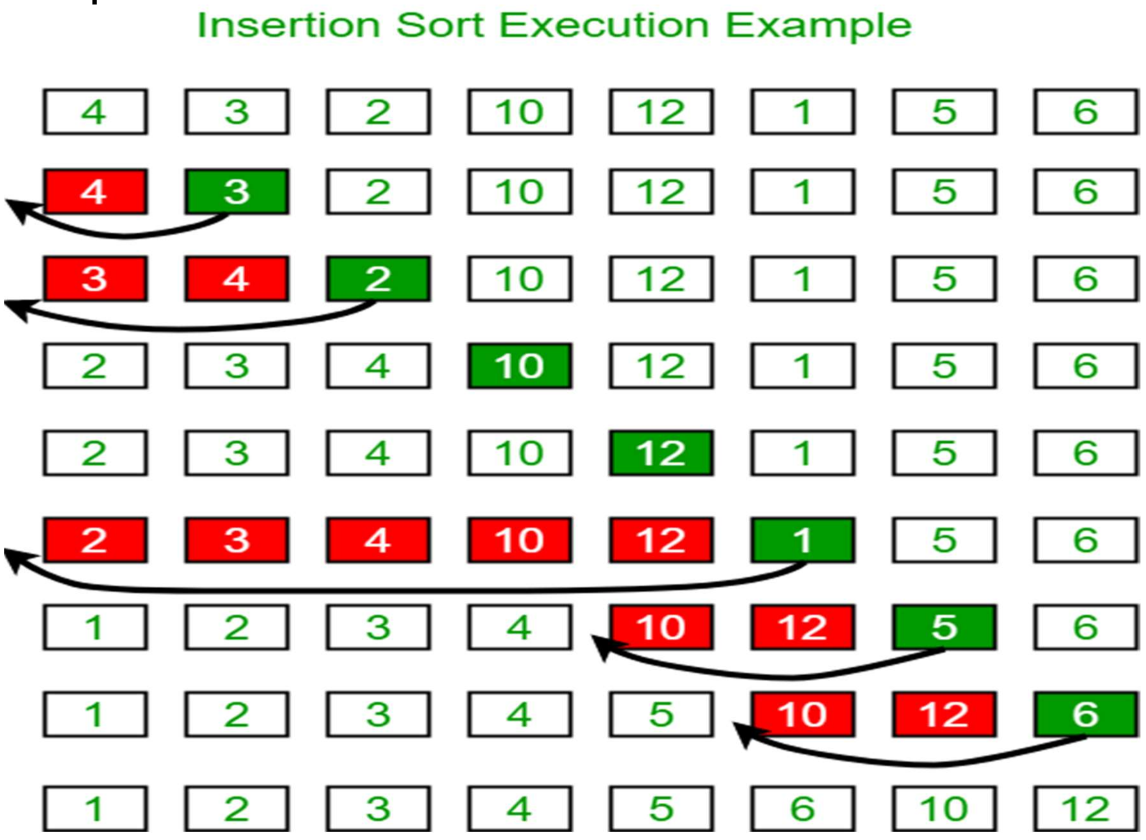
Insertion Sort

- 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.

Algorithm
To sort an array of size n in ascending order:

Step-1: Iterate from arr[1] to arr[n] over the array.
Step-2: Compare the current element (key) to its predecessor.
Step-3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

Example:



Another Example:

12, 11, 13, 5, 6

Let us loop for i = 1 (second element of the array) to 4 (last element of the array)

i = 1. Since 11 is smaller than 12, move 12 and insert 11 before 12
11, 12, 13, 5, 6

i = 2. 13 will remain at its position as all elements in A[0..I-1] are smaller than 13
11, 12, 13, 5, 6

i = 3. 5 will move to the beginning and all other elements from 11 to 13 will move one position ahead of their current position.
5, 11, 12, 13, 6

i = 4. 6 will move to position after 5, and elements from 11 to 13 will move one position ahead of their current position.
5, 6, 11, 12, 13

```
// C++ program for insertion sort
#include <bits/stdc++.h>
using namespace std;

/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
   int i, key, j;
   for (i = 1; i < n; i++)
   {
      key = arr[i];
      j = i - 1;

      /* Move elements of arr[0..i-1], that are
      greater than key, to one position ahead
      of their current position */
      while (j >= 0 && arr[j] > key)
      {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}

// A utility function to print an array of size n
```

```
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

/* Driver code */
int main()
{
    int arr[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);
    printArray(arr, n);

    return 0;
}

//
```

Output:

```
5 6 11 12 13
```

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.

Algorithmic Paradigm: Incremental Approach

Sorting In Place: Yes

Stable: Yes

Online: Yes

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.

What is Binary Insertion Sort?

- We can use binary search to reduce the number of comparisons in normal insertion sort.

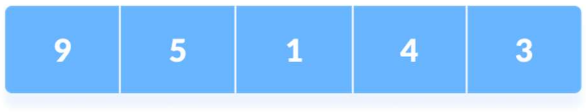
- Binary Insertion Sort uses binary search to find the proper location to insert the selected item at each iteration.
- In normal insertion, sorting takes O(i) (at ith iteration) in worst case.
- We can reduce it to O(logi) by using binary search.
- The algorithm, as a whole, still has a running worst case running time of O(n^2) because of the series of swaps required for each insertion.
-
- **How to implement Insertion Sort for Linked List?**

- Below is simple insertion sort algorithm for linked list.

```
1) Create an empty sorted (or result) list
2) Traverse the given list, do following for every node.
......a) Insert current node in sorted way in sorted or result list.
3) Change head of given linked list to head of sorted (or result) list.
```

Insertion Sort Works?

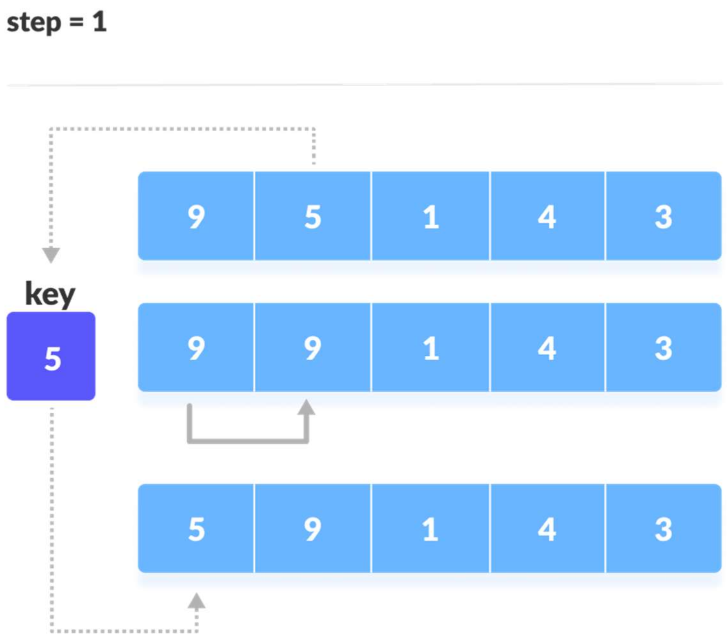
Suppose we need to sort the following array.



Initial array

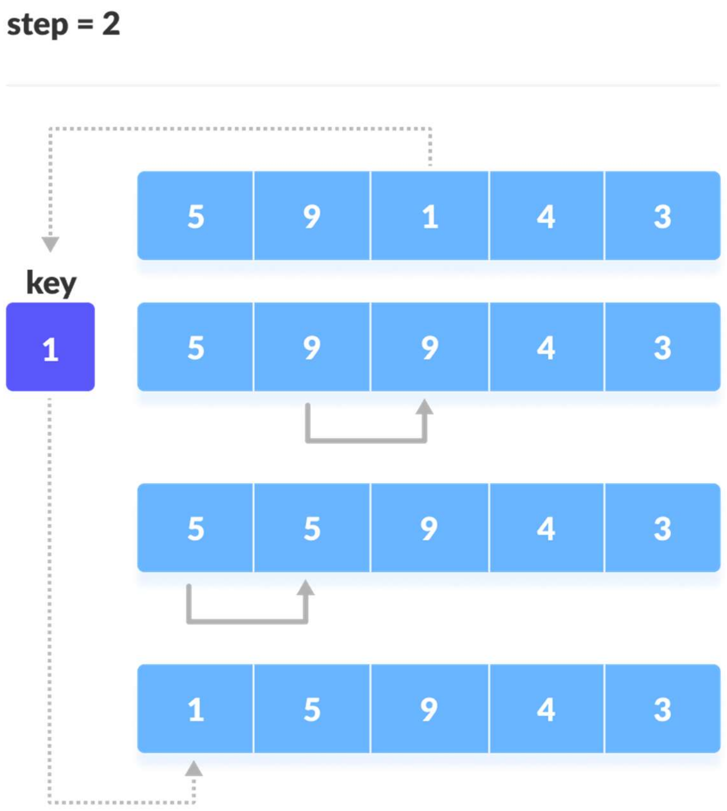
1. The first element in the array is assumed to be sorted. Take the second element and store it separately in key.

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



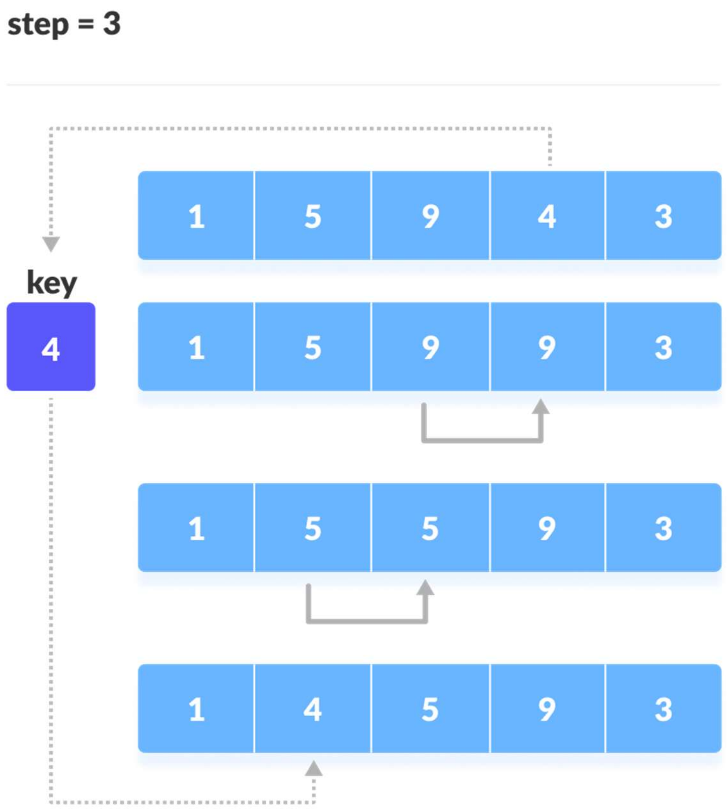
2. If the first element is greater than key, then key is placed in front of the first element.
3. Now, the first two elements are sorted.

   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

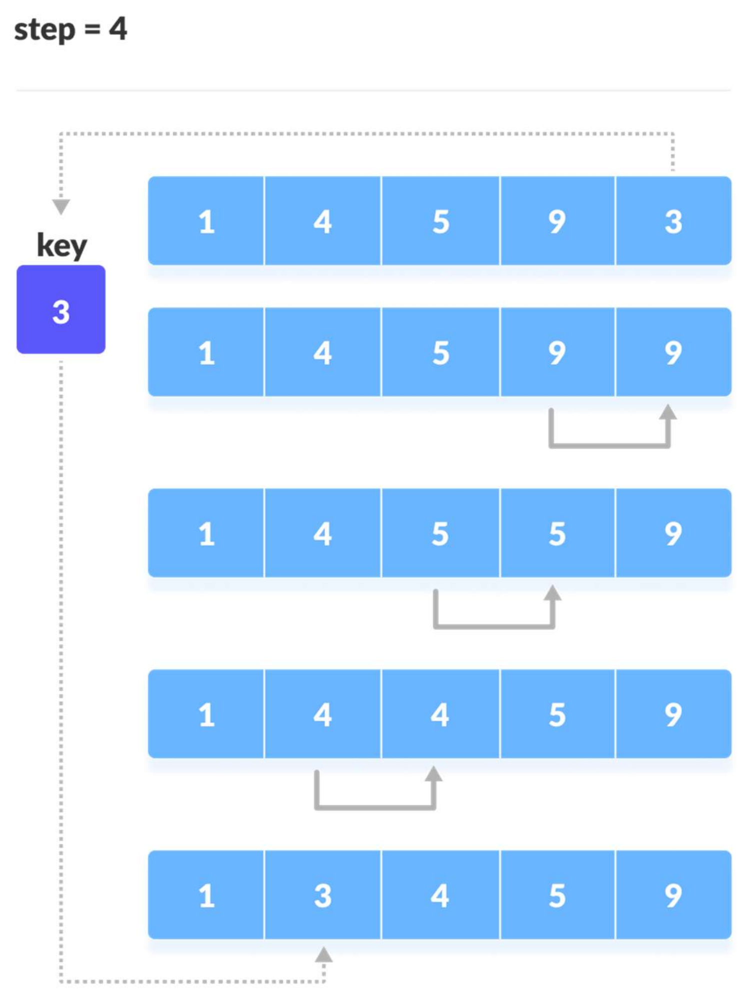


the beginning of the array.
Place 1 at the beginning

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



Place 4 behind



1
5. Place 3 behind 1 and the array is sorted

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
```

# 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)
```

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.

Insertion Sort Applications

The insertion sort is used when:

- the array is has a small number of elements
- there are only a few elements left to be sorted