**Name :** Movaliya Hasti

**Id number :** 22CP056

**Course :** Design and Analysis of Algorithms (3CP02)

**Insertion Sort Algorithm**

**Insertion sort**is a simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list. It is like sorting playing cards in your hands. You split the cards into two groups: the sorted cards and the unsorted cards. Then, you pick a card from the unsorted group and put it in the right place in the sorted group.

* We start with second element of the array as first element in the array is assumed to be sorted.
* Compare second element with the first element and check if the second element is smaller then swap them.
* Move to the third element and compare it with the first two elements and put at its correct position
* Repeat until the entire array is sorted.
* **Illustration:**
* ***arr = {23, 1, 10, 5, 2}***
* ***Initial:***
* *Current element is* ***23***
* *The first element in the array is assumed to be sorted.*
* *The sorted part until* ***0th*** *index is :* ***[23]***
* ***First Pass:***
* *Compare* ***1*** *with* ***23*** *(current element with the sorted part).*
* *Since* ***1*** *is smaller, insert* ***1*** *before* ***23*** *.*
* *The sorted part until* ***1st*** *index is:* ***[1, 23]***
* ***Second Pass:***
* *Compare* ***10*** *with* ***1*** *and* ***23*** *(current element with the sorted part).*
* *Since* ***10*** *is greater than* ***1*** *and smaller than* ***23*** *, insert* ***10*** *between* ***1*** *and* ***23*** *.*
* *The sorted part until* ***2nd*** *index is:* ***[1, 10, 23]***
* ***Third Pass:***
* *Compare* ***5*** *with* ***1*** *,* ***10*** *, and* ***23*** *(current element with the sorted part).*
* *Since* ***5*** *is greater than* ***1*** *and smaller than* ***10*** *, insert* ***5*** *between* ***1*** *and* ***10***
* *The sorted part until* ***3rd*** *index is* ***: [1, 5, 10, 23]***
* ***Fourth Pass:***
* *Compare* ***2*** *with* ***1, 5, 10*** *, and* ***23*** *(current element with the sorted part).*
* *Since* ***2*** *is greater than* ***1*** *and smaller than* ***5*** *insert* ***2*** *between* ***1*** *and* ***5*** *.*
* *The sorted part until* ***4th*** *index is:* ***[1, 2, 5, 10, 23]***
* ***Final Array:***
* *The sorted array is:* ***[1, 2, 5, 10, 23]***

**C++ Code :**

**// C++ program for implementation of Insertion Sort**

**#include <iostream>**

**using namespace std;**

**/\* Function to sort array using insertion sort \*/**

**void insertionSort(int arr[], int n)**

**{**

**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 \*/**

**void printArray(int arr[], int n)**

**{**

**for (int i = 0; i < n; ++i)**

**cout << arr[i] << " ";**

**cout << endl;**

**}**

**// Driver method**

**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 code is contributed by Hritik Shah. \*/**

**Python Code :**   
# Python program for implementation of Insertion Sort

# Function to sort array using insertion sort

def insertionSort(arr):

for i in range(1, len(arr)):

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 and key < arr[j]:

arr[j + 1] = arr[j]

j -= 1

arr[j + 1] = key

# A utility function to print array of size n

def printArray(arr):

for i in range(len(arr)):

print(arr[i], end=" ")

print()

# Driver method

if \_\_name\_\_ == "\_\_main\_\_":

arr = [12, 11, 13, 5, 6]

insertionSort(arr)

printArray(arr)

# This code is contributed by Hritik Shah.

**Complexity Analysis :**

**Time Complexity of Insertion Sort**

* **Best case: O(n)**, If the list is already sorted, where n is the number of elements in the list.
* **Average case: O(n 2 )**, If the list is randomly ordered
* **Worst case: O(n 2 )**, If the list is in reverse order

**Space Complexity of Insertion Sort**

* **Auxiliary Space:**O(1), Insertion sort requires **O(1)**additional space, making it a space-efficient sorting algorithm.