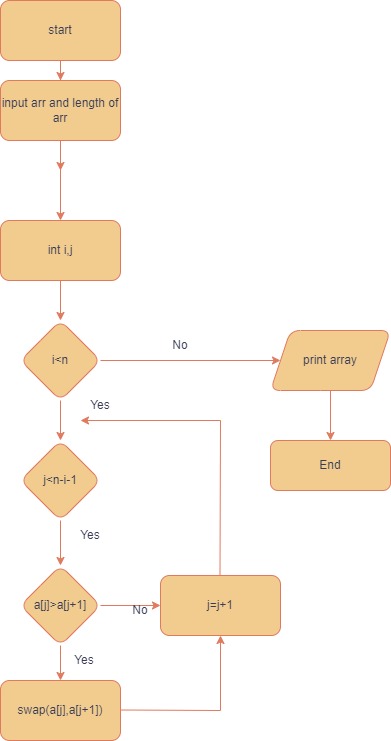
**Assignment 1: Pseudocode and Flowchart for Sorting Algorithm - Write  
pseudocode and create a flowchart for a bubble sort algorithm. Provide a  
brief explanation of how the algorithm works and a simple array of integers  
to demonstrate a dry run of your algorithm.**

**FlowChart**

****

**Pseudo code**

procedure bubbleSort(A : list of integer items)

// Determine the length of the array

n = length(A)

// Initialize a flag to check if any swaps were made in the last pass

swapped = false

// Outer loop to control the number of passes

repeat

// Reset the swapped flag for each pass

swapped = false

// Inner loop to perform the actual sorting

for i = 1 to n-1 inclusive do

// Compare each pair of adjacent elements

if A[i-1] > A[i] then

// If the elements are in the wrong order, swap them

swap(A[i-1], A[i])

// Set the swapped flag to true since a swap occurred

swapped = true

end if

end for

// Decrement the counter for the next pass

n = n - 1

// Continue the loop until no more swaps are made

until not swapped

end

**Dry run algorithm**

Initial Array: [4, 2, 1, 3]

First Pass:

Comparing 4 and 2: Since 4 > 2, swap them. Array becomes [2, 4, 1, 3].

Comparing 4 and 1: Since 4 > 1, swap them. Array becomes [2, 1, 4, 3].

Comparing 4 and 3: Since 4 > 3, swap them. Array becomes [2, 1, 3, 4].

Second Pass:

Comparing 2 and 1: Since 2 > 1, swap them. Array becomes [1, 2, 3, 4].

Comparing 2 and 3: No swap needed. Array remains [1, 2, 3, 4].

Third Pass:

Comparing 1 and 2: No swap needed. Array remains [1, 2, 3, 4].

Comparing 2 and 3: No swap needed. Array remains [1, 2, 3, 4].

**Assignment 2: Recursive  
Function and Efficiency Analysis - Write a recursive function pseudocode and  
calculate the nth Fibonacci number and use Big O notation to analyze its  
efficiency. Compare this with an iterative approach and discuss the pros and  
cons in terms of space and time complexity.**

**Pseudo code**

**Recursive function**

fibonacci(n) returns integer

if n <= 1

return n

else

//here we are making recursive call

return fibonacci(n - 1) + fibonacci(n - 2)

in the above pseudo code we can see each recursive call is making 2 other recursive calls which makes the time complexity exponentially larger

**Iterative function**

function fibonacci(n) returns integer

if n <= 1

return n

else

fib = [0, 1]

for i from 2 to n // Loop through upto n

fib[i] = fib[i - 1] + fib[i - 2]

return fib[n]

Here the time complexity is simply O(n) as the loop goes from 2 to n

Recursive approach:

Pros: Simple and intuitive implementation

.Cons: Exponential time complexity leads to poor performance for large values of n.

Iterative approach:

Pros: Efficient time complexity of O(n). Lower space complexity due to constant extra space requirement.

Cons: More complex implementation compared to the recursive approach.