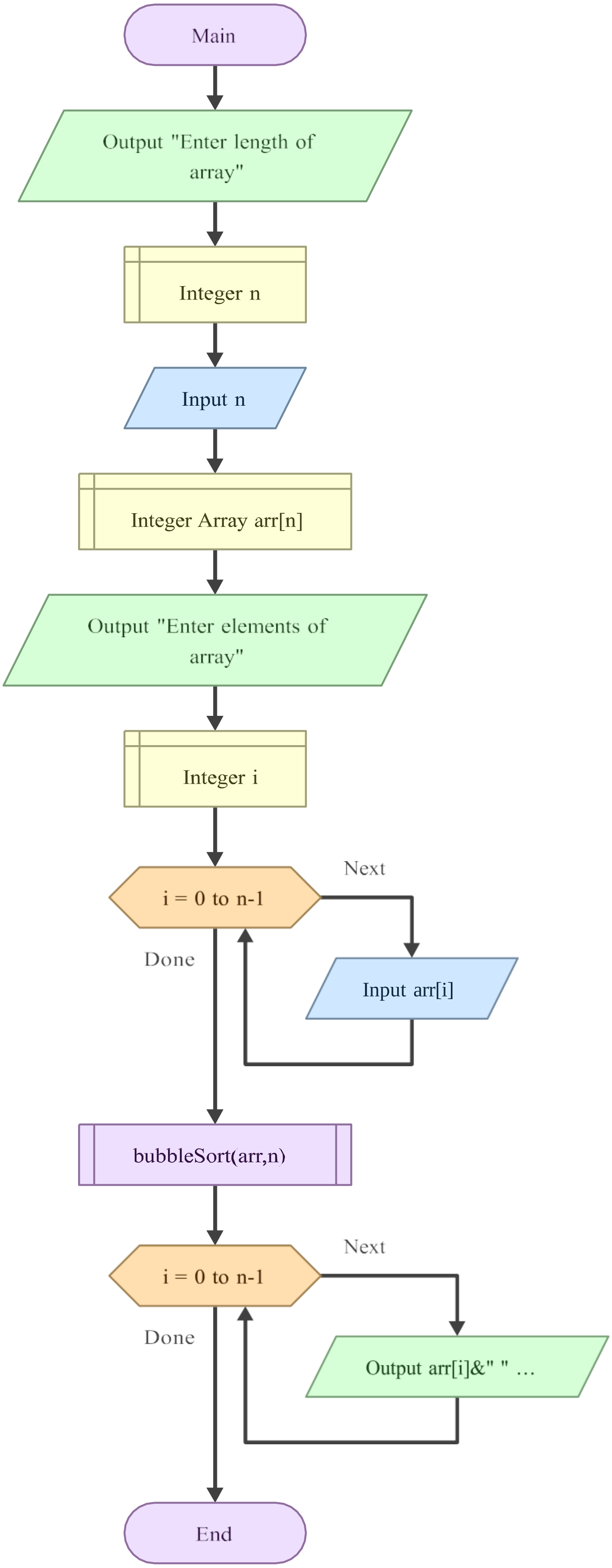
Assignment 1: Pseudocode and Flowchart for Sorting Algorithm - Write pseudocode and create a flowchart for a bubble sort algorithm. Provide a brief explanation of how the algorithm works and a simple array of integers to demonstrate a dry run of your algorithm.

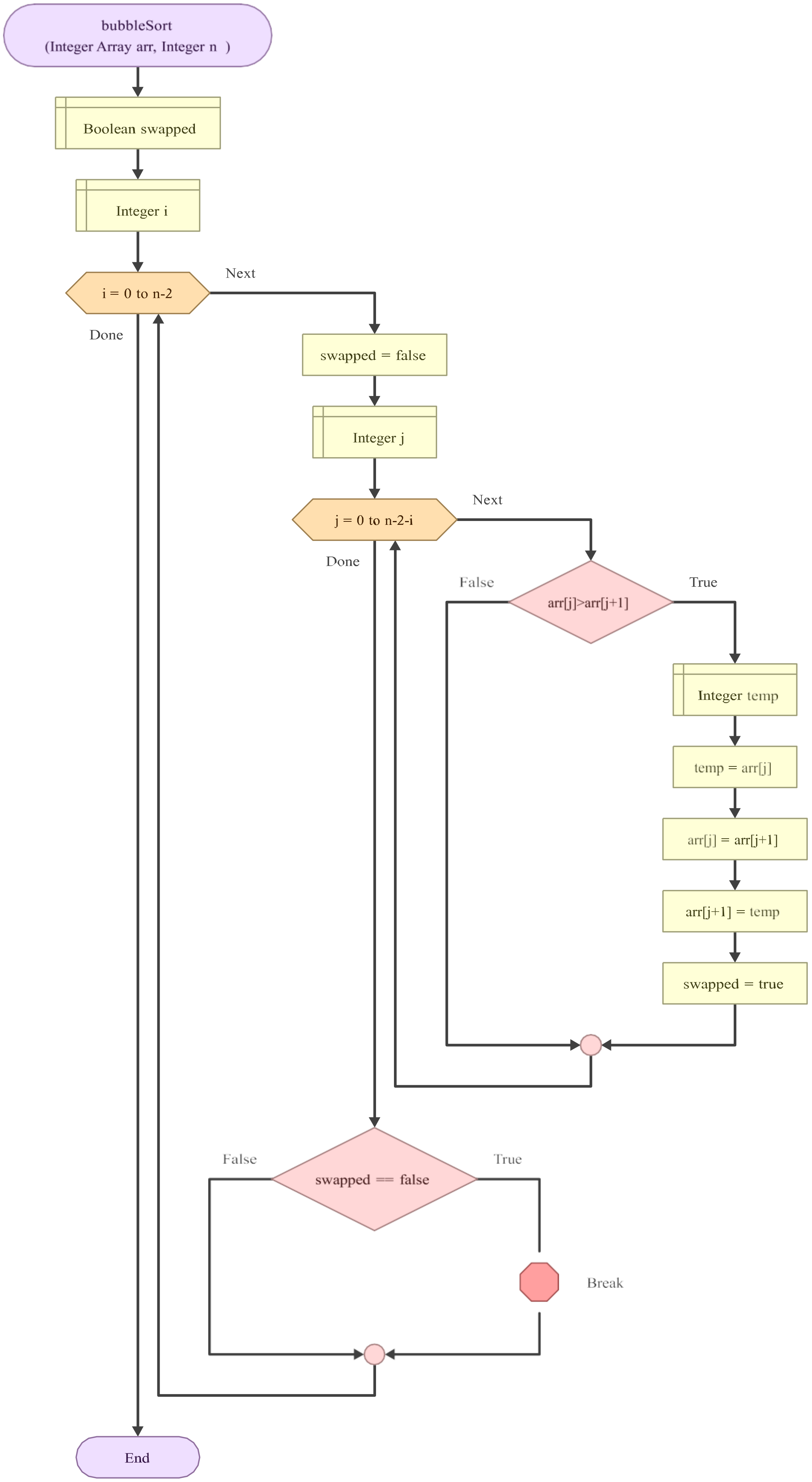
Solution:

PseudoCode:

```
BEGIN
FUNCTION bubbleSort(array)
   n = length of array
   FOR i FROM 0 TO n - 2
      swapped = false
      FOR j FROM 0 TO n - 2 - i
         IF array[j] > array[j + 1] THEN
            // Swap elements
            temp = array[j]
            array[j] = array[j + 1]
            array[j + 1] = temp
            swapped = true
         ENDIF
      ENDFOR
      IF swapped = false THEN
         BREAK   // Array is already sorted
      ENDIF
   ENDFOR
END FUNCTION
MAIN PROGRAM
   DISPLAY "Enter length of array"
   INPUT n
   DECLARE array of size n
   DISPLAY "Enter elements of array"
   FOR i FROM 0 TO n - 1
      INPUT array[i]
   ENDFOR
   CALL bubbleSort(array)
   FOR i FROM 0 TO n - 1
      PRINT array[i]
   ENDFOR
   END
```

FlowChart:


Main
Output "Enter length of array"
Integer n
Input n
Integer Array arr[n]
Output "Enter elements of array"
Integer i
i = 0 to n-1
Next
Done
Input arr[i]
bubbleSort(arr,n)
i = 0 to n-1
Next
Done
Output arr[i]&" " …
End


bubbleSort
(Integer Array arr, Integer n)
Boolean swapped
Integer i
i = 0 to n-2
Next
Done
swapped = false
Integer j
j = 0 to n-2-i
Next
Done
arr[j]>arr[j+1]
False
True
Integer temp
temp = arr[j]
arr[j] = arr[j+1]
arr[j+1] = temp
swapped = true
swapped == false
False
True
Break
End

Explanation:

1. Compare Adjacent Elements:
   - Start from the first element and compare it with the next element.
   - If the first element is greater than the second element, swap them.
2. Repeat the Process:
   - Continue this process for each pair of adjacent elements in the array.
   - After the first pass, the largest element will be at the end of the array.
   - Repeat the process for the remaining elements until the entire array is sorted.

Dry Run:

int arr[]=	**5**	**1**	**4**	**2**	**8**
index	**0**	**1**	**2**	**3**	**4**

Pass1:

i=0,j=0

arr[j](arr[0])=5>arr[j+1](arr[1])=1 (true)

swap(arr[j],arr[j+1]), after swapping

arr[j](arr[0])=1

arr[j+1](arr[1])=5

int arr[]=	1	5	4	2	8
index	0	1	2	3	4

j=1

arr[j](arr[1])=5>arr[j+1](arr[2])=4 (true)

swap(arr[j],arr[j+1]), after swapping

arr[j](arr[1])=4

arr[j+1](arr[2])=5

int arr[]=	1	4	5	2	8
index	0	1	2	3	4

j=2

arr[j](arr[2])=5>arr[j+1](arr[3])=2 (true)

swap(arr[j],arr[j+1]), after swapping

arr[j](arr[2])=2

arr[j+1](arr[3])=5

int arr[]=	1	4	2	5	8
index	0	1	2	3	4

j=3

arr[j](arr[3])=5>arr[j+1](arr[4])=8 (false)

no swapping

int arr[]=	1	4	2	5	8
index	0	1	2	3	4

Pass2:

i=1,j=0

arr[j](arr[0])=1>arr[j+1](arr[1])=4 (false)

no swapping

int arr[]=	1	4	2	5	8
index	0	1	2	3	4

j=1

arr[j](arr[1])=4>arr[j+1](arr[2])=2 (true)

swap(arr[j],arr[j+1]), after swapping

arr[j](arr[1])=2

arr[j+1](arr[2])=4

int arr[]=	1	2	4	5	8
index	0	1	2	3	4

j=2

arr[j](arr[2])=4>arr[j+1](arr[3])=5 (false)

no swapping

int arr[]=	1	2	4	5	8
index	0	1	2	3	4

Pass3:

i=2,j=0

arr[j](arr[0])=1>arr[j+1](arr[1])=2 (false)

no swapping

int arr[]=	1	2	4	5	8
index	0	1	2	3	4

j=1

arr[j](arr[1])=2>arr[j+1](arr[2])=4 (false)

no swapping

int arr[]=	1	2	4	5	8
index	0	1	2	3	4

After Sorting

int arr[]=	**1**	**2**	**4**	**5**	**8**
index	0	1	2	3	4