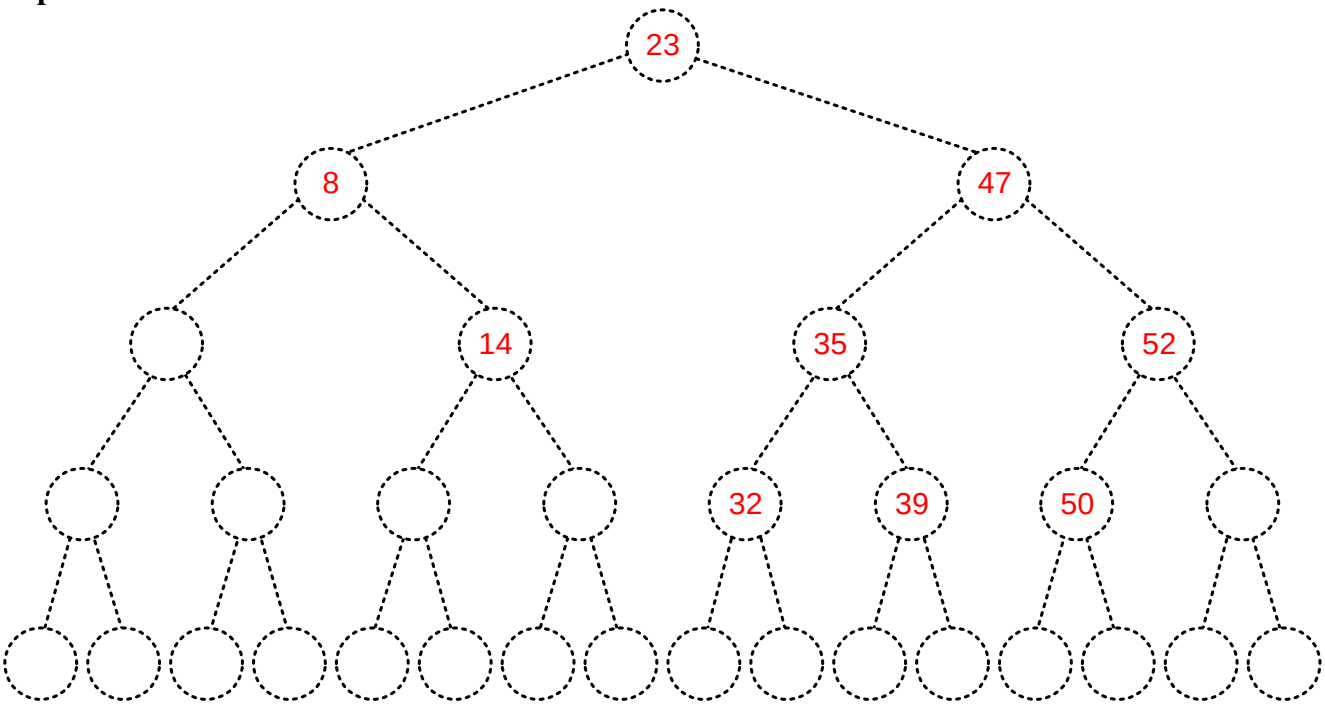
Réponse :



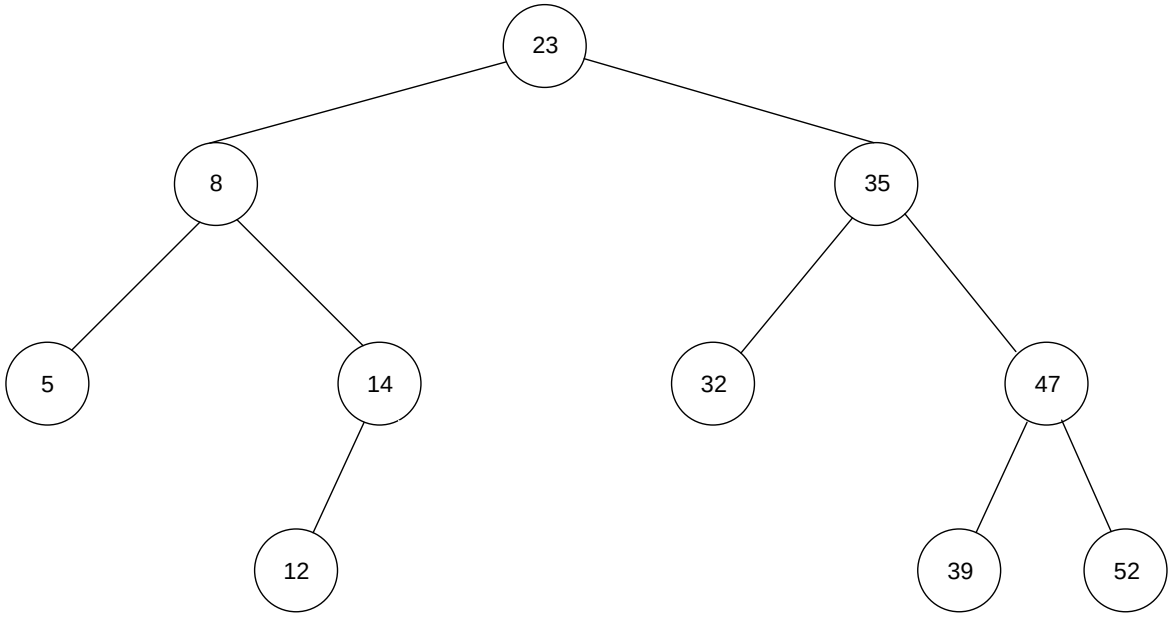
5.2) **(0.5 point)** Insérez 50.



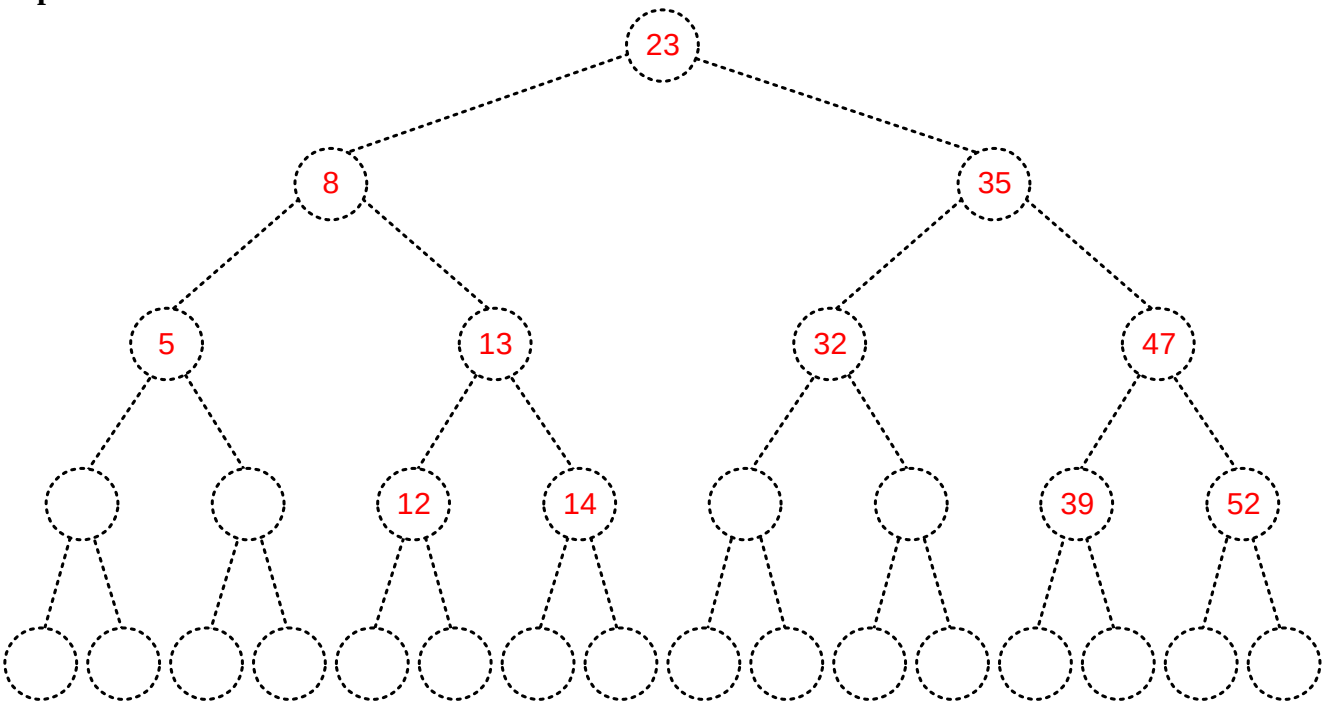
Réponse :



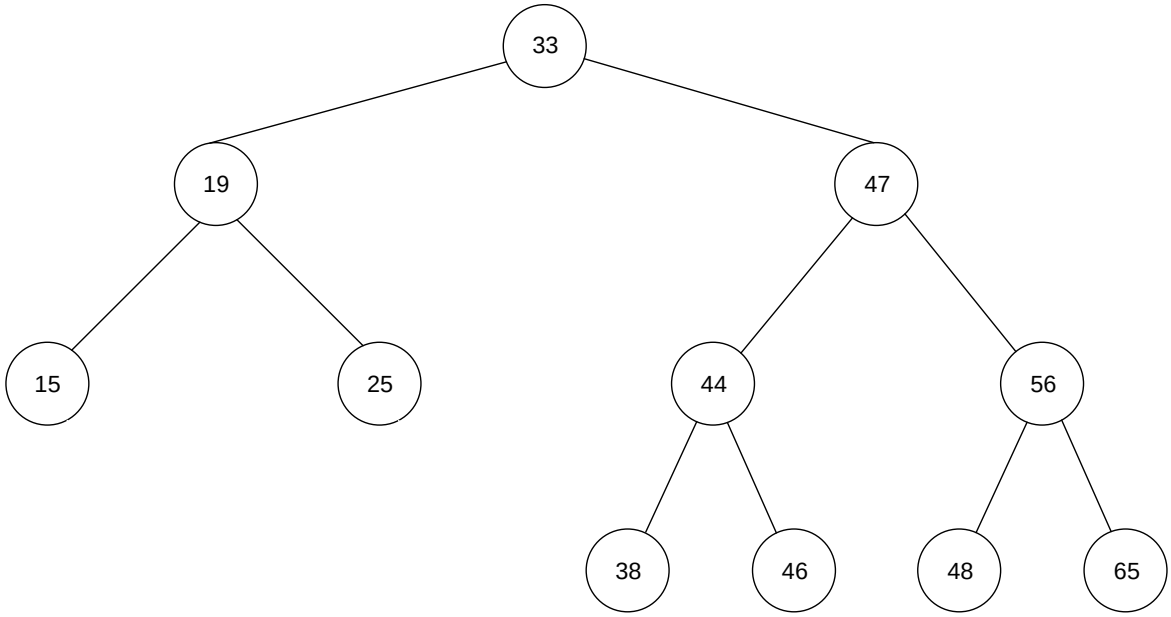
5.3) **(0.5 point)** Insérez 13.



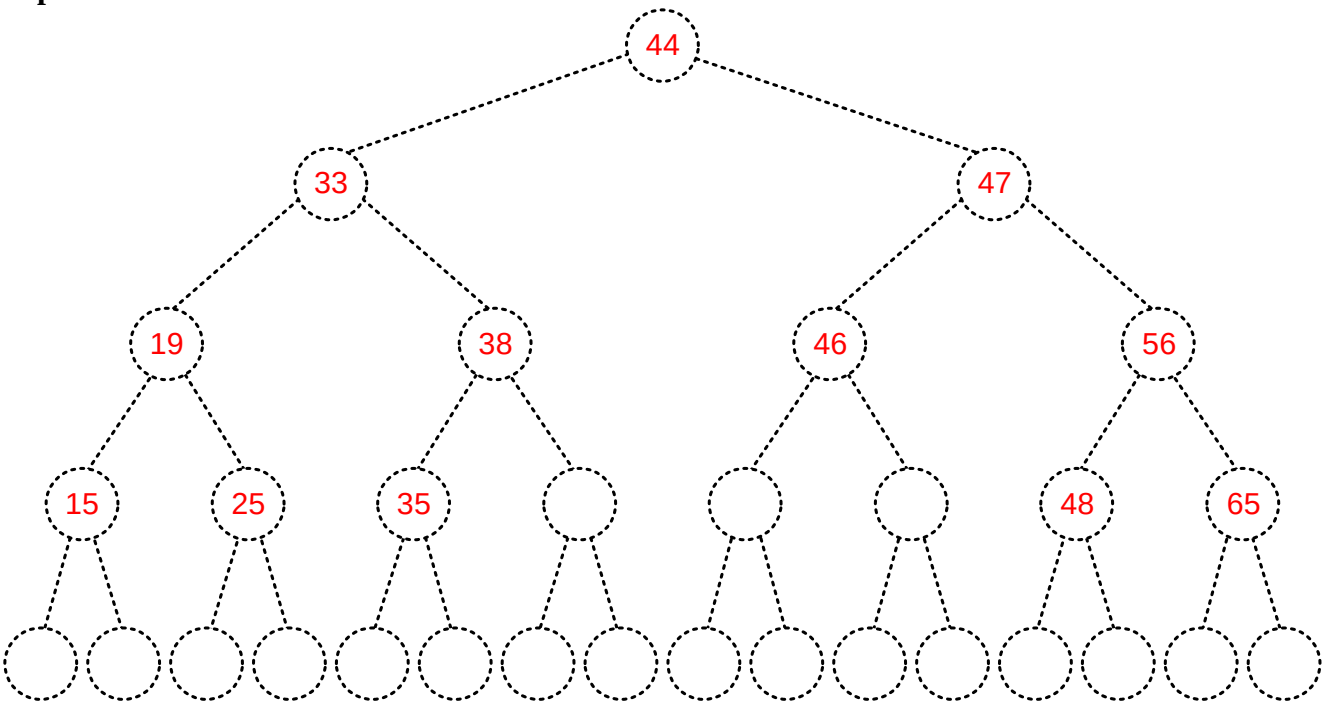
Réponse :



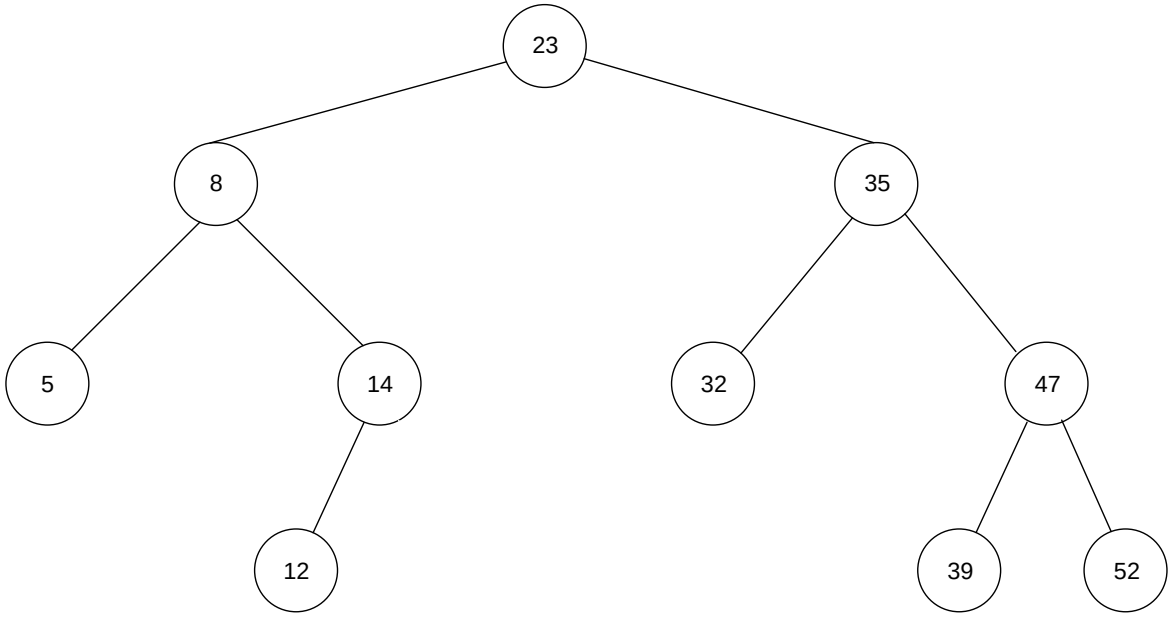
5.4) **(0.5 point)** Insérez 35.



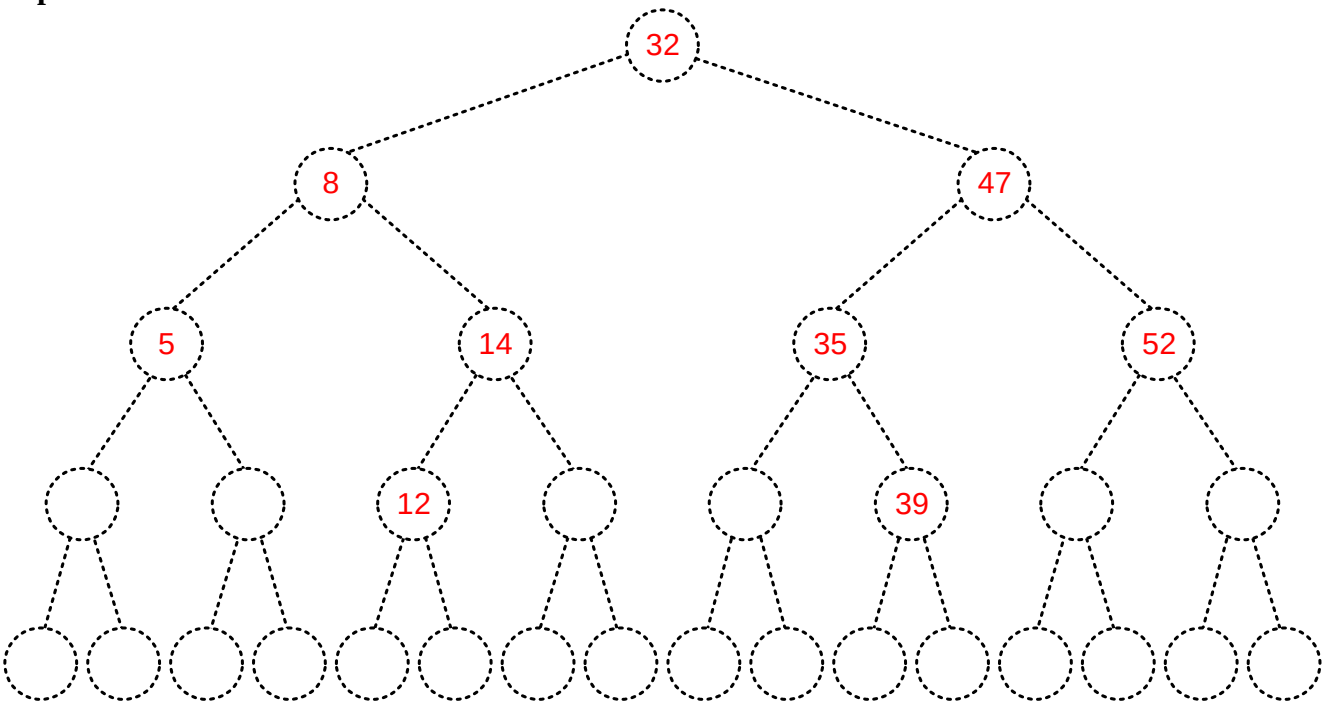
Réponse :



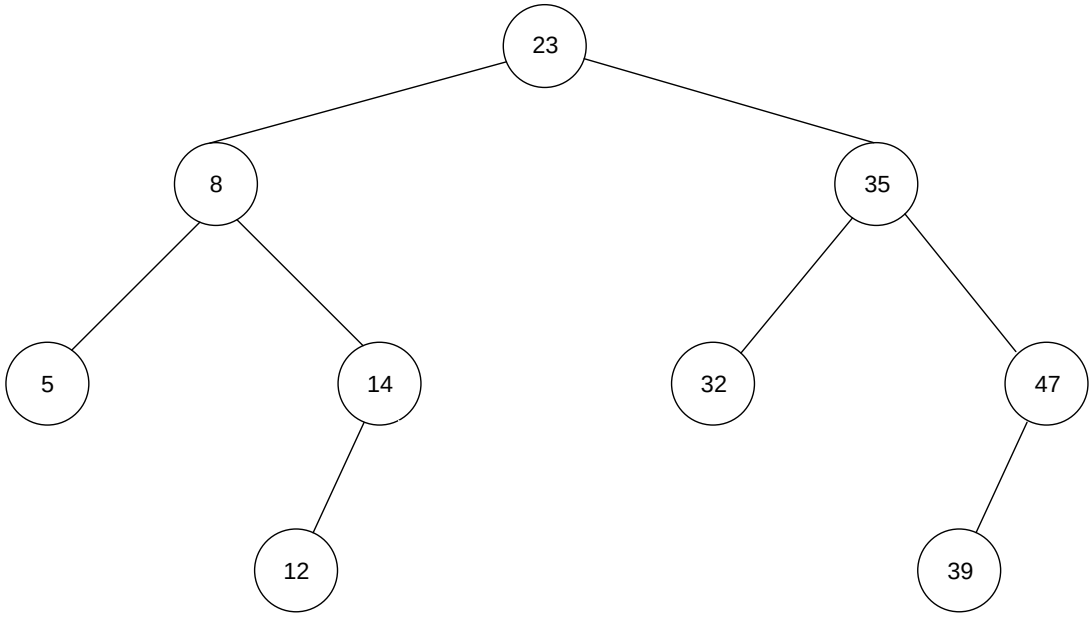
5.5) **(1 point)** Supprimez la racine. Référez-vous au code de l'Annexe 4 pour ce faire.



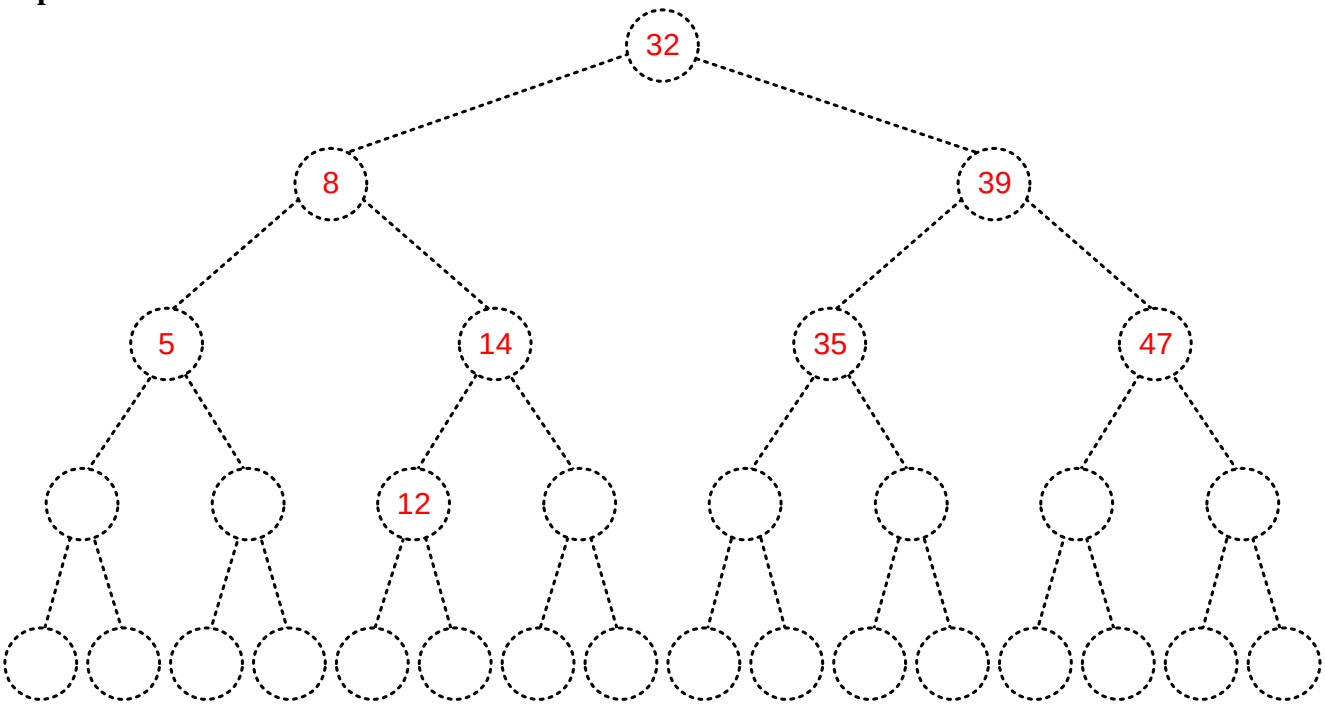
Réponse :



5.6) **(1 point)** Supprimez la racine. Référez-vous au code de l'Annexe 4 pour ce faire.



Réponse :



Question 6 : Généralités (2.5/20 points)

Répondez aux assertions suivantes par « vrai » ou par « faux ». Justifier votre réponse brièvement. Une réponse non justifiée ne sera pas considérée.

6.1) **(0.5 point)** La fusion de deux listes doublement chainées de taille *n* en une liste doublement chainée triée peut s'effectuer en $O(n)$ au mieux.

Vrai. Cela survient par exemple si les deux listes sont déjà triées.

6.2) **(0.5 point)** La signature suivante est correcte pour implémenter un itérateur sur la liste `MaListe`.

```
public class MaListe<T> implements Iterable<T>
{
    private int theSize;
    private T[] theItems;
        ...
    public java.util.Iterator<T> iterator( )
    { return new MonIterateur<T>( this ); }

    public class MonIterateur implements java.util.Iterator<T>
    {
          ...
    }
}
```

Faux. La classe MonIterateur devrait être privée.

6.3) **(0.5 point)** Il est toujours possible d'insérer un élément dans une table de dispersement utilisant une résolution de collision par sondage quadratique dont la taille est un nombre premier.

Faux. Il faut également que le facteur de compression soit de 50% au plus.

6.4) **(0.5 point)** L'algorithme `QuickSort` a une complexité O(*n*) en meilleur cas.

Faux. Il est O(nlog(n)) en meilleur cas.

6.5) **(0.5 point)** Un arbre AVL de hauteur h=7 possède au plus 127 nœuds.

Faux:
$N_{max} = 2^{h+1}-1$; h=7 => $N_{max} = 2^{8}-1=255$.