ÉCOLE POLYTECHNIQUE MONTRÉAL

Le génie sans frontières

Questionnaire 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**	**20101**
Professeur		***Local***	***Téléphone***
Ettore Merlo, responsable – Tarek Ould Bachir, chargé de cours		**A-622/A-638**	
Jour	***Date***	***Durée***	***Heure***
Mercredi	**20 octobre 2010**	**2h00**	**18h30**

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

Directives particulières

- ✓ N'oubliez pas d'inscrire vos nom, prénom, matricule et groupe sur votre cahier d'examen.
- ✓ N'oubliez pas d'inscrire vos nom, prénom, matricule et groupe sur les pages 4 et 6 à 13.
- ✓ N'oubliez pas de détacher et d'inclure les pages 4 et 6 à 13 de cet énoncé avec votre cahier d'examen.

Bonne chance à tous!

Important

Cet examen contient **7** questions sur un total de **15** 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 : Généralités (10 points)

Répondez aux assertions suivantes par « vrai » ou par « faux » en justifiant brièvement.

a) La signature suivante est bonne pour implémenter un itérateur sur la liste `MaListe`. **(2 pts)**

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

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

b) L'algorithme QuickSort a une complexité O(n log(n)) en pire cas **(2 pts)**.

c) L'algorithme MergeSorte a une complexité O(n) en meilleur cas **(2 pts)**.

d) Une table de dispersement utilisant une résolution de collision par sondage quadratique doit toujours avoir une taille qui est un nombre premier **(2 pts)**.

e) L'opération de retrait usuelle, telle qu'effectuée sur un arbre binaire de recherche standard, effectuée sur un arbre AVL produit un arbre AVL **(2 pts)**.

Question 2 : Algorithme et étude de complexité (12 points)

Considérez le code Java suivant :

```
 1   public static <AnyType extends Comparable<? super AnyType>>
 2   void QueFaisJe( AnyType [ ] a )
 3   {
 4      int k;
 5
 6      for( int i = a.length / 2; i > 0; i /= 2 )
 7         for( int j = i; j < a.length; j++ )
 8         {
 9            AnyType tmp = a[ j ];
10
11            for( k = j; k >= i && tmp.compareTo( a[ k - i ] ) < 0; k -= i )
12                        a[ k ] = a[ k - i ];
13
14            a[ k ] = tmp;
15         }
16   }
```

a) Expliquez pourquoi la variable locale `k` n'est pas déclarée à l'intérieur de la boucle `for` de la ligne 11 comme le furent `i` et `j` aux lignes 6 et 7 respectivement **(2 pts)**.

b) Donnez l'état du vecteur `a[]` après l'exécution de la fonction `QueFaisJe( a )` si le `a[]` reçu par cette dernière est : `a = {2,10,1,5}`. Détaillez succinctement chaque étape de l'exécution en indiquant les valeurs prises par `i`, `j` et `k` **(4 pts)**.

c) Donnez la séquence des valeurs prises par `i` si `a[]` a une longueur de 16 **(1 pt)**.

d) Donnez la séquence des valeurs prises par `i` si `a[]` a une longueur de 31 **(1 pt)**.

e) Supposons que `a[]` ait une longueur de $n=2^m$, où m est un entier strictement positif. Donnez le nombre de valeurs prises par `i` en fonction de m ; et donnez le nombre de valeurs prises par `j` pour chaque `i` en fonction de `i` et de m. Justifiez votre réponse **(2 pts)**.

f) Supposez que `a[]` ait une longueur de $n=2^m$, où m est un entier strictement positif. Donnez la complexité asymptotique de l'algorithme exécuté par `QueFaisJ` si les lignes 11 à 14 étaient retirées. Justifiez votre réponse **(2 pts)**.

Rappel : $\lim_{m\to\infty} \sum_{i=1}^{m} \frac{1}{2^i} = 1$

Question 3 : Arbres binaire de recherche (16 points)

L'affichage pré-ordre d'un arbre binaire de recherche donné a produit :

52, 30, 17, 14, 25, 48, 31, 57, 65, 70

a) On vous demande de dessiner cet arbre binaire de recherche en vous servant de l'**annexe 1 (4 pts)**.

b) Est-ce l'arbre obtenu est un arbre binaire complet ? Justifiez votre réponse **(4 pts)**.

L'affichage en ordre d'un arbre binaire de recherche donné a produit

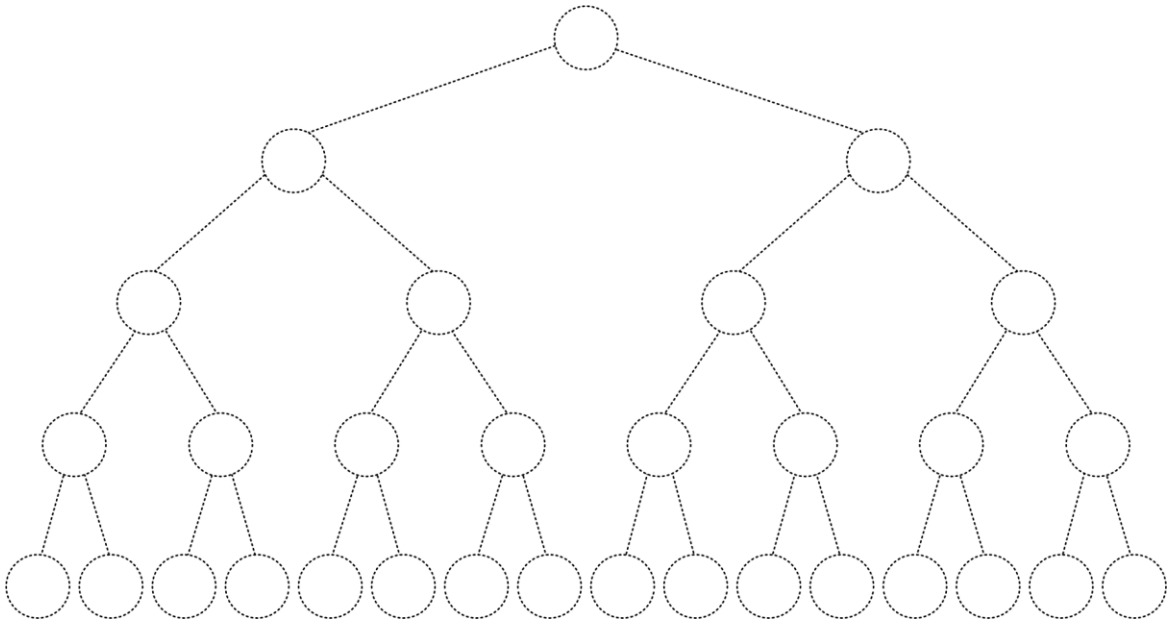
14, 17, 25, 30, 31, 48, 52, 57, 67, 70

c) On vous demande de dessiner cet arbre binaire de recherche en vous servant de l'**annexe 1** sachant qu'il s'agit d'un arbre complet **(4 pts)**.

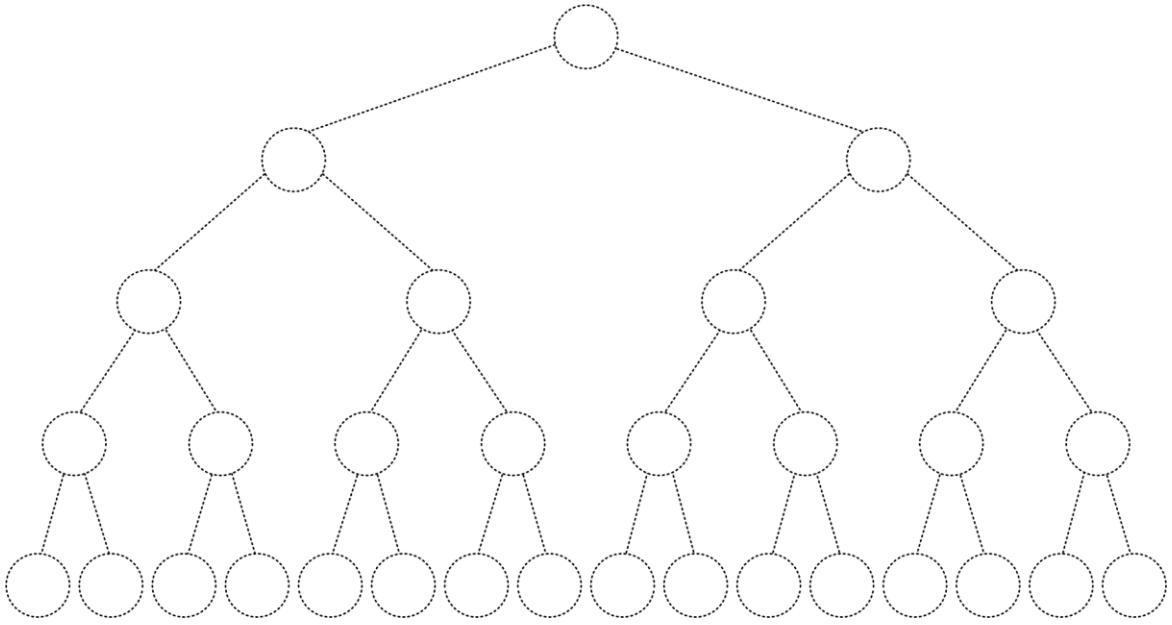
d) Auriez-vous pu dessiner cet arbre si vous ne saviez pas qu'il était complet et saviez seulement qu'il s'agissait d'un arbre binaire de recherche ? Justifiez votre réponse **(4 pts)**.

Annexe 1

a) Dessinez l'arbre ayant produit l'affichage pré-ordre :



c) Dessinez l'arbre ayant produit l'affichage en ordre :



Nom :_____ Matricule :_____ Groupe :_____

Question 4 : Arbre AVL (20 points)

Considérons l'arbre AVL suivant (figure 1) :

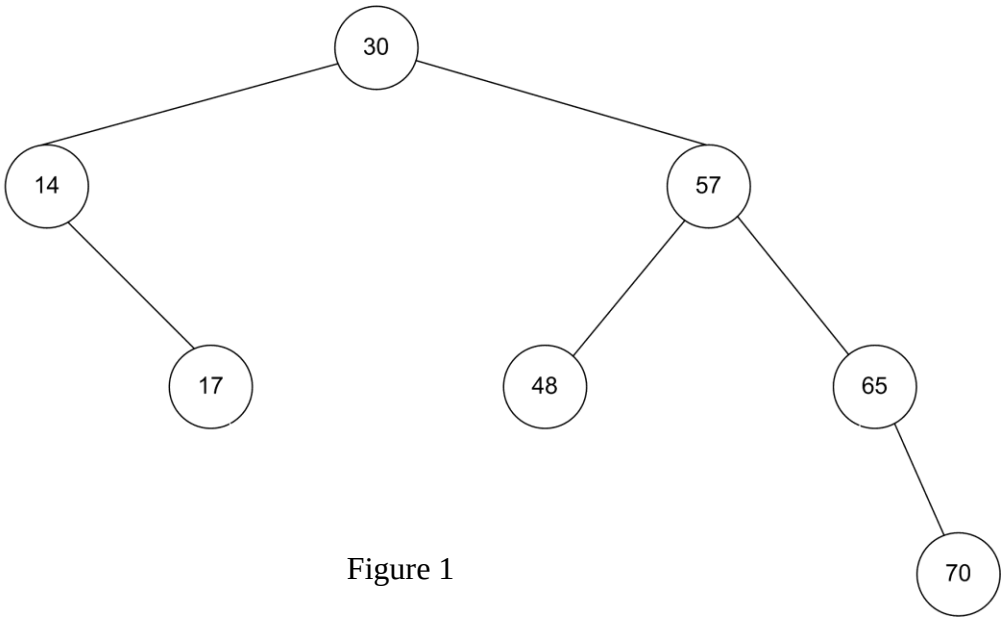


Figure 1

On vous demande d'effectuer les opérations suivantes dans l'ordre :

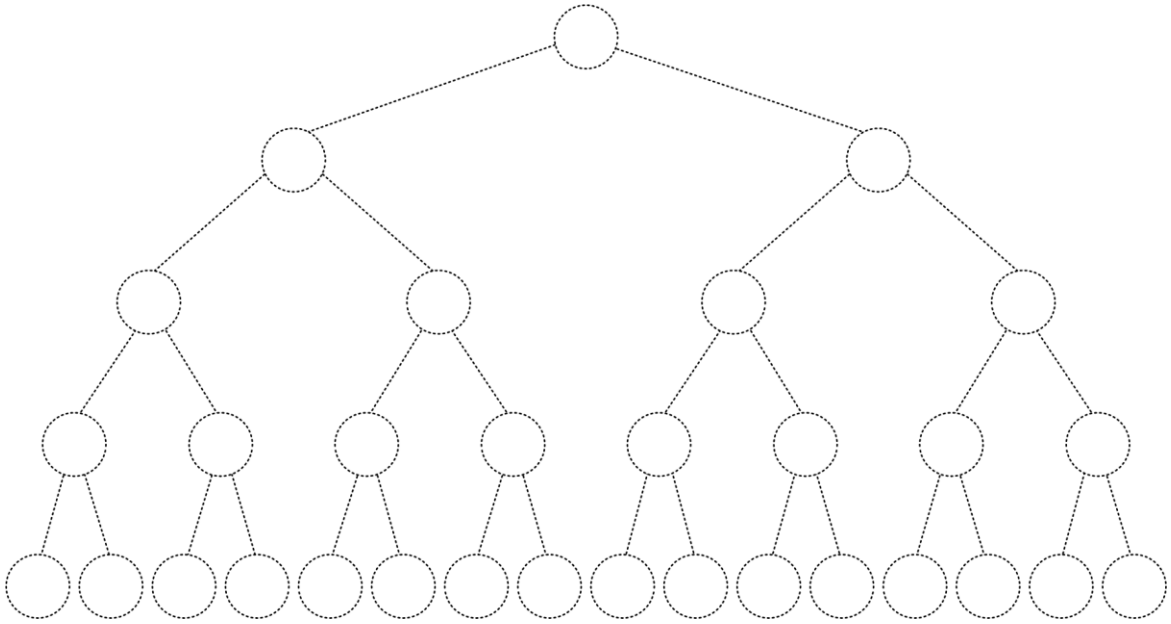
a) Insérez 25
b) Insérez 52
c) Insérez 49
d) Insérez 31
e) Retirez 49

L'opération de retrait doit être réalisée en relâchant la contrainte de l'AVL (autrement dit, retirez le nœud comme vous l'auriez fait avec un arbre binaire de recherche usuel).

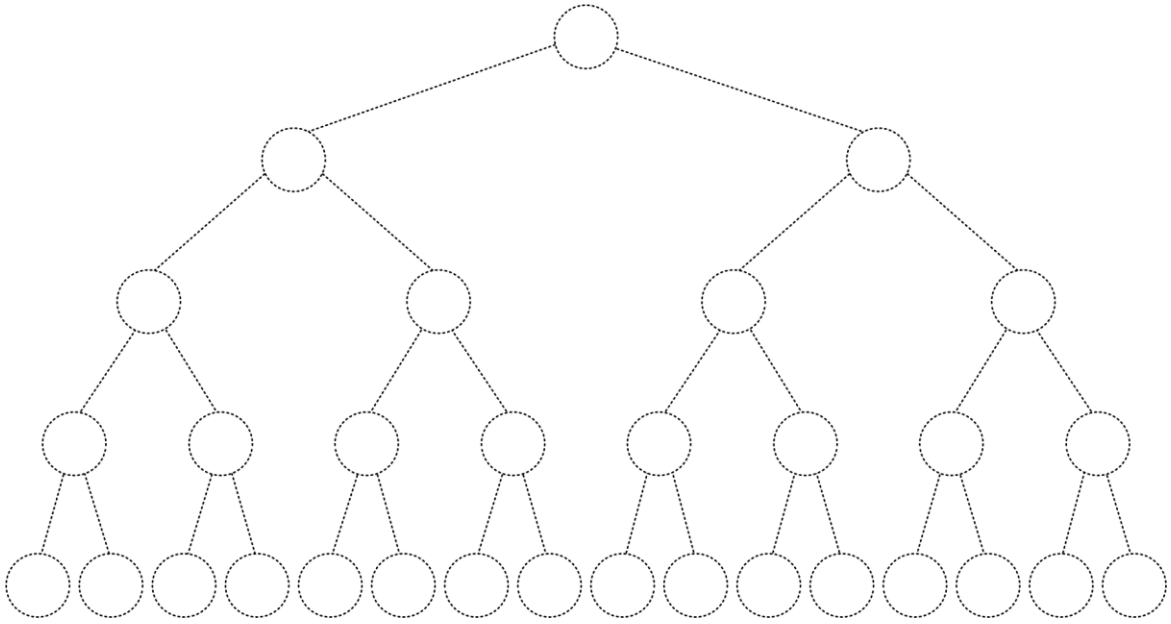
Répondez à l'**annexe 2** que vous compléterez de votre nom et que vous joindrez à votre cahier de réponse.

Annexe 2

a) Insérez 25.

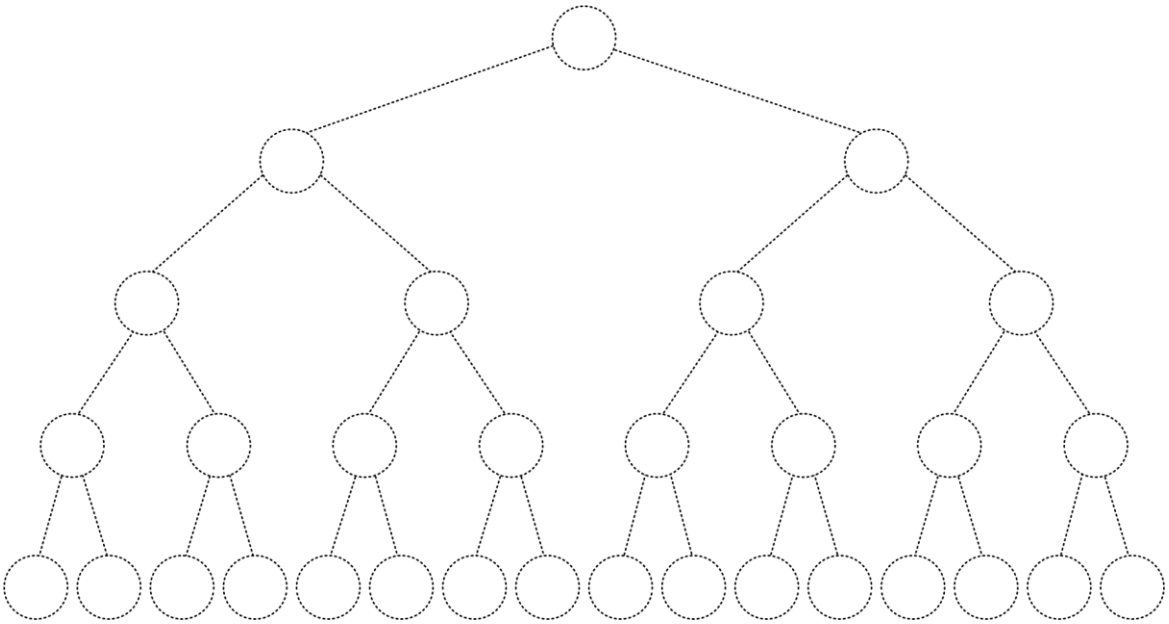


b) Insérez 52.

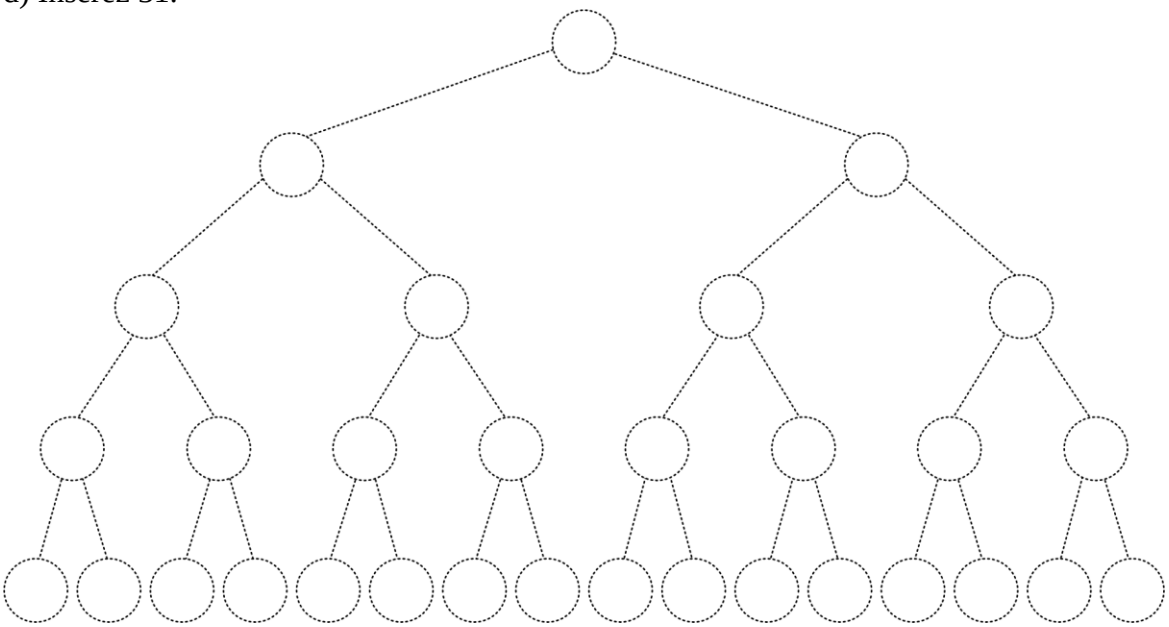


Nom :_____ Matricule :_____ Groupe :_____

c) Insérez 49.

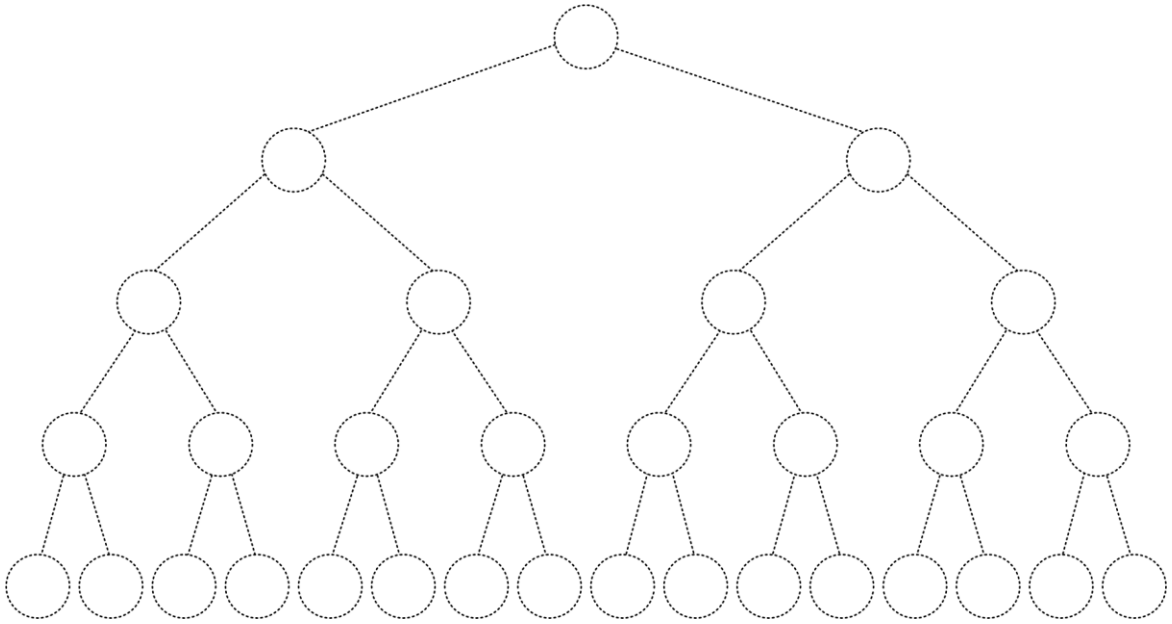


d) Insérez 31.



Nom :_____ Matricule :_____ Groupe :_____

e) Retirez 49.



Nom :_____ Matricule :_____ Groupe :_____

Question 5 – Tables de dispersement (16 points)

Considérez :

- la fonction de dispersement h(x) = x % 11
- une table de dispersement de dimension 11
- les clefs : 30, 74, 51, 66, 50, 11.

a) Considérez une table de dispersement par débordement progressif avec "*sondage*" linéaire (f(i) = i)

a.1) Quelles sont les valeurs i et $h_i(x)$ utilisées pour l'insertion de chaque clef indiquée ? Répondez sur le cahier.

a.2) Dessinez l'état de la table après l'insertion de toutes les clefs indiquées en remplissant la table de figure 2a.

b) Considérez une table de dispersement par débordement progressif avec "*sondage*" quadratique ($f(i) = i^2$).

b.1) Quelles sont les valeurs i et $h_i(x)$ utilisées pour l'insertion de chaque clef indiquée ? Répondez sur le cahier.

b.2) Dessinez l'état de la table après l'insertion de toutes les clefs indiquées en remplissant la table de la figure 2b.

Position	Table
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Figure 2a

Position	Table
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Figure 2b

Nom :_____ Matricule :_____ Groupe :_____

Question 6 – Tri rapide (calcul) (10 points)

Considérez le vecteur suivant :

V =	46	20	12	37	47	91	99	67	88	40	27	71

Remplissez les tableaux a) à g) avec les pivots et les partitions obtenus à partir du vecteur indiqué en utilisant l'algorithme de tri rapide (« *Quicksort* »).

Appliquez la limite inférieure (« *cutoff* ») = 3 pour les calculs.

a) Pivot par médiane -3 (V) : | |

V_g = Partition gauche (V) :

V_d = Partition droite (V) :

b) Pivot par médiane -3 (V_g) : | |

V_{gg} = Partition gauche (V_g) :

V_{gd} = Partition droite (V_g) :

Nom :_____ Matricule :_____ Groupe :_____

c) Pivot par médiane -3 (V_d) :

V_{dg} = Partition gauche (V_d) :

V_{dd} = Partition droite (V_d) :

d) Pivot par médiane -3 (V_{gg}) :

Partition gauche (V_{gg}) :

Partition droite (V_{gg}) :

e) Pivot par médiane -3 (V_{gd}) :

Partition gauche (V_{gd}) :

Partition droite (V_{gd}) :

Nom :_____ Matricule :_____ Groupe :_____

f) Pivot par médiane -3 (V_{dg}) :

Partition gauche (V_{dg}) :

Partition droite (V_{dg}) :

g) Pivot par médiane -3 (V_{dd}) :

Partition gauche (V_{dd}) :

Partition droite (V_{dd}) :

Nom :_____ Matricule :_____ Groupe :_____

Question 7 – Tri rapide (algorithme) (16 points)

Considérez le code en **annexe 3**. Il représente l'algorithme de tri rapide (« *Quicksort* ») modifié en ajoutant , en éliminant ou en changeant des instructions de façon possiblement erronée.

Ajoutez, éliminez ou changez les lignes pertinentes en remplissant le tableau 1 avec les modifications syntaxiquement correctes appropriées.

N'ajoutez pas d'espace au tableau. N'écrivez pas plus d'une ligne par espace.

Tableau 1

#ligne	Nouvelles lignes modifiées

Nom :_____ Matricule :_____ Groupe :_____

Annexe 3

```
1       /**
2        * A class that contains several sorting routines,
3        * implemented as static methods.
4        * Arrays are rearranged with smallest item first,
5        * using compareTo.
6        * @author Mark Allen Weiss
7        */
8           /**
9            * Quicksort algorithm.
10           * @param a an array of Comparable items.
11           */
12          public static <AnyType extends Comparable<? super AnyType>>
13          void quicksort( AnyType [ ] a ) {
14
15              quicksort( a, 0, a.length );
16
17          }
18          private static final int CUTOFF = 3;
19          /**
20           * Return median of left, center, and right.
21           * Order these and hide the pivot.
22           */
23          private static <AnyType extends Comparable<? super AnyType>>
24          AnyType median3( AnyType [ ] a, int left, int right ) {
25
26              int center = ( left + right ) / 2;
27
28              if( a[ center ].compareTo( a[ left ] ) < 0 )
29
30                  swapReferences( a, left, center );
31
32              if( a[ right ].compareTo( a[ left ] ) > 0 )
33
34                  swapReferences( a, left, right );
35
36              if( a[ right ].compareTo( a[ center ] ) == 0 )
37
38                  swapReferences( a, center, right );
39
40                  // Place pivot at position right - 1
41
42              swapReferences( a, center, right );
43
44              return a[ right ];
45
46          }
```

```
47          /**
48           * Internal quicksort method that makes recursive calls.
49           * Uses median-of-three partitioning and a cutoff of 10.
50           * @param a an array of Comparable items.
51           * @param left the left-most index of the subarray.
52           * @param right the right-most index of the subarray.
53           */
54          private static <AnyType extends Comparable<? super AnyType>>
55          void quicksort( AnyType [ ] a, int left, int right ) {
56
57              if( left <= right ) {
58
59                  AnyType pivot = median3( a, left, right );
60
61                      // Begin partitioning
62
63                  int i = left, j = right - 1;
64
65                  for( ; ; ) {
66
67                      while( a[ ++i ].compareTo( pivot ) <= 0 ) { }
68
69                      while( a[ --j ].compareTo( pivot ) >= 0 ) { }
70
71                      while ( i == j ) {
72
73                     quicksort( a, i - 1, left );    // Sort small elements
74
75                     quicksort( a, j + 1 , right );    // Sort large elements
76
77                  }
78                      else
79
80                          break;
81
82                  }
83
84                  quicksort( a, i , j );    // Sort remaining elements
85
86                  swapReferences( a, i, right );   // Restore pivot
87
88              }
89
90              else  // Do an insertion sort on the subarray
91
92                  insertionSort( a, left, right );
93
94          }
```