IMPERATIVE PROGRAMMING 2016

MOISE GABRIEL

QUESTION 1

(a) + (b) - it can do both of the subtasks

```
def findArray2(a: Array[Int], b: Array[Int]) : (Int, Boolean) = {
    var N = a.size
    var M = b.size
    var i = 0
    var j = 0
    var found = false // have we found a common element yet?
    // res is going to be the smallest value common to a and b, if there is one
    // invariant I: a[0..i) < res ≤ a[i..N) && b[0..j) < res ≤ b[j..M) && 0≤i<N &&
0≤j<M && found = whether or not we have found a common element in the two arrays, or
found = a(i) == b(j) (after the iteration of the while-loop)
    while ((i<N) && (j<M))
    {
      if (a(i) == b(j)) { found = true ; return (a(i), found) }
      else if (a(i) < b(j))  i += 1 // res has to be bigger than a(i), otherwise we would have
found a(i) at an earlier step in b
      else if (a(i) > b(j))  j += 1 // same reasoning
    }
    // Because of the invariant, we cannot get out of bounds in the loop
    // If we get out of loop, that means we have not found a common element, so we return
false
    (0, found)
}
```

(c)

```
def findArray3 (a: Array[Int], b: Array[Int], c: Array[Int]) : Int = {
    var N = a.size
    var M = b.size
    var L = c.size
    var i = 0
    var j = 0
    var k = 0
    // invariant I: a[0..i) < res ≤ a[i..N) && b[0..j) < res ≤ b[j..M) && c[0..k) < res ≤ c[k..L)
&& 0≤i<N && 0≤j<M && 0≤k<L
```

```
    while ((i < N) && (j<M) && (k < L))
    {  if (a(i)==b(j)) {
           if (a(i)== c(k)) return a(i)
           else if (a(i) < c(k)) {i+=1 ; j+=1}
           else  k+=1}
       else if (a(i) < b(j)) {
           if (a(i) < c(k)) i+=1
           else if (a(i) == c(k)) {i+=1; k+=1}
           else k += 1 }
       else {
           if (b(j) < c(k)) j+=1
           else if (b(j) == c(k)) { j+=1 ; k+=1}
           else k += 1 }
    }
    0 // We will never get here as we are guaranteed to have a result
  }
(d)
def findArrays (as: Array [Array [int]]) : (int, Boolean) = {
    var N = as. size
    var m = new Array [int](N)
    for (i <- 0 until N) m(i) = as(i). size
    var index = new Array [int] (N)
    for (i <- 0 until N) index (i) = 0
    var ok = true
    // invariant i : as(i) [0.. index(i)) < res ≤ as(i)[index(i).. m(i)) && 0 ≤ index(i) < m(i)
for all i in [0..N) && ok = whether there might be a common element in the N arrays or not
    while (ok)
    {  var minIndex = -1
       var min = as(0)(index(0))
       for (i <- 1 until N) if (min > as(i)(index(i))) {min = as(i)(index(i)) ; minIndex = i}
       var maxIndex = -1
       var max = as(0) (index(0))
       for (i <- 1 until N) if (max < as(i)(index(i))) {max= as(i)(index(i)) ; maxIndex = i}
       if (as(minIndex)(index(minIndex)) == as(maxIndex)(index(maxIndex)))
          { ok = false //that means that all the values are equal, so we are done
            return (as(minIndex)(index(minIndex)), true) }
    else
      {  index(minIndex) += 1 //we need to go to the next element in this array
         if (index(minindex) >= n(minindex)) ok = false // we got out of bounds, so we stop
      }
  }
  (0, false)
  //This happens when there is no common element
}
```