**SheetofComplexity**

**https://www.bigocheatsheet.com/**

**NOTES OF DSA:**

**ARRAY VS LINKLIST :**

**Key Differences Between Array and Linked List**

1. An array is the data structure that contains a collection of similar type data elements whereas the Linked list is considered as non-primitive data structure contains a collection of unordered linked elements known as nodes.

2. In the array the elements belong to indexes, i.e., if you want to get into the fourth element you have to write the variable name with its index or location within the square bracket.

3. In a linked list though, you have to start from the head and work your way through until you get to the fourth element.

4. Accessing an element in an array is fast, while Linked list takes linear time, so it is quite a bit slower.

5. Operations like insertion and deletion in arrays consume a lot of time. On the other hand, the performance of these operations in Linked lists is fast.

6. Arrays are of fixed size. In contrast, Linked lists are dynamic and flexible and can expand and contract its size.

7. In an array, memory is assigned during compile time while in a Linked list it is allocated during execution or runtime.

9. Elements are stored consecutively in arrays whereas it is stored randomly in Linked lists.

10. The requirement of memory is less due to actual data being stored within the index in the array. As against, there is a need for more memory in Linked Lists due to storage of additional next and previous referencing elements.

11. In addition memory utilization is inefficient in the array. Conversely, memory utilization is efficient in the linked list.

(1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage, and in practical uses, the upper limit is rarely reached.

(2) Inserting a new element in an array of elements is expensive because a room has to be created for the new elements and to create room existing elements have to be shifted.

For example, suppose we maintain a sorted list of IDs in an array id[].

id[] = [1000, 1010, 1050, 2000, 2040, …..].

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

So Linked list provides the following two advantages over arrays

1) Dynamic size

2) Ease of insertion/deletion

Linked lists have following drawbacks:

1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do a binary search with linked lists.

2) Extra memory space for a pointer is required with each element of the list.

3) Arrays have better cache locality that can make a pretty big difference in performance.

Array supports **Random Access**, which means elements can be accessed directly using their index, like arr[0] for 1st element, arr[6] for 7th element etc.

Hence, accessing elements in an array is **fast** with a constant time complexity of O(1)

Linked List supports Sequential Access, which means to access any element/node in a linked list, we have to sequentially traverse the complete linked list, upto that element.

To access nth element of a linked list, time complexity is O(n).

Why insertion and deletion is faster in linklist ?

Array **Insertion**: big memory allocation/re-allocation, followed by big memory move. **List Insertion**: small memory allocation, followed by a couple **of** pointer changes. And that's why **insertion** into a **linked list** is most often **faster** than **insertion** into an array.

Where do we use linked lists?

**Linked lists** are often **used** because of their efficient insertion and deletion. They can be **used** to implement stacks, queues, and other abstract data types.

What is linked implementation?

A **linked** list is a linear data structure, in which the elements are not stored at contiguous memory locations. ... In simple words, a **linked** list consists of nodes where each node contains a data field and a reference(**link**) to the next node in the list.

**STACK**

What is stack and its application?

**Stack** is an abstract data type and a data structure that follows LIFO (last in first out) strategy. ... It means the element added last will be removed first. **Stack** allows two operations push and pop. Push adds an element at the top of the **stack** and pop removes an element from top of the **stack**.