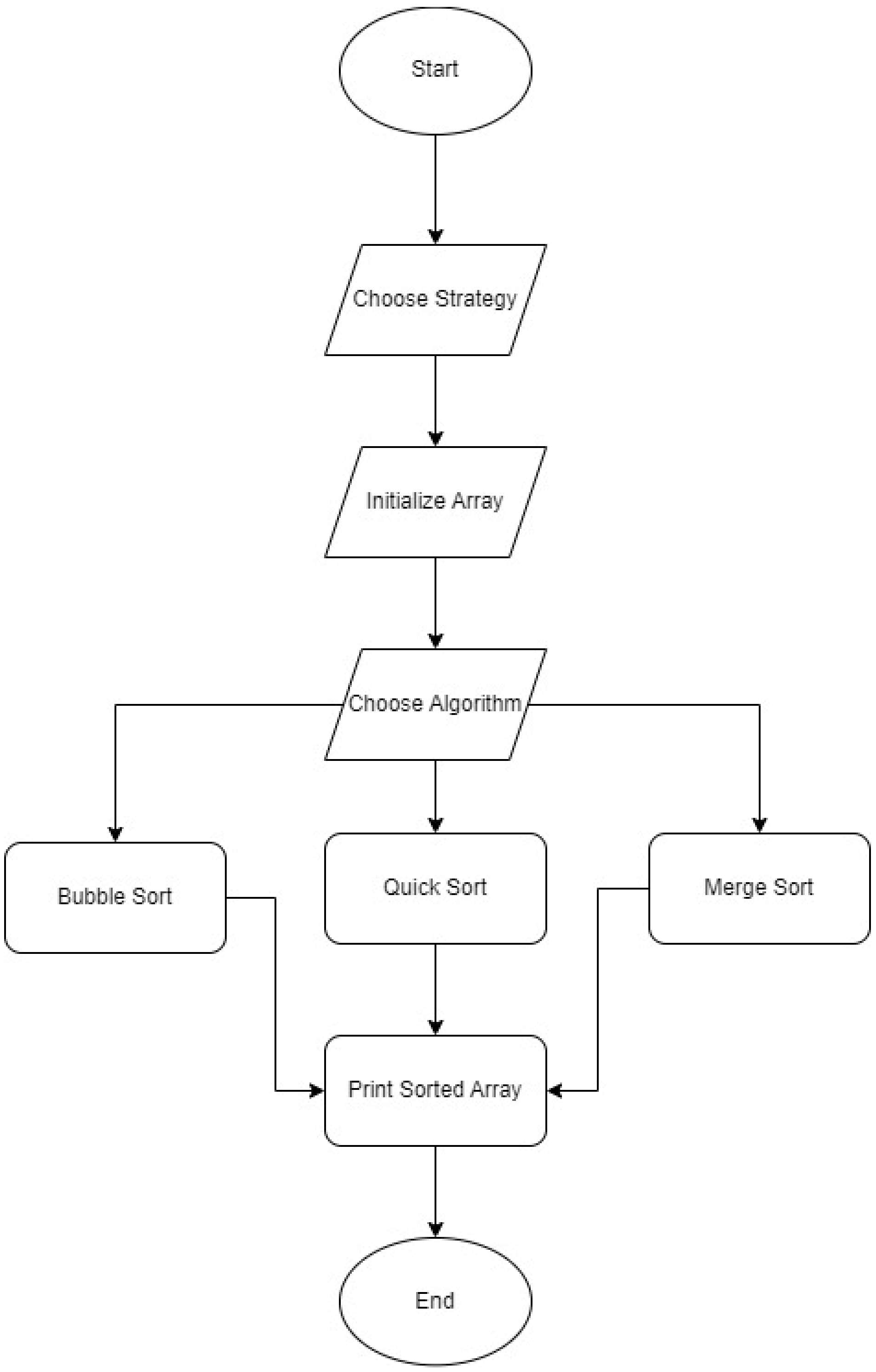
Strategy Design Pattern for Sorting Algorithms

System Components



Interfaces and Strategies

`ISortStrategy` Interface

This interface defines the contract for sorting strategies.

```
public interface ISortStrategy
{
    void Sort(int[] array);
}
```

Concrete Sorting Strategies

1. `BubbleSort`

- Implements `ISortStrategy`.
- Sorts an array using the Bubble Sort algorithm.

2. `QuickSort`

- Implements `ISortStrategy`.
- Sorts an array using the Quick Sort algorithm.

3. `MergeSort`

- Implements `ISortStrategy`.
- Sorts an array using the Merge Sort algorithm.

Context Class

`SortContext` Class

This class acts as the context that uses a strategy and can switch between different sorting strategies at runtime.

Main Program

`Program` Class

This class contains the `Main` method where instances of sorting strategies are created, and sorting is performed using the `SortContext` class.

Algorithms Used

1. Bubble Sort

Description: Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

Time Complexity:

- Best Case: O(n) (when the array is already sorted)
- Average Case: O(n^2)
- Worst Case: O(n^2)

Space Complexity:

- O(1) (in-place sorting)

2. Quick Sort

Description: Quick Sort is a divide-and-conquer algorithm that works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively.

Time Complexity:

- Best Case: O(n log n)
- Average Case: O(n log n)
- Worst Case: O(n^2) (rare, with poor pivot choices)

Space Complexity:

- O(log n) (in-place sorting)

3. Merge Sort

Description: Merge Sort is also a divide-and-conquer algorithm that divides the input array into two halves, recursively sorts each half, and then merges the sorted halves. The key step is the merging process, which combines two sorted arrays into a single sorted array.

Time Complexity:

- Best Case: O(n log n)
- Average Case: O(n log n)
- Worst Case: O(n log n)

Space Complexity:

- O(n) (requires additional space for merging)