**Linked Lists**

Linked lists do not have indexes.

A list is in a contiguous place in memory, meaning that the items in the list are right next to each other in memory.

However in a linked list, all the nodes in the linked list are in different places in memory.

In a linked list we have a head and it points to the first/next node in that linked list.

A screenshot of a computer

Description automatically generated with low confidence

How a linked list looks like in memory

Diagram

Description automatically generated with medium confidence

As each node in the linked list points from one node to the next, you can easily find the nodes without losing any data.

**Big O of Linked List**

Here we discuss how to create linked lists and Big Os involved in each method

1. Append a node to the end of the linked list, change the tail position. This is O(1)
2. Removal of an end item: To move the tail to the one before removed item, we have to set the pointer of the tail equal to another node

A screenshot of a computer

Description automatically generated with low confidence

We also have to set the tail equal to the pointer equal to the pointer coming into the new last node.

A screenshot of a computer

Description automatically generated with low confidence

Hence, we have to start from the head and iterate through the linked list until we get to the pointer that is pointing to the last node. Then the pointer that is pointing to the last node will be equated to the tail pointer. As we had to iterate through the entire list, this is O(n).

1. Adding an item to the front of the linked list: Here we just need to change the head pointer to the new starting node, and then it is completed. Hence this is O(1)
2. Removing the head node: First we need to change the head pointer to the second node which is now the new head.

Head = head.next, and the remove the first item from the linked list

This too is O(1).

1. Adding an item in the middle: We have to iterate through the entire linked list to get to the position where we want to add it to. Then node.next will be the new node and the new nodes, node.next will be the one after that. This is also O(n)

A screenshot of a video game

Description automatically generated with medium confidence

1. Removal of the newly added 4 node: Again we have to iterate through the list, fix 23.next to the pointer going from node 4 to 7.
2. Look up with a linked list: We have to iterate till we find the required node. Also even if we had a index on each node, we still have to find the index by iterating. Hence it is O(n)

Table

Description automatically generated

**Linked Lists Under the Hood**

1. Appending an item:

What is the node that you are appending?

The node consists of both the value and the pointer.

It is essentially a dictionary where {“value”:4, “next”:None}

The node previously is also a dictionary like this

Diagram

Description automatically generated

We do this by making the dict4[“next”] = dict5 like below

Graphical user interface

Description automatically generated with medium confidence

Final Linked list structure:

Text

Description automatically generated

Tail will point to the last item