Project Description: Autocomplete System

Overview:

The goal of this project is to implement an **Autocomplete System** that can efficiently find and display terms based on a given prefix. The system is designed to read a list of terms (with associated weights) from a file, and then, based on user input (a prefix), provide a list of matching terms sorted by their weight in descending order. The project also provides the functionality to count how many terms match the prefix.

The program involves creating a few classes to model the core components of the autocomplete system, including:

- **Term**: This class represents a term with a string query and an associated weight.
- **BinarySearchDeluxe**: A utility class that implements binary search to efficiently find the first and last matching terms based on the prefix.
- **Autocomplete**: This class manages the autocomplete system, handling the process of sorting terms, finding matches, and counting the number of matches.
- **Main Program**: The main program reads input from a file and allows users to query the system to find matching terms.

Components:

1. **Term Class**:

    - Represents a single term consisting of a query (string) and a weight (integer).
    - Includes methods for comparing terms based on query strings and weight.
    - Implements two comparison methods:
        - **Lexicographic Comparison** (for sorting terms alphabetically).
        - **Weight Comparison** (for sorting terms based on their associated weights).

2. **BinarySearchDeluxe Class**:

    - Implements binary search algorithms to efficiently find the index of terms that match a given prefix.
    - Contains two main methods:
        - **first_index_of**: Finds the index of the first term that matches the prefix.
        - **last_index_of**: Finds the index of the last term that matches the prefix.

3. **Autocomplete Class**:

    - Manages a list of terms and provides functionality to find all terms matching a given prefix.

    - Includes two main methods:
        - **all_matches**: Finds all terms that match a given prefix and sorts them by weight in descending order.
        - **number_of_matches**: Returns the number of terms that match the given prefix.

4. **Main Program**:

    - Reads input data (terms and their weights) from a file.
    - Allows users to input a prefix and returns matching terms sorted by their weight.
    - Provides the total count of matching terms.

Functionality:

1. **Input File**:

    - The program reads from a file containing terms, where each term consists of a weight and a query string separated by a tab.
    - The file format is expected to be:

    ```
    5
    1000    apple
    1500    banana
    1200    apricot
    800     blueberry
    1300    avocado
    ```

    - The first line specifies the number of terms, followed by each term's weight and query string on separate lines.

2. **User Input**:

    - After loading the terms from the file, the program waits for the user to enter a prefix.
    - The system will then output:
        - The number of terms matching the given prefix.
        - A list of the matching terms, sorted by their weight in descending order. Only a fixed number of top results (specified by the user) are displayed.

3. **Prefix Matching**:

    - When the user enters a prefix (e.g., "ap"), the system will find all terms whose query starts with "ap".
    - It will then sort these terms by weight and display them along with the number of matches.

Example:

Given the input file with the terms listed above, if the user enters the prefix `"ap"` , the program will output something like:

```
3 matches
1500    banana
1300    avocado
1200    apricot
```

This shows the 3 matching terms (apple, apricot, and avocado) sorted by their weight.

Purpose and Use:

The autocomplete system is useful for applications that require fast and efficient text suggestions or searches based on a prefix. This can be applied to:

- Search engines
- Autocompletion in text editors or IDEs
- E-commerce websites for product suggestions
- Data filtering systems that display relevant suggestions based on user input

Key Features:

- **Efficiency**: The use of binary search ensures that term lookup and matching are done quickly.
- **Sorting by Weight**: Matches are sorted by weight, showing the most relevant results first.
- **Prefix Matching**: Supports finding terms that start with a given prefix.

Requirements:

- **Input Data File**: A file containing terms with weights and query strings.
- **User Input**: The program will interact with the user through standard input (for prefixes) and output (for matches and count).

Future Enhancements:

- Support for case-insensitive matching.
- Support for more complex matching criteria, such as fuzzy matching.
- Optimization for large datasets by implementing advanced data structures like tries or Ternary Search Tries.

This project demonstrates an efficient way to implement an autocomplete system using Python, leveraging sorting and binary search for fast retrieval and matching.