Fichiers `array1.rb` et `array2.rb`

```ruby
class Array
  # array1.rb
  #

  #
  # Dans les methodes qui suivent, pour simplifier les exercices,
  # suppose que les elements d'un tableau sont toujours des nombres
  #

  ##########################################################################
  # select et reject
  ##########################################################################

  # Retourne les elements pairs du tableau.
  #
  # Exemple:
  #   [10, 23, 30].pairs == [10, 30]
  #
  def pairs
    select { |x| x.even? }
  end

  #
```

```ruby
# Retourne une partition du tableau, en fonction d'un 'pivot'.
#
# Plus specifiquement, retourne trois sous-tableaux du tableau
# initial:
#  - Les elements plus petits que le pivot
#  - Les elements egaux au pivot
#  - Les elements plus grands que le pivot
#
# L'ordre des elements du tableau initial est respecte.
#
# Exemple:
#   [32, 10, 30, 40].partitionner(11) == [[10], [], [32, 30, 40]]
#
def partitionner( pivot )
  [ select { |x| x < pivot },
    select { |x| x == pivot },
    select { |x| x > pivot }
  ]
end

#
```

```ruby
# Tri les elements d'un tableau, et ce en ordre croissant.
#
# Exemple:
#    [32, 10, 30, 40].trier == [10, 30, 32, 40]
#
def trier
  # Note: Semblable a la methode sort... que vous ne pouvez
  # evidemment pas utiliser! Vous devez plutot utiliser
  # partitionner, et donc trier a l'aide d'une methode style
  # quicksort!
  #
  # Rappel: [10, 20] + [12, 13] == [10, 20, 12, 13]

  return [] if empty?

  petits, egaux, grands = partitionner( self[0] )
  petits.trier + egaux + grands.trier
end


#
```

```ruby
# Retourne le tableau sans les elements nil.
#
# Exemple:
#  [10, 30, nil, false, nil].compacter == [10, 30, false]
#
def compacter
  reject { |x| x.nil? }
end

#
```

```ruby
##########################################################################
#  map
##########################################################################

#  Pour chaque nombre du tableau, retourne un tableau qui contient
#  juste des 1, autant que le nombre si positif, aucun si nul ou
#  negatif.
#
#  Exemple:
#    [-1, 3, 0, 1].juste_des_1.must_equal [ [], [1,1,1], [], [1] ]
#
#  Indice:
#    (1..3).map { |i| i } == [1, 2, 3]
#    (1..3).map { ... }.size == 3
#
def juste_des_1
  map { |n| (1..n).map { 1 } }
end

#
```

```ruby
##############################################################
#  reduce
##############################################################

#  Retourne l'element maximum du tableau, ou nil si le tableau est
#  vide.
#
#  Exemples (attention au 2e!):
#    [10, 20, 30].le_max == 30
#    [-10, -20, -30].le_max == -10
#
def le_max
  # Note: Semblable a la methode max, que vous ne pouvez evidemme
  # pas utiliser.

  return nil if empty?

  reduce(self[0]) { |m, x| x > m ? x : m }
end

#
```

```ruby
# Retourne *l'index* de l'element maximum, nil si le tableau est
#
# Exemple:
#   [10, 20, 30, 11].index_du_max == 2
#
def index_du_max
  return nil if empty?

  (0...size)
    .reduce(0) do |index_max, k|
                 self[k] > self[index_max] ? k : index_max
               end
end

#
```

```ruby
# Retourne une version inversee du tableau, i.e., avec les meme
# elements mais en ordre inverse (et le tableau initial n'est pas
# modifie).
#
# Exemple:
#   [10, 20, 30].inverse == [30, 20, 10]
#
def inverse
  # Note: Semblable a la methode reverse, que vous ne pouvez
  # evidemment pas utiliser.
  #
  # Indices:
  #   [30] + [20, 10] == [30, 20, 10]
  #
  #   a = [20, 10]
  #   a.unshift 30
  #   a == [30, 20, 10]
  #

  reduce([]) { |a, x| a.unshift x }
end


#
```

```ruby
##############################################################
#  map et reduce
##############################################################

#  Retourne le nombre d'inversions entre elements adjacents, i.e.
#  nombre de paires d'elements adjacents qui ne sont pas dans le
#  ordre.
#
#  Exemple:
#    [10, 9, 8, 10, 11, 8].nb_inversions == 3
#
#  Indice:
#    A evaluer en deux passes:
#     1. map
#     2. reduce
#
def nb_inversions
  (1...size)
    .map { |k| self[k-1] > self[k] ? 1 : 0 }
    .reduce( 0, :+ )
end

#
```

```ruby
# Produit une chaine qui represente le tableau.
#
# - format_index: format (style printf) pour afficher un index
# - terminateur: chaine a inserer apres chaque element
#
# Exemple:
#    [10, 20, 30].to_chaine("%d:: ", ";") == "0:: 10;1:: 20;2:: 3
#    [10, 20, 30].to_chaine("%d:: ") == "0:: 10\n1:: 20\n2:: 30\n
#    [10, 20, 30].to_chaine == "10\n20\n30\n"
#
def to_chaine( format_index = "", terminateur = "\n" )
  (0...size)
    .map { |i| format( "#{format_index}#{self[i]}#{terminateur}"
    .join
end

#
```

```ruby
##############################################################
#  group_by
##############################################################

#  Comme partitionner plus haut, mais cette fois en utilisant
#  group_by.
#
#  Exemple:
#    [32, 10, 30, 40].partitionner_bis(11) == [[10], [], [32, 30,
#
#  Indice:
#    x <=> y == -1 si x < y
#                0 si x == y
#                1 si x > y
#
#
#  [2, 4, 6].group_by { |x| x % 2 }[0] == [2, 4, 6]
#  [2, 4, 6].group_by { |x| x % 2 }[1] == nil
#
def partitionner_bis( pivot )
  groupes = group_by { |x| x <=> pivot }
  [groupes[-1] || [],
   groupes[0]  || [],
   groupes[1]  || []
  ]
end

#
```

```
#  array2.rb
#
#
```

```ruby
# Selectionne les elements du tableau qui satisfont le predicat
# represente par le bloc.
#
# Exemple:
#   [10, 20, 12, 88].select_ { |x| x >= 20 } == [20, 88]
#
# Remarque: La methode select est evidemment definie dans la clas
# Array (voir exemples dans array1.rb). Pour le present exercice
# on fait *comme si elle n'existait pas* et qu'on devait la defi
# a partir des iterateurs de base, i.e., each et each_index.
#
def select_
  res = []
  each do |x|
    res << x if yield(x)
  end

  res
end

#
```

```ruby
# Applique un bloc sur les elements du tableau pour produire un
# nouveau tableau (de meme taille).
#
# Exemple:
#   [1, 2, 3, 8].map_ { |x| 10 * x } == [10, 20, 30, 80]
#
# Remarque: La methode map est evidemment definie dans la classe
# Array (voir exemples dans array1.rb).  Pour le present exercice
# on fait *comme si elle n'existait pas* et qu'on devait la defi
# a partir des iterateurs de base, i.e., each et each_index.
#
# Suggestion: On peut utiliser each_index...
#
def map_
  res = []
  each do |x|
    res << yield( x )
  end

  res
end


#
```

```ruby
# Applique un bloc sur les elements du tableau pour produire un
# nouveau tableau (de meme taille).
#
# Exemple:
#   [1, 2, 3, 8].map_ { |x| 10 * x } == [10, 20, 30, 80]
#
# Remarque: La methode map est evidemment definie dans la classe
# Array (voir exemples dans array1.rb).  Pour le present exercice
# on fait *comme si elle n'existait pas* et qu'on devait la defi
# a partir des iterateurs de base, i.e., each et each_index.
#
# Suggestion: On peut utiliser each_index...
#
def map_
  res = Array.new( size )
  each_index do |i|
    res[i] = yield( self[i] )
  end

  res
end


#
```

```ruby
# Itere sur tous les elements d'un tableau, mais a partir des
# elements a la fin du tableau plutot qu'a partir du debut.
#
# Exemples:
#   [10, 20, 30].inverse_each { |x| puts x }
#   30
#   20
#   10
#
#   res = []
#   [10, 20, 30].inverse_each { |x| res << x; p res }
#   [30]
#   [30, 20]
#   [30, 20, 10]
#
# Indice: Utilisez la methode inverse introduite precedemment.
#
def inverse_each
  inverse.each do |x|
    yield x
  end
end

#
```

```ruby
# Iterateur qui genere chacun des prefixes du tableau et applique
# bloc recu sur ces prefixes.
#
# Un prefixe d'un tableau est un sous-tableau qui part du debut
# ce tableau.
#
# Exemple:
#   res = []
#   [10, 20, 30].each_prefixe { |pref| res << pref }
#   res = [[], [10], [10, 20], [10, 20, 30]]
#
# Indices:
#   [10, 20, 30][0...0] = []
#   [10, 20, 30][0...1] = [10]
#   [10, 20, 30][0...2] = [10, 20]
#   etc.
#
def each_prefixe
  (0..size).each do |k|
    yield self[0...k]
  end
end

#
```

```ruby
  # Applique un bloc, qui doit avoir deux arguments, sur chaque
  # element du tableau et combine cet element a l'element
  # correspondant de l'autre tableau via le bloc.
  #
  # Exemple:
  #   [10, 20, 30].map2( [1, 2, 3] ) { |x, y| x + y } == [11, 22,
  #
  def map2( autre )
    fail "*** Tailles differentes: #{size} vs. #{autre.size}" unles

    (0...size).map do |k|
      yield( self[k], autre[k] )
    end
  end
end
```