## **What is Sorting?**

**Sorting** refers to rearrangement of a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure. Sorting means reordering of all the elements either in ascending or in descending order.

## [Sorting Terminology:](https://www.geeksforgeeks.org/sorting-terminology/)

* **In-place Sorting:** An in-place sorting algorithm uses **constant space** for producing the output (modifies the given array only). It sorts the list only by modifying the order of the elements within the list.Examples: Selection Sort, Bubble Sort Insertion Sort and Heap Sort.
* **Internal Sorting:** Internal Sorting is when all the data is placed in the **main memory** or **internal memory**. In internal sorting, the problem cannot take input beyond its size. Example: heap sort, bubble sort, selection sort, quick sort, shell sort, insertion sort.
* **External Sorting :** External Sorting is when all the data that needs to be sorted cannot be placed in memory at a time, the sorting is called external sorting. External Sorting is used for the massive amount of data. Examples: Merge sort, Tag sort, Polyphase sort, Four tape sort, External radix sort, etc.
* **Stable sorting:** When two same data appear in the **same** **order** in sorted data without changing their position is called stable sort. Examples: Merge Sort, Insertion Sort, Bubble Sort.
* **Unstable sorting:** When two same data appear in the **different** **order** in sorted data it is called unstable sort. Examples: Quick Sort, Heap Sort, Shell Sort*.*

## **Characteristics of Sorting Algorithms:**

* **Time Complexity:** Time complexity, a measure of how long it takes to run an algorithm, is used to categorize sorting algorithms. The worst-case, average-case, and best-case performance of a sorting algorithm can be used to quantify the time complexity of the process.
* **Space Complexity:** Sorting algorithms also have space complexity, which is the amount of memory required to execute the algorithm.
* **Stability:** A sorting algorithm is said to be stable if the relative order of equal elements is preserved after sorting. This is important in certain applications where the original order of equal elements must be maintained.
* **In-Place Sorting:** An in-place sorting algorithm is one that does not require additional memory to sort the data. This is important when the available memory is limited or when the data cannot be moved.
* **Adaptivity:** An adaptive sorting algorithm is one that takes advantage of pre-existing order in the data to improve performance.

## [Applications of Sorting Algorithms:](https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-sorting-algorithm/)

* **Searching Algorithms:** Sorting is often a crucial step in search algorithms like binary search, Ternary Search, where the data needs to be sorted before searching for a specific element.
* **Data management:** Sorting data makes it easier to search, retrieve, and analyze.
* **Database optimization:** Sorting data in databases improves query performance.
* **Machine learning:** Sorting is used to prepare data for training machine learning models.
* **Data Analysis:** Sorting helps in identifying patterns, trends, and outliers in datasets. It plays a vital role in statistical analysis, financial modeling, and other data-driven fields.
* **Operating Systems:** Sorting algorithms are used in operating systems for tasks like task scheduling, memory management, and file system organization.

**Bubble Sort** is the simplest [sorting algorithm](https://www.geeksforgeeks.org/sorting-algorithms/) that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

## **Bubble Sort Algorithm (Exchanging/Swapping)**

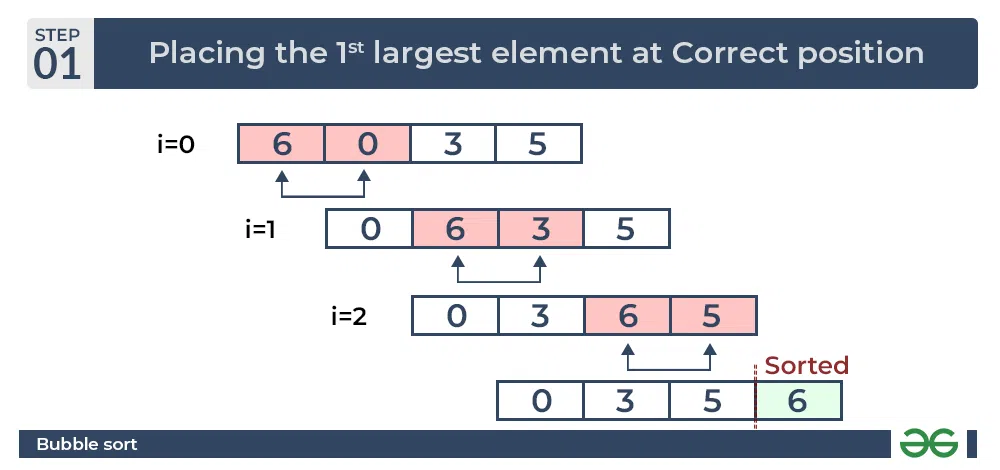
*In Bubble Sort algorithm,*

* *traverse from left and compare adjacent elements and the higher one is placed at right side.*
* *In this way, the largest element is moved to the rightmost end at first.*
* *This process is then continued to find the second largest and place it and so on until the data is sorted.*

## ***How does Bubble Sort Work?***

*Let us understand the working of bubble sort with the help of the following illustration:*

***Input:*** *arr[] = {6, 0, 3, 5}*

**

# **QuickSort – Data Structure and Algorithm Tutorials (Divide and Conquer)**

***QuickSort*** *is a sorting algorithm based on the* [*Divide and Conquer algorithm*](https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/) *that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.*

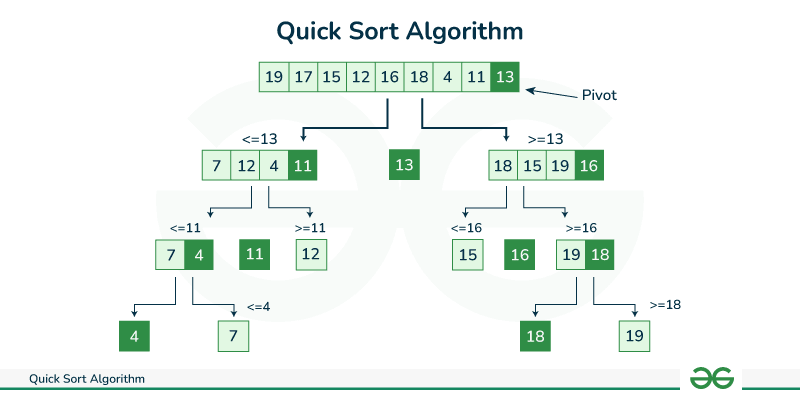
## **How does QuickSort work?**

*The key process in* ***quickSort*** *is a* ***partition()****. The target of partitions is to place the pivot (any element can be chosen to be a pivot) at its correct position in the sorted array and put all smaller elements to the left of the pivot, and all greater elements to the right of the pivot.*

*Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.*

**

Example2:



# **Merge Sort – Data Structure and Algorithms Tutorials (Divide and Conquer)**

**Merge sort** is a sorting algorithm that follows the **divide-and-conquer** approach. It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array.

In simple terms, we can say that the process of **merge sort** is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

## **How does Merge Sort work?**

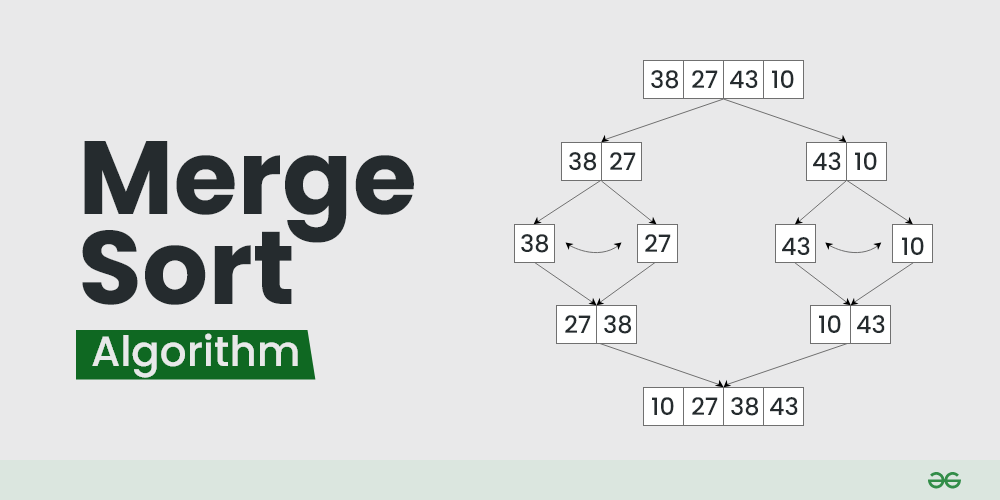
Merge sort is a popular sorting algorithm known for its efficiency and stability. It follows the **divide-and-conquer** approach to sort a given array of elements.

Here’s a step-by-step explanation of how merge sort works:

1. **Divide:** Divide the list or array recursively into two halves until it can no more be divided.
2. **Conquer:** Each subarray is sorted individually using the merge sort algorithm.
3. **Merge:** The sorted subarrays are merged back together in sorted order. The process continues until all elements from both subarrays have been merged.

### **Illustration of Merge Sort:**

Let’s sort the array or list **[38, 27, 43, 10]** using Merge Sort



*Let’s look at the working of above example:*

***Divide:***

* ***[38, 27, 43, 10]*** *is divided into* ***[38, 27****] and* ***[43, 10]****.*
* ***[38, 27]*** *is divided into* ***[38]*** *and* ***[27]****.*
* ***[43, 10]*** *is divided into* ***[43]*** *and* ***[10]****.*

***Conquer:***

* ***[38]*** *is already sorted.*
* ***[27]*** *is already sorted.*
* ***[43]*** *is already sorted.*
* ***[10]*** *is already sorted.*

***Merge:***

* *Merge* ***[38]*** *and* ***[27]*** *to get* ***[27, 38]****.*
* *Merge* ***[43]*** *and* ***[10]*** *to get* ***[10,43]****.*
* *Merge* ***[27, 38]*** *and* ***[10,43]*** *to get the final sorted list* ***[10, 27, 38, 43]***

*Therefore, the sorted list is* ***[10, 27, 38, 43]****.*

# **Heap Sort – Data Structures and Algorithms Tutorials**

***Heap sort*** *is a comparison-based sorting technique based on* [*Binary Heap*](http://www.geeksforgeeks.org/binary-heap/) *data structure. It is similar to the* [*selection sort*](http://www.geeksforgeeks.org/selection-sort/) *where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.*

## **Heap Sort Algorithm**

To solve the problem follow the below idea:

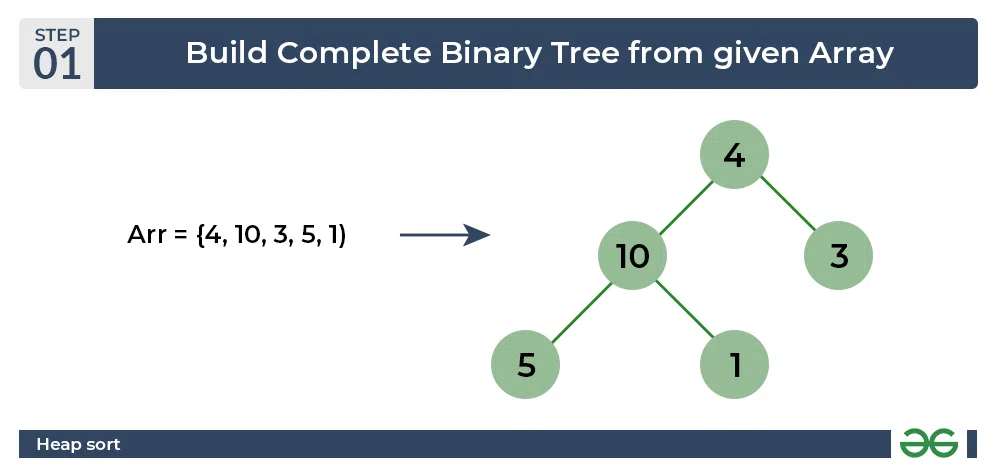
*First convert the array into heap data structure using heapify, then one by one delete the root node of the Max-heap and replace it with the last node in the heap and then heapify the root of the heap. Repeat this process until size of heap is greater than 1.*

* *Build a heap from the given input array.*
* *Repeat the following steps until the heap contains only one element:*
  + *Swap the root element of the heap (which is the largest element) with the last element of the heap.*
  + *Remove the last element of the heap (which is now in the correct position).*
  + *Heapify the remaining elements of the heap.*
* *The sorted array is obtained by reversing the order of the elements in the input array.*

## **Detailed Working of Heap Sort**

*To understand heap sort more clearly, let’s take an unsorted array and try to sort it using heap sort.  
Consider the array: arr[] = {4, 10, 3, 5, 1}.*

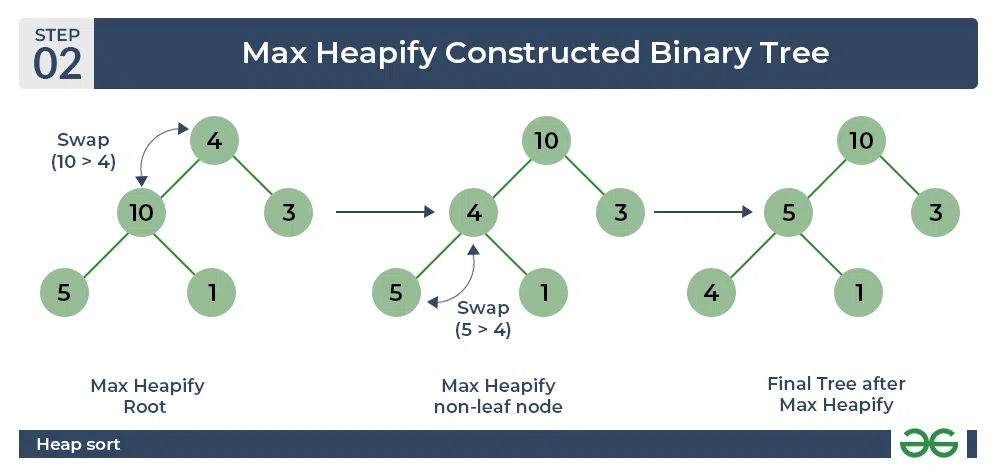
***Build Complete Binary Tree:*** *Build a complete binary tree from the array.*

**

*Heap sort algorithm | Build Complete Binary Tree*

***Transform into max heap:*** *After that, the task is to construct a tree from that unsorted array and try to convert it into* [*max heap.*](https://www.geeksforgeeks.org/difference-between-min-heap-and-max-heap/)

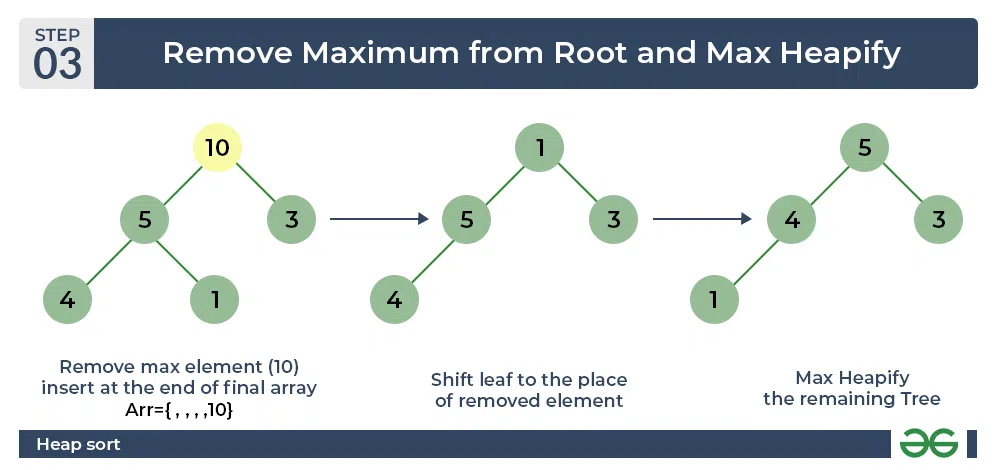
* *To transform a heap into a max-heap, the parent node should always be greater than or equal to the child nodes*
  + *Here, in this example, as the parent node* ***4*** *is smaller than the child node* ***10,*** *thus, swap them to build a max-heap.*
* *Now,* ***4*** *as a parent is smaller than the child* ***5****, thus swap both of these again and the resulted heap and array should be like this:*

**

*Heap sort algorithm | Max Heapify constructed binary tree*

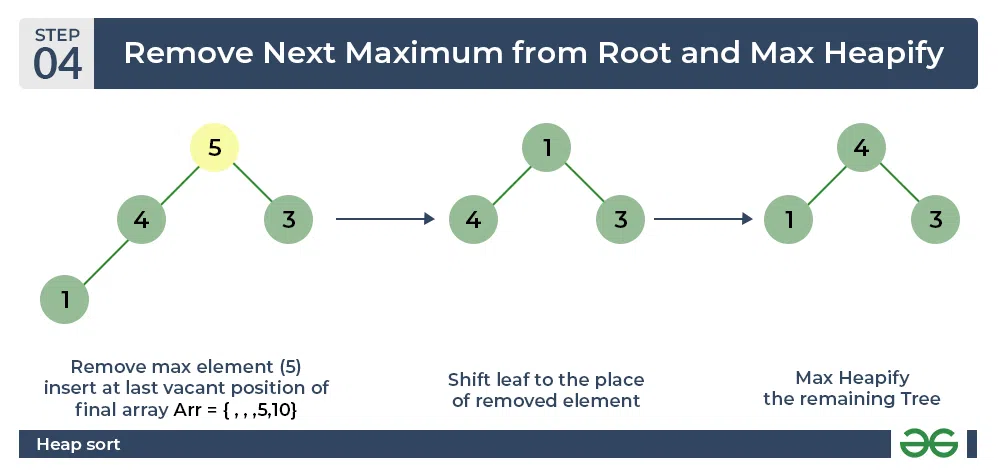
***Perform heap sort:*** *Remove the maximum element in each step (i.e., move it to the end position and remove that) and then consider the remaining elements and transform it into a max heap.*

* *Delete the root element* ***(10)*** *from the max heap. In order to delete this node, try to swap it with the last node, i.e.* ***(1).*** *After removing the root element, again heapify it to convert it into max heap.*
  + *Resulted heap and array should look like this:*

**

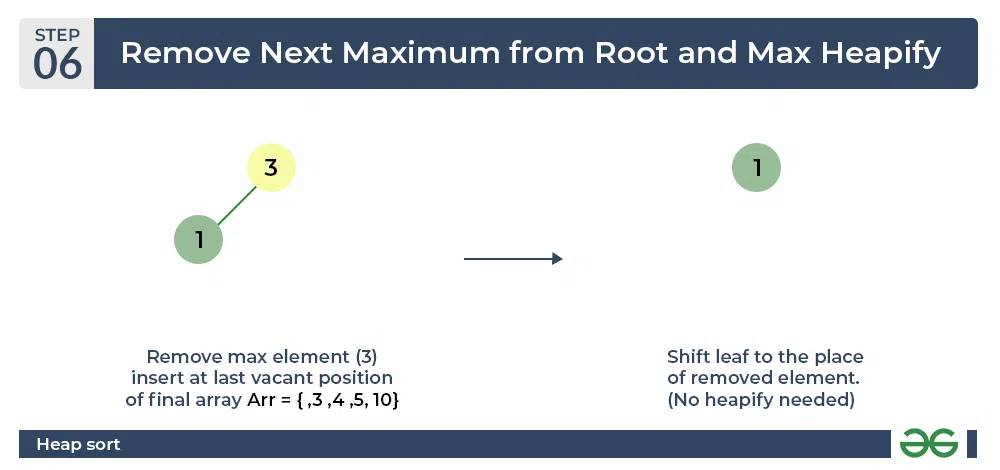
*Heap sort algorithm | Remove maximum from root and max heapify*

* *Repeat the above steps and it will look like the following:*

**

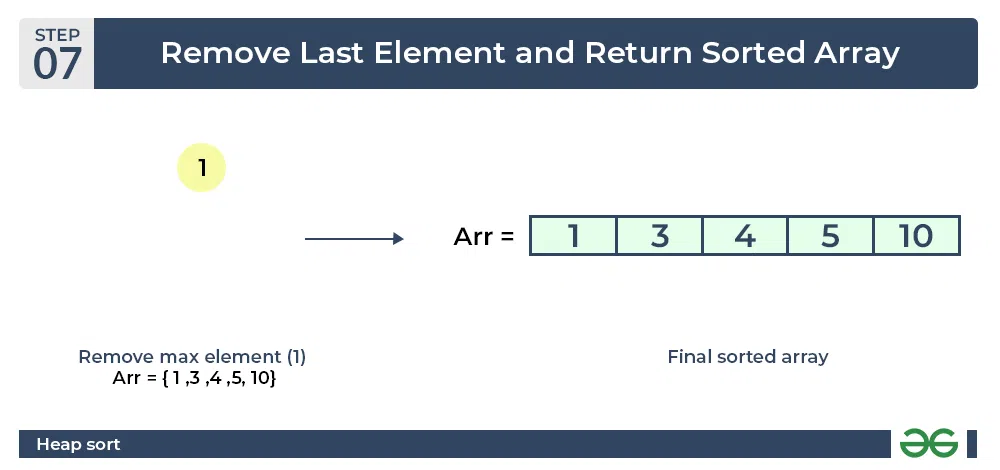
*Heap sort algorithm | Remove next maximum from root nad max heapify*

* *Now remove the root (i.e. 3) again and perform heapify.*

**

*Heap sort algorithm | Repeat previous step*

* *Now when the root is removed once again it is sorted. and the sorted array will be like* ***arr[] = {1, 3, 4, 5, 10}****.*

**

*Heap sort algorithm | Final sorted array*