Insertion Sort Algorithm

Insertion sort works similarly as we sort cards in our hand in a card game.

We assume that the first card is already sorted then, we select an unsorted card. If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left. In the same way, other unsorted cards are taken and put at their right place.

A similar approach is used by insertion sort.

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

How Insertion Sort Works?

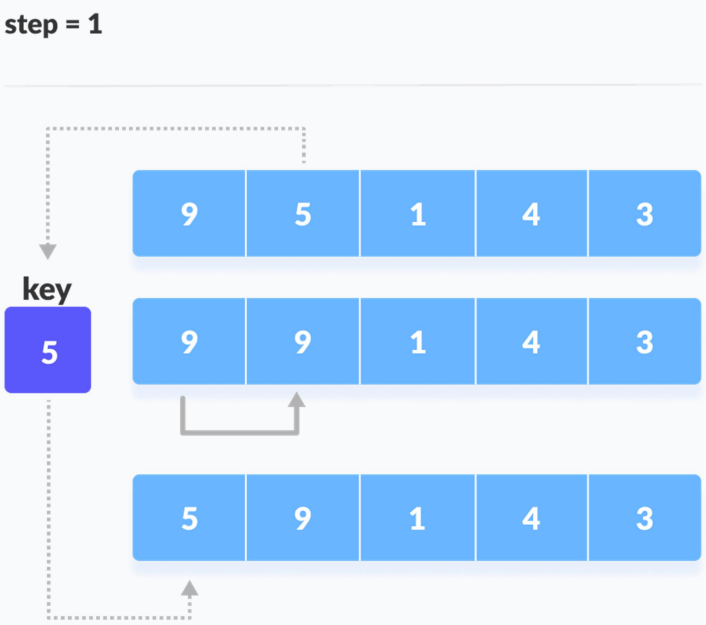
Suppose we need to sort the following array.

Initial array

1. The first element in the array is assumed to be sorted. Take the second element and store it separately in `key`.

   Compare `key` with the first element. If the first element is greater than `key`,

then `key` is placed in front of the first element.

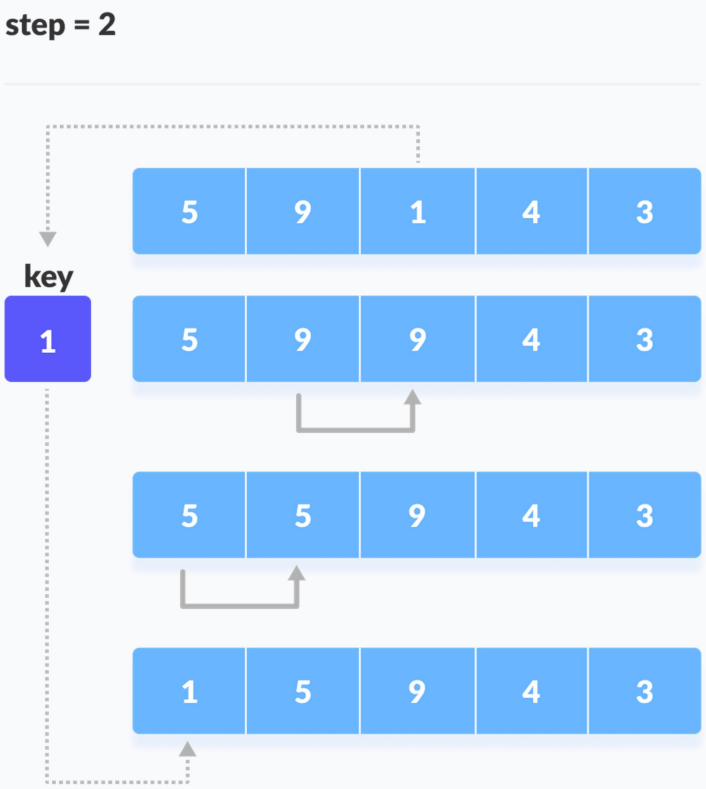


If the first element is greater than key, then key is placed in front of the first element.

2. Now, the first two elements are sorted.

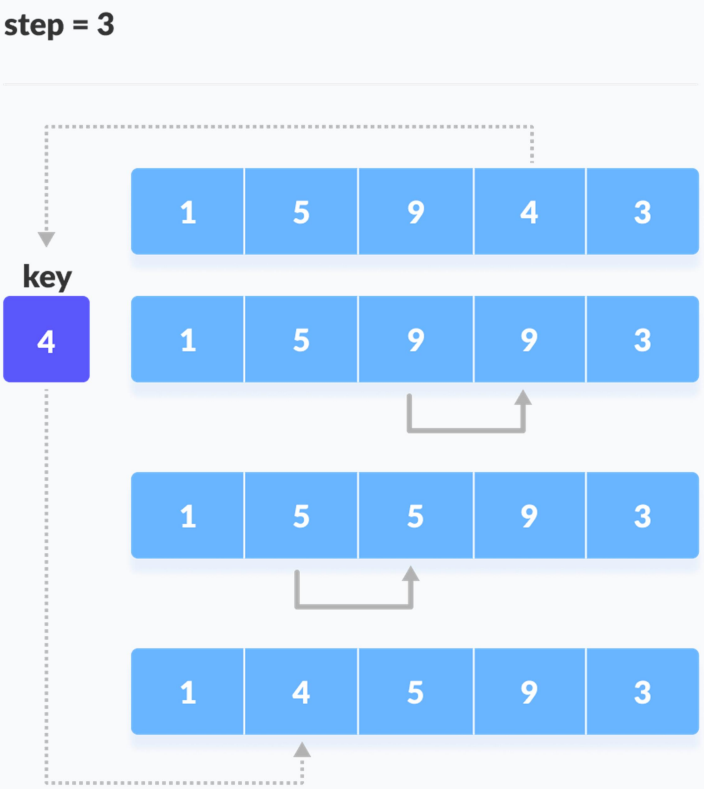
   Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller

than it, then place it at the beginning of the array.

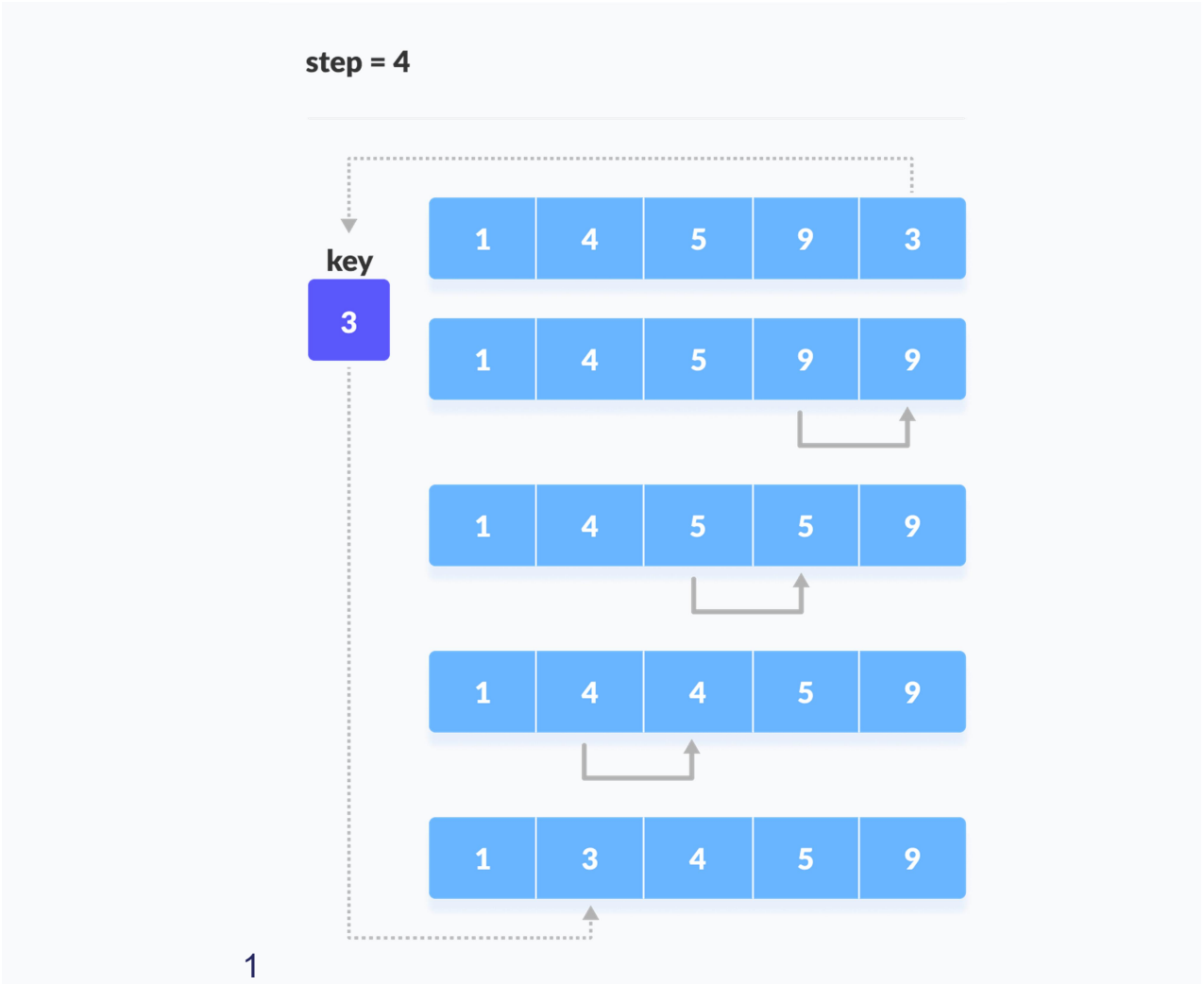


Place 1 at the beginning

3. Similarly, place every unsorted element at its correct position.



Place 4 behind



Place 3 behind 1 and the array is sorted

Insertion Sort Algorithm

```
InsertionSort(A,n)

  for j=2 to A.length

   {

     Key = A[j]

     i = j-1

     while ( i>0 && A[i]>key )

     {

      A[i+1] = A[i]

      i = i-1

      }

    A[i+1] =  key

   }
```

Complexity

Time Complexities

- **Worst Case Complexity:** $O(n^2)$

  Suppose, an array is in ascending order, and you want to sort it in descending order. In this case, worst case complexity occurs.

  Each element has to be compared with each of the other elements so, for every nth element, $(n-1)$ number of comparisons are made.

  Thus, the total number of comparisons = $n*(n-1) \sim n^2$

- **Best Case Complexity:** `O(n)`
  When the array is already sorted, the outer loop runs for `n` number of times whereas the inner loop does not run at all. So, there are only `n` number of comparisons. Thus, complexity is linear.
- **Average Case Complexity:** $O(n^2)$
  It occurs when the elements of an array are in jumbled order (neither ascending nor descending).

Space Complexity

Space complexity is `O(1)` because an extra variable `key` is used.

QUESTION-1 Illustrate the operation of Bubble sort on the array

A = {5,2,1,4,3,7,6}

QUESTION-2 Sort the following array using selection sort : A = {5,2,1,4,3}

QUICK SORT

```
Partition(a,p,q)
{
   x=a[p];
   i=p;
   for( j=i+1 ; j<=q; i++ )
   {
     If (a[j] <= x)
     {
       i=i+1;
       swap(a[i],a[j]);
     }
   }
  Swap(a[i],a[p]);
  Return(i);
}
```

40 80 90 25 54 68 16 10 35