POLYTECHNIQUE MONTRÉAL

Corrigé examen intra

INF2010

Sigle du cours

Q1	
Q2	
Q3	
Q4	
Q5	
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**	**20202**
Professeur		***Local***	***Téléphone***
Tarek Ould-Bachir		**A-343.14**	
Jour	***Date***	***Durée***	***Heure***
Lundi	**1 juin 2020**	**2h30**	**13h30**

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

Directives particulières

Ne posez pas de question durant l'examen. En cas de doute sur le sens d'une question, énoncez clairement toute supposition que vous faites. Ne communiquez avec personne durant l'examen. Remettez un fichier ZIP incluant un PDF avec vos réponses et les codes sources demandés.

Bonne chance à tous!

Important

Cet examen contient **5** questions sur un total de **9** 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 : Listes (4/20 points)

Considérez le code Java qui vous est fourni dans le répertoire .sources/q1.

Vous devez être en mesure de compiler ce code après avoir complété la ligne 9 du fichier `q1.java` avec vos informations personnelles :

```
private static final int MON_MATRICULE = ___________; // <= A COMPLÉTER
```

a) **(1 point)** Complétez la méthode `shuffleList_1(…)` qui mélange une instance de l'interface `List` en appelant les méthodes `set(…)` et `get(…)` de `List`. **Remettez le fichier modifié dans un zip avec votre examen.**

```
private static <AnyType> void shuffleList_1(List<AnyType> list) {
   int n = list.size();
   Integer[] position = getArray(n);
   pAlgorithm(position, n);

   // Permuter les elements aux indices i et j
   // utiliser les methodes list.get(...) et list.set(...)
   for(int i=0; i<list.size(); i++) {

      int j = position[i];

      if( i != j) {
         AnyType ei = list.get(i);
         AnyType ej = list.set(j, ei);
         list.set(i, ej);
      }
   }
}
```

b) **(0.5 point)** Donnez la complexité asymptotique en pire cas de la méthode `shuffleList_1(…)` si l'instance de `List` est une `ArrayList`. Justifiez clairement votre réponse.

Pour une `ArrayList`, les méthodes `list.get(…)` et `list.set(…)` s'exécutent en O(1) en pire cas. La complexité asymptotique de la méthode est donc O(n) si `list` est un `ArrayList`.

c) **(0.5 point)** Donnez la complexité asymptotique en pire cas de la méthode `shuffleList_1(…)` si l'instance de `List` est une `LinkedList`. Justifiez clairement votre réponse.

Pour une `LinkedList`, les méthodes `list.get(…)` et `list.set(…)` s'exécutent en O(n) en pire cas. La complexité asymptotique de la méthode est donc O(n^2) si `list` est un `LinkedList`.

d) **(1 point)** Complétez la méthode `shuffleList_2(…)` qui mélange une instance de l'interface liste en appelant les méthodes `next()` et `set(…)` de `ListIterator`. **Remettez le fichier modifié dans un zip avec votre examen.**

```
private static <AnyType> void shuffleList_2(List<AnyType> list) {
   int n = list.size();
   Integer[] position = getArray(n);
   pAlgorithm(position, n);

   // Permuter les elements aux indices i et j
   // Utiliser listItertator et ses methodes next(...) et set(...)
   for(int i=0; i<list.size(); i++) {
      int j = position[i];

      if( i != j) {
         ListIterator<AnyType> iti = list.listIterator(i);
         ListIterator<AnyType> itj = list.listIterator(j);

         AnyType ei = iti.next();
         AnyType ej = itj.next();

         iti.set(ej);
         itj.set(ei);
      }
   }
}
```

e) **(0.5 point)** Donnez la complexité asymptotique en pire cas de la méthode `shuffleList_2(…)` si l'instance de `List` est une `ArrayList`. Justifiez clairement votre réponse.

Pour une `ArrayList`, les méthodes `list.listIterator(i)` et `ListIterator.set(…)` s'exécutent en O(1) en pire cas. La complexité asymptotique de la méthode est donc O(n) si `list` est un `ArrayList`.

f) **(0.5 point)** Donnez la complexité asymptotique en pire cas de la méthode `shuffleList_2(…)` si l'instance de `List` est une `LinkedList`. Justifiez clairement votre réponse.

Pour une `LinkedList`, la méthode `list.listIterator(i)` s'exécutent en O(n) et `ListIterator.set(…)` s'exécute en O(1) en pire cas. La complexité asymptotique de la méthode est donc O(n^2) si `list` est un `LinkedList`.

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

Considérez le code Java qui vous est fourni dans le répertoire .sources/q2.

Vous devez être en mesure de compiler ce code après avoir complété la ligne 3 du fichier `q2.java` avec vos informations personnelles :

```
private static final int MON_MATRICULE = ___________; // <= A COMPLÉTER
```

a) **(1 point)** Complétez la méthode `getFc()` de la classe `SeparateChainingHashTable` fournie .sources/q2. La méthode doit retourner le facteur de compression de la table de dispersement. **Remettez le fichier modifié dans un zip avec votre examen.**

```
public double getFc() {
   return ((double)currentSize)/theLists.length;
}
```

b) **(1 point)** Reproduisez l'affichage résultat de l'exécution de la fonction principale `main(…)` suite à la complétion de la question 2.a et l'ajout de votre identifiant personnel à la ligne 3 du fichier `q2.java`.

L'affichage obtenu dépend du matricule utilisé. Pour 1625144, on a :

```
Rehash Factor = 2   Nb rehashes: 14          Compression factor: 0.4565618642053624
Rehash Factor = 4   Nb rehashes: 7           Compression factor: 0.48435621420920333
Rehash Factor = 8   Nb rehashes: 5           Compression factor: 0.2451563598864505
Rehash Factor = 16  Nb rehashes: 4           Compression factor: 0.12248976338854413
Rehash Factor = 32  Nb rehashes: 3           Compression factor: 0.24408252963353572
Rehash Factor = 64  Nb rehashes: 3           Compression factor: 0.30666402134663068
```

c) **(1 point)** Le rôle de la fonction principale `main` est de déterminer l'impact du facteur de rehash (`rehashFactor`) sur le comportement de la classe `SeparateChainingHashTable`. Après inspection du code fourni dans `SeparateChainingHashTable` et en considérant l'affichage obtenu à la question 2.b, quel est selon vous l'impact du `rehashFactor` sur le nombre de fois que la méthode `rehash(…)` est appelé durant l'exécution de `testHashFactor(…)` ? Justifiez clairement votre réponse.

Plus la variable `rehashFactor` est grande, moins il y a d'appels au rehash. Ceci s'explique par le fait que la taille du tableau contenant les listes chainées est de plus en plus grand par un facteur qui l'est conséquemment.

d) **(1 point)** Après inspection du code fourni dans `SeparateChainingHashTable`, la complétion de la question 2.a et en considérant l'affichage obtenu à la question 2.b, quel est selon vous l'impact du `rehashFactor` sur le facteur de compression ? Justifiez clairement votre réponse.

Il n'y a pas de lien entre le rehash factor et le compression factor. Le compression factor est le ratio entre le nombre d'éléments présents sur la taille du tableau de listes chainées. Il est donc indépendant de la variable `rehashFactor` qui indique le facteur par lequel le tableau est accru

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

Considérez le code Java qui vous est fourni dans le répertoire .sources/q3.

Les résultats dépendent du matricule. Ce qui suit s'obtient avec 1625144.

Vous devez être en mesure de compiler ce code après avoir complété la ligne 6 du fichier `q3.java` avec vos informations personnelles :

```
private static final int MON_MATRICULE = ___________; // <= A COMPLÉTER
```

a) **(0.5 point)** Suite à l'ajout de votre identifiant personnel à la ligne 6 du fichier `q3.java`, indiquez le contenu et la taille du tableau trié par les méthodes `mergesort(…)`, `quicksort(…)` et `heapsort(…)`.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Valeurs	10	8	7	9	3	4	1	6	0	2	5					

Taille du tableau: 11

b) **(0.5 point)** Au total, quel est le nombre de fois que la méthode récursive `mergeSort` aura été appelée pour exécuter le tri ? Pour éviter toute ambiguïté, la signature de la fonction est reproduite ci-après.

Signature de la fonction considérée :

```
private static <AnyType extends Comparable<? super AnyType>>
void mergeSort( AnyType [ ] a, AnyType [ ] tmpArray, int left, int right )
```

Votre réponse: 21

c) **(0.5 point)** Au total, quel est le nombre de fois que la méthode récursive `quicksort` aura été appelée pour exécuter le tri ? Pour éviter toute ambiguïté, la signature de la fonction est reproduite ci-après.

Signature de la fonction considérée :

```
private static <AnyType extends Comparable<? super AnyType>>
void quicksort( AnyType [ ] a, int left, int right )
```

Votre réponse: 7

d) **(0.5 point)** Au total, quel est le nombre de fois que la méthode `percDown` aura été appelée pour exécuter le tri ? Pour éviter toute ambiguïté, la signature de la fonction est reproduite ci-après.

Signature de la fonction considérée :

```
private static <AnyType extends Comparable<? super AnyType>>
void percDown( AnyType [ ] a, int i, int n )
```

Votre réponse: 15

e) **(1.5 point)** Donnez la complexité asymptotique de chacun des tris considérés ?

Merge Sort : $O(n\log(n))$

Quick Sort : $O(n\log(n))$

Heap Sort : $O(n\log(n))$

f) **(0.5 point)** Quelle est selon vous la méthode de tri la plus appropriée pour le problème traité ? Justifiez votre réponse.

QuickSort qui donne les meilleurs résultats en temps de calcul.

Question 4 : Arbres binaire de recherche (4/20 points)

Considérez le code Java qui vous est fourni dans le répertoire .sources/q4.

Les résultats dépendent du matricule. Ce qui suit s'obtient avec 1625144.

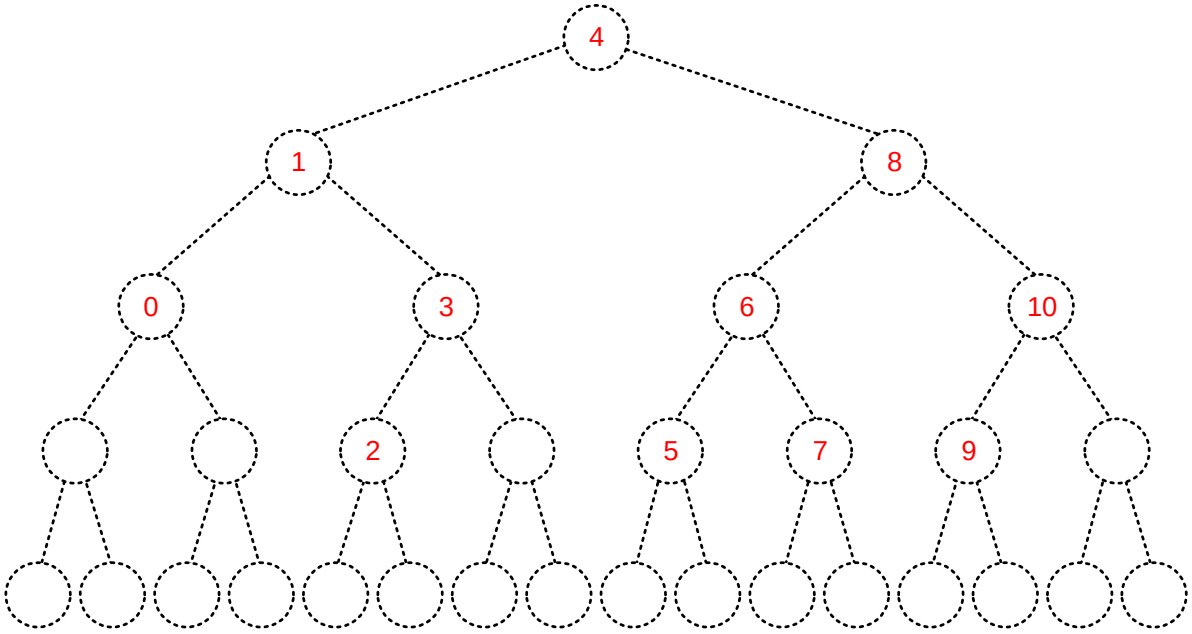
Vous devez être en mesure de compiler ce code après avoir complété la ligne 4 du fichier `q4.java` avec vos informations personnelles :

```
private static final int MON_MATRICULE = ____________; // <= A COMPLÉTER
```

a) **(1 point)** Donnez la représentation graphique de l'arbre utilisé dans l'affichage par niveaux. Reproduisez l'affichage considéré.

Affichage : `4 1 8 0 3 6 10 2 5 7 9`

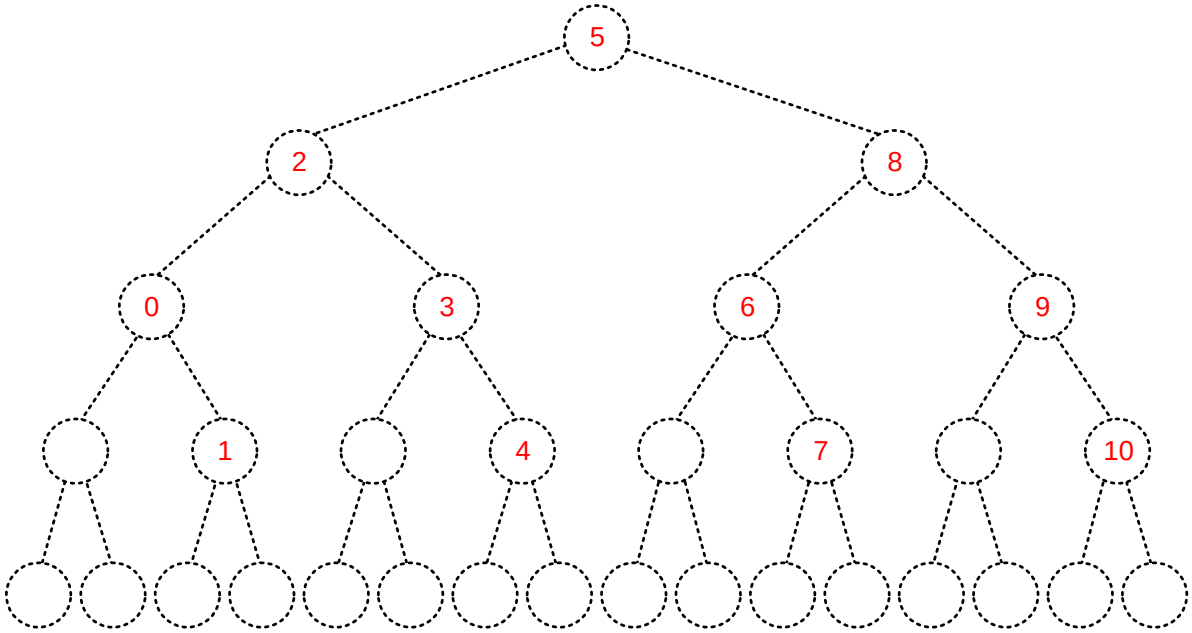
Représentation graphique de l'arbre :



b) **(1 point)** Donnez la représentation graphique de l'arbre utilisé dans l'affichage pré-ordre. Reproduisez l'affichage considéré.

Affichage : `5 2 0 1 3 4 8 6 7 9 10`

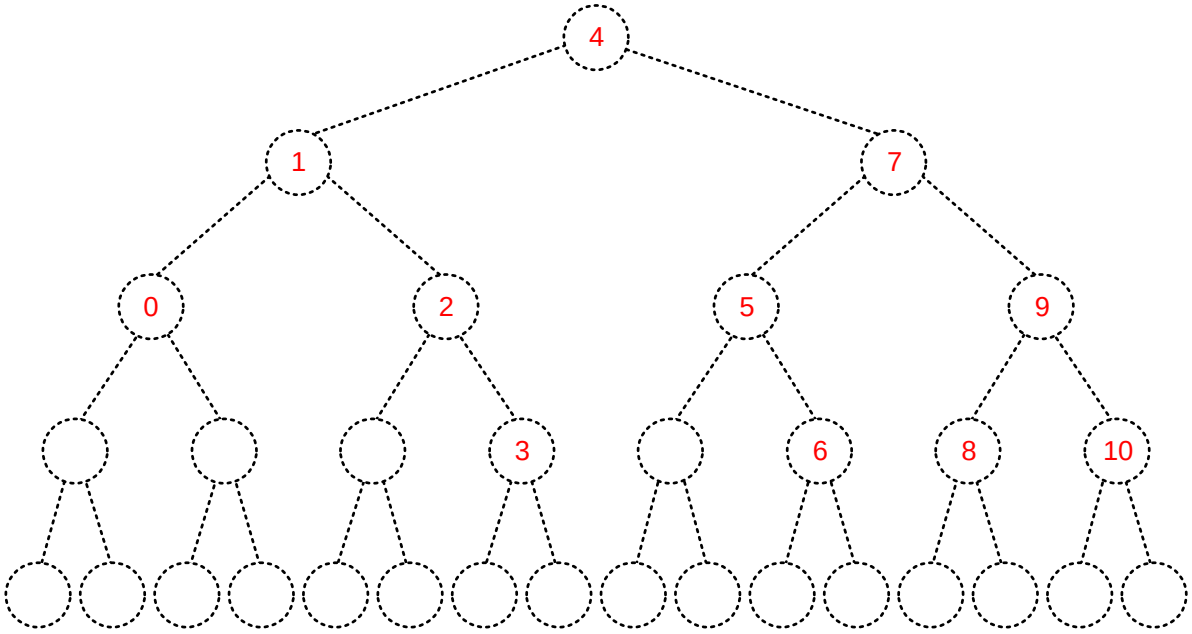
Représentation graphique de l'arbre :



c) **(1 point)** Donnez la représentation graphique de l'arbre utilisé dans l'affichage post-ordre. Reproduisez l'affichage considéré.

Affichage : `0 3 2 1 6 5 8 10 9 7 4`

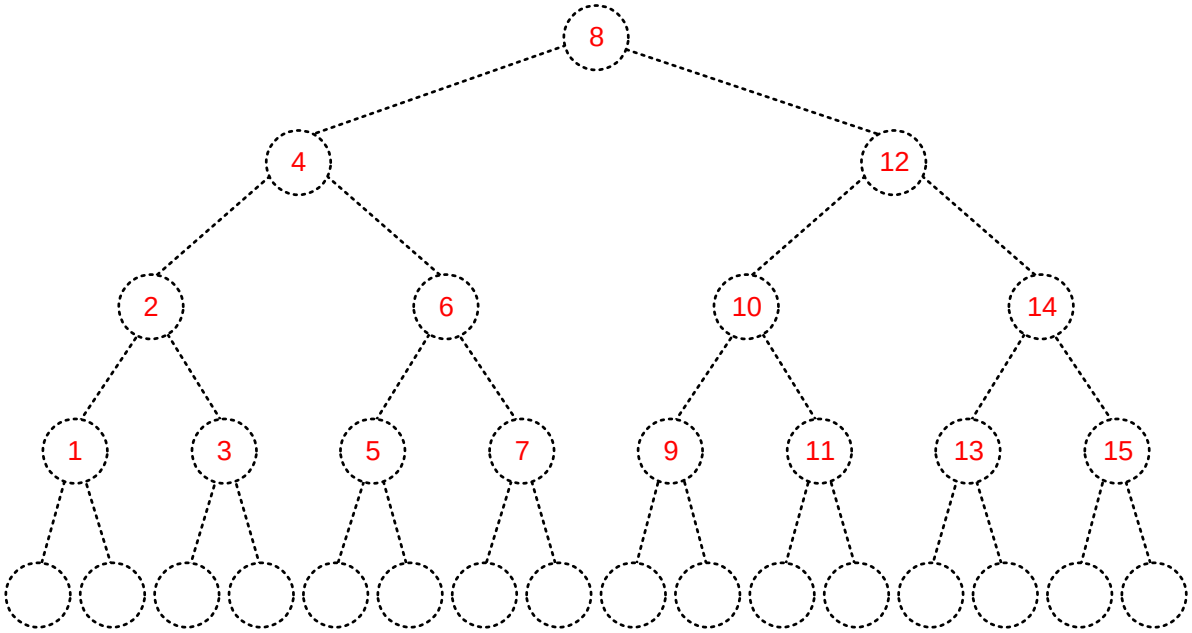
Représentation graphique de l'arbre :



d) **(1 point)** Donnez la représentation graphique de l'arbre utilisé dans l'affichage en ordre sachant qu'il s'agit d'un AVL résultant de l'insertion dans l'ordre des valeurs à 1 à 15.

Affichage : `1 2 3 4 5 6 7 8 9 10 11 12 13 14 15`

Représentation graphique de l'arbre :



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

Considérez le code Java qui vous est fourni dans le répertoire .sources/q5. Vous pouvez librement ajouter du code au fichier `q5.java` pour répondre aux questions. Cependant, il vous est interdit de modifier le fichier `AvlTree.java` qui vous est fourni.

Les résultats dépendent du matricule. Ce qui suit s'obtient avec 1625144.

Vous devez être en mesure de compiler ce code après avoir complété la ligne 4 du fichier `q4.java` avec vos informations personnelles :

```
private static final int MON_MATRICULE = ___________; // <= A COMPLÉTER
```

a) **(1 point)** Quelles sont dans l'ordre les éléments insérés dans le AVL à la partie 5.a du programme ? Une fois tous les éléments insérés, quels sont les feuilles de l'AVL ainsi obtenu ?

Éléments insérés :_ `6 17 28 39 50 61 72 83 94 105 116 127 138 149`

Feuilles de l'arbre AVL ainsi obtenu : `6 28 50 72 94 116 149`

b) **(1 point)** Les mêmes éléments que ceux utilisés dans 5.a sont insérés dans le même ordre à la partie 5.b du programme fourni dans un AVL qui tolère un débalancement d'au plus 2. Quelles sont les feuilles de l'AVL ainsi obtenu ?

Feuilles de l'arbre AVL ainsi obtenu :_ `94 6 28 50 72 116 149`

c) **(1 point)** Un AVL qui tolère un débalancement d'au plus 2 garantit-il un temps d'insertion O(log(*n*)) en pire cas? Justifiez.

Les rotations se faisant à coût constant, un débalancement de 2 maintiendra un temps d'insertion en O(log(*n*)). Il est également possible de le démontrer en exploitant la relation de récurrence :

S(0) = 1, S(1) = 2, S(2) = 3, S(h) = S(h-1) + S(h-3) + 1

h	0	1	2	3	4	5	6	7	8	9	10	11	12
S(*h*)	1	2	3	5	8	12	18	27	40	59	86	126	185

d) **(1 point)** Donnez une série d'insertions à effectuer sur un AVL qui tolère un débalancement d'au plus 2 telle qu'une double rotation est produite. On supposera le AVL initialement vide. Sur quel nœud le débalancement est-il observé au moment où la double rotation est invoquée ?

1, 4, 3, 2