1. Linked List Intro

- Why linked list over array:
    - Dynamic allocation and memory wastage in array:
        - We know that array's length is fixed.
        - Say if we currently have 10 elements in list. But if we need to increase the size then it is not allowed in array, because array is an contiguous memory(e.g address: 1000-1040). Now if we want to extend the array, then it sees whether the 1041-1044 is free,but it may be occupied.so obviously extending the array is impossible.
        - Whereas linked list is not contiguous memory allocation. So if we need to extend the array, it is very simple by just updating the pointer of nodes.
        - Say if we declare a array with size 50, but currently we have only 10 elements in it. But the other 40 blocks are being allocated to the array is unusable. So it is a waste of memory.
        - Linked list does not face this problem because it's size is not fixed.
    - Easy insertion and removing:
        - Consider an array length with 10.
            - If we try to insert an element at 2nd index then we need to first create the space for the new element at index 2 by shifting all elements one step right from index 2 till the last index and then we will insert the new element at index 2
            - Similarly if we wan to delete an element at index 2, then after removing the element at index 2, we need to shift all the elements one step left from index 3 till n`th index.
            - **So the worst time complexity for insertion or removal of element in array is 0(n)**
        - Consider a linked list:
            - If we try to insert an element at 2nd position, we just create a new node containing the new value. And now we fetch the 1st node in our linked list and take a copy of the next_addr pointer(which currently points the old 2nd node). Now update the next_addr pointer to our newly created node. Now in the newly created node, we just assign the address of old 2nd node which is copied and stored previously.
            - Here we don't need to shift elements unlike in array.
            - **But the time complexity is 0(n). This is because if we want to insert at 9th position then we need to iterate from head node to 9th node, which takes 0(n) time**

    - **Array is better in searching elements than linked list:**
        -
        - Let size of array =10. Base address of array is 1000. Type of elements in array is int(4 bits each).

    - 1000–1004–1008–1012–1016–1020–1024–1028–1032–1036
- if we try to access 5th index. It just calculates the address of 5th index by this => base_address + (index*size_of_data_type)
    - I.e.. 1000 + (5*4) = 1020
- **So time complexity of searching in array using index is 0(1).**
- But whereas in linked list, we cannot have index for searching. Instead we need to iterate from the very first node to reach the 5th node by traversing each address.
- **So time complexity for searching in linked list is 0(n).**