Assignment 2B

> Author: Isaac Archer
>
> Due Date: July 14th 13:00 2015
>
> Problem Description: $\exists$ two sets $a, b \in \mathbb{Z}\{1..n\} \mid \exists$ a set $c \in \mathbb{Z}\{1..n\} \mid |c| \leq |a|, |b| \mid c = a \cap b$

Algorithm: Array Intersection

> input: 1-Based arrays arr_a and arr_b of positive integers. denote the size of arr_a by |arr_a| and similarly for b
>
> output: 1-Based array arr_c of positive integers, such that arr_c contains exactly one copy of each integer that appears somewhere in both arr_a arr_b. Thus, |arr_c| ≤ min(|arr_a|, |arr_b|).

```
def array_intersection ← arr_a, arr_b

      arr_c ← []
      len_a ← |arr_a|
      len_b ← |arr_b|

      for i ← 1..len_a // O(n)
            for j ← 1..len_b // O(m)
                 if arr_a[i] = arr_b[j]
                        if include ← arr_c, arr_a[i] // O(o)
                              continue
                        else
                              // input: array dest, integer to append to dest
                              // base operation
                              append ← arr_c, arr_a[i]

      ↑ arr_c
```

> input: 1-Based array arr_temp and a integer num
>
> output: true if arr_temp contains num and false if arr_temp does not contain num

```
def include ← arr_temp, num

      if |arr_temp| < 1 then ↑ false

      for k ← 1..|arr_temp|
            if arr_temp[k] = num then ↑ true

      ↑ false
```

Correctness

> - Returning the Correct Value
>
>   > The algorithm checks all the elements of arr_b for arr_a and if the elements are the same and not already in the array that holds onto elements shared in arr_a and arr_b then that value is added to the array that holds onto elements shared in arr_a and arr_b. These requirements are all met and arr_c is not appended additional copies of times when arr_a[i] == arr_b[j] because of the include function.
>
> - Halting
>
>   > the loops are the same size as arr_a and arr_b or smaller for arr_c therefore the loops

> > will end resulting in a return statement which guarantees that as long as arr_a and arr_b are finite the algorithm will halt.

Worst Case

> n = |arr_a|, m = |arr_b|, o = the size of a growing array that doesn't exceed the length of n or m, whichever is smaller.
>
> This simplifies to $O(mn^2)$ or $O(nm^2)$, for whichever of n or m is smaller.
>
> O(n) because the outer loop goes n times, and O(m) because the inner loop goes m times and O(n) or O(m) because that loop runs through an array no larger than the smaller of n or m.

Generalization to allow for any number of arrays

> ∀ sets ∈ ℤ {1..n} ∃ a number n ∈ ℤ {1..n} of sets m ∈ ℤ {1..n} | ∃ a set c ∈ ℤ {1..n} | ∀ sets m, |c| ≤ |the smallest array in m| | c = the intersection of sets in m

Algorithm 2 for Generalization

> input: an array of arrays of integers
>
> outout: an array of integers shared by the arrays in the input such that no integer appears more than once in the output array.

```
def narray_intersect ← arr_all

      len_all ← |arr_all|
      arr_c ← []

      for i ← 1..len_all
            for j ← 1..len_all
                  if i = j
                        next
                  else
                        temp_intersect ← array_intersection ← arr_all[i], arr_all[j]
                        len_temp ← |temp_intersect|
                        for k ← 1..len_temp
                              if include ← arr_c, temp_intersect[k]
                                    next
                              else
                                    // input: array dest, integer to append to dest
                                    // base operation
                                    append ← arr_c, temp_intersect[k]


      ↑ arr_c
```

> input: 1-Based arrays arr_a and arr_b of positive integers. denote the size of arr_a by |arr_a| and similarly for b
>
> output: 1-Based array arr_c of positive integers, such that arr_c contains exactly one copy of each integer that appears somewhere in both arr_a arr_b. Thus, |arr_c| ≤ min(|arr_a|, |arr_b|).

```
def array_intersection ← arr_a, arr_b

    arr_c ← []
    len_a ← |arr_a|
    len_b ← |arr_b|

    for i ← 1..len_a // O(n)
          for j ← 1..len_b // O(m)
               if arr_a[i] = arr_b[j]
                    if include ← arr_c, arr_a[i] // O(o)
                         continue
                    else
                         // input: array dest, integer to append to dest
                         // base operation
                         append ← arr_c, arr_a[i]

    ↑ arr_c
```

> input: 1-Based array arr_temp and a integer num
>
> output: true if arr_temp contains num and false if arr_temp does not contain num

```
def include ← arr_temp, num

    if |arr_temp| < 1 then ↑ false

    for k ← 1..|arr_temp|
          if arr_temp[k] = num then ↑ true


    ↑ false
```

Correctness Algorithm 2

> - Returning the Correct Value
>
>   > I built the second algorithm to use the previous one, so if the previous one is assumed to be correct then those parts in the n array intersect are still correct. The n array intersect uses a similar strategy and does an n^2 number of calls to the previous algorithm with an additional n. The algorithm for n array intersect returns the correct value as it checks all possible sets of different array combinations from the array of arrays and only adds to the array intersect array when that array does not contain the value shared by the current arr_all[i] and arr_all[j]. therefore all values in the returned array are unique and all possible combinations of arrays are checked, thus the intersection of the arrays is correctly computed.
>
> - Halting
>
>   > Like the previous algorithm this one operates on a finite array of finite arrays and so the loops will end and the return will be called halting the algorithm.

Implementation

> Implementation in Ruby (for both algorithms) with Testing

```
#! /usr/bin/env ruby
# Author: Isaac Archer
# Description:
# test for an algorithm to find the array
```

```ruby
# intersect of two arrays of length n and m

# input: two arrays of integers
# output: array of integers of the
# intersect of the arrays
def array_intersect(arr_a, arr_b)

      arr_c = []

      0.upto(arr_a.length - 1) do |i|
            0.upto(arr_b.length - 1) do |j|
                  if arr_a[i] == arr_b[j]
                        if arr_c.include? arr_a[i]
                              next
                        else
                              arr_c.push(arr_a[i])
                        end
                  end
            end
      end

      return arr_c
end

# input: array of arrays of integers
# output array of integers that is the
# intersect of the arrays in the input
def narray_intersect(arr_all)

      len_all = arr_all.length - 1
      arr_c = []

      0.upto(len_all) do |i|
            0.upto(len_all) do |j|
                  if i == j
                        next
                  else
                        temp_intersect = array_intersect(arr_all[i], arr_all[j])
                        temp_intersect.each do |k|
                              if arr_c.include? k
                                    next
                              else
                                    arr_c.push(k)
                              end
                        end
                  end
            end
      end

      return arr_c

end

def test_arr_all_int(num_arr, arr_len)

      arr_all = []

      0.upto(num_arr - 1) do |l|
            temp_arr = []
            0.upto(arr_len - 1) { |m| temp_arr.push(rand(arr_len) + 1) }
            arr_all.push(temp_arr)
            temp_arr = nil
      end

      i = 1

      arr_all.each do |a|
```

```
      arr_all.each do |a|
            print "ARRAY #{i}: "
            STDOUT.flush
            puts a.to_s
            i = i + 1
      end

      arr_c = narray_intersect(arr_all)

      print "INTERSECT ARRAY: "
      STDOUT.flush
      puts arr_c.to_s
end

def test_arr_int(len_a, len_b)

      arr_a = []
      arr_b = []

      0.upto(len_a - 1) { |i| arr_a.push(rand(len_a) + 1) }
      0.upto(len_b - 1) { |i| arr_b.push(rand(len_b) + 1) }

      print "ARRAY A: "
      STDOUT.flush
      puts arr_a.to_s

      print "ARRAY B: "
      STDOUT.flush
      puts arr_b.to_s

      arr_c = array_intersect(arr_a, arr_b)

      print "INTERSECT ARRAY: "
      STDOUT.flush
      puts arr_c.to_s

      return 0
end

def main()

      if ARGV.length != 4
            puts "invalid arguments"
            return 0
      end

      len_a = ARGV[0].to_i
      len_b = ARGV[1].to_i

      num_arr = ARGV[2].to_i
      arr_len = ARGV[3].to_i

      test_arr_int(len_a, len_b)

      test_arr_all_int(num_arr, arr_len)
end

main
```

> The algorithm slows down considerably as the input size increases above small values like 100 or 1000.
>
> - n and m size 100 completed in 0.044s
> - n and m size 1000 completed in 0.204s

> - n and m size 10000 completed in 17.774s
> - n and m size 100000 completed in a really long time (significantly over an hour)