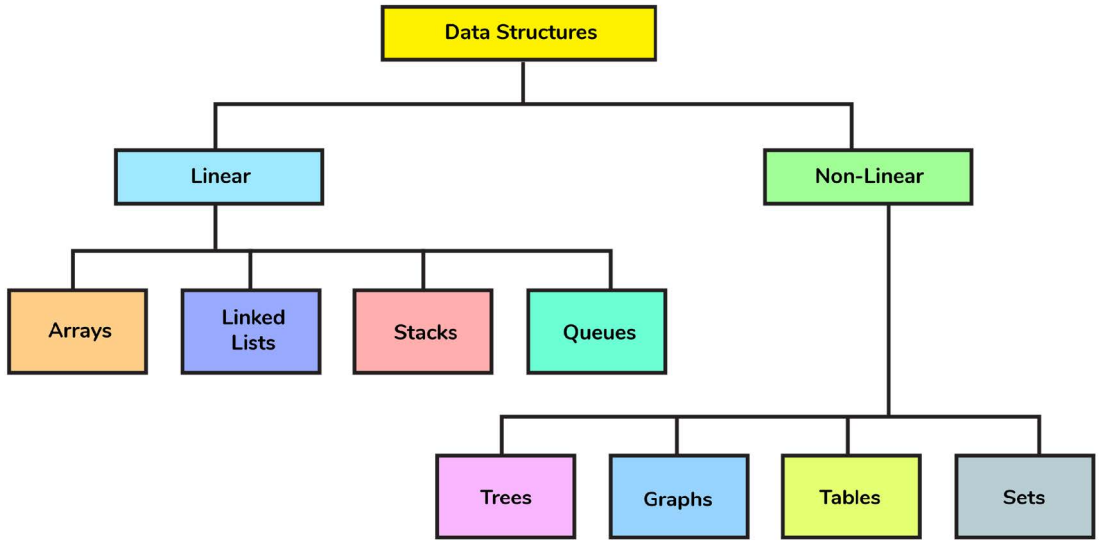
What is Data Structure?

- Data structure is a fundamental concept of any programming language, essential for algorithmic design.
- It is used for the efficient organization and modification of data.
- DS is how data and the relationship amongst different data is represented, that aids in how efficiently various functions or operations or algorithms can be applied.

Types

- There are two types of data structures:
    - Linear data structure: If the elements of a data structure result in a sequence or a linear list then it is called a linear data structure. Example: Arrays, Linked List, Stacks, Queues etc.
    - Non-linear data structure: If the elements of data structure results in a way that traversal of nodes is not done in a sequential manner, then it is a non linear data structure. Example: Trees, Graphs etc.

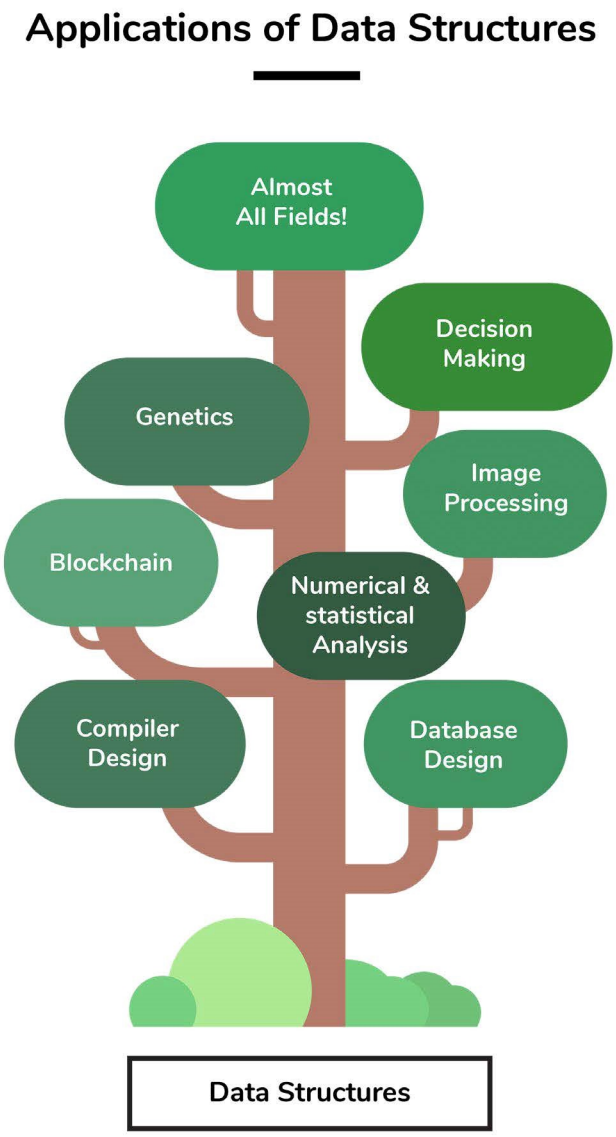


Applications

Data structures form the core foundation of software programming as any efficient algorithm to a given problem is dependent on how effectively a data is structured.

- Identifiers look ups in compiler implementations are built using hash tables.
- The B-trees data structures are suitable for the databases implementation.
- Some of the most important areas where data structures are used are as follows:
    1. Artificial intelligence
    2. Compiler design
    3. Machine learning
    4. Database design and management
    5. Blockchain

6. Numerical and Statistical analysis
7. Operating system development
8. Image & Speech Processing
9. Cryptography



Benefits of Learning Data Structures

Any given problem has constraints on how fast the problem should be solved (time) and how much less resources the problem consumes(space). That is, a problem is constrained by the space and time complexity within which it has to be solved efficiently.

- In order to do this, it is very much essential for the given problem to be represented in a proper structured format upon which efficient algorithms could be applied.

- Selection of proper data structure becomes the most important step before applying algorithm to any problem.

Having knowledge of different kinds of data structures available helps the programmer in choosing which data structure suits the best for solving a problem efficiently. It is not just important to make a problem work, it is important how efficiently you make it work.

Data structures in C, Java

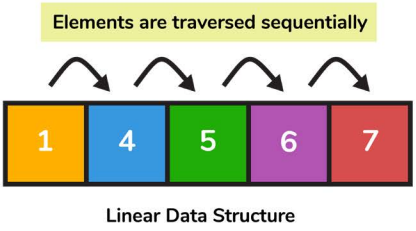
- The core concepts of data structures remains the same across all the programming languages. Only the implementation differs based on the syntax or the structure of the programming language.
    - The implementation in procedural languages like C is done with the help of structures, pointers, etc.
    - In an objected oriented language like Java, data structures are implemented by using classes and objects.
- Having sound knowledge of the concepts of each and every data structures helps you to stand apart in any interviews as selecting right data structure is the first step towards solving problem efficiently.

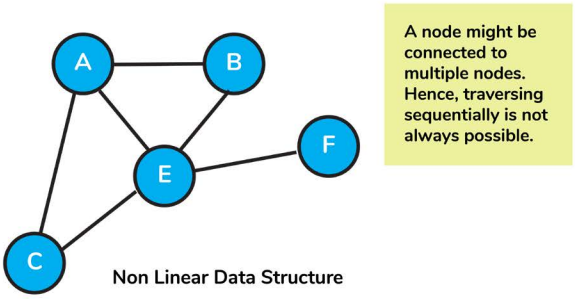
1. Can you explain the difference between file structure and storage structure?

- File Structure: Representation of data into secondary or auxiliary memory say any device such as hard disk or pen drives that stores data which remains intact until manually deleted is known as a file structure representation.
- Storage Structure: In this type, data is stored in the main memory i.e RAM, and is deleted once the function that uses this data gets completely executed.
- The difference is that storage structure has data stored in the memory of the computer system, whereas file structure has the data stored in the auxiliary memory.

2. Can you tell how linear data structures differ from non-linear data structures?

- If the elements of a data structure result in a sequence or a linear list then it is called a linear data structure. Whereas, traversal of nodes happens in a non-linear fashion in non-linear data structures.
- Lists, stacks, and queues are examples of linear data structures whereas graphs and trees are the examples of non-linear data structures.



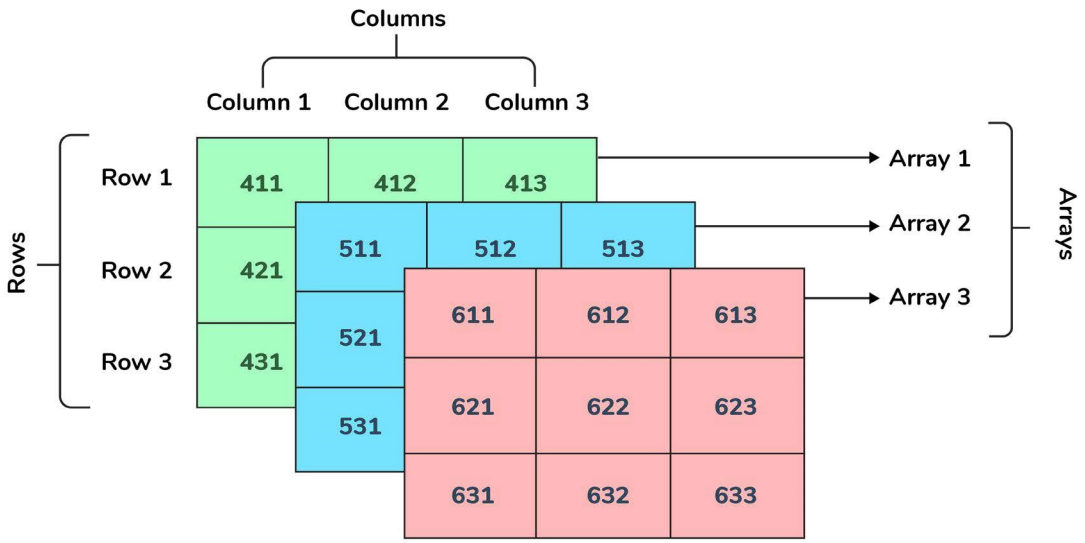


3. What is an array?

- Arrays are the collection of **similar** types of data stored at **contiguous** memory locations.
- It is the simplest data structure where the data element can be accessed randomly just by using its index number.

4. What is a multidimensional array?

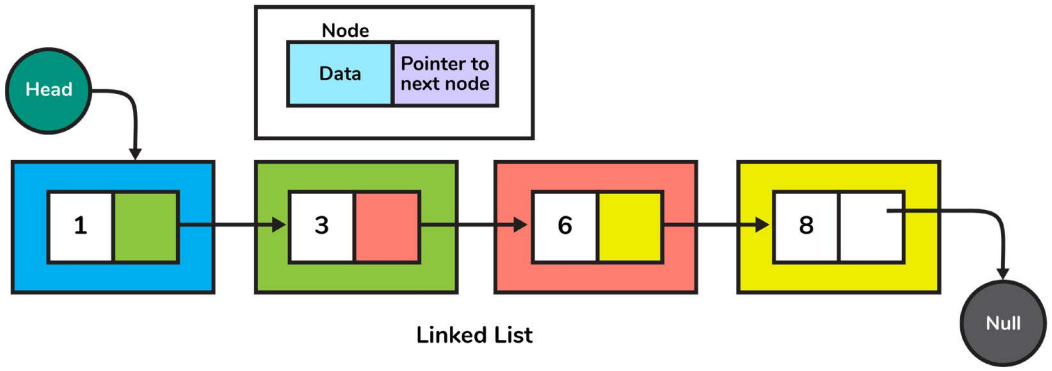
- Multi-dimensional arrays are those data structures that span across more than one dimension.
- This indicates that there will be more than one index variable for every point of storage. This type of data structure is primarily used in cases where data cannot be represented or stored using only one dimension. Most commonly used multidimensional arrays are 2D arrays.
    - 2D arrays emulates the tabular form structure which provides ease of holding the bulk of data that are accessed using row and column pointers.



5. What is a linked list?

A linked list is a data structure that has **sequence of nodes** where every node is connected to the next node by means of a reference pointer. The elements are **not stored in adjacent** memory locations. They are linked using pointers to form a chain. This forms a chain-like link for data storage.

- Each node element has two parts:
  - a data field
  - a reference (or pointer) to the next node.
- The first node in a linked list is called the head and the last node in the list has the pointer to NULL. Null in the reference field indicates that the node is the last node. When the list is empty, the head is a null reference.



6. Are linked lists of linear or non-linear type?

Linked lists can be considered both linear and non-linear data structures. This depends upon the application that they are used for.

- When linked list is used for access strategies, it is considered as a linear data-structure. When they are used for data storage, it can be considered as a non-linear data structure.

7. How are linked lists more efficient than arrays?

1. **Insertion and Deletion**
   - Insertion and deletion process is expensive in an array as the room has to be created for the new elements and existing elements must be shifted.
   - But in a linked list, the same operation is an easier process, as we only update the address present in the next pointer of a node.
2. **Dynamic Data Structure**
   - Linked list is a dynamic data structure that means there is no need to give an initial size at the time of creation as it can grow and shrink at runtime by allocating and deallocating memory.
   - Whereas, the size of an array is limited as the number of items is statically stored in the main memory.
3. **No wastage of memory**
   - As the size of a linked list can grow or shrink based on the needs of the program, there is no memory wasted because it is allocated in runtime.

- In arrays, if we declare an array of size 10 and store only 3 elements in it, then the space for 3 elements is wasted. Hence, chances of memory wastage is more in arrays.

8. Explain the scenarios where you can use linked lists and arrays.

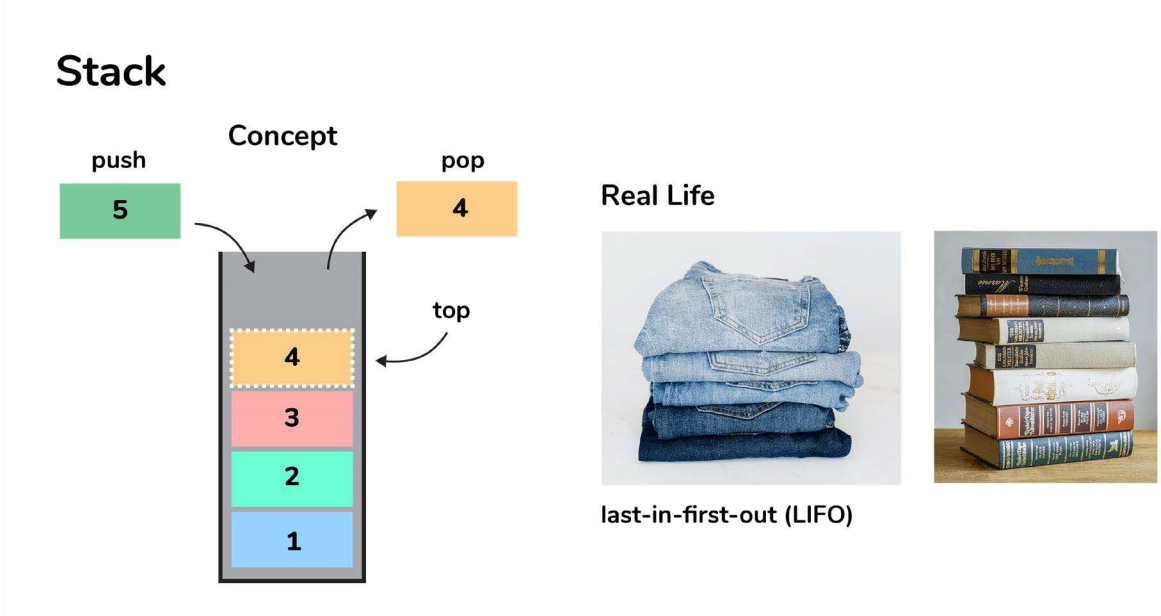
- Following are the scenarios where we use linked list over array:
  - When we do not know the exact number of elements beforehand.
  - When we know that there would be large number of add or remove operations.
  - Less number of random access operations.
  - When we want to insert items anywhere in the middle of the list, such as when implementing a priority queue, linked list is more suitable.
- Below are the cases where we use arrays over the linked list:
  - When we need to index or randomly access elements more frequently.
  - When we know the number of elements in the array beforehand in order to allocate the right amount of memory.
  - When we need speed while iterating over the elements in the sequence.
  - When memory is a concern:
    1. Due to the nature of arrays and linked list, it is safe to say that filled arrays use less memory than linked lists.
    2. Each element in the array indicates just the data whereas each linked list node represents the data as well as one or more pointers or references to the other elements in the linked list.
- To summarize, requirements of space, time, and ease of implementation are considered while deciding which data structure has to be used over what.

9. What is a doubly-linked list (DLL)? What are its applications.

- This is a complex type of a linked list wherein a node has two references:
  - One that connects to the next node in the sequence
  - Another that connects to the previous node.
- This structure allows traversal of the data elements in both directions (left to right and vice versa).
- Applications of DLL are:
  - A music playlist with next song and previous song navigation options.
  - The browser cache with BACK-FORWARD visited pages
  - The undo and redo functionality on platforms such as word, paint etc, where you can reverse the node to get to the previous page.

10. What is a stack? What are the applications of stack?

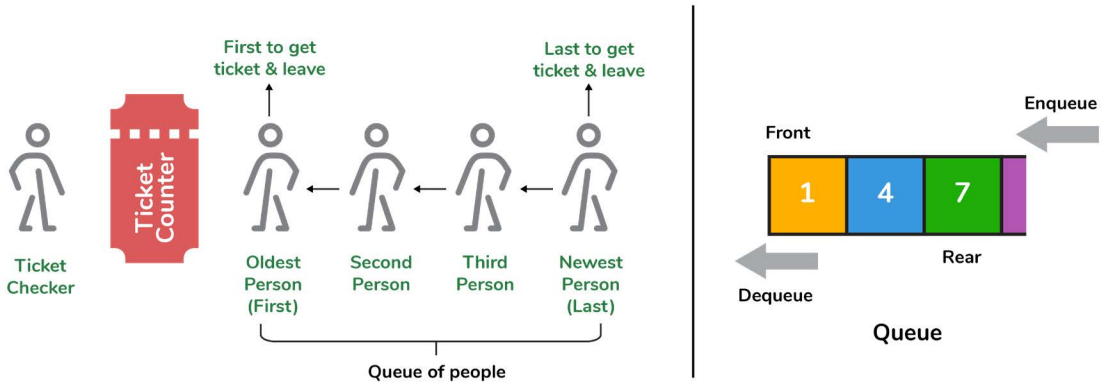
- Stack is a linear data structure that follows LIFO (Last In First Out) approach for accessing elements.
- Push, pop, and top (or peek) are the basic operations of a stack.



- Following are some of the applications of a stack:
  - Check for balanced parentheses in an expression
  - Evaluation of a postfix expression
  - Problem of Infix to postfix conversion
  - Reverse a string

11. What is a queue? What are the applications of queue?

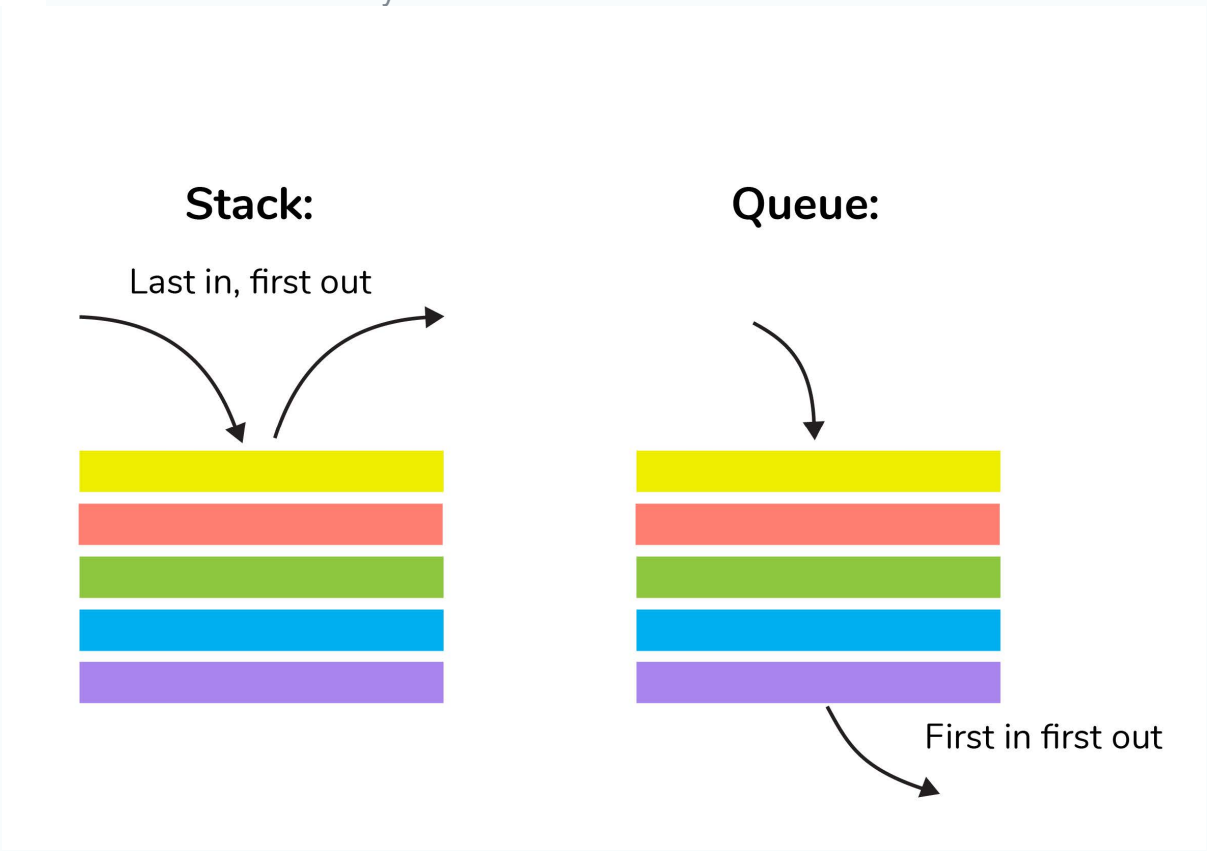
- A queue is a linear data structure that follows the FIFO (First In First Out) approach for accessing elements.
- Dequeue from the queue, enqueue element to the queue, get front element of queue, and get rear element of queue are basic operations that can be performed.



- Some of the applications of queue are:
  - CPU Task scheduling
  - BFS algorithm to find shortest distance between two nodes in a graph.
  - Website request processing
  - Used as buffers in applications like MP3 media player, CD player, etc.
  - Managing an Input stream

12. How is a stack different from a queue?

- In a stack, the item that is most recently added is removed first whereas in queue, the item least recently added is removed first.



13. Explain the process behind storing a variable in memory.

- A variable is stored in memory based on the amount of memory that is needed. Following are the steps followed to store a variable:
    - The required amount of memory is assigned first.
    - Then, it is stored based on the data structure being used.
        - Using concepts like dynamic allocation ensures high efficiency and that the storage units can be accessed based on requirements in real time.

14. How to implement a queue using stack?

- A queue can be implemented using **two stacks**. Let `q` be the queue and `stack1` and `stack2` be the 2 stacks for implementing `q`. We know that stack supports push, pop, peek operations and using these operations, we need to emulate the operations of queue - enqueue and dequeue. Hence, queue `q` can be implemented in two methods (Both the methods use auxillary space complexity of O(n)):
    1. **By making enqueue operation costly:**
        - Here, the oldest element is always at the top of `stack1` which ensures dequeue operation to occur in O(1) time complexity.
        - To place element at top of stack1, stack2 is used.
        - **Pseudocode:**
            - Enqueue: Here time complexity will be O(n)

```
enqueue(q, data):

  While stack1 is not empty:
      Push everything from stack1 to stack2.
      Push data to stack1
      Push everything back to stack1.
```

- Dequeue: Here time complexity will be O(1)

```
deQueue(q):

  If stack1 is empty then error
  else
      Pop an item from stack1 and return it
```

2. **By making dequeue operation costly:**
   - Here, for enqueue operation, the new element is pushed at the top of `stack1`. Here, the enqueue operation time complexity is O(1).
   - In dequeue, if `stack2` is empty, all elements from `stack1` are moved to `stack2` and top of `stack2` is the result. Basically, reversing the list by pushing to a stack and returning the first enqueued element. This operation of pushing all elements to new stack takes O(n) complexity.
   - **Pseudocode:**
     - Enqueue: Time complexity: O(1)

```
enqueue(q, data):
    Push data to stack1
```

     - Dequeue: Time complexity: O(n)

```
dequeue(q):

    If both stacks are empty then raise error.
    If stack2 is empty:
        While stack1 is not empty:
            push everything from stack1 to stack2.
    Pop the element from stack2 and return it.
```

15. How do you implement stack using queues?

- A stack can be implemented using two queues. We know that a queue supports enqueue and dequeue operations. Using these operations, we need to develop push, pop operations.
- Let stack be 's' and queues used to implement be 'q1' and 'q2'. Then, stack 's' can be implemented in two ways:

1. **By making push operation costly:**
   - This method ensures that newly entered element is always at the front of 'q1', so that pop operation just dequeues from 'q1'.
   - 'q2' is used as auxillary queue to put every new element at front of 'q1' while ensuring pop happens in O(1) complexity.
   - **Pseudocode:**
     - Push element to stack s : Here push takes O(n) time complexity.

```
push(s, data):

    Enqueue data to q2

    Dequeue elements one by one from q1 and enqueue to q
2.

    Swap the names of q1 and q2
```

     - Pop element from stack s: Takes O(1) time complexity.

```
pop(s):
    dequeue from q1 and return it.
```

2. **By making pop operation costly:**
   - In push operation, the element is enqueued to q1.
   - In pop operation, all the elements from q1 except the last remaining element, are pushed to q2 if it is empty. That last element remaining of q1 is dequeued and returned.
   - **Pseudocode:**
     - Push element to stack s : Here push takes O(1) time complexity.

```
push(s,data):
    Enqueue data to q1
```

     - Pop element from stack s: Takes O(n) time complexity.

```
pop(s):
    Step1: Dequeue every elements except the last element from q1 and enqueue to q2.
    Step2: Dequeue the last item of q1, the dequeued item is stored in result variable.
    Step3: Swap the names of q1 and q2 (for getting updated data after dequeue)
    Step4: Return the result.
```