**Difference between Stack and Queue Data Structures**

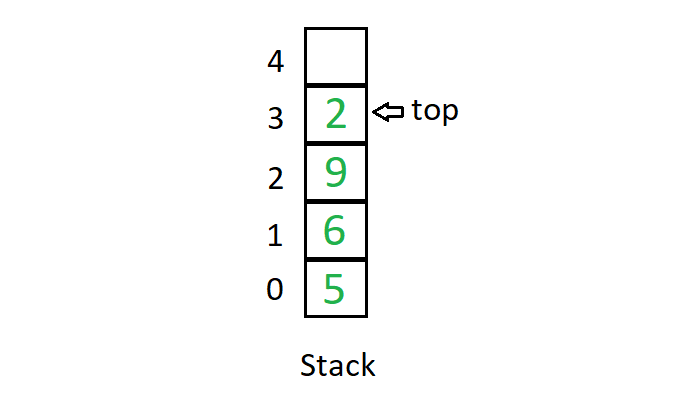
* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 15 Sep, 2022

 Read

 Discuss

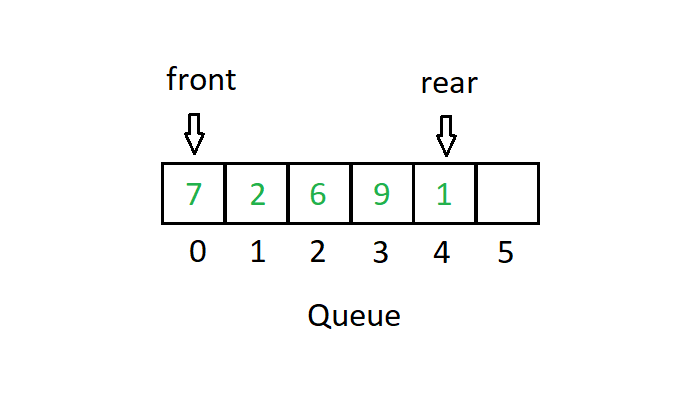
**Stack** A [stack](https://www.geeksforgeeks.org/stack-data-structure-introduction-program/) is a linear data structure in which elements can be inserted and deleted only from one side of the list, called the **top**. A stack follows the **LIFO** (Last In First Out) principle, i.e., the element inserted at the last is the first element to come out. The insertion of an element into the stack is called **push** operation, and the deletion of an element from the stack is called **pop** operation. In stack, we always keep track of the last element present in the list with a pointer called **top**.

The diagrammatic representation of the stack is given below:



[Queue](https://www.geeksforgeeks.org/queue-set-1introduction-and-array-implementation/) is a linear data structure in which elements can be inserted only from one side of the list called **rear**, and the elements can be deleted only from the other side called the **front**. The queue data structure follows the **FIFO** (First In First Out) principle, i.e. the element inserted at first in the list, is the first element to be removed from the list. The insertion of an element in a queue is called an **enqueue** operation and the deletion of an element is called a **dequeue** operation. In queue, we always maintain two pointers, one pointing to the element which was inserted at the first and still present in the list with the **front** pointer and the second pointer pointing to the element inserted at the last with the **rear** pointer.

The diagrammatic representation of the queue is given below:



Difference between Stack and Queue Data Structures are as follows:

| **Stacks** | **Queues** |
| --- | --- |
| Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list. | Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list. |
| Insertion and deletion in stacks takes place only from one end of the list called the top. | Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list. |
| Insert operation is called push operation. | Insert operation is called enqueue operation. |
| Delete operation is called pop operation. | Delete operation is called dequeue operation. |
| In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list. | In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element. |
| Stack is used in solving problems works on recursion. | Queue is used in solving problems having sequential processing. |
| Stack does not have any types. | Queue is of three types – 1. Circular Queue 2. Priority queue 3. double-ended queue. |
| Can be considered as a vertical collection visual. | Can be considered as a horizontal collection visual. |

# Linked List vs Array

* **Difficulty Level :** [Basic](https://www.geeksforgeeks.org/basic/)
* **Last Updated :** 02 Sep, 2022

 Read

 Discuss

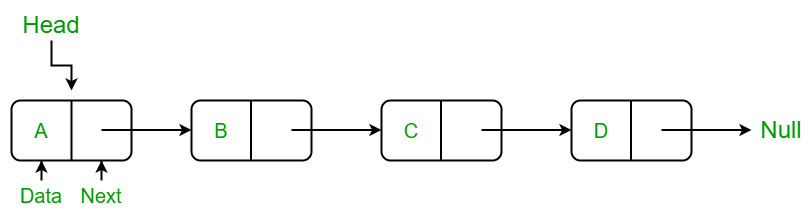
**Array:** [Arrays](https://www.geeksforgeeks.org/array-data-structure/) store elements in contiguous memory locations, resulting in easily calculable addresses for the elements stored and this allows faster access to an element at a specific index.

**Linked List:** [Linked lists](https://www.geeksforgeeks.org/data-structures/linked-list/) are less rigid in their storage structure and elements are usually not stored in contiguous locations, hence they need to be stored with additional tags giving a reference to the next element.

This difference in the data storage scheme decides which data structure would be more suitable for a given situation.



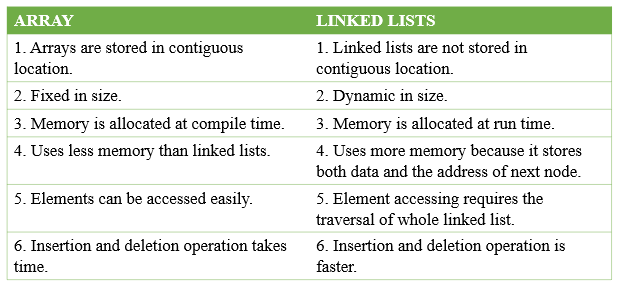
Data storage scheme of an array



Linked-List representation

## Major differences between array and linked-list are listed below:

* **Size:** Since data can only be stored in contiguous blocks of memory in an array, its size cannot be altered at runtime due to the risk of overwriting other data.   
  However, in a linked list, each node points to the next one such that data can exist at scattered (non-contiguous) addresses; this allows for a dynamic size that can change at runtime.
* **Memory allocation:** For arrays at compile time and at runtime for linked lists. but, a dynamically allocated array also allocates memory at runtime.
* **Memory efficiency:** For the same number of elements, linked lists use more memory as a reference to the next node is also stored along with the data. However, size flexibility in linked lists may make them use less memory overall; this is useful when there is uncertainty about size or there are large variations in the size of data elements;   
  Memory equivalent to the upper limit on the size has to be allocated (even if not all of it is being used) while using arrays, whereas linked lists can increase their sizes step-by-step proportionately to the amount of data.
* **Execution time:** Any element in an array can be directly accessed with its index. However, in the case of a linked list, all the previous elements must be traversed to reach any element.   
  Also, better cache locality in arrays (due to contiguous memory allocation) can significantly improve performance. As a result, some operations (such as modifying a certain element) are faster in arrays, while others (such as inserting/deleting an element in the data) are faster in linked lists.
* **Insertion:** In an array, insertion operation takes more time but in a linked list these operations are fast. For example, if we want to insert an element in the array at the end position in the array and the array is full then we copy the array into another array and then we can add an element whereas if the linked list is full then we find the last node and make it next to the new node
* **Dependency**: In an array, values are independent of each other but   
  In the case of linked list nodes are dependent on each other. one node is dependent on its previous node. If the previous node is lost then we can’t find its next subsequent nodes.



Array vs Linked List

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

## Advantages of Linked Lists:

* 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 usage, and in practical uses, the upper limit is rarely reached.
* 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 a room existing elements have to be shifted.

**Example:**

suppose we maintain a sorted list of IDs in an array 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 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

## Disadvantages of Linked Lists:

* 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.
* Extra memory space for a pointer is required for each element of the list.
* Arrays have a better cache locality that can make a pretty big difference in performance.
* It takes a lot of time in traversing and changing the pointers.
* It will be confusing when we work with pointers.

## Advantages of Arrays:

* Arrays store multiple data of similar types with the same name.
* It allows random access to elements.
* As the array is of fixed size and stored in contiguous memory locations there is no memory shortage or overflow.
* It is helpful to store any type of data with a fixed size.
* Since the elements in the array are stored at contiguous memory locations it is easy to iterate in this data structure and unit time is required to access an element if the index is known.

## Disadvantages of Arrays:

* The array is static in nature. Once the size of the array is declared then we can’t modify it.
* Insertion and deletion operations are difficult in an array as elements are stored in contiguous memory locations and the shifting operations are costly.
* The number of elements that have to be stored in an array should be known in advance.
* Wastage of memory is the main problem in the array. If the array size is big the less allocation of memory leads to wastage of memory.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.