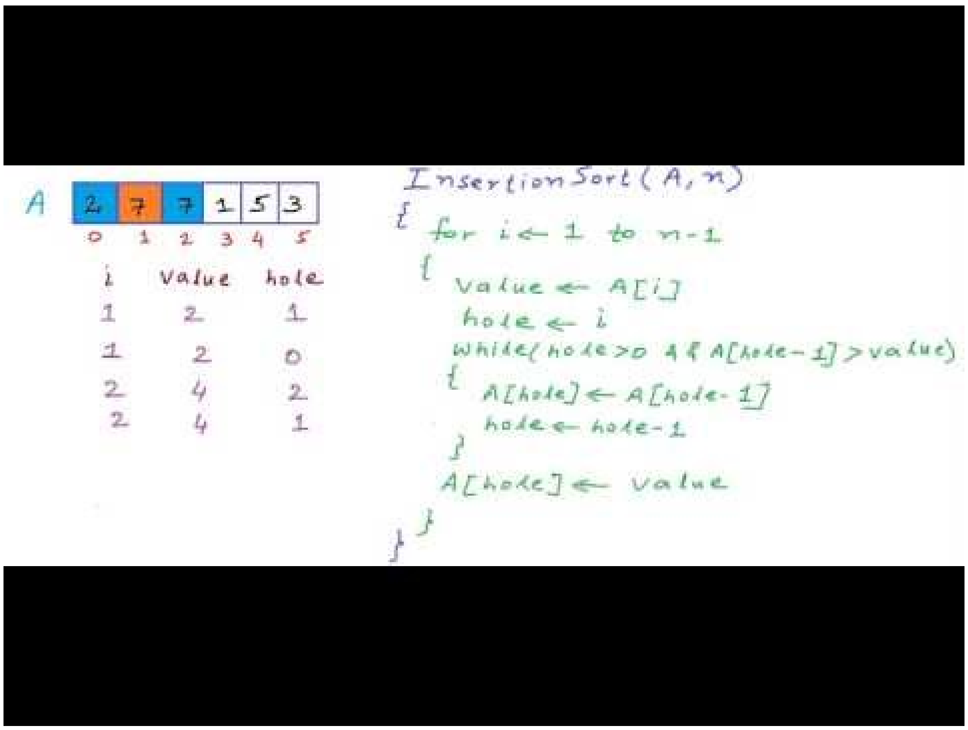
Insertion sort

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Algorithm:

```
// Sort an arr[] of size n
insertionSort(arr, n)
Loop from i = 1 to n-1
    a) Pick element arr[i] and insert it into sorted sequence arr[0…i-1]
```



- *Time Complexity:* O(n^2)
- *Auxiliary Space:* O(1)
- *Boundary Cases:* Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.
- *Sorting In Place:* Yes
- *Stable:* Yes
- *Online:* Yes

Algorithmic Paradigm: Incremental Approach

An **online algorithm** is one that can process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start.

Uses: Insertion sort is used when number of elements is small. It can also be useful when input array is almost sorted, only few elements are misplaced in complete big array.

Next

- What is Binary Insertion Sort?
- How to implement Insertion Sort for Linked List?

Concepts Problems

Given an array arr[], a pair arr[i] and arr[j] forms an inversion if arr[i] < arr[j] and i > j. For example, the array {1, 3, 2, 5} has one inversion (3, 2) and array {5, 4, 3} has inversions (5, 4), (5, 3) and (4, 3).

If we take a closer look at the insertion sort code, we can notice that every iteration of while loop reduces one inversion. The while loop executes only if i > j and arr[i] < arr[j]. Therefore total number of while loop iterations (For all values of i) is same as number of inversions. Therefore overall time complexity of the insertion sort is O(n + f(n)) where f(n) is inversion count. If the inversion count is O(n), then the time complexity of insertion sort is O(n). In worst case, there can be n*(n-1)/2 inversions. The worst case occurs when the array is sorted in reverse order. So the worst case time complexity of insertion sort is $O(n^2)$.

Merge sort can also be used to count inversion

Insertion sort is faster for small input size (n) because *Quick Sort* has extra overhead from the recursive function calls. Insertion sort is also more stable than Quick sort and requires less memory.