Data Structure And Algorithm

Insertion Sort

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

- 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 in their right place.

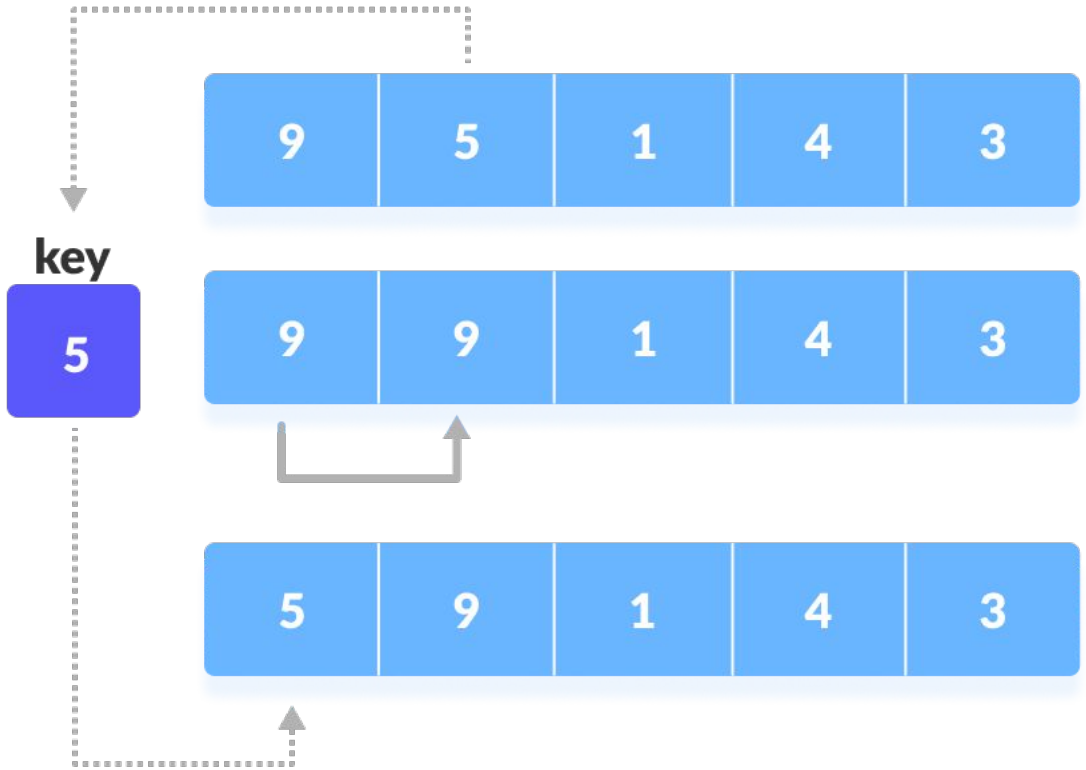
Working of Insertion Sort

- Suppose we need to sort the following array.
-

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

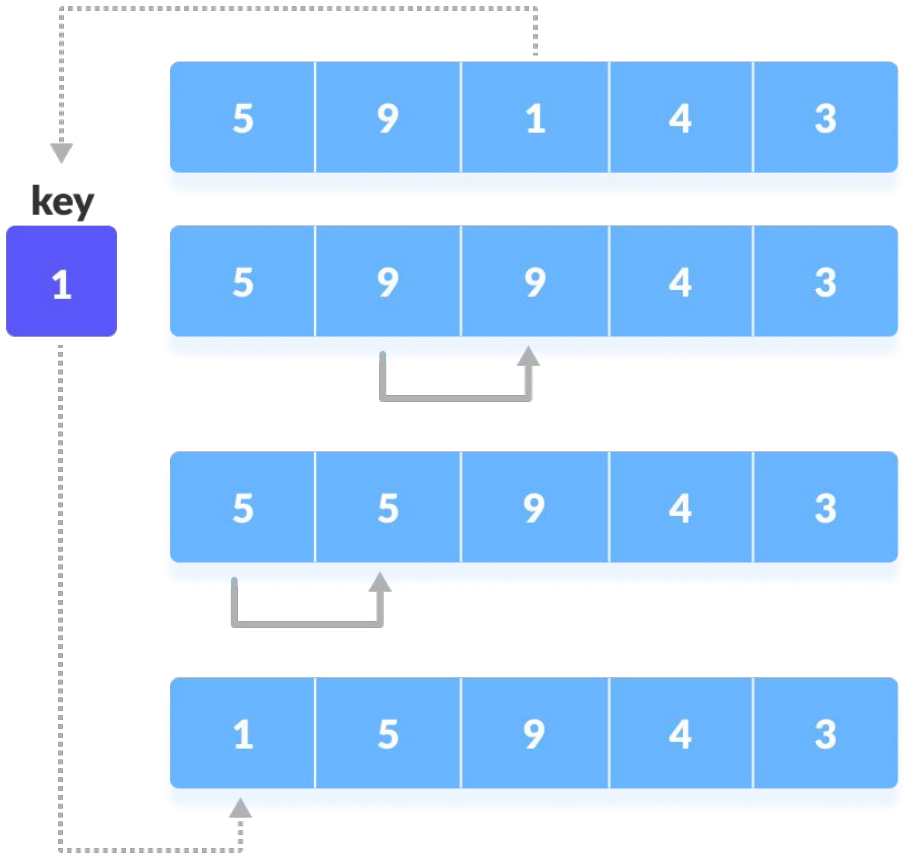
- Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element.

step = 1



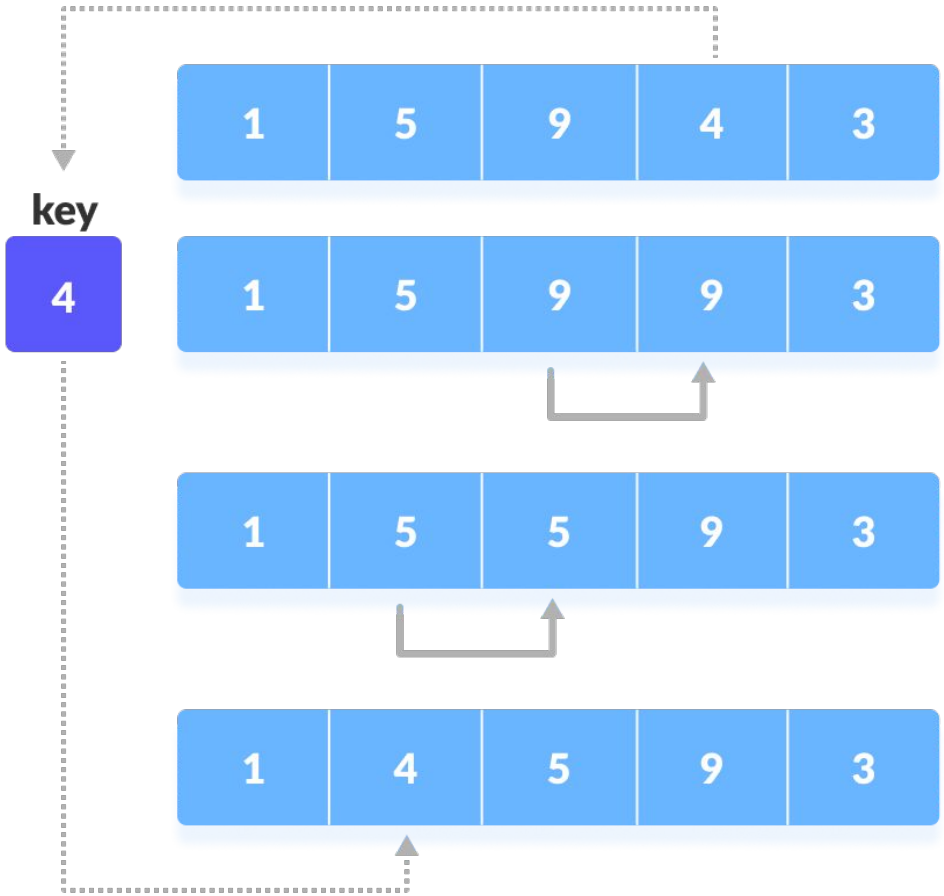
- Now, the first two elements are sorted.
- Take the third element and compare it with the elements on the left of it. Place it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.

step = 2

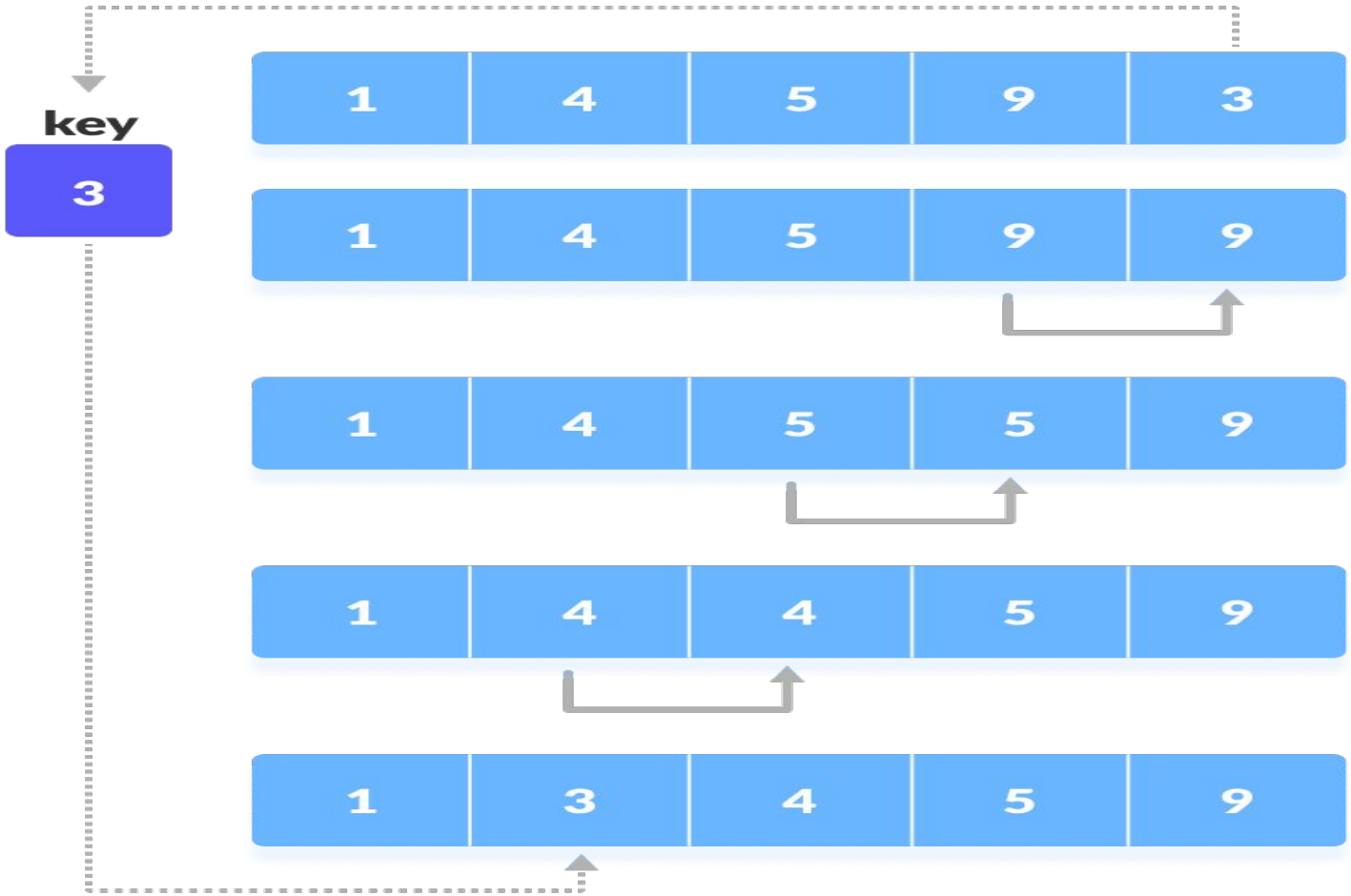


- Similarly, place every unsorted element at its correct position.

step = 3



step = 4



Insertion sort 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

```
void insertionSort(int array[]) {
  int size = array.length;

  for (int step = 1; step < size; step++) {
    int key = array[step];
    int 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 && key < array[j]) {
      array[j + 1] = array[j];
      --j;
    }

    // Place key at after the element just smaller than it.
    array[j + 1] = key;
  }
}
```

- Time Complexity
- Best O(n)
- Worst $O(n^2)$
- Average $O(n^2)$