**DATA STRUCTURES:**

Organizing the data in a main memory is known as data structures .There are two types of data structures those are

Linear data structure

Non-linear data structure

The arrangement of data in a sequential manner is known as a linear data structure. The data structures used for this purpose are Arrays, Linked list, Stacks, and Queues. In these data structures, one element is connected to only one another element in a linear form.

**When one element is connected to the 'n' number of elements known as a non-linear data structure. The best example is trees and graphs. In this case, the elements are arranged in a random manner.**

Major Operations

The major or the common operations that can be performed on the data structures are:

ADVERTISEMENT

ADVERTISEMENT

* **Searching:** We can search for any element in a data structure.
* **Sorting:** We can sort the elements of a data structure either in an ascending or descending order.
* **Insertion:** We can also insert the new element in a data structure.
* **Updation:** We can also update the element, i.e., we can replace the element with another element.
* **Deletion:** We can also perform the delete operation to remove the element from the data structure.

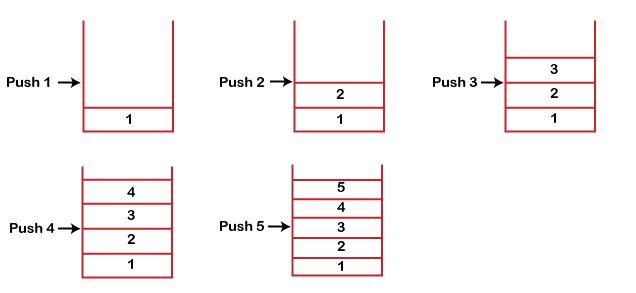
# **Stack**

A Stack is a linear data structure that follows the **LIFO (Last-In-First-Out)** principle. Stack has one end, whereas the Queue has two ends (**front and rear**). It contains only one pointer **top pointer** pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack. In other words, a **stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.**

Standard Stack Operations

**The following are some common operations implemented on the stack:**

* **push():** When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
* **pop():** When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
* **isEmpty():** It determines whether the stack is empty or not.
* **isFull():** It determines whether the stack is full or not.'
* **peek():** It returns the element at the given position.
* **count():** It returns the total number of elements available in a stack.
* **change():** It changes the element at the given position.
* **display():** It prints all the elements available in the stack.



* If we try to insert the element in a stack, and the stack is full, then the ***overflow*** condition occurs.
* If we try to delete the element from the empty stack, then the ***underflow*** condition occurs.

# **Queue**

1. A queue can be defined as an ordered list which enables insert operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**.

2. Queue is referred to be as First In First Out list.

3. For example, people waiting in line for a rail ticket form a queue.



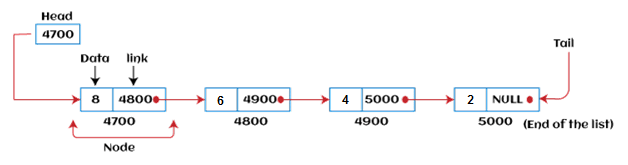
# **Linked list**

In this article, we will see the introduction of linked list.

Linked list is a linear data structure that includes a series of connected nodes. Linked list can be defined as the nodes that are randomly stored in the memory. A node in the linked list contains two parts, i.e., first is the data part and second is the address part. The last node of the list contains a pointer to the null. After array, linked list is the second most used data structure. In a linked list, every link contains a connection to another link.

### Representation of a Linked list

Linked list can be represented as the connection of nodes in which each node points to the next node of the list. The representation of the linked list is shown below -



Till now, we have been using array data structure to organize the group of elements that are to be stored individually in the memory. However, Array has several advantages and disadvantages that must be known to decide the data structure that will be used throughout the program.

Now, the question arises why we should use linked list over array?

### Why use linked list over array?

Linked list is a data structure that overcomes the limitations of arrays. Let's first see some of the limitations of arrays -

* The size of the array must be known in advance before using it in the program.
* Increasing the size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
* All the elements in the array need to be contiguously stored in the memory. Inserting an element in the array needs shifting of all its predecessors.

Linked list is useful because -

* It allocates the memory dynamically. All the nodes of the linked list are non-contiguously stored in the memory and linked together with the help of pointers.
* In linked list, size is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

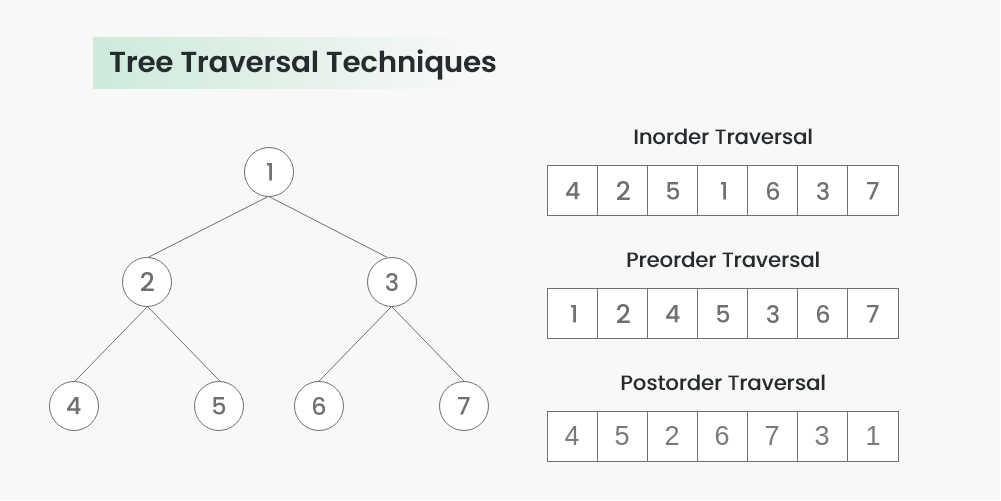
**Tree:**

*A****tree data structure****is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.*

*The topmost node of the tree is called the root, and the nodes below it are called the child nodes. Each node can have multiple child nodes, and these child nodes can also have their own child nodes, forming a recursive structure.*

## Basic Operation Of Tree Data Structure:

* **Create** – create a tree in the data structure.
* **Insert** − Inserts data in a tree.
* **Search** − Searches specific data in a tree to check whether it is present or not.
* **Traversal**:
  + **Preorder Traversal** – perform Traveling a tree in a pre-order manner in the data structure.
  + **In order Traversal** – perform Traveling a tree in an in-order manner.
  + **Post-order Traversal** –perform Traveling a tree in a post-order manner.



**Inorder🡪 left , root , right**

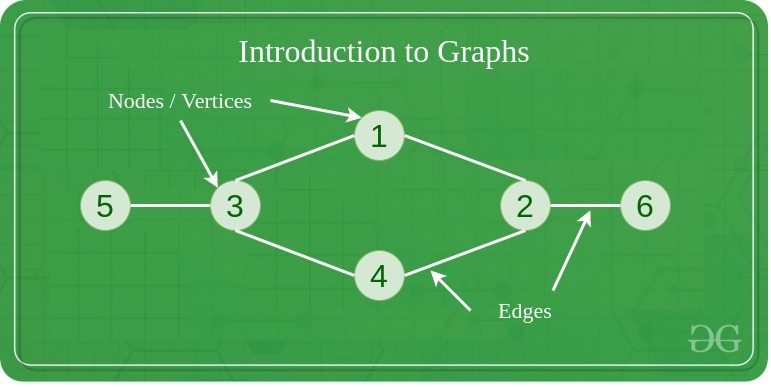
**Preorder🡪 root , left , right**

**Postorder🡪 left , right , root**

*A****Graph****is a****non-linear data structure****consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices( V ) and a set of edges( E ). The graph is denoted by G(E, V).*

## Components of a Graph

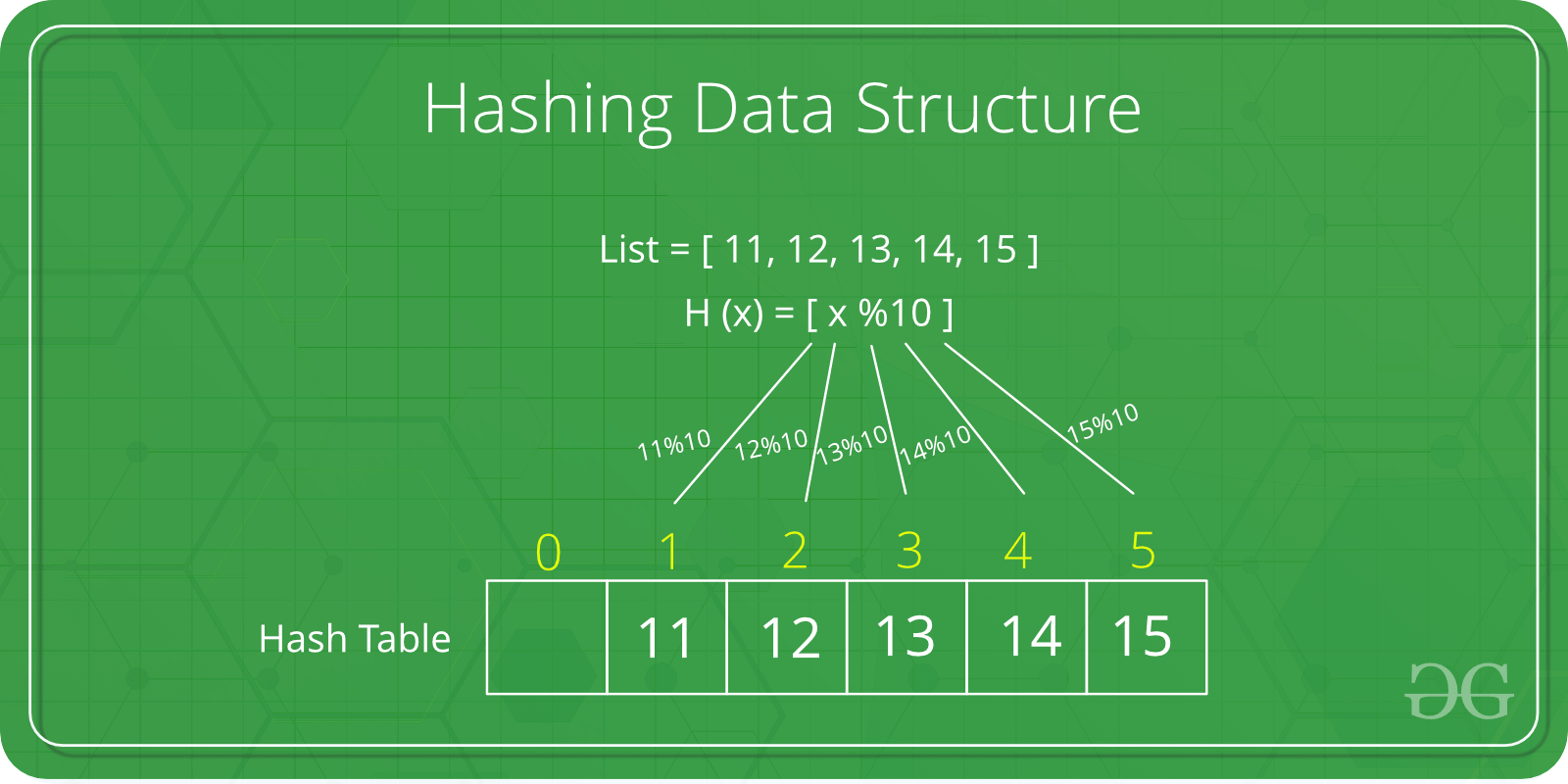
* **Vertices:** Vertices are the fundamental units of the graph. Sometimes, vertices are also known as vertex or nodes. Every node/vertex can be labeled or unlabelled.
* **Edges:** Edges are drawn or used to connect two nodes of the graph. It can be ordered pair of nodes in a directed graph. Edges can connect any two nodes in any possible way. There are no rules. Sometimes, edges are also known as arcs. Every edge can be labeled/unlabelled.



## [**What is Hashing?**](https://www.geeksforgeeks.org/introduction-to-hashing-data-structure-and-algorithm-tutorials/)

**Hashing** is a technique or process of mapping keys, and values into the hash table by using a **hash function**. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

Let a **hash function H(x)** maps the value **x** at the index **x%10** in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



# **Bubble sort Algorithm:**

n this article, we will discuss the Bubble sort Algorithm. The working procedure of bubble sort is simplest. This article will be very helpful and interesting to students as they might face bubble sort as a question in their examinations. So, it is important to discuss the topic.

Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order. It is called bubble sort because the movement of array elements is just like the movement of air bubbles in the water. Bubbles in water rise up to the surface; similarly, the array elements in bubble sort move to the end in each iteration.

Although it is simple to use, it is primarily used as an educational tool because the performance of bubble sort is poor in the real world. It is not suitable for large data sets. The average and worst-case complexity of Bubble sort is **O(n2),** where **n** is a number of items.

# **Insertion Sort Algorithm:**

In this article, we will discuss the Insertion sort Algorithm. The working procedure of insertion sort is also simple. This article will be very helpful and interesting to students as they might face insertion sort as a question in their examinations. So, it is important to discuss the topic.

Insertion sort works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.

The same approach is applied in insertion sort. The idea behind the insertion sort is that first take one element, iterate it through the sorted array. Although it is simple to use, it is not appropriate for large data sets as the time complexity of insertion sort in the average case and worst case is **O(n2)**, where n is the number of items. Insertion sort is less efficient than the other sorting algorithms like heap sort, quick sort, merge sort, etc.

Insertion sort has various advantages such as -

Now, let's see the working of the insertion sort Algorithm.

To understand the working of the insertion sort algorithm, let's take an unsorted array. It will be easier to understand the insertion sort via an example.

Let the elements of array are -

Insertion Sort Algorithm

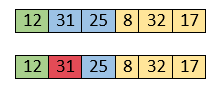
Initially, the first two elements are compared in insertion sort.

Insertion Sort Algorithm

Here, 31 is greater than 12. That means both elements are already in ascending order. So, for now, 12 is stored in a sorted sub-array.

Insertion Sort Algorithm

Now, move to the next two elements and compare them.

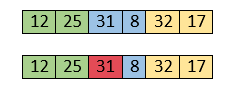


Here, 25 is smaller than 31. So, 31 is not at correct position. Now, swap 31 with 25. Along with swapping, insertion sort will also check it with all elements in the sorted array.

For now, the sorted array has only one element, i.e. 12. So, 25 is greater than 12. Hence, the sorted array remains sorted after swapping.

Insertion Sort Algorithm

Now, two elements in the sorted array are 12 and 25. Move forward to the next elements that are 31 and 8.



Both 31 and 8 are not sorted. So, swap them.

Insertion Sort Algorithm

After swapping, elements 25 and 8 are unsorted.

Insertion Sort Algorithm

So, swap them.

Insertion Sort Algorithm

Now, elements 12 and 8 are unsorted.

Insertion Sort Algorithm

So, swap them too.

Insertion Sort Algorithm

Now, the sorted array has three items that are 8, 12 and 25. Move to the next items that are 31 and 32.

Insertion Sort Algorithm

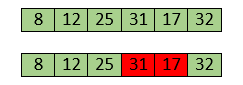
Hence, they are already sorted. Now, the sorted array includes 8, 12, 25 and 31.

Insertion Sort Algorithm

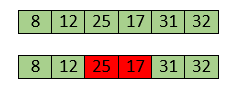
Move to the next elements that are 32 and 17.

Insertion Sort Algorithm

17 is smaller than 32. So, swap them.



Swapping makes 31 and 17 unsorted. So, swap them too.



Now, swapping makes 25 and 17 unsorted. So, perform swapping again.

Insertion Sort Algorithm

Now, the array is completely sorted.

# **Merge Sort Algorithm:**

In this article, we will discuss the merge sort Algorithm. Merge sort is the sorting technique that follows the divide and conquer approach. This article will be very helpful and interesting to students as they might face merge sort as a question in their examinations. In coding or technical interviews for software engineers, sorting algorithms are widely asked. So, it is important to discuss the topic.

Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It is one of the most popular and efficient sorting algorithm. It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the **merge()** function to perform the merging.

The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process. The sorted two-element pairs is merged into the four-element lists, and so on until we get the sorted list.

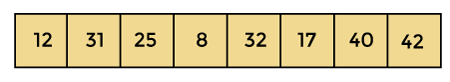
Now, let's see the algorithm of merge sort.

## **Working of Merge sort Algorithm**

Now, let's see the working of merge sort Algorithm.

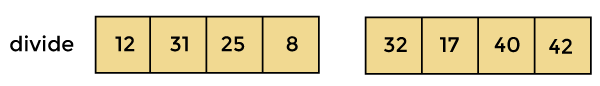
To understand the working of the merge sort algorithm, let's take an unsorted array. It will be easier to understand the merge sort via an example.

Let the elements of array are -

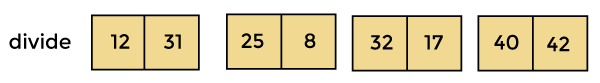


According to the merge sort, first divide the given array into two equal halves. Merge sort keeps dividing the list into equal parts until it cannot be further divided.

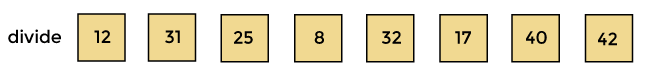
As there are eight elements in the given array, so it is divided into two arrays of size 4.



Now, again divide these two arrays into halves. As they are of size 4, so divide them into new arrays of size 2.



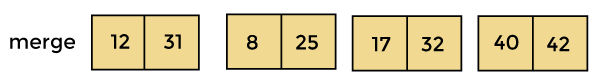
Now, again divide these arrays to get the atomic value that cannot be further divided.



Now, combine them in the same manner they were broken.

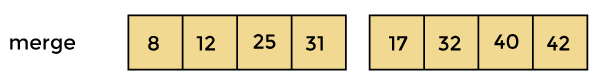
In combining, first compare the element of each array and then combine them into another array in sorted order.

So, first compare 12 and 31, both are in sorted positions. Then compare 25 and 8, and in the list of two values, put 8 first followed by 25. Then compare 32 and 17, sort them and put 17 first followed by 32. After that, compare 40 and 42, and place them sequentially.

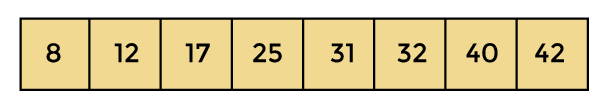


ADVERTISEMENT

In the next iteration of combining, now compare the arrays with two data values and merge them into an array of found values in sorted order.



Now, there is a final merging of the arrays. After the final merging of above arrays, the array will look like -



Now, the array is completely sorted.

# **Quick Sort Algorithm**

In this article, we will discuss the Quicksort Algorithm. The working procedure of Quicksort is also simple. This article will be very helpful and interesting to students as they might face quicksort as a question in their examinations. So, it is important to discuss the topic.

Sorting is a way of arranging items in a systematic manner. Quicksort is the widely used sorting algorithm that makes **n log n** comparisons in average case for sorting an array of n elements. It is a faster and highly efficient sorting algorithm. This algorithm follows the divide and conquer approach. Divide and conquer is a technique of breaking down the algorithms into subproblems, then solving the subproblems, and combining the results back together to solve the original problem.

**Divide:** In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.

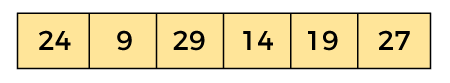
**Conquer:** Recursively, sort two subarrays with Quicksort.

## **Working of Quick Sort Algorithm**

Now, let's see the working of the Quicksort Algorithm.

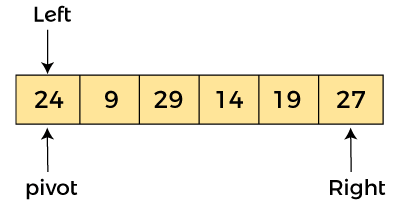
To understand the working of quick sort, let's take an unsorted array. It will make the concept more clear and understandable.

Let the elements of array are -

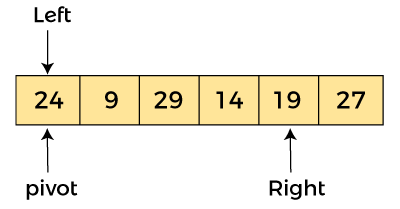


In the given array, we consider the leftmost element as pivot. So, in this case, a[left] = 24, a[right] = 27 and a[pivot] = 24.

Since, pivot is at left, so algorithm starts from right and move towards left.

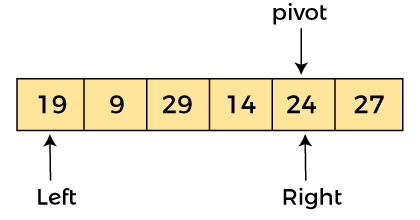


Now, a[pivot] < a[right], so algorithm moves forward one position towards left, i.e. -



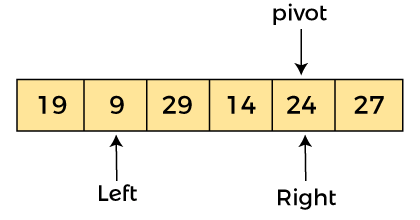
Now, a[left] = 24, a[right] = 19, and a[pivot] = 24.

Because, a[pivot] > a[right], so, algorithm will swap a[pivot] with a[right], and pivot moves to right, as -

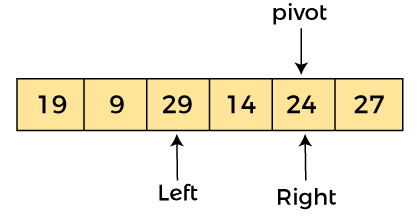


Now, a[left] = 19, a[right] = 24, and a[pivot] = 24. Since, pivot is at right, so algorithm starts from left and moves to right.

As a[pivot] > a[left], so algorithm moves one position to right as -



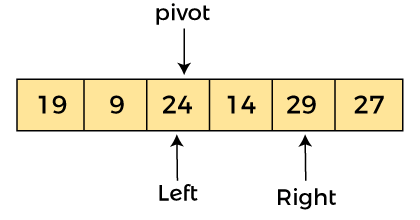
Now, a[left] = 9, a[right] = 24, and a[pivot] = 24. As a[pivot] > a[left], so algorithm moves one position to right as -



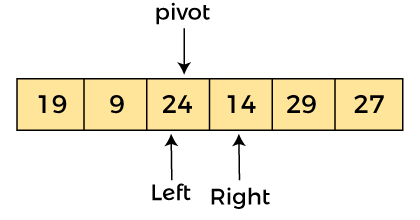
ADVERTISEMENT

ADVERTISEMENT

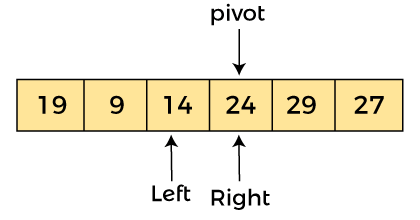
Now, a[left] = 29, a[right] = 24, and a[pivot] = 24. As a[pivot] < a[left], so, swap a[pivot] and a[left], now pivot is at left, i.e. -



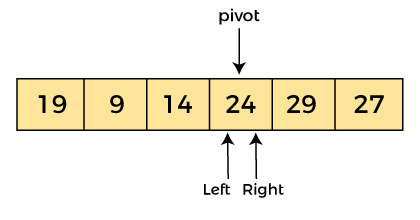
Since, pivot is at left, so algorithm starts from right, and move to left. Now, a[left] = 24, a[right] = 29, and a[pivot] = 24. As a[pivot] < a[right], so algorithm moves one position to left, as -



Now, a[pivot] = 24, a[left] = 24, and a[right] = 14. As a[pivot] > a[right], so, swap a[pivot] and a[right], now pivot is at right, i.e. -



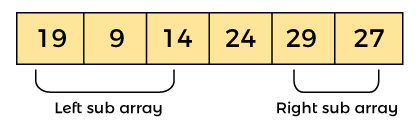
Now, a[pivot] = 24, a[left] = 14, and a[right] = 24. Pivot is at right, so the algorithm starts from left and move to right.



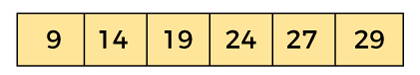
Now, a[pivot] = 24, a[left] = 24, and a[right] = 24. So, pivot, left and right are pointing the same element. It represents the termination of procedure.

Element 24, which is the pivot element is placed at its exact position.

Elements that are right side of element 24 are greater than it, and the elements that are left side of element 24 are smaller than it.



Now, in a similar manner, quick sort algorithm is separately applied to the left and right sub-arrays. After sorting gets done, the array will be -



# **Selection Sort Algorithm:**

In this article, we will discuss the Selection sort Algorithm. The working procedure of selection sort is also simple. This article will be very helpful and interesting to students as they might face selection sort as a question in their examinations. So, it is important to discuss the topic.

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. It is also the simplest algorithm. It is an in-place comparison sorting algorithm. In this algorithm, the array is divided into two parts, first is sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and unsorted part is the given array. Sorted part is placed at the left, while the unsorted part is placed at the right.

In selection sort, the first smallest element is selected from the unsorted array and placed at the first position. After that second smallest element is selected and placed in the second position. The process continues until the array is entirely sorted.

The average and worst-case complexity of selection sort is **O(n2)**, where **n** is the number of items. Due to this, it is not suitable for large data sets.

## **Working of Selection sort Algorithm**

Now, let's see the working of the Selection sort Algorithm.

To understand the working of the Selection sort algorithm, let's take an unsorted array. It will be easier to understand the Selection sort via an example.

Let the elements of array are -

selection Sort Algorithm

Now, for the first position in the sorted array, the entire array is to be scanned sequentially.

ADVERTISEMENT

At present, **12** is stored at the first position, after searching the entire array, it is found that **8** is the smallest value.

selection Sort Algorithm

So, swap 12 with 8. After the first iteration, 8 will appear at the first position in the sorted array.

selection Sort Algorithm

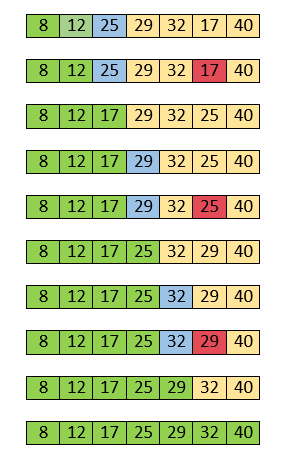
For the second position, where 29 is stored presently, we again sequentially scan the rest of the items of unsorted array. After scanning, we find that 12 is the second lowest element in the array that should be appeared at second position.

selection Sort Algorithm

Now, swap 29 with 12. After the second iteration, 12 will appear at the second position in the sorted array. So, after two iterations, the two smallest values are placed at the beginning in a sorted way.

selection Sort Algorithm

The same process is applied to the rest of the array elements. Now, we are showing a pictorial representation of the entire sorting process.



Now, the array is completely sorted.

# **Heap Sort Algorithm**

In this article, we will discuss the Heapsort Algorithm. Heap sort processes the elements by creating the min-heap or max-heap using the elements of the given array. Min-heap or max-heap represents the ordering of array in which the root element represents the minimum or maximum element of the array.

Heap sort basically recursively performs two main operations -

* Build a heap H, using the elements of array.
* Repeatedly delete the root element of the heap formed in 1st phase.

Before knowing more about the heap sort, let's first see a brief description of **Heap.**

### What is a heap?

A heap is a complete binary tree, and the binary tree is a tree in which the node can have the utmost two children. A complete binary tree is a binary tree in which all the levels except the last level, i.e., leaf node, should be completely filled, and all the nodes should be left-justified.

### What is heap sort?

Heapsort is a popular and efficient sorting algorithm. The concept of heap sort is to eliminate the elements one by one from the heap part of the list, and then insert them into the sorted part of the list.

Heapsort is the in-place sorting algorithm.

Now, let's see the algorithm of heap sort.

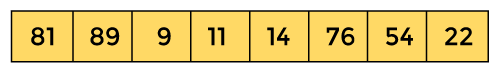
## **Working of Heap sort Algorithm**

Now, let's see the working of the Heapsort Algorithm.

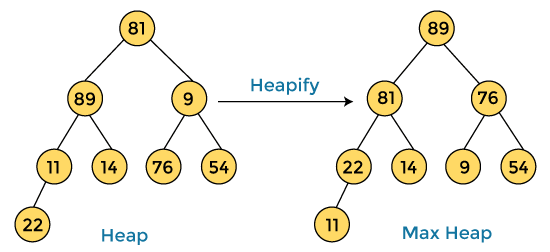
In heap sort, basically, there are two phases involved in the sorting of elements. By using the heap sort algorithm, they are as follows -

* The first step includes the creation of a heap by adjusting the elements of the array.
* After the creation of heap, now remove the root element of the heap repeatedly by shifting it to the end of the array, and then store the heap structure with the remaining elements.

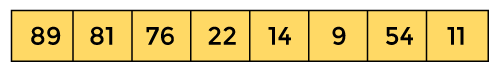
Now let's see the working of heap sort in detail by using an example. To understand it more clearly, let's take an unsorted array and try to sort it using heap sort. It will make the explanation clearer and easier.



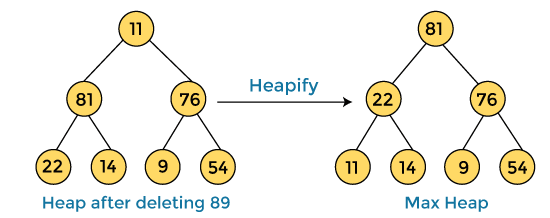
First, we have to construct a heap from the given array and convert it into max heap.



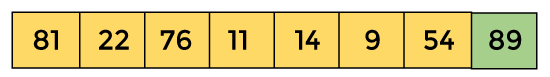
After converting the given heap into max heap, the array elements are -



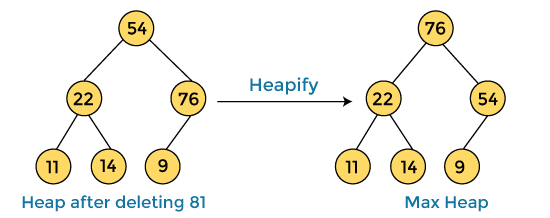
Next, we have to delete the root element **(89)** from the max heap. To delete this node, we have to swap it with the last node, i.e. **(11).** After deleting the root element, we again have to heapify it to convert it into max heap.



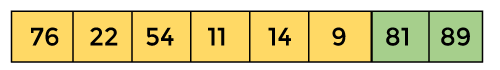
After swapping the array element **89** with **11,** and converting the heap into max-heap, the elements of array are -



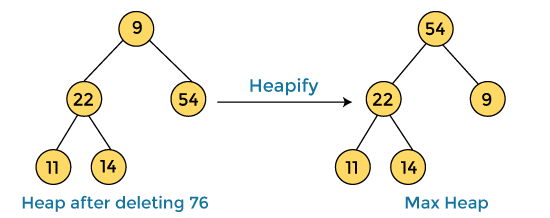
In the next step, again, we have to delete the root element **(81)** from the max heap. To delete this node, we have to swap it with the last node, i.e. **(54).** After deleting the root element, we again have to heapify it to convert it into max heap.



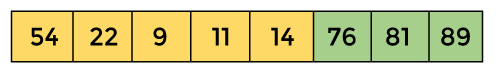
After swapping the array element **81** with **54** and converting the heap into max-heap, the elements of array are -



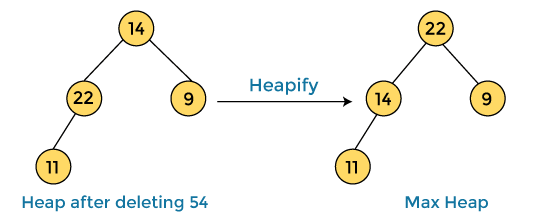
In the next step, we have to delete the root element **(76)** from the max heap again. To delete this node, we have to swap it with the last node, i.e. **(9).** After deleting the root element, we again have to heapify it to convert it into max heap.



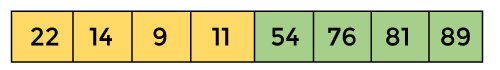
After swapping the array element **76** with **9** and converting the heap into max-heap, the elements of array are -



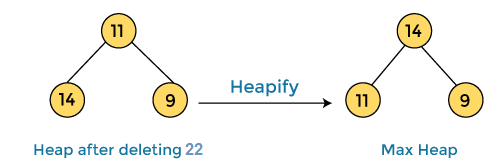
In the next step, again we have to delete the root element **(54)** from the max heap. To delete this node, we have to swap it with the last node, i.e. **(14).** After deleting the root element, we again have to heapify it to convert it into max heap.



After swapping the array element **54** with **14** and converting the heap into max-heap, the elements of array are -



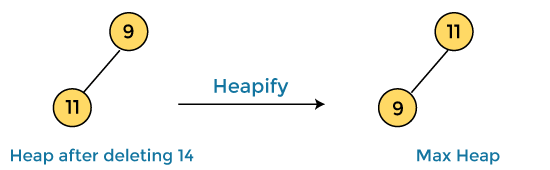
In the next step, again we have to delete the root element **(22)** from the max heap. To delete this node, we have to swap it with the last node, i.e. **(11).** After deleting the root element, we again have to heapify it to convert it into max heap.



After swapping the array element **22** with **11** and converting the heap into max-heap, the elements of array are -



In the next step, again we have to delete the root element **(14)** from the max heap. To delete this node, we have to swap it with the last node, i.e. **(9).** After deleting the root element, we again have to heapify it to convert it into max heap.



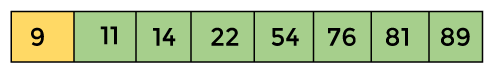
After swapping the array element **14** with **9** and converting the heap into max-heap, the elements of array are -



In the next step, again we have to delete the root element **(11)** from the max heap. To delete this node, we have to swap it with the last node, i.e. **(9).** After deleting the root element, we again have to heapify it to convert it into max heap.



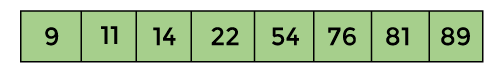
After swapping the array element **11** with **9,** the elements of array are -



Now, heap has only one element left. After deleting it, heap will be empty.



After completion of sorting, the array elements are -



Now, the array is completely sorted.