Most of the algorithms that we've been dealing with have been pretty slow, and seem inefficient. However, they tend to come up a lot in computer science courses and theoretical explanations because they are often used as the naive approach, or the simplest implementation of sorting a collection.

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.

Insertion Algorithms: Steps on how it works:

1. If it is the first element, it is already sorted.

2. Pick the next element.

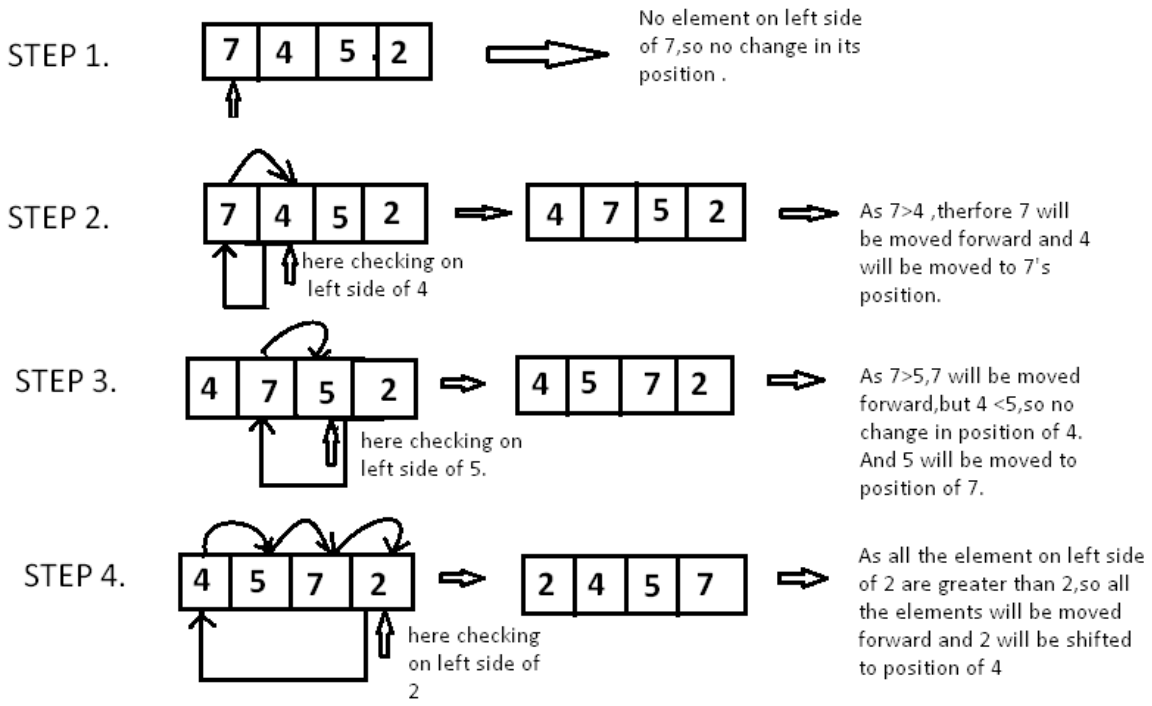
3. Compare with all the elements in sorted sub-list.

4. Shift all the elements in sorted sub-list that is greater than the value to be sorted, one location on right side.

5. Insert the value.

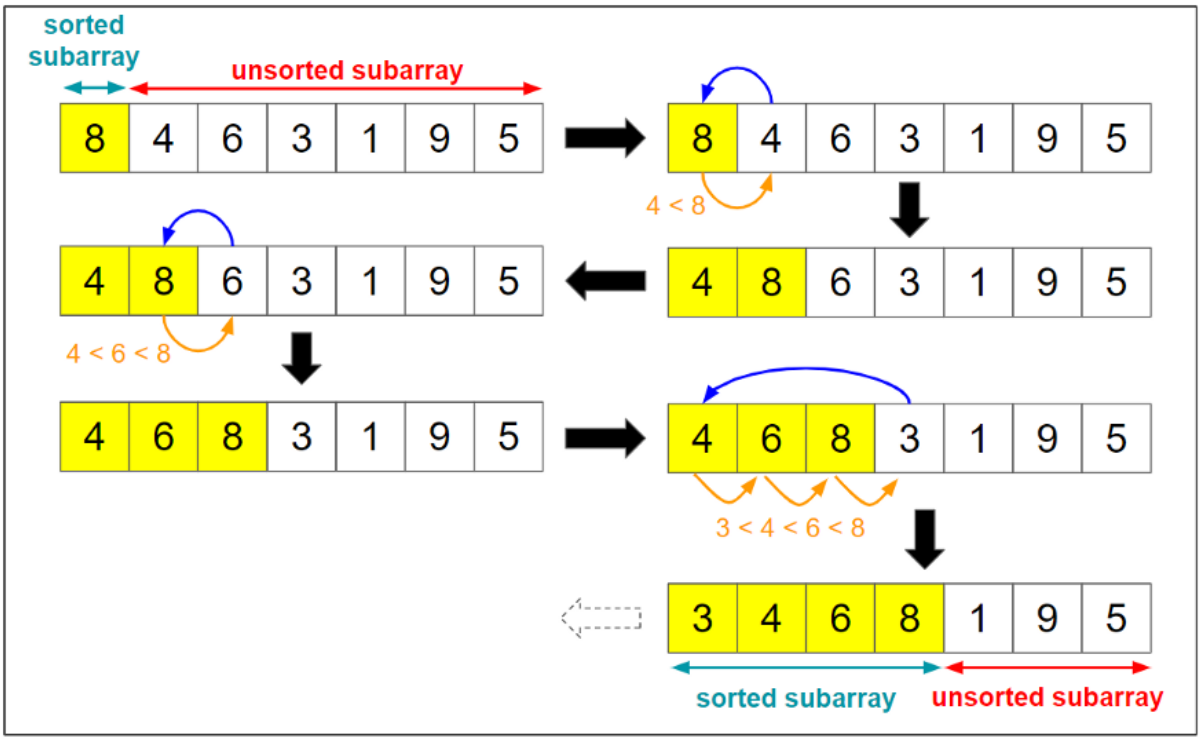
6. Repeat until list is sorted.

Below is an array of 4 numbers, which need to be sorted. We will use Insertion Sort Algorithm, to sort this array:



Since 7 is the first element and has no other element to be compared with, it remains at its position. Now when on moving towards 4, 7 is the largest element in the sorted list and greater than 4. So, move 4 to its correct position, which is before 7. Similarly with 5, as 7 (largest element in the sorted list) is greater than 5, move 5 to its correct position. Finally for 2, all the

elements on the left side of 2 (sorted list) are moved one position forward as all are greater than 2 and then 2 is placed in the first position. Finally, the given array will result in a sorted array.



Algorithm

insertionSort(array)
mark first element as sorted
for each unsorted element X
'extract' the element X
for j < - lastSortedIndex down to 0
if current element j > X
move sorted element to the right by 1
break loop and insert X here
end insertionSort

Characteristics of Insertion Sort

1. It is efficient for smaller data sets, but very inefficient for larger lists.

2. Insertion Sort is adaptive, that means it reduces its total number of steps if given a partially sorted list, hence it increases its efficiency.

3. Its space complexity is less. Insertion sort requires a single additional memory space.

4. Overall time complexity of Insertion sort is O(n2).

Implementation in Python

```
# Insertion sort in Python


def insertionSort(array):

  for step in range(1, len(array)):
    key = array[step]
    j = step - 1

    # Compare key with each element on the left of it until an element smaller than it is found
    # For descending order, change key<array[j] to key>array[j].
    while j >= 0 and key < array[j]:
      array[j + 1] = array[j]
      j = j - 1

    # Place key at after the element just smaller than it.
    array[j + 1] = key


data = [9, 5, 1, 4, 3]
insertionSort(data)
print('Sorted Array in Ascending Order:')
print(data)
```