Polina Anis'kina and Edouard Larouche
ID's: 26991092 and 27826907
COMP-352-F
October 21-2016

Assignment #2 (Programming part)

Pseudocode of a recursive algorithm

```
Algorithm: RightWingCave(A,i,counter)
Input: array A at index i, with counter that counts how many iterations were done
Output: true if it reaches the last index, false if it does not.

if i<0 or i≥ A.length or counter > A.length
    return false
else if A[i]⟵0
    return true
else
    return RightWingCave(A, i + A[i], ++counter) or RightWingCave(A, i - A[i], ++counter)
```

Pseudocode using ArrayList

```
Algorithm: RightWingCaveArrayList(A)
Input: array A
Output: true if it reaches the last index, false if it does not.

ArrayList<Integer> list ⟵ new ArrayList<Integer> (A.length)
for i = 0 to n
    list.add(A[i])
end loop
i ⟵ 0
j ⟵ 0
counter ⟵ 0

while true do
    if list.get(i)==0 then
        return true
    if list.size() - i > list.get(i) then
        j ⟵ i + list.get(i)
    else if list.get(i) < i then
        j ⟵ i - list.get(i)
    else
        return false
    if counter > list.size()
        return false;
    counter++
    i ⟵ j
end loop
```

Part A

The time complexity of the recursive algorithm is O(n) because the last recursive steps happens n times, and the space complexity is n because the length of the array is n.
For the iterative algorithm it is O(n) because at max it will loop n times (the length of the array), and the space complexity is n because it uses an ArrayList of size n.

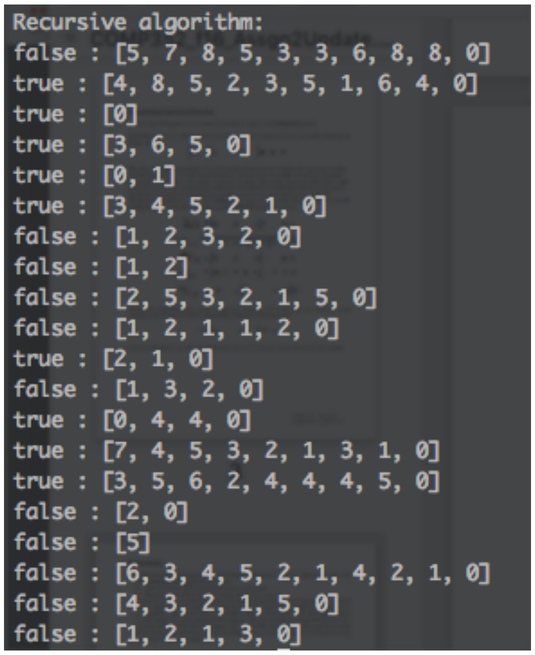
Part B

This is a linear tails recursive algorithm does not have an impact on the time or memory complexity.

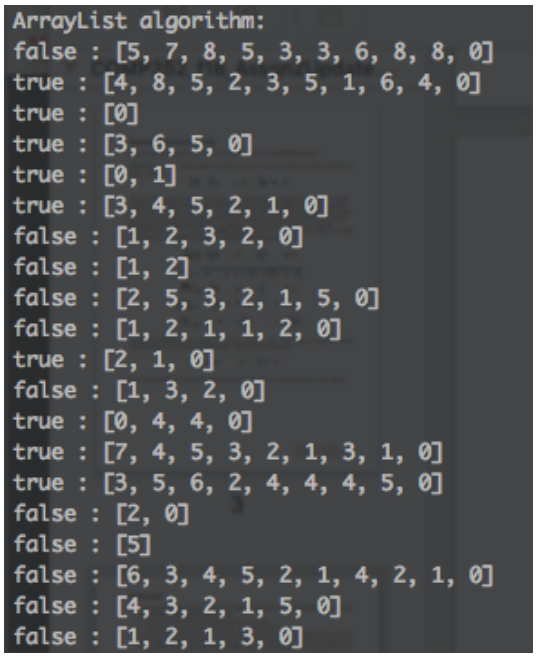
Part C

For the iterative algorithm we chose to implement it using an ArrayList because the problem does not require to add or remove values but only check them therefore, there would be no purpose in using a queue or a stack.

Part D



```
Recursive algorithm:
false : [5, 7, 8, 5, 3, 3, 6, 8, 8, 0]
true : [4, 8, 5, 2, 3, 5, 1, 6, 4, 0]
true : [0]
true : [3, 6, 5, 0]
true : [0, 1]
true : [3, 4, 5, 2, 1, 0]
false : [1, 2, 3, 2, 0]
false : [1, 2]
false : [2, 5, 3, 2, 1, 5, 0]
false : [1, 2, 1, 1, 2, 0]
true : [2, 1, 0]
false : [1, 3, 2, 0]
true : [0, 4, 4, 0]
true : [7, 4, 5, 3, 2, 1, 3, 1, 0]
true : [3, 5, 6, 2, 4, 4, 4, 5, 0]
false : [2, 0]
false : [5]
false : [6, 3, 4, 5, 2, 1, 4, 2, 1, 0]
false : [4, 3, 2, 1, 5, 0]
false : [1, 2, 1, 3, 0]
```



```
ArrayList algorithm:
false : [5, 7, 8, 5, 3, 3, 6, 8, 8, 0]
true : [4, 8, 5, 2, 3, 5, 1, 6, 4, 0]
true : [0]
true : [3, 6, 5, 0]
true : [0, 1]
true : [3, 4, 5, 2, 1, 0]
false : [1, 2, 3, 2, 0]
false : [1, 2]
false : [2, 5, 3, 2, 1, 5, 0]
false : [1, 2, 1, 1, 2, 0]
true : [2, 1, 0]
false : [1, 3, 2, 0]
true : [0, 4, 4, 0]
true : [7, 4, 5, 3, 2, 1, 3, 1, 0]
true : [3, 5, 6, 2, 4, 4, 4, 5, 0]
false : [2, 0]
false : [5]
false : [6, 3, 4, 5, 2, 1, 4, 2, 1, 0]
false : [4, 3, 2, 1, 5, 0]
false : [1, 2, 1, 3, 0]
```

Part E

To detect unsolvable array configurations:

- To see if there is a value at index of the array which is greater than the length of the array.
- If the last element of the array is not equal to 0.
- Check if (A[i]==A[i+A[i]]) || (A[i]==A[i-A[i]]) is true .

Checking these cases right off the start, increases the chance of speeding up the execution time because returns false right away.