**Insertion Sort**

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst-case complexity are of Ο(n2), where **n** is the number of items.

**Algorithm**   
To sort an array of size n in ascending order:   
1: Iterate from arr[1] to arr[n] over the array.   
2: Compare the current element (key) to its predecessor.   
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.

Pseudocode

procedure insertionSort( A : array of items )

int holePosition

int valueToInsert

for i = 1 to length(A) inclusive do:

/\* select value to be inserted \*/

valueToInsert = A[i]

holePosition = i

/\*locate hole position for the element to be inserted \*/

while holePosition > 0 and A[holePosition-1] > valueToInsert do:

A[holePosition] = A[holePosition-1]

holePosition = holePosition -1

end while

/\* insert the number at hole position \*/

A[holePosition] = valueToInsert

end for

end procedure

**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**

|  |  |
| --- | --- |
| **Time Complexity** |  |
| Best | O(n) |
| Worst | O(n2) |
| Average | O(n2) |
| **Space Complexity** | O(1) |
| **Stability** | Yes |

**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) ~ 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 having a small number of elements
* there are only a few elements left to be sorted