Solutions de l'Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017

NOM : ..... PRENOM : ..... SECTION : .....

Partie 1 – Question 1 – Fraction irréductible

Deux solutions différentes sont proposées. La première trie les élémets des deux tableaux auparavant, ce qui permet ensuite un parcours avec un nombre d'itérations égal à la somme des longueurs de listes. La seconde solution nécessite au pire un nombre d'itérations égal au produit des longueurs de listes.

```
public static int[] reduce(int[] fp, int[] fq) {
  Arrays.sort(fp);
  Arrays.sort(fq);
  int[] reduced= new int[] { 1, 1 };
  int i= 0, j= 0;
  while ((i < fp.length) && (j < fq.length)) {
    if (fp[i] < fq[j]) {
      reduced[0] *= fp[i];
      i++;
    } else if (fp[i] > fq[j]) {
      reduced[1] *= fq[j];
      j++;
    } else {
      i++;
      j++;
    }
  }
  while (i < fp.length)
    reduced[0] *= fp[i++];
  while (j < fq.length)
    reduced[1] *= fq[j++];
  return reduced;
}
```

```
public static int[] reduce(int[] fp, int[] fq) {
  int[] reduced= new int[] { 1, 1 };
  for (int i= 0; i < fp.length; i++) {
    for (int j= 0; j < fq.length; j++) {
      if (fp[i] == fq[j]) {
        fp[i]= fq[j]= 1;
        break; // fp[i] reduced -> no more opportunity
      }
    }
  }
  int i= 0, j= 0;
  while (i < fp.length)
    reduced[0] *= fp[i++];
  while (j < fq.length)
    reduced[1] *= fq[j++];
  return reduced;
}
```

Solutions de l'Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017

NOM : ..... PRENOM : ..... SECTION : .....

Partie 1 – Question 2 – Ensemble ordonné

Les méthodes ci-dessous sont définies dans la class `ArrayOfListSet<E>`.

Q2a – `findListIndex`

```
/** Détermine l'index de la liste dans laquelle un élément doit
 * être inséré selon son ordre naturel (interface {@code Comparable})
 *
 * @param elt élément à insérer
 * @return l'index de la liste
 */
private int findListIndex(E elt) {
  int i= 0;
  /* Note : stop before N-1 as we insert the new element in the last list
   *        even if the new element is greater than the last element */
  while ((i < N-1) &&
         (lists[i].size() > 0) &&
         ((lists[i].get(lists[i].size()-1)).compareTo(elt) < 0))
    i++;
  return i;
}
```

Q2b – `insert`

```
/** Insère un élément dans la liste donnée. L'élément sera positionné
 * dans la liste selon son ordre naturel (interface {@code Comparable})
 *
 * @param i index de la liste dans laquelle insérer
 * @param elt élément à insérer
 */
private void insert(int i, E elt) throws Exception {
  int j= 0;
  int r= -1;
  List<E> l= lists[i];
  while ((j < l.size()) && ((r= l.get(j).compareTo(elt)) < 0))
    j++;
  if (r == 0)
    throw new Exception("Duplicate element \"" + elt + "\"");
  l.add(j, elt);
}
```

Q2c – `equilibrate`

```
/** Après un ajout dans une liste, propage l'élément excédentaire
```

```
 * vers le haut ou vers le bas si nécessaire.
 *
 * @param i index de la liste ou l'insertion a eu lieu
 */
private void equilibrate(int i) {
  while (i != numElts % N) {
    if (i < numElts % N) {
      lists[i+1].add(0, lists[i].remove(lists[i].size()-1));
        i++;
    } else {
      lists[i-1].add(lists[i].remove(0));
      i--;
    }
  }
}
```

Partie 2 – Question 1 – Tri de liste chaînée

Les méthodes ci-dessous sont définies dans la classe `LinkedList<E>` qui contient la classe interne `Node`.

```
private Node findMin(Node l, Comparator<E> c) {
  Node min= null;
  while (l != null) {
    if ((min == null) || (c.compare(l.data, min.data) < 0))
      min= l; l= l.next;
  }
  return min;
}

private void exch(Node n1, Node n2) {
  E tmp= n1.data;
  n1.data= n2.data;
  n2.data= tmp;
}

public void sort(Comparator<E> c)
{
  Node l= head;
  while (l != null) {
    Node min= findMin(l, c);
    exch(l, min);
    l= l.next;
  }
}
```

Solutions de l'Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017

NOM : ..... PRENOM : ..... SECTION : .....

Partie 2 – Question 2 – Itérateur pour liste chaînée

Les méthodes ci-dessous sont définies dans la classe `LinkedList<E>` qui contient la classe interne `Node`. `Filter` est une interface fonctionnelle. Elle permet notamment la définition de filtres sous la forme d'expressions lambda.

Q2a – `Filter`

```
public static interface Filter<T> {
  boolean isFiltered(T elt);
}
```

Q2b – `LinkedListIterator`

```
private class LinkedListIterator implements Iterator<E> {

  private Node next= head;

  public LinkedListIterator() {
    findNext();
  }

  private void findNext() {
    while ((next != null) && filter.isFiltered(next.data))
      next= next.next;
  }

  public boolean hasNext() {
    return (next != null);
  }

  public E next() {
    E tmp= next.data;
    next= next.next;
    findNext();
    return tmp;
  }

}
```

Partie 2 – Question 3 – Good luck !

Cette question peut être résolue en compilant et exécutant le programme fourni dans l'énoncé.