Assignment - 08 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.

The Bubble Sort Algorithm is a straightforward technique suitable for small datasets, where nearby elements are repeatedly swapped until the list is sorted. However, its performance degrades significantly for larger datasets, making it unsuitable for such scenarios. The basic idea involves iterating through the list and swapping adjacent elements if they are out of order until no more swaps are needed or the list is fully sorted.

In essence, the algorithm starts with the first element and, depending on whether it's sorting in ascending or descending order, swaps it with the next element if necessary. This process continues until the end of the list is reached, constituting a pass. Multiple passes are made until either all elements are sorted or no more swaps are required.

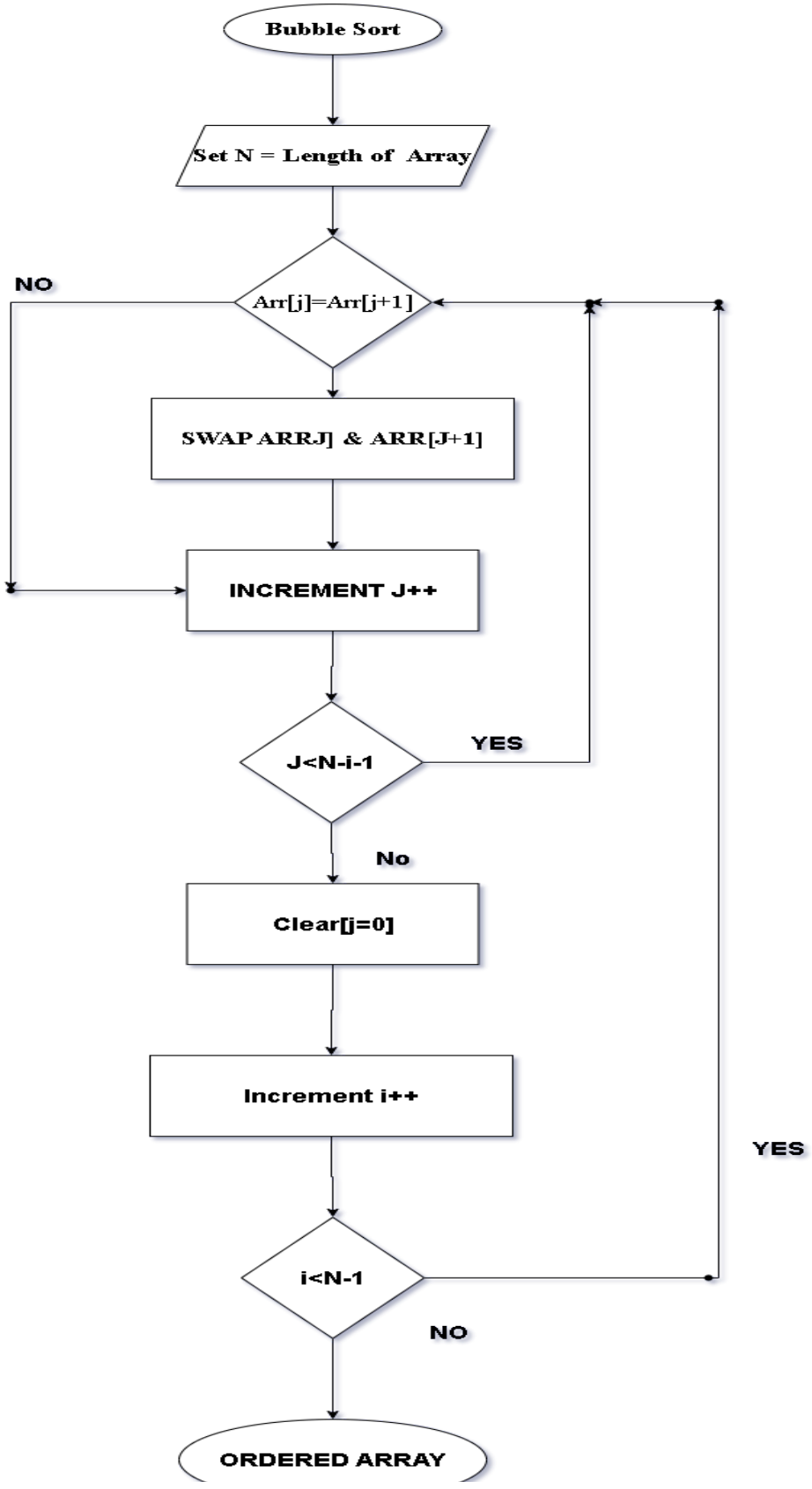
Despite its simplicity, Bubble Sort's time complexity can be high, especially in the worst-case scenarios. Hence, it's generally not recommended for sorting large volumes of data where more efficient algorithms like QuickSort or MergeSort would be more appropriate.

Bubble Sort Algorithm Pseudo Code:

Consider the below-given pseudo-code for implementing a bubble sort:

```
Bubble Sort(a[],n)
For i=0 to n-1
Swap=false
For j=i+1 to n
    if a[j-1] >a[j]
        Swap(a[j-1],a[j])
        Swap=true
    Break if not swapped
```

Bubble Sort Algorithm Flow chart: To help you understand better you can look at the flowchart for the bubble sort given below:



Dry Run Demonstration: -

Consider a simple array of integers: [5, 3, 8, 4, 2]

First Pass:
Compare 5 and 3, swap since 5 > 3: [3, 5, 8, 4, 2]
Compare 5 and 8, no swap since 5 < 8: [3, 5, 8, 4, 2]
Compare 8 and 4, swap since 8 > 4: [3, 5, 4, 8, 2]
Compare 8 and 2, swap since 8 > 2: [3, 5, 4, 2, 8]

Second Pass:
Compare 3 and 5, no swap since 3 < 5: [3, 5, 4, 2, 8]
Compare 5 and 4, swap since 5 > 4: [3, 4, 5, 2, 8]
Compare 5 and 2, swap since 5 > 2: [3, 4, 2, 5, 8]
Compare 5 and 8, no swap since 5 < 8: [3, 4, 2, 5, 8]

Third Pass:
Compare 3 and 4, no swap since 3 < 4: [3, 4, 2, 5, 8]
Compare 4 and 2, swap since 4 > 2: [3, 2, 4, 5, 8]
Compare 4 and 5, no swap since 4 < 5: [3, 2, 4, 5, 8]

Fourth Pass:
Compare 3 and 2, swap since 3 > 2: [2, 3, 4, 5, 8]
At this point, the list is sorted, but the algorithm does not know it yet and will go through the entire list one more time without any swap, confirming that the list is sorted.