The given C++ code defines an `AutocompleteSystem` class that provides functionality for inputting and retrieving autocomplete suggestions based on user input. The system utilizes a trie data structure for efficient storage and retrieval of sentences. Here's an explanation of the key components and how the system works:

Trie Node (Node):

- **Description**: The `Node` class represents a node in a trie. A trie is a tree-like data structure used to store a dynamic set of strings where the keys are usually strings. Each node in the trie can have multiple children, and each child is associated with a character. The trie stores sentences character by character.
- **Properties**:
  - `children`: An unordered map from characters to pointers of `Node`. It represents the possible next characters for a given prefix.
  - `counter`: An unordered map from strings to integers, which holds the sentences and their corresponding frequencies (the number of times the sentence has been typed).

Autocomplete System (AutocompleteSystem):

- **Description**: The `AutocompleteSystem` class provides methods to input characters and retrieve a list of autocomplete suggestions. It utilizes a trie structure to store and retrieve sentences efficiently.
- **Constructor** (`AutocompleteSystem`):
  - Initializes a new trie (`root`).
  - Inserts each sentence from the input array into the trie along with its frequency.
- **Public Methods**:
  - `input`: Handles the input of a single character and returns autocomplete suggestions based on the current input.
- **Private Members**:
  - `root`: A pointer to the root node of the trie.
  - `curNode`: A pointer to the current node in the trie during traversal.
  - `prefix`: A string that holds the current input being typed by the user.

Method Details:

- **insert**: A private method used to insert a sentence and its frequency into the trie. It traverses the trie based on the characters in the sentence, creating new nodes as necessary. It updates the frequency of the sentence at the last node of each sentence.
- **input**: The primary method for user interaction. It takes a single character as input and performs different actions based on the character:
  - If the input is '#', it treats the current `prefix` as a complete sentence and inserts it into the trie (updating frequency). It then resets the `prefix` and `curNode` for new input.
  - For other characters, it appends the character to the `prefix` and attempts to move `curNode` to the corresponding child node.
  - If the next node exists, it retrieves and sorts all sentences passing through that node based on their frequency and lexicographical order using a priority queue. It then returns the top 3 sentences.
  - If the next node doesn't exist, it means there are no sentences with the current prefix, and it returns an empty list.

Key Components:

- **Trie**: The backbone data structure of the system. It allows for efficient retrieval of sentences based on prefixes.
- **Priority Queue (pq)**: Used to maintain and retrieve the top 3 hot sentences. It is sorted first by frequency (higher frequency sentences come first) and then by lexicographical order (ASCII value) in case of a tie.

Printable Summary:

The `AutocompleteSystem` is a C++ class designed to provide quick and efficient autocomplete suggestions based on previously typed sentences. It leverages a trie data structure to store the sentences character by character, allowing for fast retrieval of suggestions as the user types. The system supports updating sentence frequency and retrieving the top 3 suggestions based on the current input. The trie nodes are dynamically created and linked, with each node capable of holding multiple sentences and their respective frequencies. The use of a priority queue ensures that suggestions are always ordered by their "hotness" and lexicographical order. As the user types, the system updates its current state and provides relevant, timely, and sorted autocomplete suggestions.