Insertion Sort

Algorithm:

Insertion Sort works similar to how we sort a hand of playing cards.

Imagine that you are playing a card game. You're holding the cards in your hand, and these cards are sorted. The dealer hands you exactly one new card. You have to put it into the correct place so that the cards you're holding are still sorted. In selection sort, each element that you add to the sorted subarray is no smaller than the elements already in the sorted subarray.

But in our card example, the new card could be smaller than some of the cards you're already holding, and so you go down the line, comparing the new card against each card in your hand until you find the place to put it. You insert the new card in the right place, and once again, your hand holds fully sorted cards. Then the dealer gives you another card, and you repeat the same procedure. Then another card, and another card, and so on, until the dealer stops giving you cards. This is the idea behind the **insertion sort**. Loop over positions in the array, starting with index 1. Each new position is like the new card handed to you by the dealer, and you need to insert it into the correct place in the sorted subarray to the left of that position.

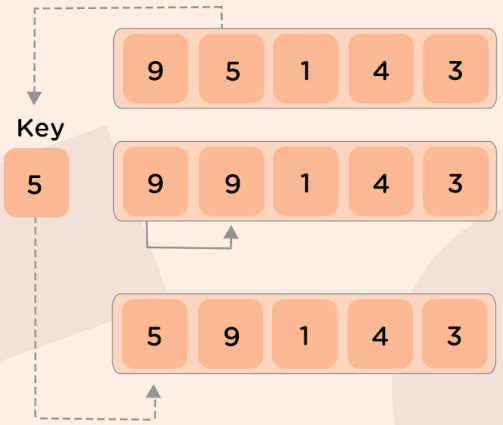
Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.

Consider the following depicted array as an example. You want to sort this array in increasing order.

Arr[] = [9,4,5,1,3]

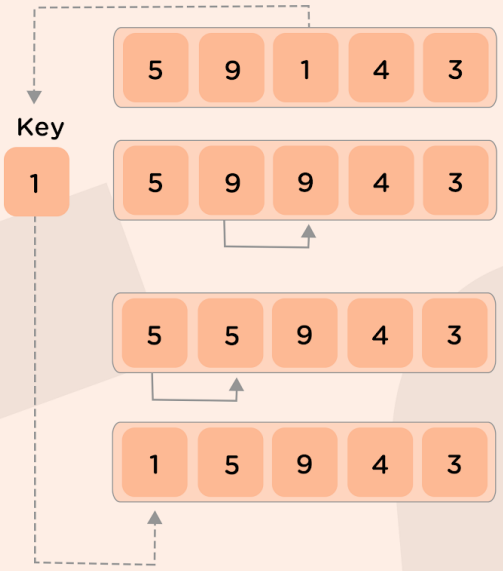
Following is the pictorial diagram for a better explanation of how it works:

STEP- 1

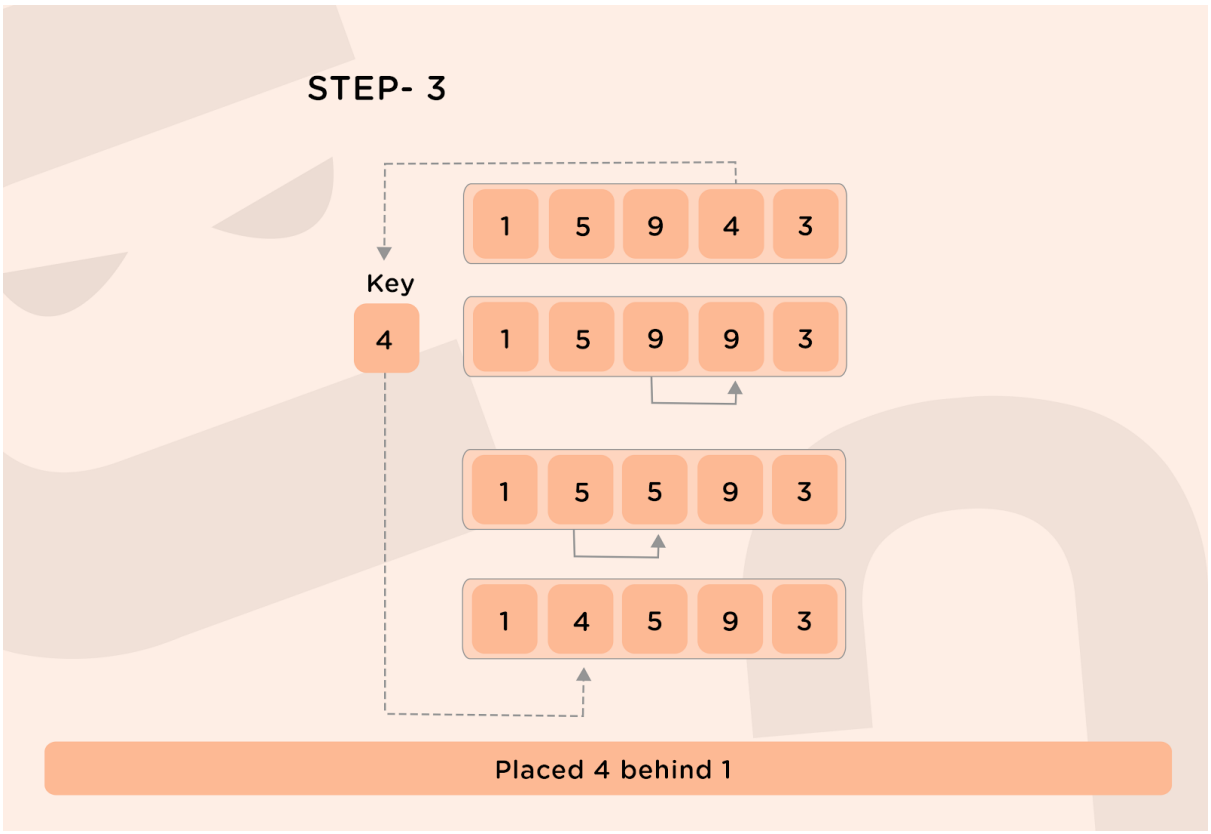


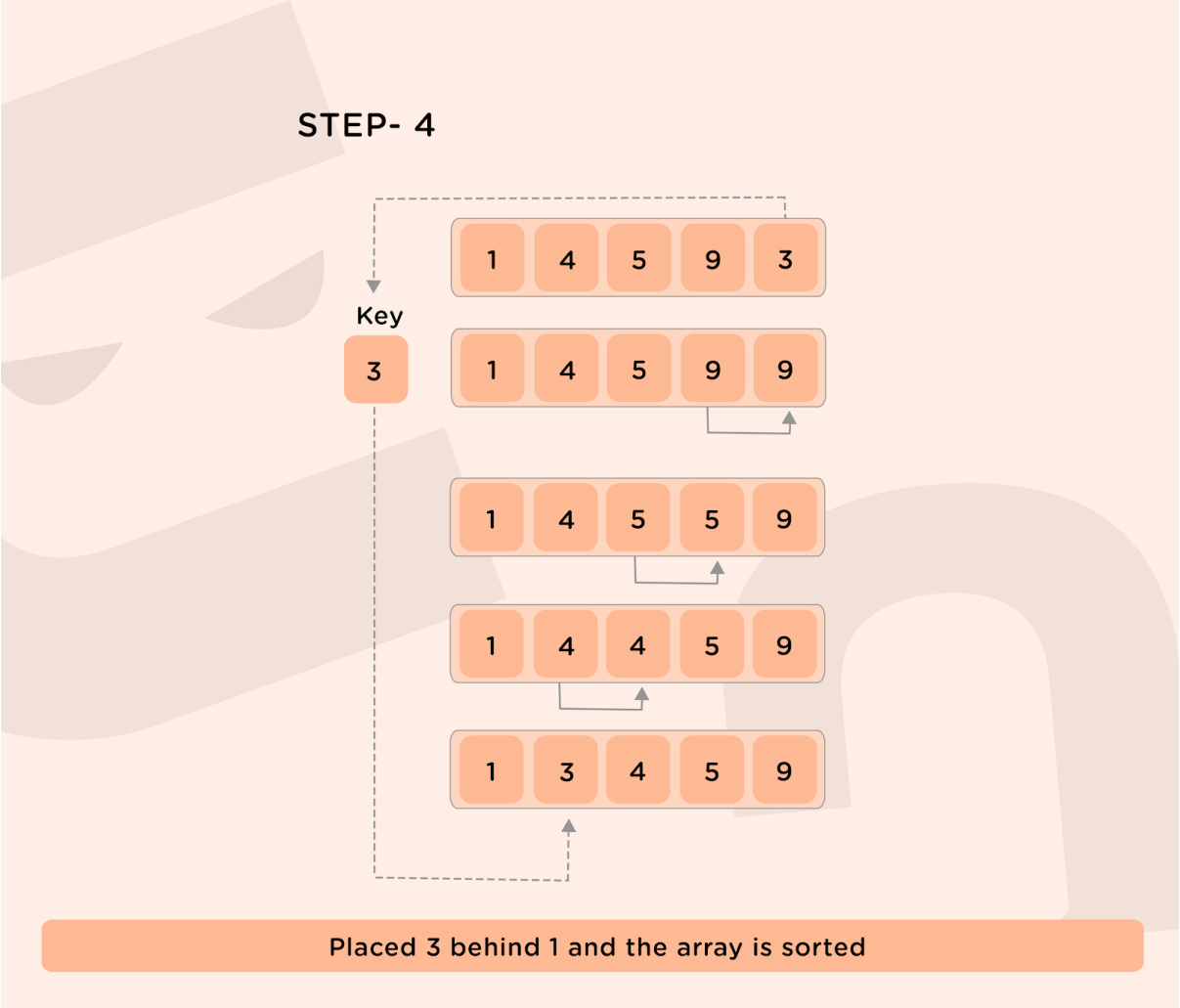
If the first element is greater than key, then key is placed in front of the element.

STEP- 2



Placed 1 at the beginning


STEP- 3
Key
4
1 5 9 4 3
1 5 9 9 3
1 5 5 9 3
1 4 5 9 3
Placed 4 behind 1



Pseudocode:

```
/*
      array of size N from 0 to N-1 is considered
*/
function insertionSort(arr, N)

        for idx = 1 to N-1
                cur = arr[idx]
                //   Finding the appropriate position for the element from 0 to idx-1.
                jdx = idx-1
                while arr[jdx] > cur and jdx >=0
                        //   Creating space for the element at idx
                        arr[jdx+1] = arr[jdx]
                        jdx -= 1
```

```
//   Placing the element at idx, making the array sorted from 0 to idx
arr[jdx+1] = cur
```

Time complexity: O(N^2), in the worst case.

As for finding the correct position for the ith element, we need i iterations from 0 to i-1th index, so the total number of comparisons = 1+2+3+ … + N-1 = (N*(N-1))/2 and the total number of exchanges in the worst case: N-1.

So, the overall complexity becomes O(N^2).

Space complexity: O(1), as no extra space is required.