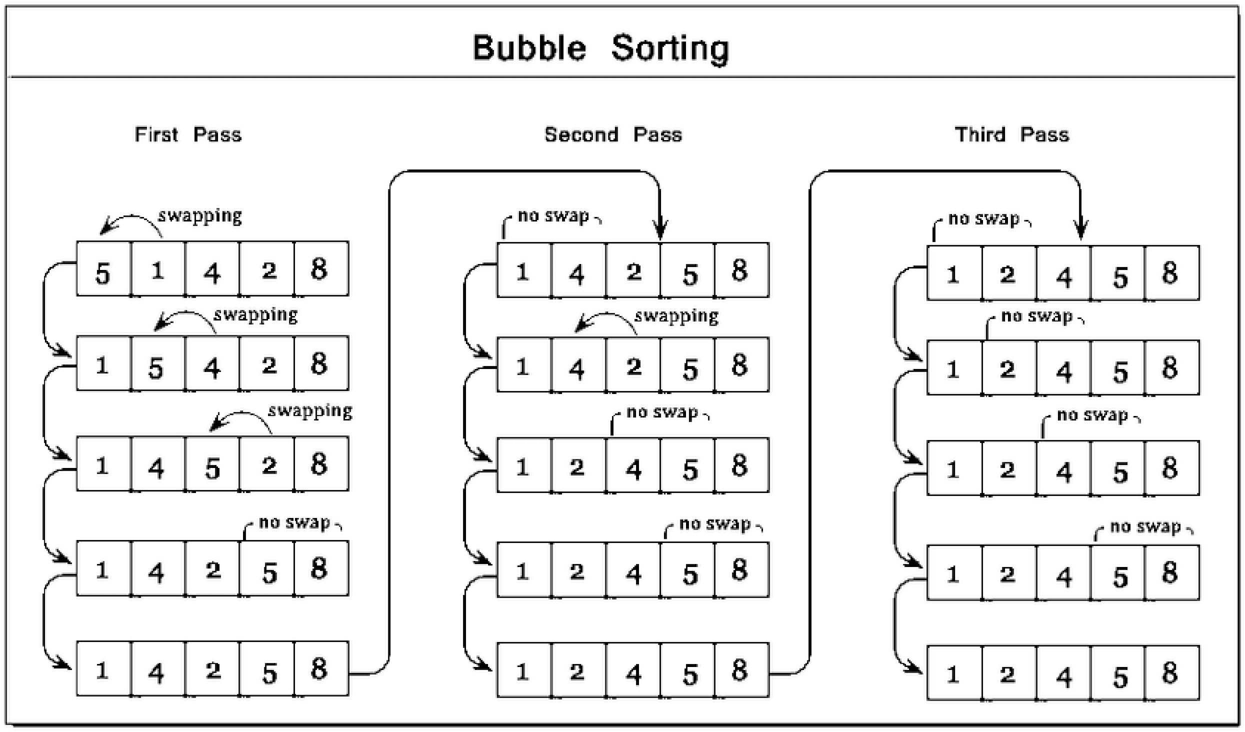
بِسْمِ ٱللَّٰهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Bubble Sort Algorithm

Bubble Sort is the simplest sorting algorithm 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.

How Bubble Sort Works?



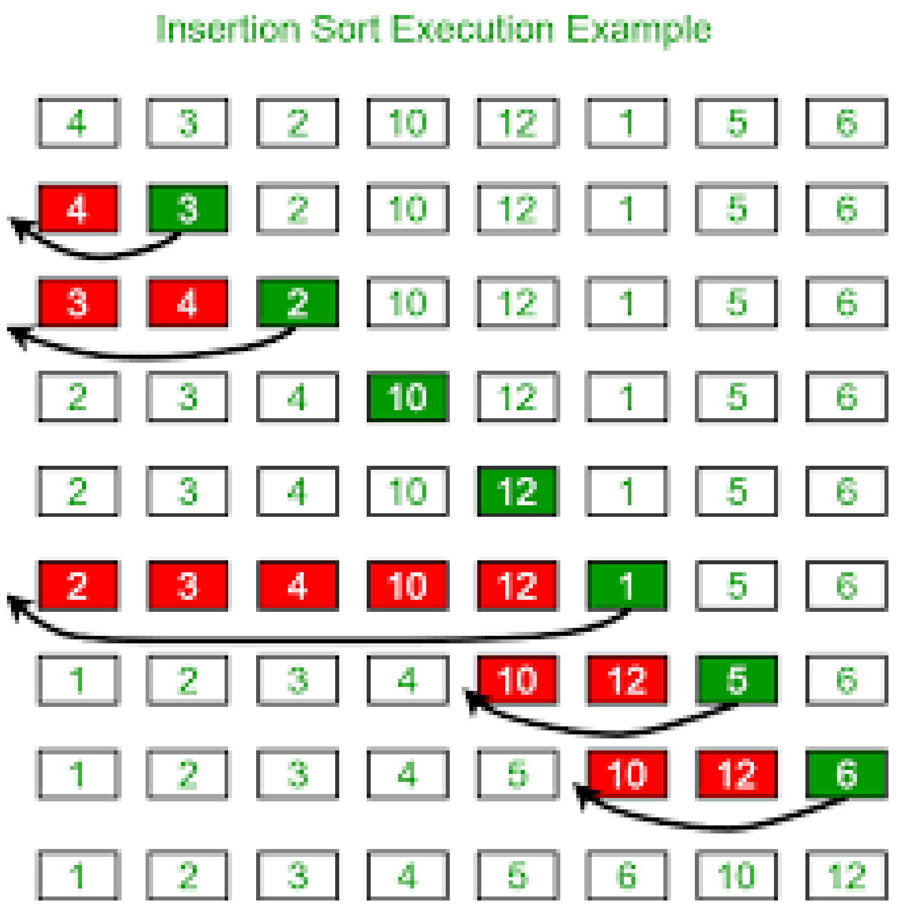
Insertion Sort Algorithm

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Characteristics of Insertion Sort:

- **This algorithm is one of the simplest algorithm with simple implementation**
- **Basically, Insertion sort is efficient for small data values**
- **Insertion sort is adaptive in nature, i.e. it is appropriate for data sets which are already partially sorted.**

Working of Insertion Sort algorithm:



Selection Sort Algorithm

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

- **The subarray which is already sorted.**
- **Remaining subarray which is unsorted.**

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

How selection sort works?

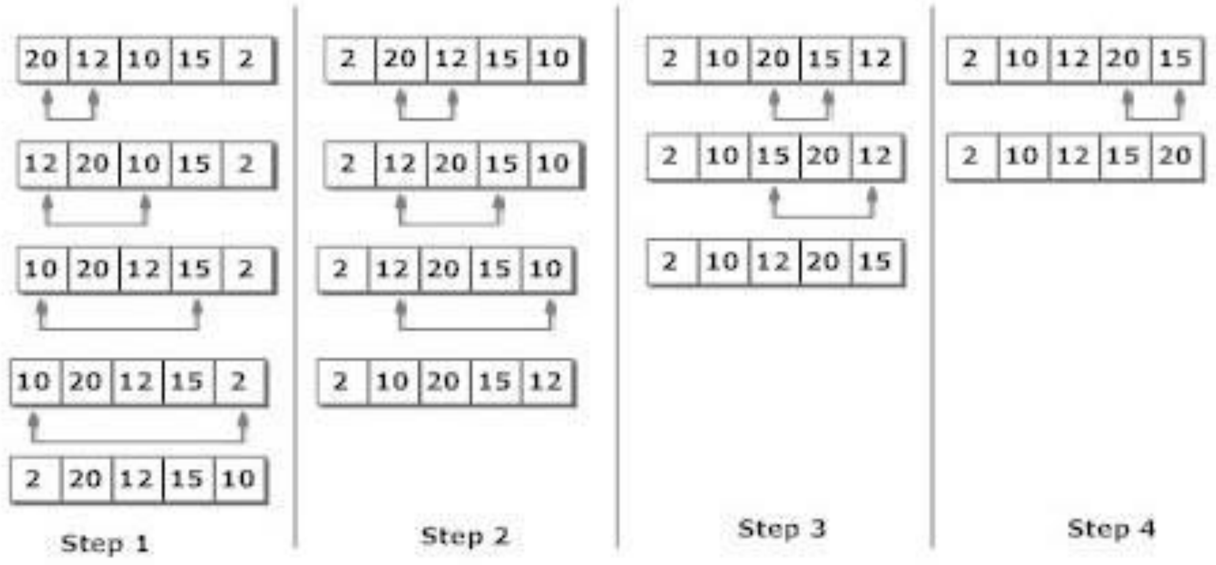
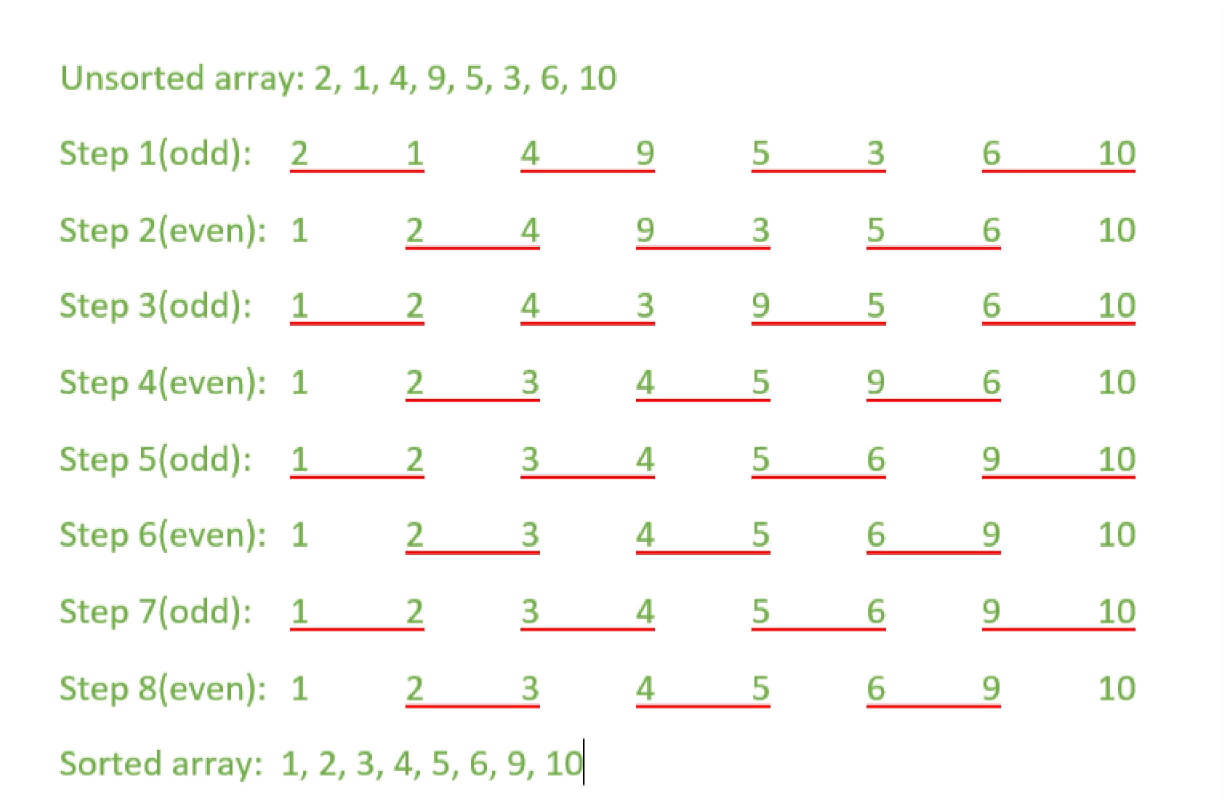


Figure: Selection Sort

Odd-Even Sort / Brick Sort

This is basically a variation of bubble sort . This algorithm is divided into two phases- Odd and Even Phase. The algorithm runs until the array elements are sorted and in each iteration two phases occurs- Odd and Even Phases.
In the odd phase, we perform a bubble sort on odd indexed elements and in the even phase, we perform a bubble sort on even indexed elements.



Unsorted array: 2, 1, 4, 9, 5, 3, 6, 10

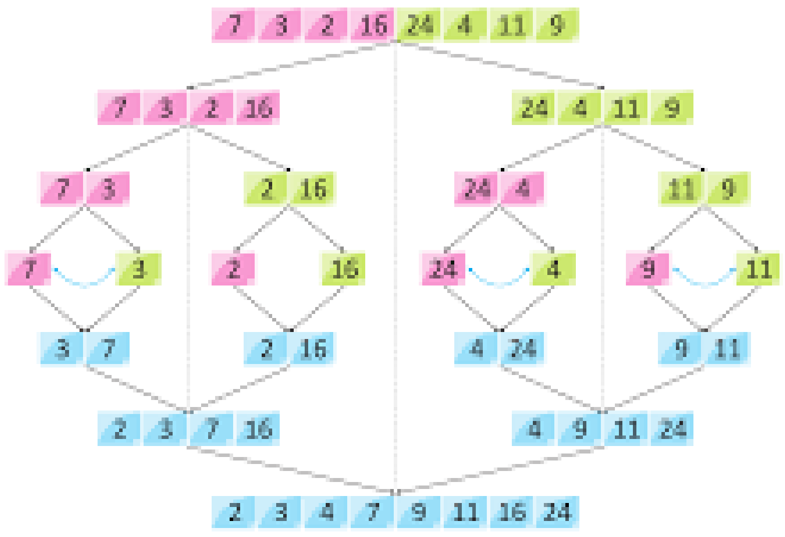
Step 1(odd):	$\underline{2 \quad 1}$		$\underline{4 \quad 9}$		$\underline{5 \quad 3}$		$\underline{6 \quad 10}$	
Step 2(even):	1	$\underline{2 \quad 4}$		$\underline{9 \quad 3}$		$\underline{5 \quad 6}$		10
Step 3(odd):	$\underline{1 \quad 2}$		$\underline{4 \quad 3}$		$\underline{9 \quad 5}$		$\underline{6 \quad 10}$	
Step 4(even):	1	$\underline{2 \quad 3}$		$\underline{4 \quad 5}$		$\underline{9 \quad 6}$		10
Step 5(odd):	$\underline{1 \quad 2}$		$\underline{3 \quad 4}$		$\underline{5 \quad 6}$		$\underline{9 \quad 10}$	
Step 6(even):	1	$\underline{2 \quad 3}$		$\underline{4 \quad 5}$		$\underline{6 \quad 9}$		10
Step 7(odd):	$\underline{1 \quad 2}$		$\underline{3 \quad 4}$		$\underline{5 \quad 6}$		$\underline{9 \quad 10}$	
Step 8(even):	1	$\underline{2 \quad 3}$		$\underline{4 \quad 5}$		$\underline{6 \quad 9}$		10

Sorted array: 1, 2, 3, 4, 5, 6, 9, 10

Merge Sort

The Merge Sort algorithm is a sorting algorithm that is considered an example of the divide and conquer strategy. So, in this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner. We can think of it as a recursive algorithm that continuously splits the array in half until it cannot be further divided. This means that if the array becomes empty or has only one element left, the dividing will stop, i.e. it is the base case to stop the recursion. If the array has multiple elements, we split the array into halves and recursively invoke the merge sort on each of the halves. Finally, when both the halves are sorted, the merge operation is applied. Merge operation is the process of taking two smaller sorted arrays and combining them to eventually make a larger one.

- Declare left variable to 0 and right variable to n-1
- Find mid by medium formula. mid = (left+right)/2
- Call merge sort on (left,mid)
- Call merge sort on (mid+1,rear)
- Continue till left is less than right
- Then call merge function to perform merge sort.



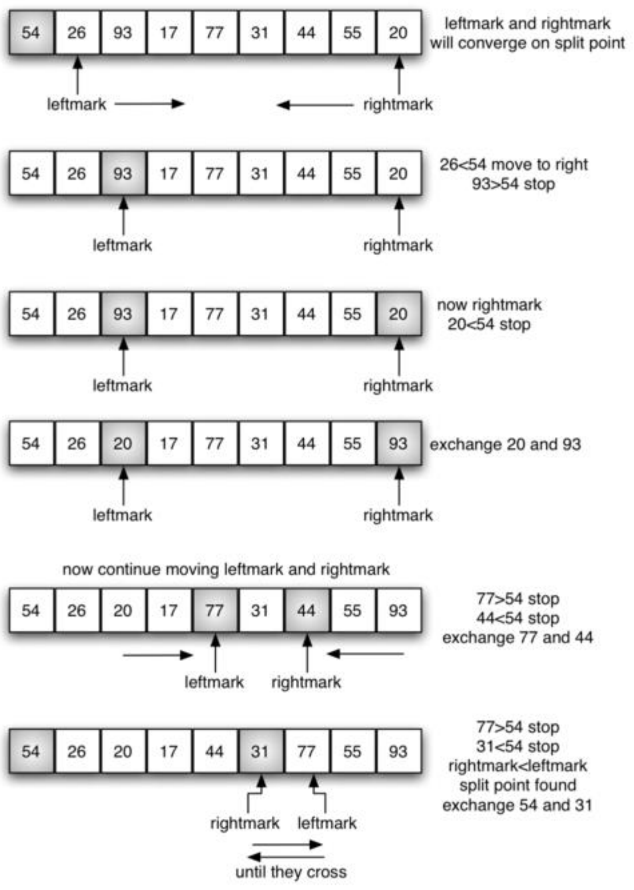
Quick Sort

Like *Merge Sort*, **Quick Sort** is a *Divide and Conquer algorithm*. It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of **quick Sort** that pick pivot in different ways.

- Always pick the first element as a pivot.
- Always pick the last element as a pivot (implemented below)
- Pick a random element as a pivot.
- Pick median as the pivot.

The key process in **quickSort** is a **partition().** The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Partition Algorithm:
There can be many ways to do partition, following pseudo-code adopts the method given in the CLRS book. The logic is simple, we start from the leftmost element and keep track of the index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap the current element with arr[i]. Otherwise, we ignore the current element.



Created By Mohammed ElIdrissi Laoukili

لا اله الا الله