**Linked List Rearrangement Algorithms**

**Algorithm Overview**

1. Append if Miss

* The algorithm adds a new element to the end of the linked list only if it's not already present in the list.
* Traverse the linked list.
* Check if the given element is already present in the list.
* If the element is not present, append it to the end of the list.

2. Move to Front

* The algorithm moves an accessed element to the front of the list, assuming it will be accessed again soon.
* Traverse the linked list to find the target element.
* If the target element is found, remove it from its current position.
* Add the target element to the front of the list.

3. Frequency Count

* The algorithm rearranges the elements in the linked list based on their access frequency, with the most frequently accessed elements at the front.
* Maintain an auxiliary data structure (e.g., a HashMap) to store the frequency count of each element in the list.
* Whenever an element is accessed, increment its frequency count in the auxiliary data structure.
* If an element's frequency count changes, rearrange the list by moving the element to the appropriate position based on its frequency.
* Ensure that the list remains sorted in descending order of frequency.