Insertion Sort – Data Structure and Algorithm Tutorials

Insertion sort is a simple sorting algorithm that works similarly 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 in the correct position in the sorted part.

Insertion Sort Algorithm

> *To sort an array of size N in ascending order iterate over the array and compare the current element (key) to its predecessor, 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.*

Working of Insertion Sort algorithm

Consider an example: arr[]: ***{12, 11, 13, 5, 6}***

12	***11***	***13***	***5***	***6***

First Pass:

Skip to content

- *Initially, the first two elements of the array are compared in insertion sort.*

12	***11***	*13*	*5*	*6*

- *Here, 12 is greater than 11 hence they are not in the ascending order and 12 is not at its correct position. Thus, swap 11 and 12.*
- *So, for now 11 is stored in a sorted sub-array.*

11	***12***	*13*	*5*	*6*

Second Pass:

- *Now, move to the next two elements and compare them*

11	***12***	***13***	*5*	*6*

- *Here, 13 is greater than 12, thus both elements seems to be in ascending order, hence, no swapping will occur. 12 also stored in a sorted sub-array along with 11*

Third Pass:

- *Now, two elements are present in the sorted sub-array which are **11** and **12***
- *Moving forward to the next two elements which are 13 and 5*

11	*12*	***13***	***5***	*6*

- *Both 5 and 13 are not present at their correct place so swap them*

11	*12*	***5***	***13***	*6*

- *After swapping, elements 12 and 5 are not sorted, thus swap again*

11	***5***	***12***	*13*	*6*

Job Fair 2024

Explore Our Geeks Community

Insertion Sort - Data Structure and Algorithm Tutorials

Insertion Sort in Different languages

Binary Insertion Sort

Binary Insertion Sort in Different languages

Recursive Insertion Sort in Different languages

5	***11***	*12*	*13*	*6*

- *Here, 5 is at its correct position*

Fourth Pass:

- *Now, the elements which are present in the sorted sub-array are **5, 11** and **12***
- *Moving to the next two elements 13 and 6*

5	*11*	*12*	***13***	***6***

- *Clearly, they are not sorted, thus perform swap between both*

Skip to content

5	*11*	*12*	***6***	***13***

- *Now, 6 is smaller than 12, hence, swap again*

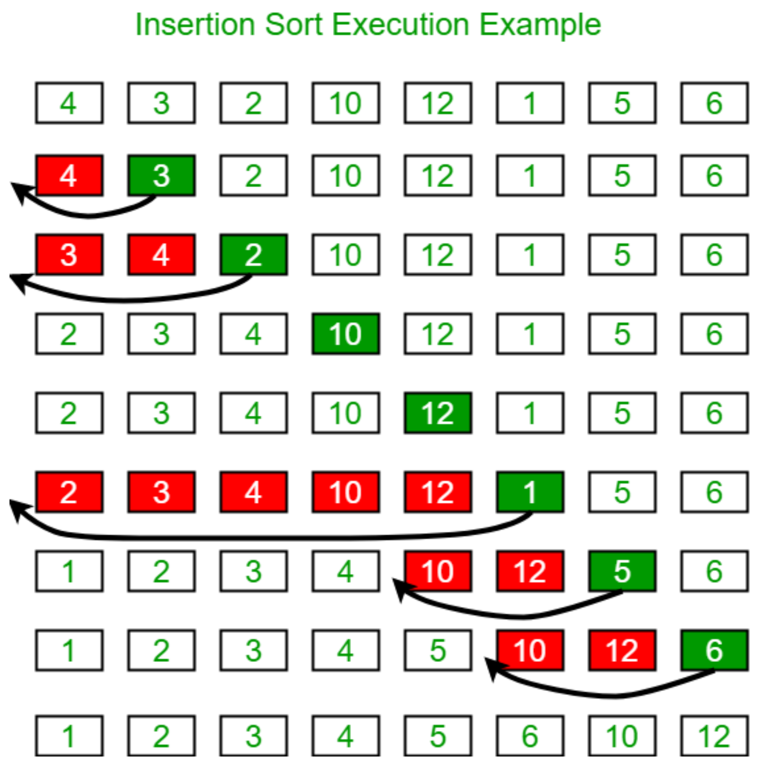
5	*11*	***6***	***12***	*13*

- *Here, also swapping makes 11 and 6 unsorted hence, swap again*

5	***6***	***11***	*12*	*13*

- *Finally, the array is completely sorted.*

Illustrations:



Skip to content

Implementation of Insertion Sort Algorithm

Recommended Problem

Insertion Sort

Sorting Algorithms Microsoft MAQ Software +6 more

Solve Problem

Submission count: 1.2L

Below is the implementation of the iterative approach:

C++

```
// 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)
{
```

Skip to content

```
    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;
}
// This is code is contributed by rathbhupendra
```

C

```
// C program for insertion sort
#include <math.h>
#include <stdio.h>

/* 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;
    }
}
```

Skip to content

```
// A utility function to print an array of size n
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

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

    return 0;
}
```

Java

```
// Java program for implementation of Insertion Sort
public class InsertionSort {
    /*Function to sort array using insertion sort*/
    void sort(int arr[])
    {
        int n = arr.length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
            int 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 array of size n*/
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");

        System.out.println();
    }

    // Driver method
    public static void main(String args[])
    {
        int arr[] = { 12, 11, 13, 5, 6 };

        InsertionSort ob = new InsertionSort();
        ob.sort(arr);

        printArray(arr);
    }
};


/* This code is contributed by Rajat Mishra. */
```

Python

```
# Python program for implementation of Insertion Sort

# Function to do insertion sort
def insertionSort(arr):

    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):

        key = arr[i]

        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i-1
        while j >= 0 and key < arr[j] :
                arr[j + 1] = arr[j]
                j -= 1
```

Skip to content

```
            arr[j + 1] = key


# Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
for i in range(len(arr)):
    print ("% d" % arr[i])

# This code is contributed by Mohit Kumra
```

C#

```
// C# program for implementation of Insertion Sort
using System;

class InsertionSort {

    // Function to sort array
    // using insertion sort
    void sort(int[] arr)
    {
        int n = arr.Length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
            int 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
    // array of size n
    static void printArray(int[] arr)
    {
        int n = arr.Length;
```

Skip to content

```
            for (int i = 0; i < n; ++i)
                Console.Write(arr[i] + " ");

            Console.Write("\n");
        }

        // Driver Code
        public static void Main()
        {
            int[] arr = { 12, 11, 13, 5, 6 };
            InsertionSort ob = new InsertionSort();
            ob.sort(arr);
            printArray(arr);
        }
    }

    // This code is contributed by ChitraNayal.
```

Javascript

```
// Javascript program for insertion sort

// Function to sort an array using insertion sort
function insertionSort(arr, n)
{
    let 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
```

Skip to content

```
function printArray(arr, n)
{
    let i;
    for (i = 0; i < n; i++)
        document.write(arr[i] + " ");
    document.write("<br>");
}

// Driver code
    let arr = [12, 11, 13, 5, 6 ];
    let n = arr.length;

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

// This code is contributed by Mayank Tyagi
```

PHP

```
<?php
// PHP program for insertion sort

// Function to sort an array
// using insertion sort
function insertionSort(&$arr, $n)
{
    for ($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;
    }
```

Skip to content

```
}

// A utility function to
// print an array of size n
function printArray(&$arr, $n)
{
    for ($i = 0; $i < $n; $i++)
        echo $arr[$i]." ";
    echo "\n";
}

// Driver Code
$arr = array(12, 11, 13, 5, 6);
$n = sizeof($arr);
insertionSort($arr, $n);
printArray($arr, $n);

// This code is contributed by ChitraNayal.
?>
```

Learn **Data Structures & Algorithms** with GeeksforGeeks

Output

```
5 6 11 12 13
```

Time Complexity: O(N^2)
Auxiliary Space: O(1)

Complexity Analysis of Insertion Sort:

Time Complexity of Insertion Sort

- The **worst-case** time complexity of the Insertion sort is **O(N^2)**
- The **average case** time complexity of the Insertion sort is **O(N^2)**
- The time complexity of the **best case** is **O(N)**.

Space Complexity of Insertion Sort

Skip to content

The auxiliary space complexity of Insertion Sort is **O(1)**

Characteristics of Insertion Sort

- This algorithm is one of the simplest algorithms with a simple implementation
- Basically, Insertion sort is efficient for small data values
- Insertion sort is adaptive in nature, i.e. it is appropriate for data sets that are already partially sorted.

Frequently Asked Questions on Insertion Sort

Q1. What are the Boundary Cases of the Insertion Sort algorithm?

Insertion sort takes the maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.

Q2. What is the Algorithmic Paradigm of the Insertion Sort algorithm?

The Insertion Sort algorithm follows an incremental approach.

Q3. Is Insertion Sort an in-place sorting algorithm?

Yes, insertion sort is an in-place sorting algorithm.

Q4. Is Insertion Sort a stable algorithm?

Yes, insertion sort is a stable sorting algorithm.