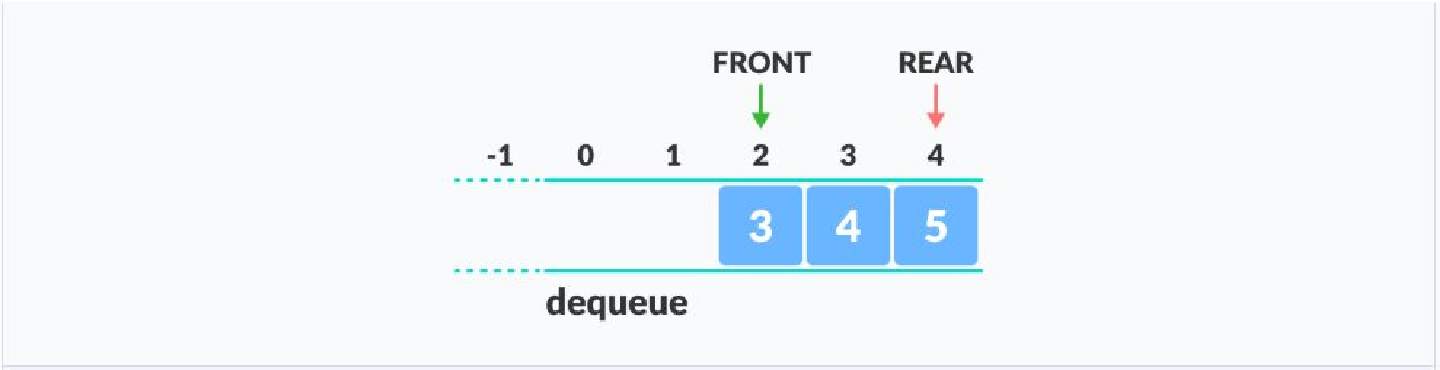
Date : 6th - 10 - 2020
Morning Session : 9am – 11.00 PM
By ~ Rohan Kumar

Topics: Circular Queue and Linked List

Circular Queue : Circular queue avoids the wastage of space in a regular queue implementation using arrays.



Limitation of the regular Queue

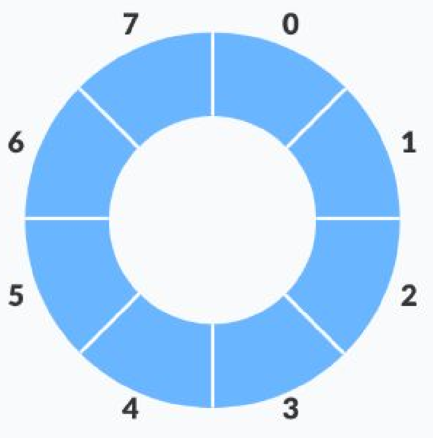
As you can see in the above image, after a bit of enqueuing and dequeuing, the size of the queue has been reduced.

The indexes 0 and 1 can only be used after the queue is reset when all the elements have been dequeued.

How Circular Queue Works

Circular Queue works by the process of circular increment i.e. when we try to increment the pointer and we reach the end of the queue, we start from the beginning of the queue.

Here, the circular increment is performed by modulo division with the queue size. That is,



Circular queue representation

Circular Queue Operations

The circular queue work as follows:

- two pointers `FRONT` and `REAR`
- `FRONT` track the first element of the queue
- `REAR` track the last elements of the queue
- initially, set value of `FRONT` and `REAR` to -1

1. Enqueue Operation

- check if the queue is full
- for the first element, set value of `FRONT` to 0
- circularly increase the `REAR` index by 1 (i.e. if the rear reaches the end, next it would be at the start of the queue)
- add the new element in the position pointed to by `REAR`

2. Dequeue Operation

- check if the queue is empty
- return the value pointed by `FRONT`
- circularly increase the `FRONT` index by 1
- for the last element, reset the values of `FRONT` and `REAR` to -1

However, the check for full queue has a new additional case:

- Case 1: `FRONT` = 0 && `REAR == SIZE - 1`
- Case 2: `FRONT = REAR + 1`

The second case happens when `REAR` starts from 0 due to circular increment and when its value is just 1 less than `FRONT`, the queue is full.

- ## Circular Queue Complexity Analysis
- The complexity of the enqueue and dequeue operations of a circular queue is `O(1)` for (array implementations).

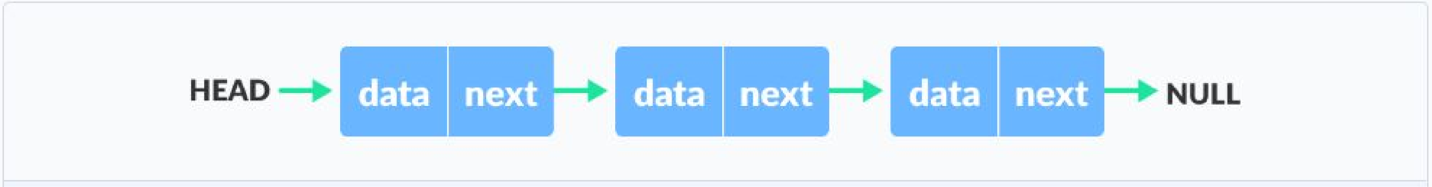
Applications of Circular Queue

- CPU scheduling
- Memory management
- Traffic Management

Queue Data Structure

Linked List :

A linked list data structure includes a series of connected nodes. Here, each node store the data and the address of the next node. For example,



Linkedin Data Structure

You have to start somewhere, so we give the address of the first node a special name called HEAD.

Also, the last node in the linked list can be identified because its next portion points to NULL.

You might have played the game Treasure Hunt, where each clue includes the information about the next clue. That is how the linked list operates.

Why Linked List?

Arrays can be used to store linear data of similar types, but arrays have the following limitations.

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.

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

For example, in a system, if 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.

Advantages over arrays

1) Dynamic size

2) Ease of insertion/deletion

Drawbacks:

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

Binary Search Linked List

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

3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

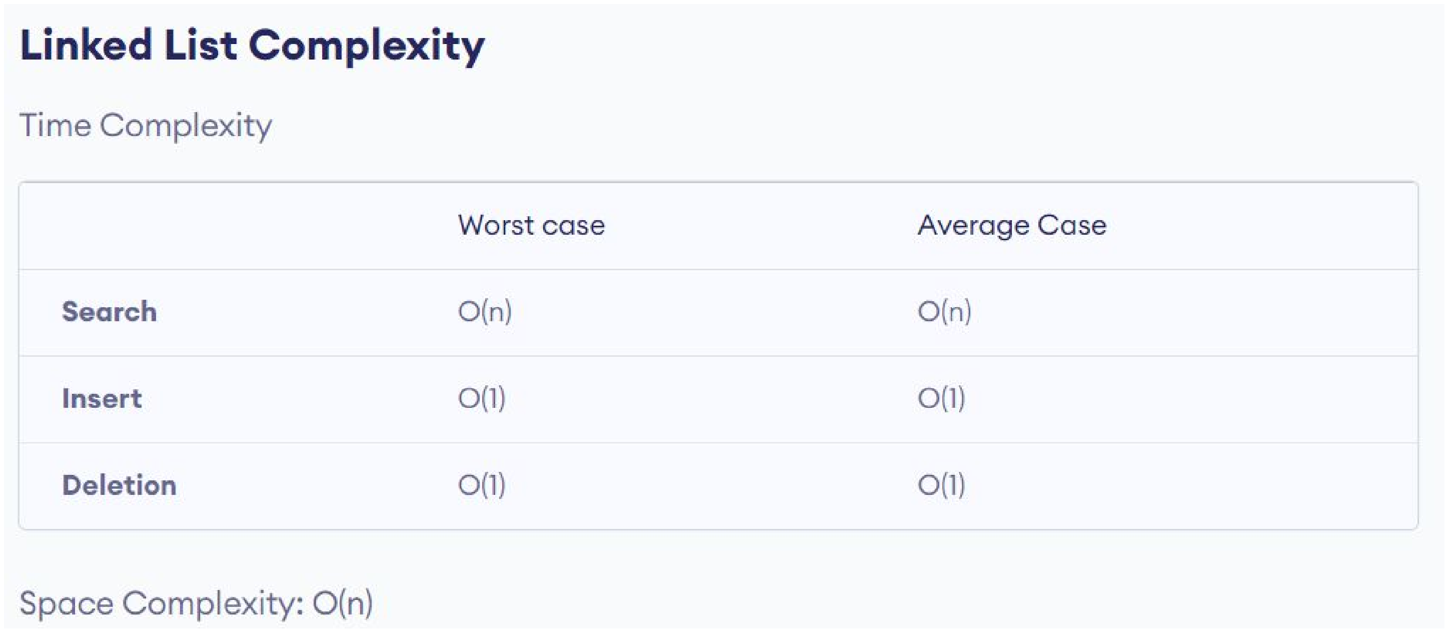
Representation:

A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.

Each node in a list consists of at least two parts:

1) data

2) Pointer (Or Reference) to the next node



Linked List Complexity

Time Complexity

	Worst case	Average Case
Search	O(n)	O(n)
Insert	O(1)	O(1)
Deletion	O(1)	O(1)

Space Complexity: O(n)

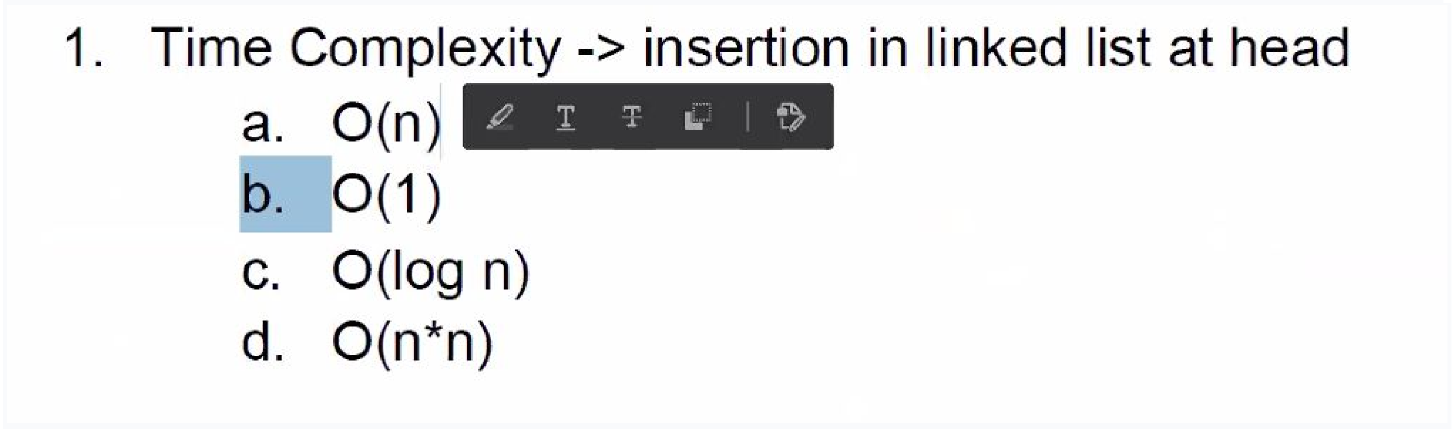
Linked List Applications

- Dynamic memory allocation
- Implemented in stack and queue
- In undo functionality of softwares
- Hash tables, Graphs

[Linked List Video]

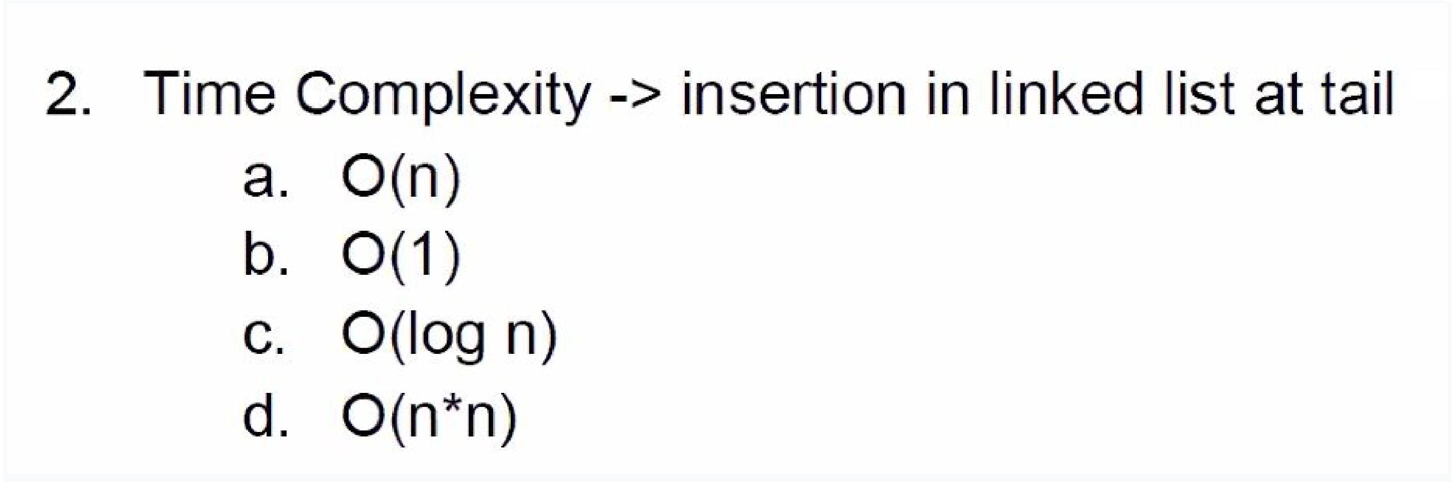
https://www.programiz.com/dsa/linked-list

MCQ's:



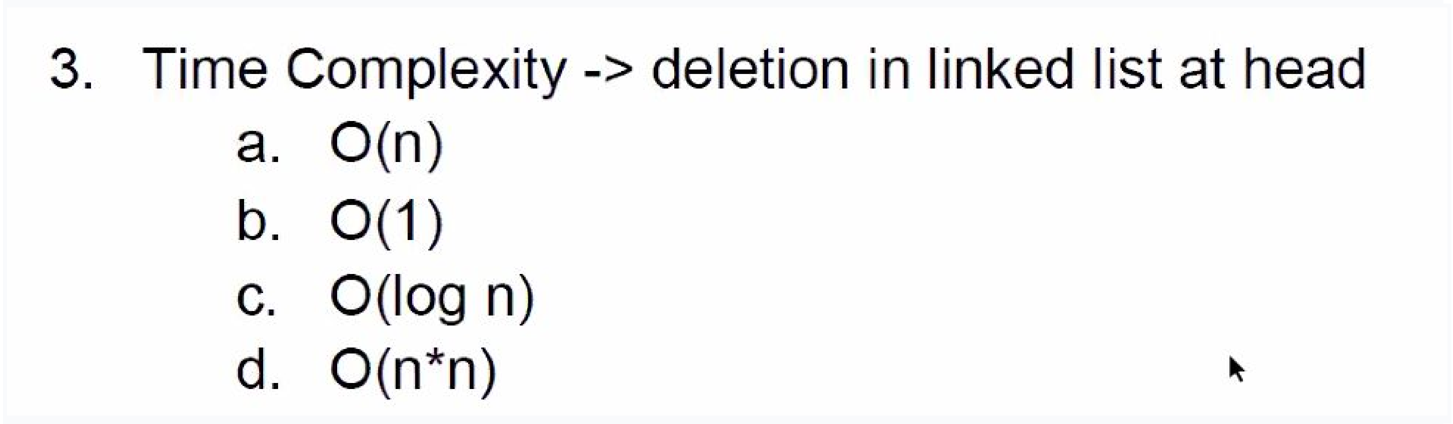
1. Time Complexity -> insertion in linked list at head
    a. O(n)
    b. O(1)
    c. O(log n)
    d. O(n*n)

Answer: B, O(1)



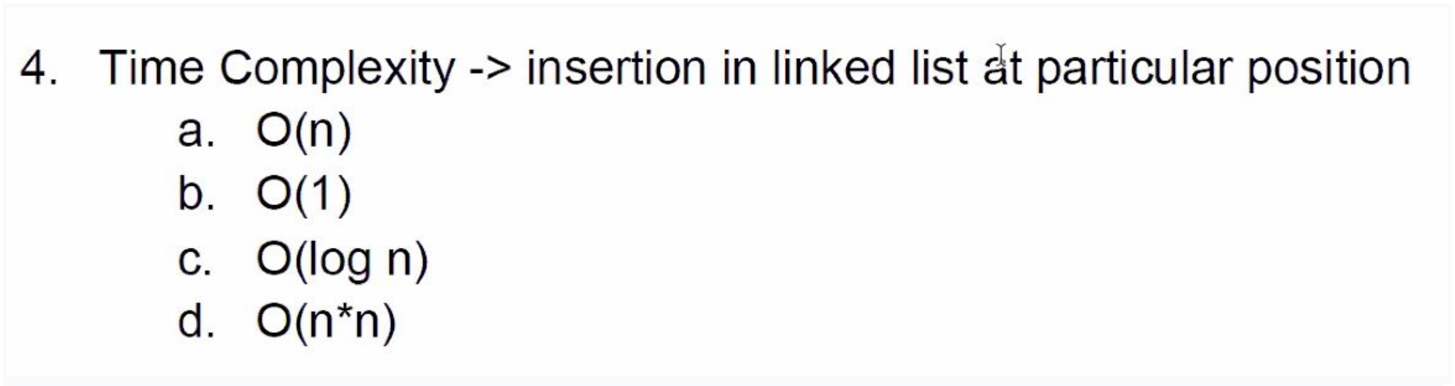
2. Time Complexity -> insertion in linked list at tail
    a. O(n)
    b. O(1)
    c. O(log n)
    d. O(n*n)

Answer: A, O(n)



3. Time Complexity -> deletion in linked list at head
    a. O(n)
    b. O(1)
    c. O(log n)
    d. O(n*n)

Answer: B, O(1)



4. Time Complexity -> insertion in linked list at particular position
    a. O(n)
    b. O(1)
    c. O(log n)
    d. O(n*n)

Answer : A, O(n)