Worksheet
Linked Lists

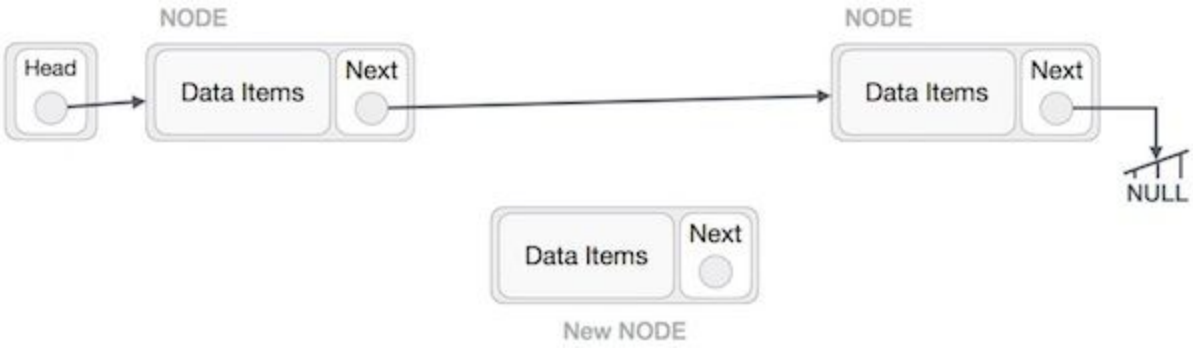
Summary

1. Subject: Data Structure
2. Topic: Insertion, Deletion & Reverse
3. Level: Easy

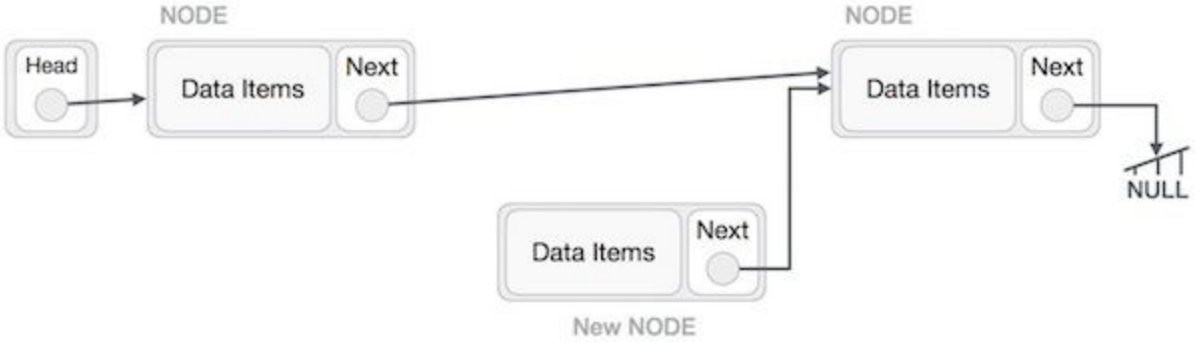
Insertion

Insertion is a 3 step operation:

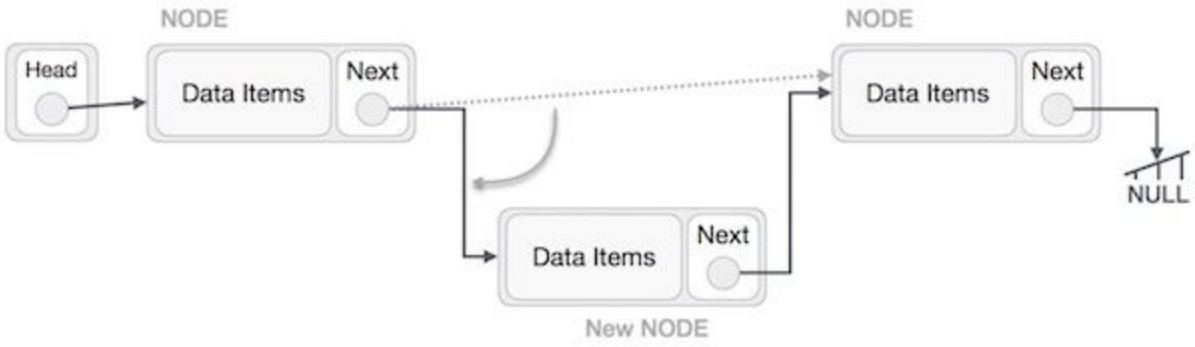
1) Get to the index of insertion $O(n)$



2) Set our `NewNode` to point to the `RightNode`, if there is no node to the right, then we point to `NULL` $O(1)$



3) The `LeftNode` should point to our `NewNode` $O(1)$



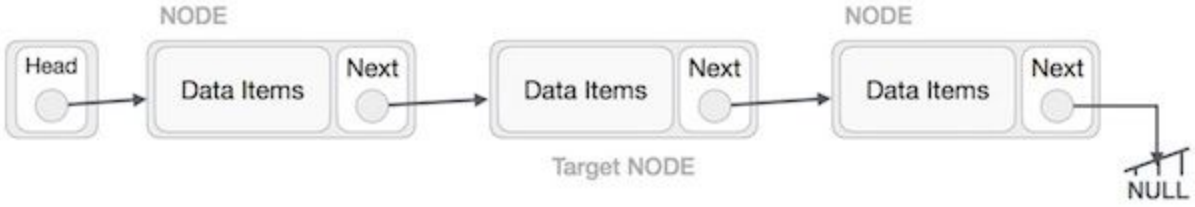
Based on your interviewer, they may or may not be strict about when insertion is $O(n)$ vs $O(1)$. Generally we say that linked list insertion is $O(1)$ because all we do is manipulate pointers and we can leave step 1 to be the job of a `search()` function external to `insert()`. Here are the cases when insertion is $O(n)$ vs $O(1)$

Insertion at the head	O(1)
Insertion in the middle of a list	O(n)
Insertion at the tail (given a tail ptr)	O(1)
Insertion at the tail (without a tail ptr)	O(n)

Deletion

Deletion is a 3 step operation:

1. Get to the index of deletion $O(n)$



2. The previous node `LeftNode` of the `TargetNode` should point to the next of `TargetNode`.

3. Set `TargetNode` to point to `NULL`

Based on your interviewer, they may or may not be strict about when deletion is $O(n)$ vs $O(1)$. Generally we say that linked list deletion is $O(1)$ because all we do is manipulate pointers and we can leave step 1 to be the job of a `search()` function external to `deletion()`. Here are the cases when deletion is $O(n)$ vs $O(1)$

Deletion at the head	O(1)
Deletion in the middle of a list	O(n)
Deletion at the tail	O(n)

Reverse

We are going to do a linked list reversal in place (O(1) space). We will need three pointers:

```
1. prev = NULL;
2. current = head;
3. next = NULL;
```

We iterate through the linked list. In our loop we will

1. Store the next node of `current` in `next`
2. Have `current` point to `prev`
3. Move `prev` and `current` forward

Let's illustrate these three steps in code:

```
next = current.next;

current.next = prev;

prev = current;

current = next;
```

Here is a nice GIF to visualize the code
https://media.geeksforgeeks.org/wp-content/cdn-uploads/RGIF2.gif

Time Complexity: O(n)

Space Complexity: O(1)