

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	20101
<i>Professeur</i>		<i>Local</i>	<i>Téléphone</i>
Ettore Merlo, responsable – Tarek Ould Bachir, chargé de cours		A-622/A-638	
<i>Jour</i>	<i>Date</i>	<i>Durée</i>	<i>Heure</i>
Mercredi	20 octobre 2010	2h00	18h30
<i>Documentation</i>		<i>Calculatrice</i>	
<input type="checkbox"/> Toute	<input type="checkbox"/> Aucune	Les calculatrices, agendas électroniques ou télecvertisseurs sont interdits.	
<input checked="" type="checkbox"/> Aucune	<input type="checkbox"/> Programmable		
<input checked="" type="checkbox"/> Voir directives particulières	<input checked="" type="checkbox"/> Non programmable		
<i>Directives particulières</i>			
<ul style="list-style-type: none"> ✓ 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. 			
<i>Bonne chance à tous!</i>			
Important	Cet examen contient 7 questions sur un total de 23 pages (excluant cette page)		
	La pondération de cet examen est de 30 %		
	Vous devez répondre <input type="checkbox"/> le questionnaire <input type="checkbox"/> le cahier <input checked="" type="checkbox"/> les deux		
	Vous devez remettre le questionnaire : <input checked="" type="checkbox"/> oui <input type="checkbox"/> 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>
    {
        ...
    }
}
```

Vrai. L’itérateur est déclaré ici comme une classe privée non statique implémentant une interface publique d’itérateur.

- b) L’algorithme QuickSort a une complexité $O(n \log(n))$ en pire cas (2 pts).

Faux. En pire cas, QuickSort a une complexité $O(n^2)$.

- c) L’algorithme MergeSort a une complexité $O(n)$ en meilleur cas (2 pts).

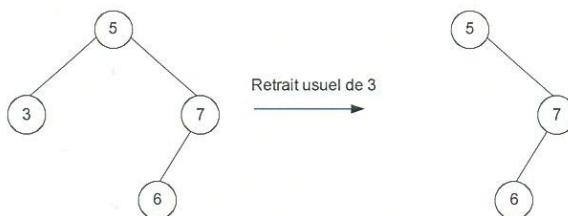
Faux. MergeSort a une complexité $O(n \log(n))$ en tout temps.

- 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).

Vrai. Sans cette condition, il se peut que des entrées ne puissent pas être introduites dans la table. Une seconde condition est que la table soit pleine au plus à 50%.

- 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).

Faux. Une opération de retrait standard modifie l’AVL de telle sorte qu’il puisse perdre ses propriétés. Exemple : Retrait d’une feuille de la branche la moins haute d’un AVL de hauteur $h=2$ et de taille $N=4$.



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).

Cette déclaration est justifiée par l'utilisation de la variable `k` à la ligne 14 en dehors de la boucle `for` des lignes 11-12.

- 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).

```

Au départ de i = 2
  a = 2 10 1 5
Au départ de j = 2
  a = 2 10 1 5
Au départ de k = 2
  a = 2 10 1 5
  À la fin de k = 2
  a = 2 10 2 5
  À la fin de j = 2
  a = 1 10 2 5
Au départ de j = 3
  a = 1 10 2 5
Au départ de k = 3
  a = 1 10 2 5
  À la fin de k = 3
  a = 1 10 2 10
  À la fin de j = 3
  a = 1 5 2 10
```

```

À la fin de  i = 2
    a = 1 5 2 10
Au départ de i = 1
    a = 1 5 2 10
Au départ de j = 1
    a = 1 5 2 10
À la fin de j = 1
    a = 1 5 2 10
Au départ de j = 2
    a = 1 5 2 10
    Au départ de k = 2
        a = 1 5 2 10
    À la fin de k = 2
        a = 1 5 5 10
    À la fin de j = 2
        a = 1 2 5 10
    Au départ de j = 3
        a = 1 2 5 10
    À la fin de j = 3
        a = 1 2 5 10
À la fin de i = 1
    a = 1 2 5 10

```

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

Si $a.length = 16$, alors i prendra les valeurs 8, 4, 2, 1

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

Si $a.length = 31$, alors i prendra les valeurs 15, 7, 3, 1

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).

Si $a.length = 2^m$, alors i prendra les valeurs $2^{m-1}, 2^{m-2}, \dots, 2, 1$, soit m valeurs différentes. Dans ce cas, j prendra les valeurs allant de i à 2^m-1 , soit $2^m-1-i+1=2^m-i$ valeurs différentes.

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 QueFaisJe si les lignes 11 à 14 étaient retirées. Justifiez votre réponse (2 pts).

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

$$\sum_{i=0}^{m-1} 2^m - 2^i = m2^m - \sum_{i=0}^{m-1} 2^i = m2^m - 2^m \sum_{i=1}^m 2^{-i} \approx m2^m - 2^m = 2^m(m-1)$$

Il en ressort que la complexité asymptotique de l'algorithme sans ces lignes est $O(n \log(n))$

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**).

Non. Le niveau 2 est incomplet et le niveau 3 n'est pas complet de gauche à droite.

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

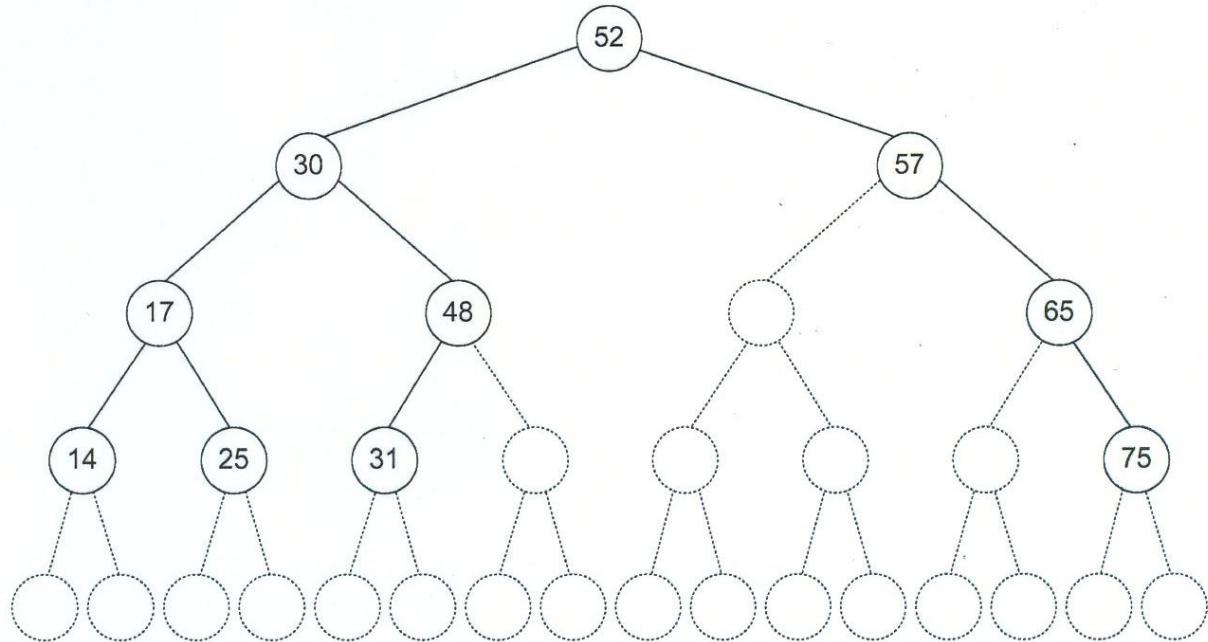
14, 17, 25, 30, 31, 48, 52, 57, 67, 70

- 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**).

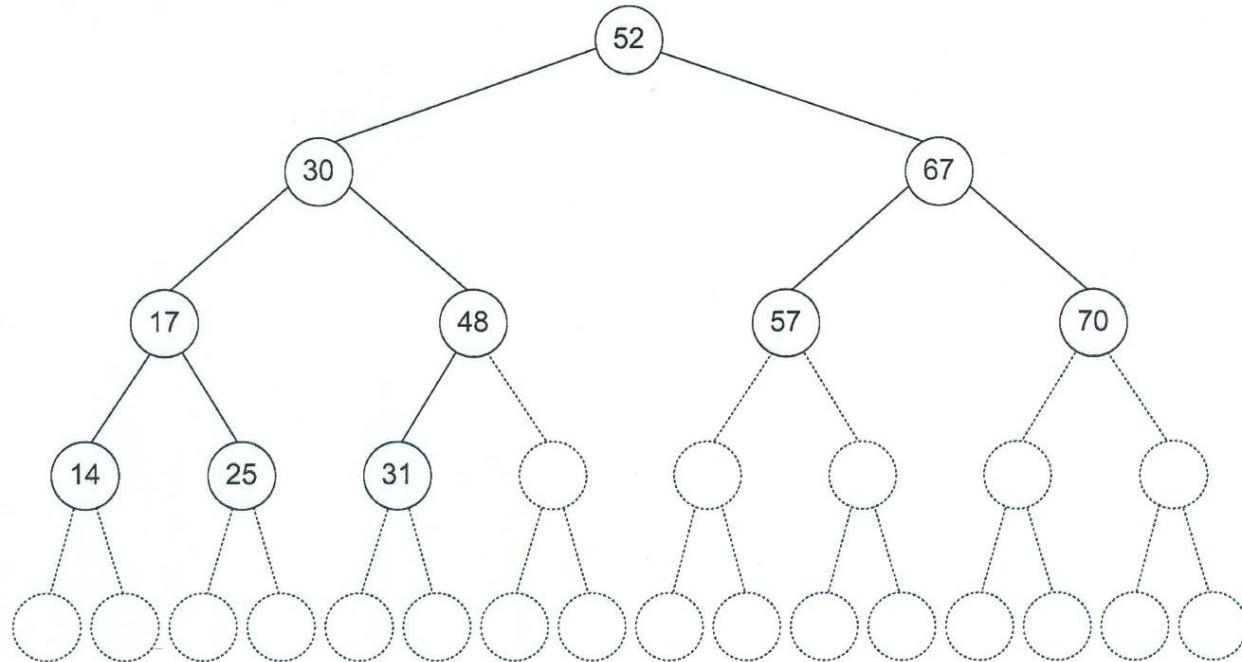
Sans cette information, il n'aurait pas été possible de dessiner l'arbre car l'affichage en ordre d'un arbre binaire de recherche produit les valeurs dans l'ordre trié, rendant impossible 'identifier les racines.

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) :

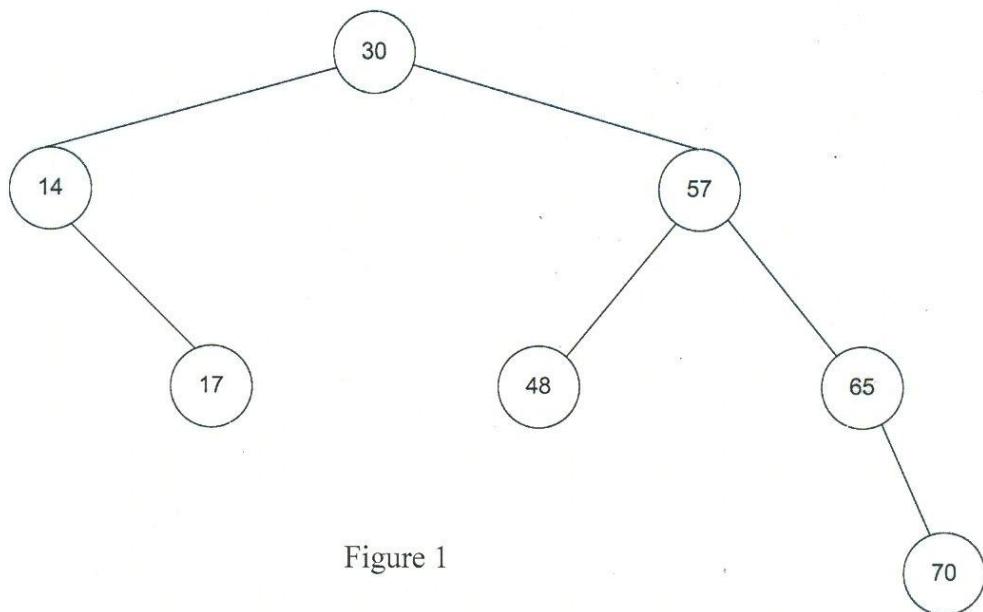


Figure 1

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

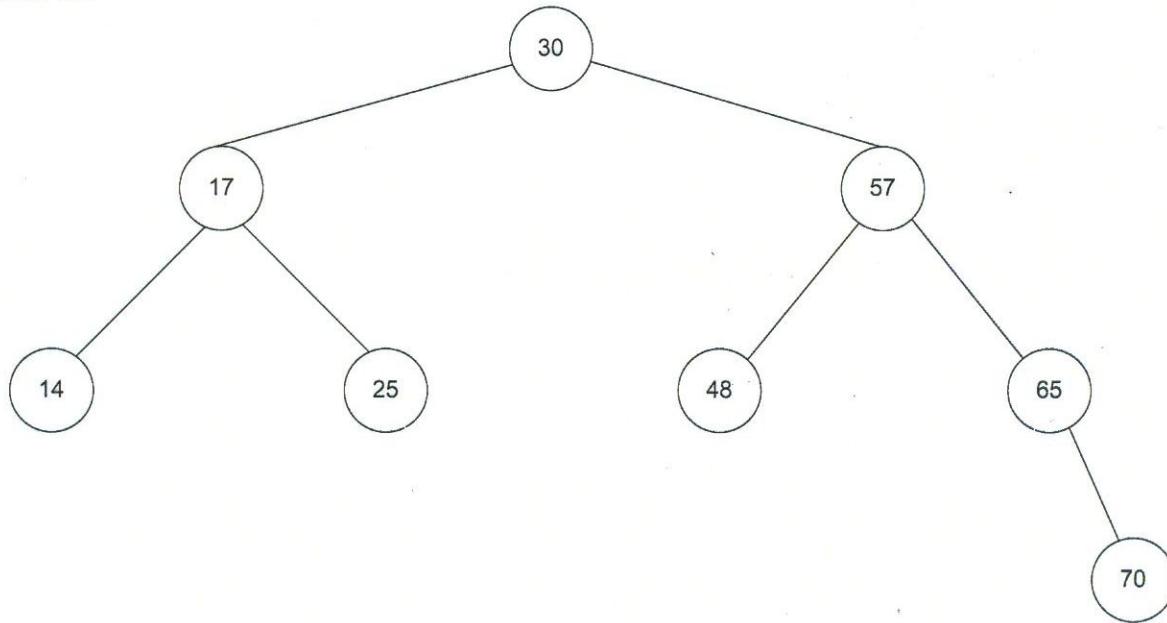
- a) Insérez 25
- b) Insérez 52
- c) Insérez 49
- d) Insérez 31
- e) Retirez 49

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).

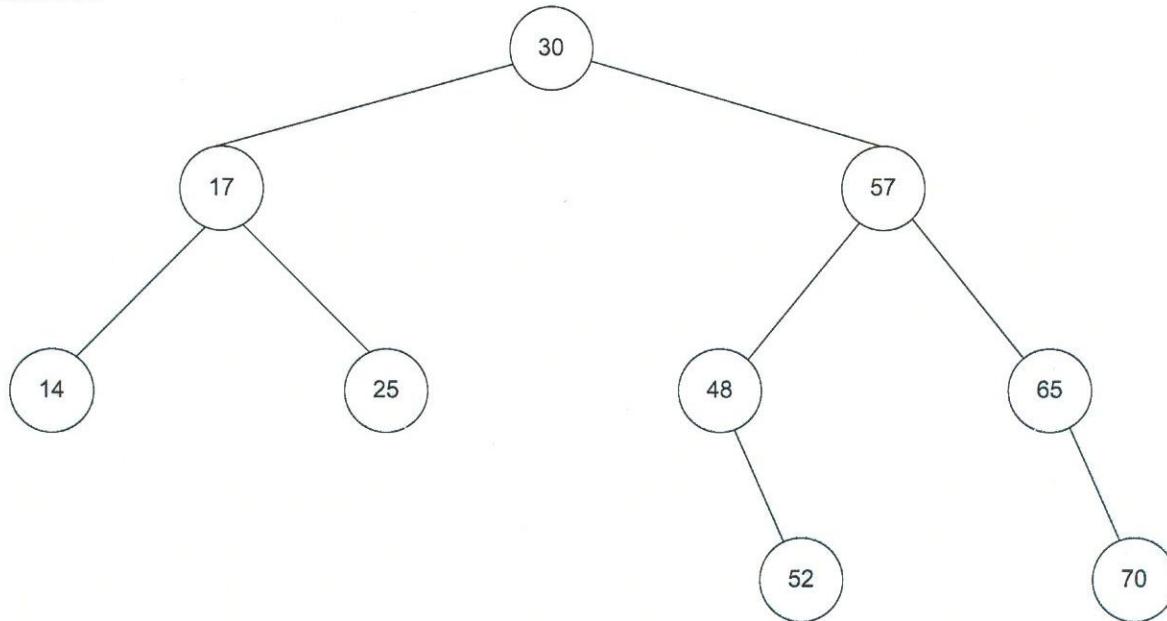
Répondez à l'**annexe 2** que vous compléterez de votre nom et que vous joindrez à votre cahier de réponse.

Annexe 2

a) Insérez 25.

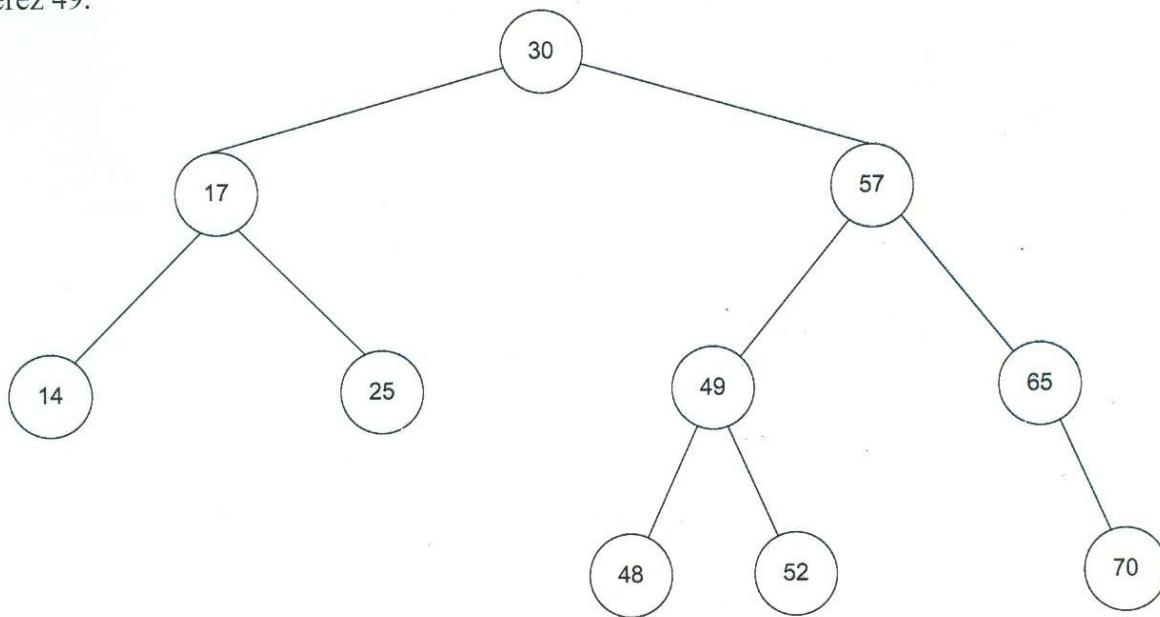


b) Insérez 52.

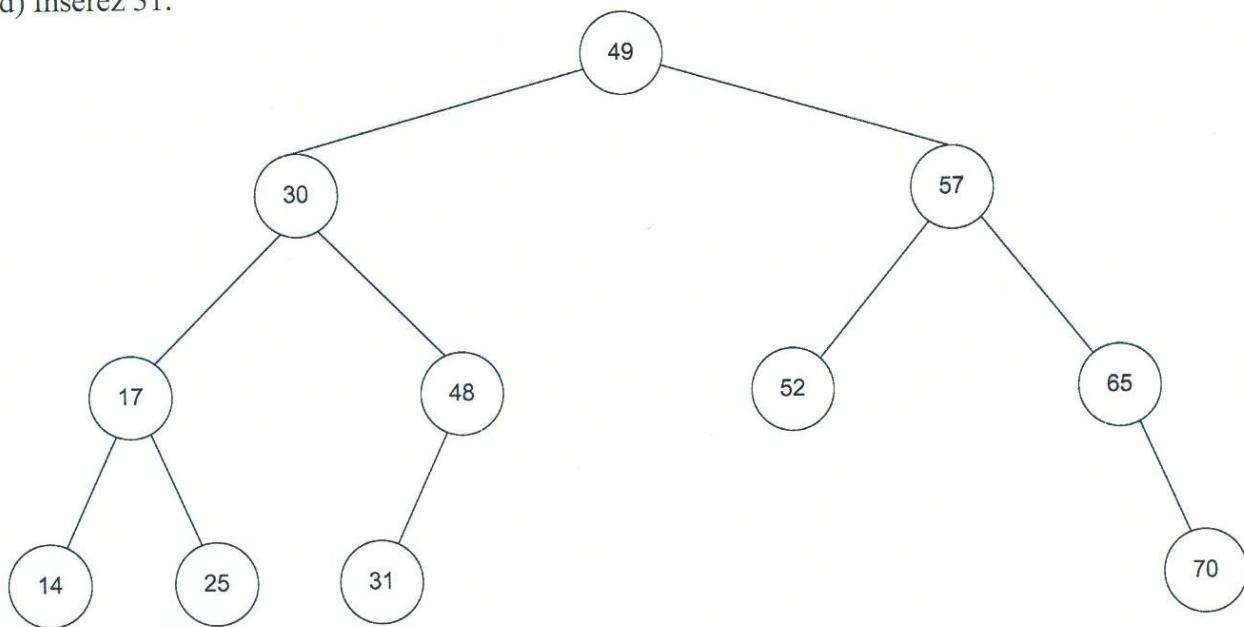


Nom : _____	Matricule : _____
Groupe : _____	

c) Insérez 49.

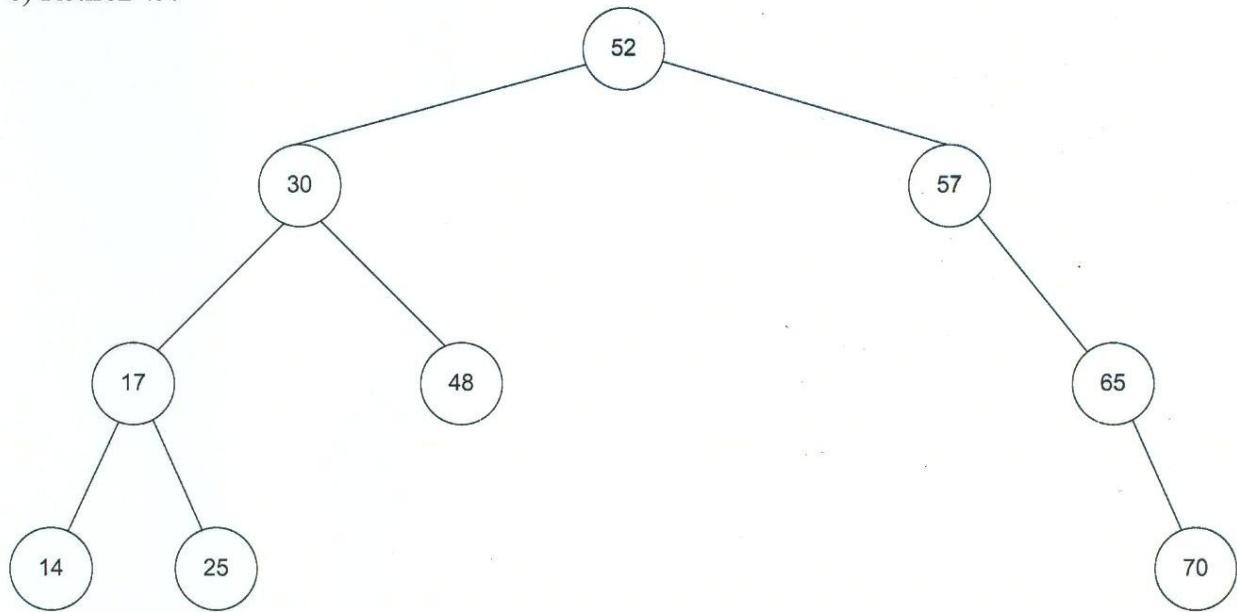


d) Insérez 31.



Nom : _____ Matricule : _____
Groupe : _____

e) Retirez 49.



Nom : _____ Matricole : _____
Groupe : _____

Question 5 – Tables de dispersion

(16 points)

Considérez :

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

- a) Considérez une table de dispersion 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 dispersion 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	66
1	11
2	
3	
4	
5	
6	50
7	51
8	30
9	74
10	

Figure 2a

Position	Table
0	66
1	11
2	
3	
4	
5	
6	50
7	51
8	30
9	74
10	

Figure 2b

Nom : _____ Matricule : _____ Groupe : _____

(Q5)

a)

key	i	h
30	0	8
74	1	9
51	0	7
66	0	8
50	0	6
11	1	1

b)

key	i	h
30	0	8
74	1 ²	9
51	0	7
66	0	8
50	0	6
11	1 ²	1

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) :

71

V_g = Partition gauche (V) :

46	20	12	37	47	27	40	67					
----	----	----	----	----	----	----	----	--	--	--	--	--

V_d = Partition droite (V) :

99	88	91										
----	----	----	--	--	--	--	--	--	--	--	--	--

b) Pivot par médiane -3 (V_g) :

46

V_{gg} = Partition gauche (V_g) :

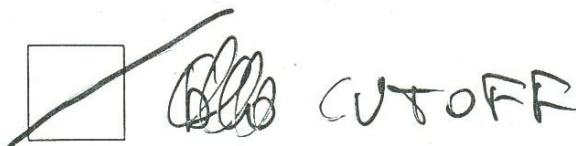
37	20	12	40	27								
----	----	----	----	----	--	--	--	--	--	--	--	--

V_{gd} = Partition droite (V_g) :

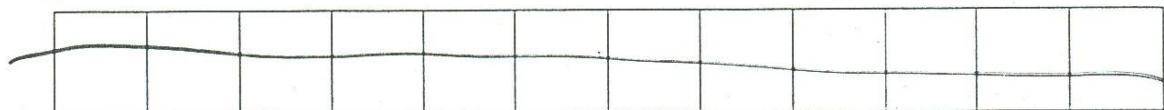
47	67											
----	----	--	--	--	--	--	--	--	--	--	--	--

Nom : _____	Matricule : _____	Groupe : _____
-------------	-------------------	----------------

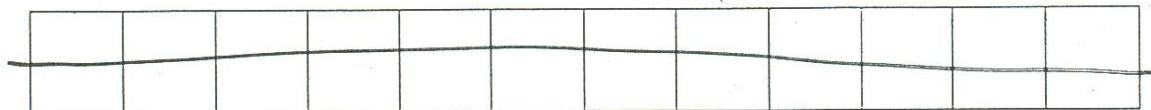
c) Pivot par médiane -3 (V_d) :



V_{dg} = Partition gauche (V_d) :



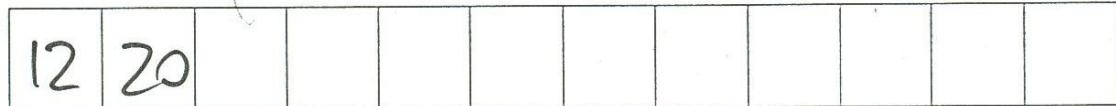
V_{dd} = Partition droite (V_d) :



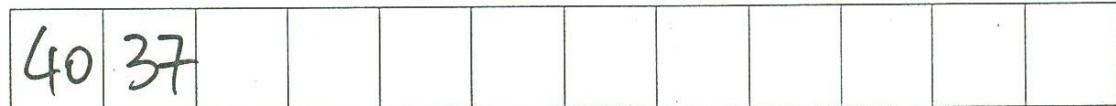
d) Pivot par médiane -3 (V_{gg}) :

27

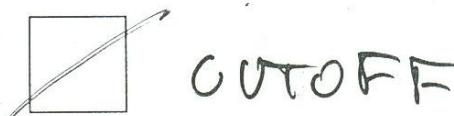
Partition gauche (V_{gg}) :



Partition droite (V_{gg}) :



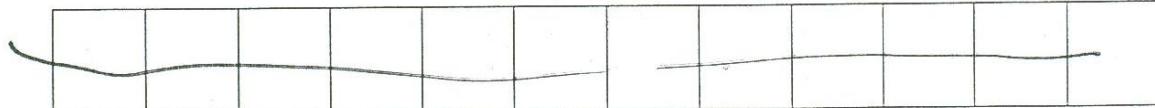
e) Pivot par médiane -3 (V_{gd}) :



Partition gauche (V_{gd}) :

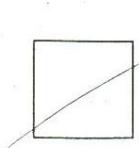


Partition droite (V_{gd}) :



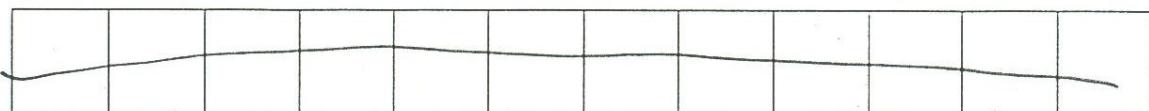
Nom : _____ Matricule : _____ Groupe : _____

f) Pivot par médiane -3 (V_{dg}) :

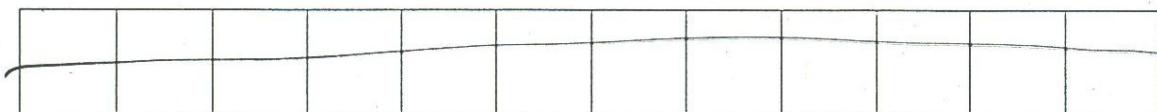


6UT0FR

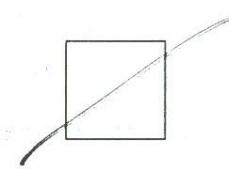
Partition gauche (V_{dg}) :



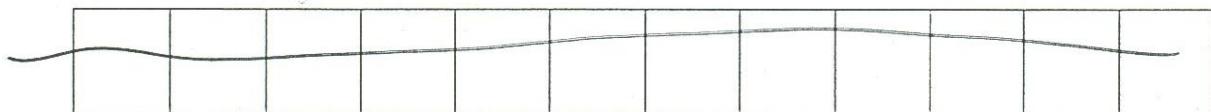
Partition droite (V_{dg}) :



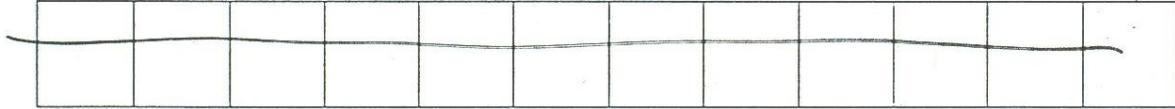
g) Pivot par médiane -3 (V_{dd}) :



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
15	quicksort(a, \emptyset , a. center - 1);
32	if (a[right]. compareTo(a[left]) < 0)
36	if (a[right]. compareTo(a[center]) < 0)
42	swapReferences(a, center, right - 1);
44	return (a[right - 1]);
57	if (left + CUTOFF <= right) {
67	while (a[i + i]. compareTo(pivot) < 0) { }
69	while (a[E - i]. compareTo(pivot) > 0) { }
71	if (i < j)
73-76	swapReferences(a, i, j);
83	swapReferences(a, i, right - 1);
84	quicksort(a, left, i - 1);
85	quicksort(a, i + 1, right);
86	• "effacer"

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  }
```

Oct 25, 10 11:24

Sort_double_n.java

Page 1/2

```

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 - 1 );
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 - 1 );
43
44     return a[ right - 1 ];
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 + CUTOFF <= right ) {
58
59         AnyType pivot = median3( a, left, right );
60
61         // Begin partitioning
62
63         int i = left, j = right - 1;
64
65         for( ; ; ) {

```

Oct 25, 10 11:24

Sort_double_n.java

Page 2/2

```
66
67         while( a[ ++i ].compareTo( pivot ) < 0 ) { }
68
69         while( a[ --j ].compareTo( pivot ) > 0 ) { }
70
71         if( i < j )
72             swapReferences( a, i, j );
73
74         else
75             break;
76
77     }
78
79     swapReferences( a, i, right - 1 );    // Restore pivot
80
81     quicksort( a, left, i - 1 );        // Sort small elements
82
83     quicksort( a, i + 1, right );       // Sort large elements
84
85 }
86
87 else // Do an insertion sort on the subarray
88     insertionSort( a, left, right );
89
90 }
91
92 }
```

Oct 19, 10 13:58

res.rep

Page 1/4

```

TO BE SORTED --->
    ARRAY: --->
        INDEX: 0 ELEMENT: 46
        INDEX: 1 ELEMENT: 20
        INDEX: 2 ELEMENT: 12
        INDEX: 3 ELEMENT: 37
        INDEX: 4 ELEMENT: 47
        INDEX: 5 ELEMENT: 91
        INDEX: 6 ELEMENT: 99
        INDEX: 7 ELEMENT: 67
        INDEX: 8 ELEMENT: 88
        INDEX: 9 ELEMENT: 40
        INDEX: 10 ELEMENT: 27
        INDEX: 11 ELEMENT: 71
    <---- ARRAY
<---- TO BE SORTED

SORTING --->
    ARRAY: --->
        INDEX: 0 ELEMENT: 46
        INDEX: 1 ELEMENT: 20
        INDEX: 2 ELEMENT: 12
        INDEX: 3 ELEMENT: 37
        INDEX: 4 ELEMENT: 47
        INDEX: 5 ELEMENT: 91
        INDEX: 6 ELEMENT: 99
        INDEX: 7 ELEMENT: 67
        INDEX: 8 ELEMENT: 88
        INDEX: 9 ELEMENT: 40
        INDEX: 10 ELEMENT: 27
        INDEX: 11 ELEMENT: 71
    <---- ARRAY
<---- SORTING

MEDIAN --->
    LEFT
        INDEX: 0
            ELEMENT: 46
    RIGHT
        INDEX: 11
            ELEMENT: 71
    CENTER
        INDEX: 5
            ELEMENT: 91
    MEDIAN
        VAL: 71
<---- MEDIAN

PARTITIONS --->
    PARTITIONED ARRAY: --->
        INDEX: 0 ELEMENT: 46
        INDEX: 1 ELEMENT: 20
        INDEX: 2 ELEMENT: 12
        INDEX: 3 ELEMENT: 37
        INDEX: 4 ELEMENT: 47
        INDEX: 5 ELEMENT: 27
        INDEX: 6 ELEMENT: 40
        INDEX: 7 ELEMENT: 67
        INDEX: 8 ELEMENT: 71
        INDEX: 9 ELEMENT: 99
        INDEX: 10 ELEMENT: 88
        INDEX: 11 ELEMENT: 91
    <---- PARTITIONED ARRAY

    INDEXES

```

Oct 19, 10 13:58

res.rep

Page 2/4

```

I: 8
J: 7
LEFT: 0
PIVOT: 8
RIGHT: 11

PIVOT VALUE: 71

LEFT PARTITION --->
    INDEX: 0 ELEMENT: 46
    INDEX: 1 ELEMENT: 20
    INDEX: 2 ELEMENT: 12
    INDEX: 3 ELEMENT: 37
    INDEX: 4 ELEMENT: 47
    INDEX: 5 ELEMENT: 27
    INDEX: 6 ELEMENT: 40
    INDEX: 7 ELEMENT: 67
<--- LEFT PARTITION

PIVOT --->
    INDEX: 8 ELEMENT: 71
<--- PIVOT

RIGHT PARTITION --->
    INDEX: 9 ELEMENT: 99
    INDEX: 10 ELEMENT: 88
    INDEX: 11 ELEMENT: 91
<--- RIGHT PARTITION

<--- PARTITIONS

SORTING --->
    ARRAY: --->
        INDEX: 0 ELEMENT: 46
        INDEX: 1 ELEMENT: 20
        INDEX: 2 ELEMENT: 12
        INDEX: 3 ELEMENT: 37
        INDEX: 4 ELEMENT: 47
        INDEX: 5 ELEMENT: 27
        INDEX: 6 ELEMENT: 40
        INDEX: 7 ELEMENT: 67
    <--- ARRAY

<--- SORTING

MEDIAN --->
    LEFT
        INDEX: 0
            ELEMENT: 46
    RIGHT
        INDEX: 7
            ELEMENT: 67
    CENTER
        INDEX: 3
            ELEMENT: 37
    MEDIAN
        VAL: 46
<--- MEDIAN

PARTITIONS --->
    PARTITIONED ARRAY: --->
        INDEX: 0 ELEMENT: 37
        INDEX: 1 ELEMENT: 20
        INDEX: 2 ELEMENT: 12
        INDEX: 3 ELEMENT: 40
        INDEX: 4 ELEMENT: 27
        INDEX: 5 ELEMENT: 46
        INDEX: 6 ELEMENT: 47
        INDEX: 7 ELEMENT: 67

```

Oct 19, 10 13:58

res.rep

```

<--- PARTITIONED ARRAY

INDEXES
    I: 5
    J: 4
    LEFT: 0
    PIVOT: 5
    RIGHT: 7

PIVOT VALUE: 46

LEFT PARTITION --->
    INDEX: 0 ELEMENT: 37
    INDEX: 1 ELEMENT: 20
    INDEX: 2 ELEMENT: 12
    INDEX: 3 ELEMENT: 40
    INDEX: 4 ELEMENT: 27

<--- LEFT PARTITION
PIVOT --->
    INDEX: 5 ELEMENT: 46

<--- PIVOT
RIGHT PARTITION --->
    INDEX: 6 ELEMENT: 47
    INDEX: 7 ELEMENT: 67

<--- RIGHT PARTITION
<--- PARTITIONS

SORTING --->
ARRAY: --->
    INDEX: 0 ELEMENT: 37
    INDEX: 1 ELEMENT: 20
    INDEX: 2 ELEMENT: 12
    INDEX: 3 ELEMENT: 40
    INDEX: 4 ELEMENT: 27

<--- ARRAY
<--- SORTING

MEDIAN --->
    LEFT
        INDEX: 0
        ELEMENT: 37
    RIGHT
        INDEX: 4
        ELEMENT: 27
    CENTER
        INDEX: 2
        ELEMENT: 12
    MEDIAN
        VAL: 27
<--- MEDIAN

PARTITIONS --->
    PARTITIONED ARRAY: --->
        INDEX: 0 ELEMENT: 12
        INDEX: 1 ELEMENT: 20
        INDEX: 2 ELEMENT: 27
        INDEX: 3 ELEMENT: 40
        INDEX: 4 ELEMENT: 37

<--- PARTITIONED ARRAY

INDEXES
    I: 2
    J: 1
    LEFT: 0
    PIVOT: 2

```

Oct 19, 10 13:58

res.rep

Page 4/4

```

RIGHT: 4

PIVOT VALUE: 27

LEFT PARTITION --->
    INDEX: 0 ELEMENT: 12
    INDEX: 1 ELEMENT: 20
<--- LEFT PARTITION
PIVOT --->
    INDEX: 2 ELEMENT: 27
<--- PIVOT
RIGHT PARTITION --->
    INDEX: 3 ELEMENT: 40
    INDEX: 4 ELEMENT: 37
<--- RIGHT PARTITION
<--- PARTITIONS

SORTING --->
    ARRAY: --->
        INDEX: 0 ELEMENT: 12
        INDEX: 1 ELEMENT: 20
    <--- ARRAY
<--- SORTING

SORTING --->
    ARRAY: --->
        INDEX: 3 ELEMENT: 40
        INDEX: 4 ELEMENT: 37
    <--- ARRAY
<--- SORTING

SORTING --->
    ARRAY: --->
        INDEX: 6 ELEMENT: 47
        INDEX: 7 ELEMENT: 67
    <--- ARRAY
<--- SORTING

SORTING --->
    ARRAY: --->
        INDEX: 9 ELEMENT: 99
        INDEX: 10 ELEMENT: 88
        INDEX: 11 ELEMENT: 91
    <--- ARRAY
<--- SORTING

SORTED --->
    ARRAY: --->
        INDEX: 0 ELEMENT: 12
        INDEX: 1 ELEMENT: 20
        INDEX: 2 ELEMENT: 27
        INDEX: 3 ELEMENT: 37
        INDEX: 4 ELEMENT: 40
        INDEX: 5 ELEMENT: 46
        INDEX: 6 ELEMENT: 47
        INDEX: 7 ELEMENT: 67
        INDEX: 8 ELEMENT: 71
        INDEX: 9 ELEMENT: 88
        INDEX: 10 ELEMENT: 91
        INDEX: 11 ELEMENT: 99
    <--- ARRAY
<--- SORTED

```