**Software Design Document (SDD)**  Tag Sort Algorithm

**1. Introduction**

The purpose of this document is to provide a detailed design for the **Tag Sort Algorithm** project. This includes the system architecture, data structures, flow of control, and detailed descriptions of the core components.

**2. System Overview**

The system is designed to sort a given array of elements using the **Tag Sort Algorithm**, where each element is associated with a tag or index. Tags are sorted based on the values of the corresponding elements, and the array is then rearranged based on the sorted tags.

**3. Design Considerations**

**3.1 Assumptions**

* Input is a finite array of integers or floating-point numbers.
* Input array has unique elements (for simplicity, duplicate elements can be handled with a slight modification in tag sorting).

**3.2 Constraints**

* The algorithm must be implemented using an auxiliary tag array.
* The solution must minimize memory usage and optimize sorting time.

**3.3 Dependencies**

* Standard libraries for input/output and array handling (e.g., stdio.h, stdlib.h in C or equivalent libraries in other languages).

**4. Architecture Design**

**4.1 High-Level Architecture**

The system consists of the following main components:

1. **Input Module:** Takes input for the unsorted array.
2. **Tag Initialization Module:** Creates an array of tags corresponding to the indices of the elements.
3. **Tag Sorting Module:** Sorts the tags based on the values in the original array.
4. **Rearrangement Module:** Rearranges the original array based on the sorted tags.
5. **Output Module:** Displays the sorted array as output.

**4.2 Flow of Control**

The control flow of the system follows these steps:

* Initialize the array and tags.
* Sort the tags by comparing the values in the original array.
* Rearrange the original array using the sorted tags.
* Output the sorted array.

**5. Detailed Design**

**5.1 Modules**

**5.1.1 Input Module**

* **Functionality:** Read the unsorted array from the user or from a file.
* **Data:** The original array A[] and size n.

**5.1.2 Tag Initialization Module**

* **Functionality:** Create an array T[] of size n, where each tag corresponds to an index of the original array.
* **Data:** An integer array T[] initialized as T[i] = i for i = 0 to n-1.

**5.1.3 Tag Sorting Module**

* **Functionality:** Sort the tag array T[] based on the values of the original array A[]. Sorting can be performed using algorithms like QuickSort or MergeSort for efficiency.
* **Algorithm:**

for each pair (i,j) in T[]:

if A[T[i]] > A[T[j]]:

Swap T[i] and T[j]

**5.1.4 Rearrangement Module**

* **Functionality:** Create a new array B[] and copy elements from A[] into B[] using the sorted tags in T[]. Copy B[] back to A[].
* **Algorithm:**

for i = 0 to n-1:

B[i] = A[T[i]]

Copy B[] back to A[]

**5.1.5 Output Module**

* **Functionality:** Output the sorted array to the user.
* **Data:** The sorted array A[].

**6. Data Structures**

* **Array A[]:** The original array of elements to be sorted.
* **Array T[]:** An auxiliary array of tags that stores the indices of the elements in A[].
* **Array B[]:** A temporary array used to rearrange the elements of A[].

**7. Algorithmic Complexity**

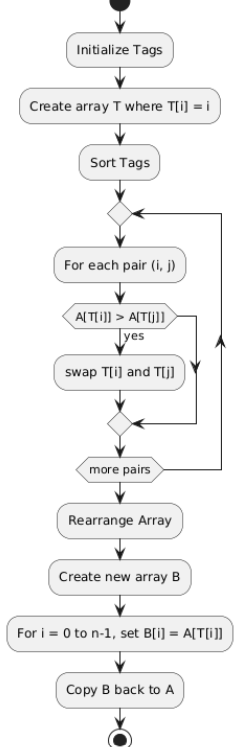
**7.1 Time Complexity**

* **Best Case:** O(n log n)
* **Average Case:** O(n log n)
* **Worst Case:** O(n²) (if a less optimal sorting algorithm like Bubble Sort is used for tag sorting).

**7.2 Space Complexity**

* O(n), due to the use of auxiliary arrays T[] and B[].

**8. Flowchart**

****

**9. Test Plan**

**9.1 Test Cases**

1. **Test Case 1:** Input a random unsorted array and verify if the output is sorted.
   * Input: A[] = {40, 10, 20, 30}
   * Expected Output: A[] = {10, 20, 30, 40}
2. **Test Case 2:** Input a reverse-sorted array and check performance.
   * Input: A[] = {50, 40, 30, 20, 10}
   * Expected Output: A[] = {10, 20, 30, 40, 50}
3. **Test Case 3:** Input an already sorted array to evaluate best-case performance.
   * Input: A[] = {10, 20, 30, 40, 50}
   * Expected Output: A[] = {10, 20, 30, 40, 50}

**9.2 Testing Tools**

* Manual testing using sample inputs.
* Automated testing scripts for large datasets.

**10. Conclusion**

The design of the Tag Sort Algorithm ensures that the sorting process is carried out efficiently using auxiliary tags. The system is designed to handle varying input sizes and sorts the array while maintaining optimal time and space complexity where possible.