Priority Dispatch Selection Using Sorting Algorithms

1 Problem Description

Modern applications often maintain a dynamic queue in which items are ordered according to multiple priority rules. Common examples include music playback queues, ride-sharing dispatch systems, and task schedulers. In such systems, items may be inserted at arbitrary times, but the order in which they are processed depends on a combination of attributes rather than simple first-in-first-out behavior.

In this project, the goal is to determine which request should be served next by sorting a collection of requests using well-defined prioritization criteria. Each request is associated with several attributes that influence its position in the queue, such as priority level, urgency, and time of arrival. The ordering must respect these attributes in a fixed hierarchy and preserve the original input order when all priority values are equal.

The project emphasizes the use of comparison-based sorting algorithms to implement this behavior. By applying different sorting techniques to the same input, students can observe how algorithmic properties such as stability affect the correctness and consistency of the resulting order. This problem models real-world queueing behavior while providing a concrete setting for analyzing and comparing sorting algorithms.

2 Data Model

Each request is represented by the following attributes:

- **id**: A unique string identifier for the request.
- **priority**: An integer value where 1 indicates high priority and 0 indicates normal priority.
- **urgency**: A non-negative integer; smaller values indicate higher urgency.
- **time**: A non-negative integer representing the time at which the request was created.

Requests are assumed to be provided in the order they were received.

3 Sorting Rules

Requests must be ordered according to the following criteria, in order of precedence:

1. Requests with higher priority are ordered before those with lower priority.
2. Requests with lower urgency values are ordered before those with higher urgency values.
3. Requests with earlier timestamps are ordered before those with later timestamps.
4. If all attributes above are equal, the original input order must be preserved.

4 Input Specification

The input consists of a single line with the following format:

```
<algorithm> | <request_1> ; <request_2> ; ...  ; <request_n>
```

The sorting algorithm is one of:

- `merge`
- `quick`
- `heap`

Each request is formatted as:

```
id,priority,urgency,time
```

Whitespace around delimiters may be ignored.

5 Output Specification

The output consists of a single line containing the **id** of the request that should be served next after sorting according to the rules described above.

6 Example

Input:

```
merge | R1,0,5,100 ; R2,1,7,101 ; R3,1,3,102
```

Output:

```
R3
```

7 Learning Objectives

Upon completing this project, students will:

- Implement sorting algorithms with custom comparison functions.
- Understand the importance of sorting stability.
- Apply multi-key ordering rules used in real-world systems.
- Compare algorithmic behavior across different sorting techniques.