1. How do you implement a stack in Python using a list, and how would you handle underflow and overflow conditions?

Answer:
A **stack** is a data structure that follows the **LIFO** (Last In, First Out) principle. You can implement a stack using Python's built-in list because lists have methods to add and remove elements from the end of the list, mimicking stack operations.

- **Push**: Add an element to the top of the stack.
- **Pop**: Remove and return the top element of the stack.
- **Peek**: View the top element without removing it.
- **IsEmpty**: Check if the stack is empty.
- **Overflow and Underflow**:
  In Python, you do not usually deal with overflow since the memory is dynamically managed. However, underflow (attempting to pop from an empty stack) needs to be handled.

2. How would you implement a queue using two stacks in Python?

Answer:
A **queue** is a data structure that follows the **FIFO** (First In, First Out) principle. Implementing a queue using two stacks is a common interview problem. The trick is to use one stack for enqueue operations and the other for dequeue operations.

- **Enqueue**: Push elements onto the first stack.
- **Dequeue**: Pop elements from the second stack. If the second stack is empty, transfer all elements from the first stack to the second stack.

Algorithm:

- **Enqueue (element)**:
  - Push the element into `stack1`.
- **Dequeue**:
  - If `stack2` is empty, pop all elements from `stack1` and push them onto `stack2`. Then, pop the top element from `stack2`.

Explanation:

- The **amortized time complexity** for both `enqueue` and `dequeue` operations is O(1) due to the transfer of elements between stacks happening only when `stack2` is empty.

3. How would you detect and remove a cycle in a singly linked list?

Answer:
To detect a cycle in a singly linked list, we can use **Floyd's Cycle Detection Algorithm** (Tortoise and Hare method). If a cycle exists, the fast pointer (hare) will eventually meet the slow pointer (tortoise).

Once a cycle is detected, the next step is to find the node where the cycle begins. After that, we can remove the cycle.

Algorithm:

1. **Cycle Detection**:
   - Initialize two pointers: `slow` (moves one step at a time) and `fast` (moves two steps at a time).
   - If `fast` equals `slow`, a cycle is detected.
2. **Cycle Removal**:
   - After detecting the cycle, place one pointer (`ptr1`) at the head of the list and another pointer (`ptr2`) at the meeting point.
   - Move both pointers one step at a time until they meet. This meeting point is the start of the cycle.
   - To remove the cycle, find the node just before the start of the cycle and set its next pointer to `None`.

Explanation:

- **Time Complexity**: O(n)
- **Space Complexity**: O(1)

4. How do you implement a Least Recently Used (LRU) cache in Python?

Answer:
The **LRU Cache** is a cache replacement algorithm that removes the least recently used item when the cache exceeds its capacity. You can implement an LRU Cache using a **doubly linked list** and a **hash map** (dictionary in Python).

Key Concepts:

- The hash map stores keys, and each key points to a node in the doubly linked list.
- The doubly linked list stores the order of usage, with the least recently used at the head and the most recently used at the tail.

Operations:

- **Get(key)**: Fetch the value if the key exists and move the node to the end of the list (most recently used).
- **Put(key, value)**: Insert a new key-value pair or update an existing one. If the cache exceeds capacity, remove the least recently used item.