**Tag Sort Algorithm**

**Software Requirements Specification**

Date 19/09/2024

Team 10

Project Leader

Sneha J. Kumbhar

Project member1

Aaditi V. Digraje

Project member2

Shruti A. Ramteke

Project Guide

Dr. A. J. Umbarkar

Prepared for

ITPM-IT-AJU-WCE

2024-25

**Table of Contents**

1. Introduction 1.1 Purpose  
   1.2 Scope  
   1.3 Definitions, Acronyms, and Abbreviations  
   1.4 References  
   1.5 Overview
2. General Description  
   2.1 Product Perspective  
   2.2 Product Functions  
   2.3 User Characteristics  
   2.4 General Constraints  
   2.5 Assumptions and Dependencies
3. Specific Requirements  
   3.1 External Interface Requirements  
   3.1.1 User Interfaces  
   3.1.2 Hardware Interfaces  
   3.1.3 Software Interfaces  
   3.1.4 Communications Interfaces  
   3.2 Functional Requirements  
   3.2.1 Sorting Logic  
   3.2.2 Input and Output Handling  
   3.3 Use Cases  
   3.4 Classes / Objects  
   3.5 Non-Functional Requirements  
   3.5.1 Performance  
   3.5.2 Reliability  
   3.5.3 Maintainability  
   3.6 Inverse Requirements  
   3.7 Design Constraints  
   3.8 Logical Database Requirements  
   3.9 Other Requirements
4. Analysis Models  
   4.1 Sequence Diagrams  
   4.2 Data Flow Diagrams (DFD)  
   4.3 State-Transition Diagrams (STD)
5. Change Management Process
6. Appendices

**1. Introduction**

**1.1 Purpose**

The purpose of this project is to learn a sorting algorithm, called the Tag Sort Algorithm, which associates each element in an array with a tag (or index) and sorts the array based on the values of its elements. The algorithm aims to provide an efficient solution applicable to any comparable data type, specifically for academic analysis of sorting techniques.

**1.2 Scope**

The Tag Sort Algorithm will be implemented in a high-level programming language, Python. The scope includes designing, implementing, and testing the algorithm across various datasets to validate its correctness and performance. This project is primarily academic and is not intended for production deployment.

**1.3 Definitions, Acronyms, and Abbreviations**

* **Tag Sort Algorithm**: A sorting technique that associates each array element with a tag and sorts the array while maintaining the relationship between the elements and their tags.
* **O(n log n)**: Time complexity of efficient sorting algorithms such as Merge Sort and Quick Sort.

**1.4 References**

* IEEE Guide to Software Requirements Specification (ANSI/IEEE Std. 830-1984).

**1.5 Overview**

This document details the functional and non-functional requirements of the Tag Sort Algorithm project. It provides the software design, performance criteria, and validation methods necessary to complete the project

**2. General Description**

**2.1 Product Perspective**

The Tag Sort Algorithm is a standalone academic project, developed to enhance the understanding of sorting algorithms and their time complexities. It is intended to be run as a Python script that accepts an input list and outputs a sorted list while retaining corresponding tags.

**2.2 Product Functions**

* **Sorting Function**: The algorithm will sort an array of elements.
* **Tag Association**: Each element in the array will be associated with a tag, and the algorithm will maintain this relationship throughout the sorting process.
* **Array Reordering**: After sorting, the original array will be reordered based on the sorted values.

**2.3 User Characteristics**

Users of this project include students, instructors, and anyone interested in studying sorting algorithms. Basic knowledge of programming and sorting techniques is required.

**2.4 General Constraints**

* The algorithm will have a time complexity of **O(n log n)**.
* Space complexity should remain within **O(n)**.
* The project is limited to comparable numerical data types.

**2.5 Assumptions and Dependencies**

* The algorithm will be written in Python.
* The datasets used for testing will be of small to medium size.
* No external libraries will be required for core functionality.

**3. Specific Requirements**

**3.1 External Interface Requirements**

**3.1.1 User Interfaces**

The algorithm will be executed through a command-line interface where the user can input an array and receive a sorted output. No graphical interface will be developed.

**3.1.2 Hardware Interfaces**

There are no specific hardware interfaces required, as the algorithm will run on any system capable of executing Python.

**3.1.3 Software Interfaces**

* **Development Environment**: Python 3.x
* **Testing Environment**: PyTest or similar unit testing frameworks for Python.

**3.1.4 Communications Interfaces**

No specific communication protocols are required.

**3.2 Functional Requirements**

**3.2.1 Sorting Logic**

* The algorithm will implement a sorting technique with a time complexity of **O(n log n)**. It may employ a modified version of Merge Sort or Quick Sort to include tag management.
* During execution, the system will generate a list of tags that will be sorted along with the array values, ensuring that the original relationships are maintained.

**3.2.2 Input and Output Handling**

* **Input**: The system will accept an array of elements, each associated with a tag.
* **Output**: The system will output the sorted array and the corresponding reordered tags.

**3.3 Use Cases**

* **Use Case 1**: The user provides an unsorted array, and the algorithm outputs the sorted array and rearranged tags.

**3.4 Classes / Objects**

* **Class 1**: Sorter
  + **Attributes**: Array of elements, corresponding tags.
  + **Functions**: sort(), reorder()

**3.5 Non-Functional Requirements**

**3.5.1 Performance**

* The system will maintain **O(n log n)** time complexity for sorting.
* Space complexity will remain within **O(n)**.

**3.5.2 Reliability**

* The algorithm should handle invalid inputs by notifying users of errors.

**3.5.3 Maintainability**

* The algorithm should be modular and well-documented, enabling future updates or improvements.

**3.6 Inverse Requirements**

* The system should not handle non-comparable data types, such as strings or objects, in its core sorting functionality.

**3.7 Design Constraints**

* The sorting algorithm must adhere to Python’s built-in data structures and avoid reliance on external libraries.

**3.8 Logical Database Requirements**

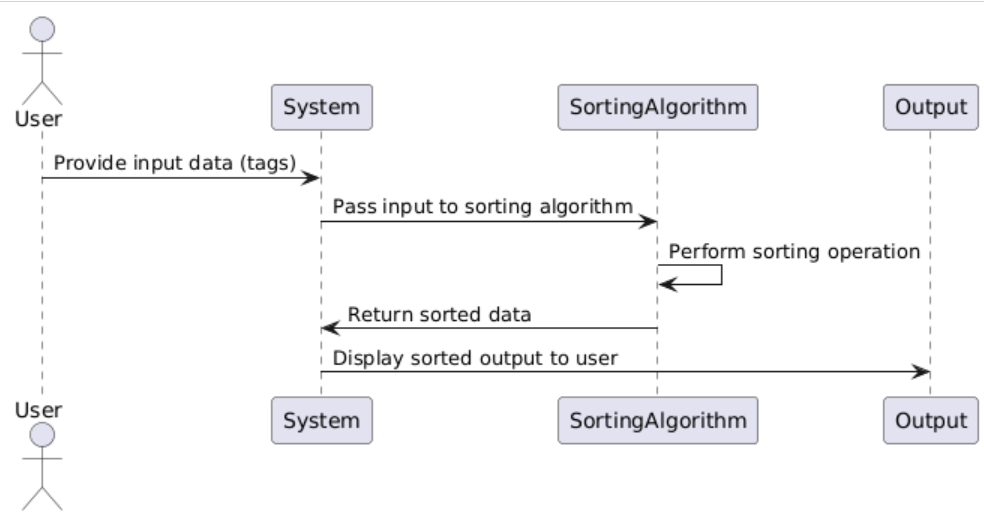
* No database is required for this project.

**3.9 Other Requirements**

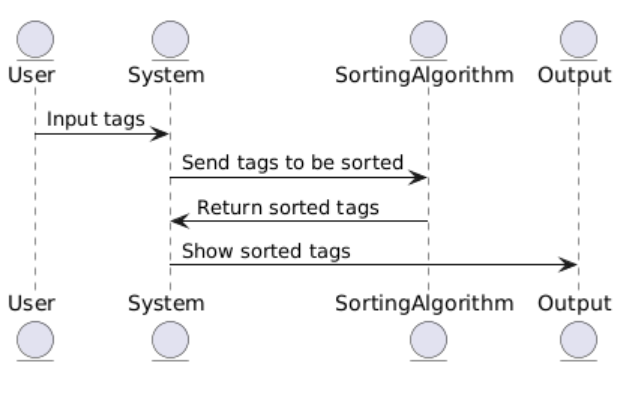
* The solution must be implemented using standard Python libraries.

**4. Analysis Models**

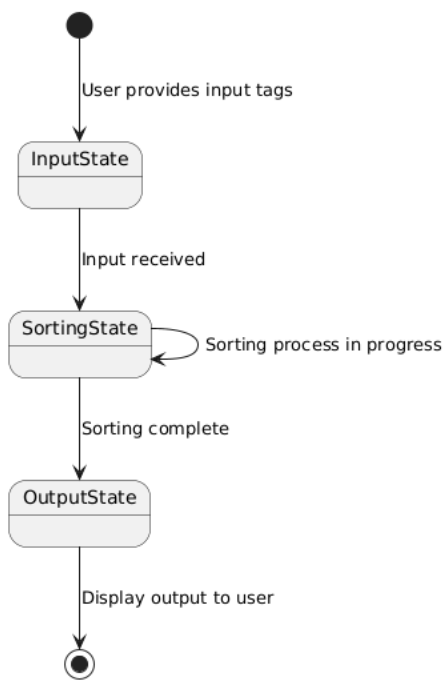
**4.1 Sequence Diagrams**



**4.2 Data Flow Diagrams (DFD)**



**4.3 State-Transition Diagrams (STD)**

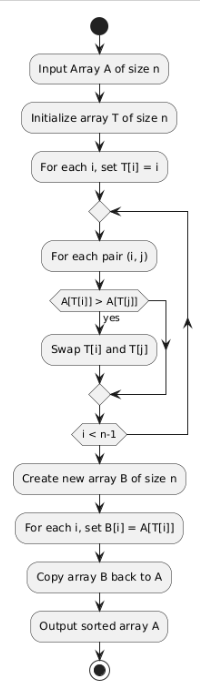


**5. Change Management Process**

Changes to this SRS will be managed through a formal review process, which will require approval from both the project leader and the instructor. Change requests can be submitted via email or the project management tool.

**6. Appendices**

**Appendix1 :Activity Diagram**

****

**Appendix 2: Pseudocode for the Tag Sort Algorithm.**

Input: An array A of size n.

Output: The sorted array A.

1. Initialize array T of size n:

For each i from 0 to n-1, set T[i] = i.

2. Sort the tags in array T:

For each i from 0 to n-1:

For each j from i+1 to n-1:

If A[T[i]] > A[T[j]], then:

Swap T[i] and T[j].

3. Create a new array B of size n:

For each i from 0 to n-1:

Set B[i] = A[T[i]].

4. Copy array B back to A:

For each i from 0 to n-1:

Set A[i] = B[i].

5. End: Array A is now sorted.