Neal Noble  
IT333  
Week 2 [Pseudocode](https://egator.greenriver.edu/courses/1273210/modules/items/16924257)  
April 2016

Note: for each of the problems below give well-formed pseudocode, as described in class. An example is given below:

algorithm add(first, second)  
//input: integers first and second  
//output: the sum of first and second  
  
result <-- first + second  
return result

**1. Give the pseudocode for an algorithm that reorders the elements of the following array:  
A = [x, y, z]  --> A = [y, z, x]**

**algorithm** **rotateLeft**(***arrayOfItems***)  
*// input: array of items  
// output: All items in array are rotated one position to the left and  
 first item is added to end of array*

***firstItem*** 🡨 ***arrayOfItems*** [0]  
**initialize *i***  
while ***i*** < **arrayOfItems.length - 1**  
 ***arrayOfItems***[i] 🡨 ***arrayOfItems*** [pre-increment ***i***]

***arrayOfItems*** [***arrayOfItems***.**length** - 1] 🡨 firstItem;  
return ***arrayOfItems***

 What is the minimum number of assignment statements needed?

For this algorithm the minimum assignment is array.length+1.

A three element array of “*[x, y, z]*” would require 4 assignments

**2. Give the pseudocode for an algorithm that finds the sum of all integers in an array.**

**algorithm** **sumArray**(***intArray***)  
*// input: array of sorted Integers lowest to highest  
// output: the sum total of all Integers in array*

**initialize** *total*

**initialize *i*  
while i < *intArray*.length   
 total 🡨 total + *intArray* [post increment** i**];**

**return *total***

**3. Give the pseudocode for an algorithm that takes as an input a list of n integers in increasing order and produces a list of all values that occur more than once.**

**algorithm** **findDuplicateNumbers**(***intArray***)  
*// input: array of sorted Integers lowest to highest  
// output: List of integers that are duplicate numbers*

**initialize** previousNumber as integer

**initialize** previousDuplicateNumber as integer

**initialize** duplicateListas Arrray List

previousNumber 🡨 ***intArray*** [0]  
previousDuplicateNumber 🡨 ***intArray*** [***intArray***.**length** - 1]

// if the array has zero or one elements, then there are no duplicate numbers.

// No point of going further. Exit  
**if intArray**.**length** = 0 or **intArray**.**length** = 1  
 **return**

// beyond this point, we know there are at least two elements in array.  
  
// If first and last elements are equal, then all elements in-between

// are the same number. Add number to dup list and exit.  
**if intArray** [0] = **intArray** [**intArray**.**length** - 1]  
 add previousNumber to duplicateList  
 **return**  
  
  
// During loop check if current element matches previous element, and then

// check if element matches previously added duplicate number. Only add

// number to duplicate list if it has not been previously added

**for** i 🡨 1 to ***intArray***.**length**  
 **if *intArray*** [i] = previousNumber

**and** ***intArray*** [i]

**not** previousDuplicateNumber  
 **add** ***intArray*** [i] **to** duplicateList  
 previousDuplicateNumber 🡨 ***intArray*** [i]  
   
 previousNumber 🡨 ***intArray*** [i]

return duplicateList

**4. Give the pseudocode for an algorithm that takes as an input a list of n integers in increasing order and returns the mode of the list (the value that appears most often).**

**algorithm mode**(***intArray***)  
// input: array of sorted Integers lowest to highest  
// output: Mode of the array  
 **initialize** dupCount 🡨 0  
 **initialize** dupNumber 🡨 ***intArray*** [0]  
 **initialize** mostDupCount 🡨 1  
 **initialize** mostDupNumber 🡨 ***intArray*** [0]  
  
  
 // if first and last elements are equal, then all elements in-between are   
 // the same number. Return full length of array. Exit**if *intArray***[0] = ***intArray*** [***intArray***.**length** - 1]  
 **return *intArray***.**length**  
   
  
 // If the array is one, then it is the largest. exit  
**if *intArray***.**length** = 1  
 **return *intArray***[0]  
  
   
 // Check if current element match previous element, if true then   
 // increment duplicate count. Continue**for** i 🡨 1 to ***intArray***.**length**  
 **if** (***intArray***[i] = dupNumber  
 increment dupCount  
  
 // Always continue except for the very last element, let last element   
 // fall through**if** i < ***intArray***.**length** - 1  
 **continue**  
  
  
 // Update mostDupCount and mostDupCount if dupCount is greater than previous  
 // duplicate count ceiling**if** dupCount > mostDupCount  
 mostDupCount 🡨 dupCount  
 mostDupNumber 🡨 dupNumber  
   
 *// End of* i *loop. Reset duplicate count and number* dupCount 🡨 1  
 dupNumber 🡨 ***intArray*** [i]  
   
 // End of algorithm, return results   
 **return** mostDupNumber

**5. Give the pseudocode for an algorithm that inserts an integer x in the appropriate position of a list of sorted integers.***Example inputs: A = [2, 5, 13, 21],   x = 10          
Example list after the algorithm completes: A = [2, 5, 10, 13, 21]*

**algorithm insertValue**(intArray, insertNum)  
*// parameter 1: array of sorted Integers lowest to highest*

*// parameter 2: number to be inserted into array  
// output: array with value inserted at appropriate position in list.*  
 **initialize** index 🡨 0  
 **initialize** newArray 🡨 **the size of** intArray.**length** + 1

// create a mirror of original array upto where insertNum is greater than

// current index element   
 **while** intArray[index] < insertNum  
 newArray[index] 🡨 intArray[index]  
 increment index  
   
 // insert number at the current index, increment index  
 newArray[index] 🡨 insertNum  
 increment index

// Add remaining elements from original array   
 **while** index < intArray.**length**  
 newArray [index] 🡨 intArray [index - 1]  
 increment index

// end of algorithm   
**return** newArray