Array vs Linked List - A comparison

- An array is a datatype which is pretty much ubiquitous as a default type in every language. Depending on their use, they are easy to use and fast. However, they are bounded by size and are costly if trying to resize.
- Linked Lists are not as ubiquitous, but most languages give a similar implementation on par with a vector in C++. Linked lists have no bounded size, and grow and shrink easily (with an overhead cost of managing pointers).
- Lets look more deeply:

Arrays

1. An array is a collection of elements of the *same data type*. (*Can arrays hold different types of data?*)
2. Arrays support `Random Access`, which means elements can be accessed directly using an index. Hence, accessing elements in an array is fast with a constant time complexity of `O(1)`.
3. In an array, elements are stored in `contiguous` memory locations (meaning the locations are sizeof(datatype) away from each other (smacked together with no gaps). For example if I have an `integer` array of size `10`, and the starting address is `1234`, then I know since ints are 4 bytes that the next address is `1238` and the next is `1242` and so on.
4. In an array, `Insertion` and `Deletion` operations take more time (assuming that we cannot allow for empty slots).
    - If a value is deleted, we must shift other items to close the empty slot, or
    - If we insert a value we must shift other items to make a hole for the item.
5. Arrays can use:
    - `Static Memory Allocation` (known at compile time), or
    - `Dynamic Memory Allocation` (during run time).
6. Arrays have direct access to any element just by knowing its index value.
7. Arrays can be:
    - Single Dimensional
    - Two Dimensional
    - Multi Dimensional
8. The size of an array must be specified at time of creation and resizing the array is costly since more memory needs to be allocated, and then have all the items copied over.
9. Statically declared arrays get allocated in the `Run Time Stack`, where dynamically declared arrays get allocated in the `Heap`.

Linked Lists

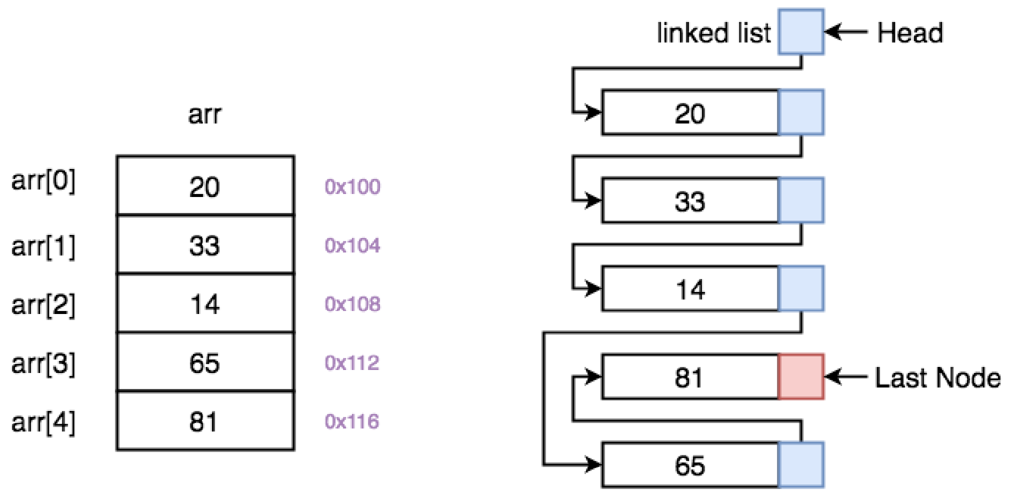
1. A Linked List is an collection of elements of same type, which are connected to each other using pointers. (*Can you link different types together?*)
2. Linked List supports `Sequential Access`, which means to access any element/node in a linked list, we have to sequentially traverse the list to find the element.
    - To access the `nth` element of a linked list, time complexity is `O(n)`.
3. In a linked list, new elements can be stored anywhere in the memory. The address of the memory location allocated to the new element is stored in the previous node of the linked list, hence forming a link between the two nodes/elements.

4. In case of linked list, `Insertion` and `Deletion` operations can be done in:
    - constant time or `O(number of pointers to update)`which is constant time, and is ignored.
    - Or if the item needs to be placed in an ordered position, then this could at worst case be `O(n)`
5. a new element is stored at the first free and available memory location, with only a single overhead step of storing the address of memory location in the previous node of linked list. **Insertion and Deletion** operations are fast in linked list.
6. Linked lists use **Dynamic Memory Allocation**.
    - *Is it even possible for lists to use Statically Allocated Memory?*
7. In case of a linked list, each node/element points to the next, previous, or maybe both nodes, meaning no direct access (only sequential).
8. Linked lists are:

- **1)** Singly
- **2)** Doubly
- **3)** Circularly Linked.
- *Could we create a 2D linked list??*

9. Size of a Linked list is variable. It grows and shrinks as more nodes are added or removed.
10. Linked lists are always allocated in the **Heap**

Looking at an image of an array and a linked list, it should be apparent why insertions and deletions (in the middle of the structure) are easy for a linked list, and costly for an array.



Array representation