

[Courses \(/courses/\)](/courses/) / [Programming \(/courses/programming/\)](/courses/programming/) / [Arrays \(/courses/programming/topics/arrays/\)](/courses/programming/topics/arrays/) / Insertion Sort Algorithm

≡ Level 2
Arrays

[Go To Problems \(/courses/programming/topics/arrays/#problems\)](/courses/programming/topics/arrays/#problems)

TUTORIAL

1. Introduction To Pointers In C/C++

[tutorial/introduction-to-pointers-in-cc](/tutorial/introduction-to-pointers-in-cc)) (</tutorial/arrays-in-programming-fundamentals>) (</tutorial/pointers-and-arrays>) (</tutorial/pointers-and-2-d-arrays>)

2. Arrays In Programming Fundamentals

3. Pointers And Arrays

4. Pointers And 2 D Arrays

</tutorial/array-implementation-details>) (</tutorial/sorting-algorithms>) (</tutorial/merge-sort-algorithm>) (</tutorial/quicksort-algorithm>)

5. Array Implementation Details

6. Sorting Algorithms

7. Insertion Sort Algorithm

8. Merge Sort Algorithm

9. Quick Sort Algorithm

10. Sort Implementation Details

[tutorial/sort-implementation-details](#)) ([/tutorial/selection-sort](#)) ([/tutorial/bubble-sort](#))

11. Selection Sort

12. Bubble Sort

Insertion Sort Algorithm

Insertion sort is the sorting mechanism where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. The analogy can be understood from the style we arrange a deck of cards. This sort works on the principle of inserting an element at a particular position, hence the name Insertion Sort.

Quick Navigation

1. Working
2. Pseudocode
3. Example program in C, C++, Java & Python
4. Time Complexity

Insertion Sort works as follows:

1. The first step involves the comparison of the element in question with its adjacent element.
2. And if at every comparison reveals that the element in question can be inserted at a particular position, then space is created for it by shifting the other elements one position to the right and inserting the element at the suitable position.
3. The above procedure is repeated until all the element in the array is at their apt position.

Let us now understand working with the following example:

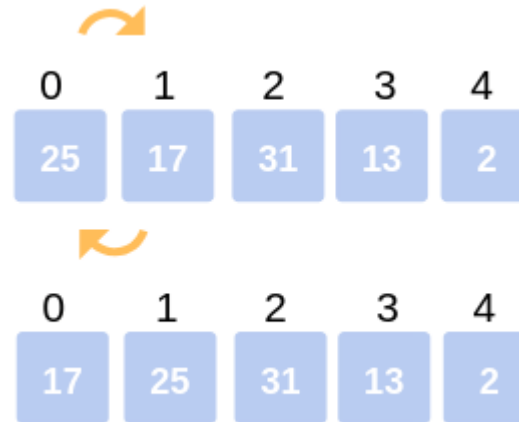
Consider the following array: 25, 17, 31, 13, 2

First Iteration: Compare 25 with 17. The comparison shows $17 < 25$. Hence swap 17 and 25.

The array now looks like:

17, 25, 31, 13, 2

First Iteration



Second Iteration: Begin with the second element (25), but it was already swapped on for the correct position, so we move ahead to the next element.

Now hold on to the third element (31) and compare with the ones preceding it.

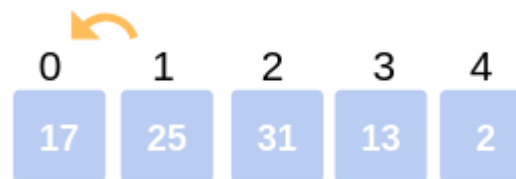
Since $31 > 25$, no swapping takes place.

Also, $31 > 17$, no swapping takes place and 31 remains at its position.

The array after the Second iteration looks like:

17, 25, 31, 13, 2

Second Iteration



Third Iteration: Start the following Iteration with the fourth element (13), and compare it with its preceding elements.

Since $13 < 31$, we swap the two.

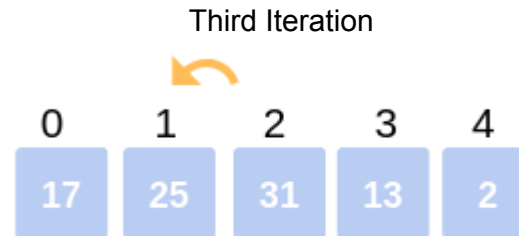
Array now becomes: 17, 25, 13, 31, 2.

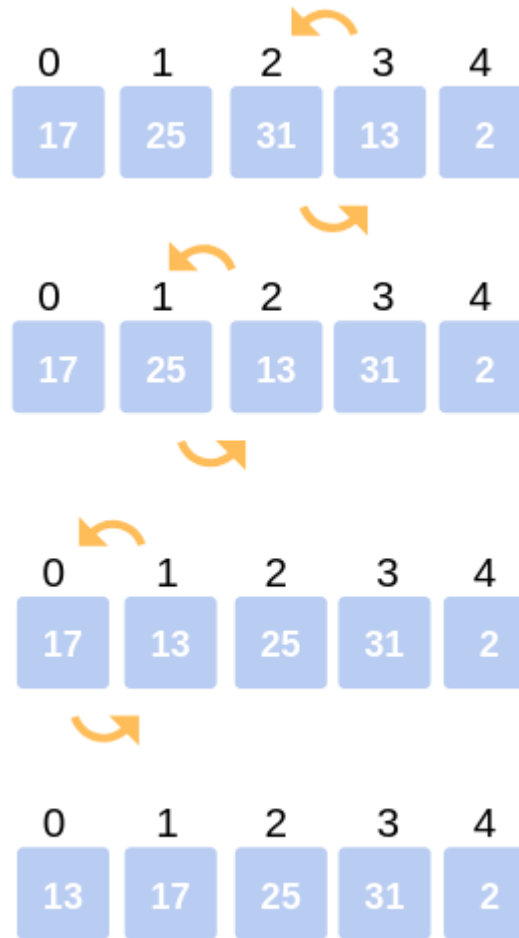
But there still exist elements that we haven't yet compared with 13. Now the comparison takes place between 25 and 13. Since, $13 < 25$, we swap the two.

The array becomes **17, 13, 25, 31, 2**.

The last comparison for the iteration is now between 17 and 13. Since $13 < 17$, we swap the two.

The array now becomes **13, 17, 25, 31, 2**.





Fourth Iteration: The last iteration calls for the comparison of the last element (2), with all the preceding elements and make the appropriate swapping between elements.

Since, $2 < 31$. Swap 2 and 31.

Array now becomes: 13, 17, 25, 2, 31.

Compare 2 with 25, 17, 13.

Since, $2 < 25$. Swap 25 and 2.

13, 17, 2, 25, 31.

Compare 2 with 17 and 13.

Since, $2 < 17$. Swap 2 and 17.

Array now becomes:

13, 2, 17, 25, 31.

The last comparison for the Iteration is to compare 2 with 13.

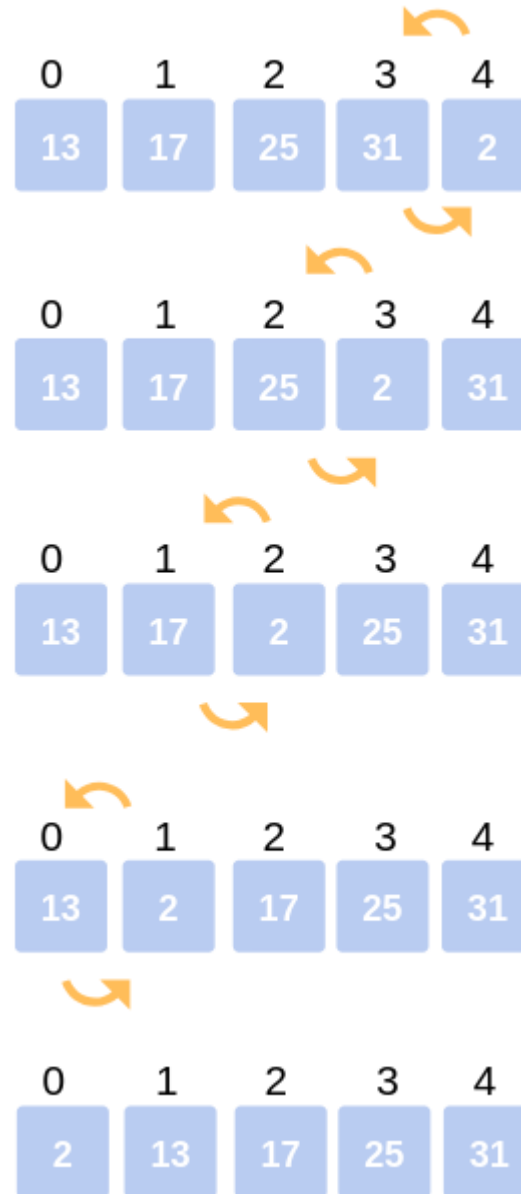
Since $2 < 13$. Swap 2 and 13.

The array now becomes:

2, 13, 17, 25, 31.

This is the final array after all the corresponding iterations and swapping of elements.

Fourth Iteration



Pseudocode

```
INSERTION-SORT(A)
  for i = 1 to n
    key ← A[i]
    j ← i - 1
    while j >= 0 and A[j] > key
      A[j+1] ← A[j]
      j ← j - 1
    End while
    A[j+1] ← key
  End for
```

Implementation:

Following are C, C++, Java and Python implementations of Insertion Sort.

[C](#)[C++](#)[Java](#)[Python](#)

Insertion sort Implementation in C:

```
#include <stdio.h>
```

```
#include <stdbool.h>
```

```
#define MAX 7    //defining size of our array
```

```
int intArray[MAX] = {4,6,3,2,1,9,7};
```

```
void printline(int count) {
```

```
    int i;
```

```
    for(i = 0; i < count-1; i++) {
```

```
        printf("=");
```

```
    }
```

```
    printf("\n");
```

```
}
```

```
void display() {
```

```
    int i;
```

```
    printf("[");
```

```
    // navigate through all items
```

```
    for(i = 0; i < MAX; i++) {
```

```
        printf("%d ", intArray[i]);
```

```
    }
```

```
    printf("]\n");
```

```
}
```

```
void insertionSort() {
```

```
    int valueToInsert;
```

```
    int holePosition;
```

```
    int i;
```

```
    // loop through all numbers
```

```
for(i = 1; i < MAX; i++) {

    // select a value to be inserted.
    valueToInsert = intArray[i];

    // select the hole position where number is to be inserted
    holePosition = i;

    // check if previous no. is larger than value to be inserted
    while (holePosition > 0 && intArray[holePosition-1] > valueToInsert) {
        intArray[holePosition] = intArray[holePosition-1];
        holePosition--;
        printf(" item moved : %d\n" , intArray[holePosition]);
    }

    if(holePosition != i) {
        printf(" item inserted : %d, at position : %d\n" , valueToInsert, holePosition);
        // insert the number at current hole
        intArray[holePosition] = valueToInsert;
    }

    printf("Iteration %d#:", i);
    display();

}

}

void main() {
    printf("Input Array: ");
    display();
    printline(50);
    insertionSort();
    printf("Output Array: ");
    display();
    printline(50);
}
```

Time Complexity Analysis:

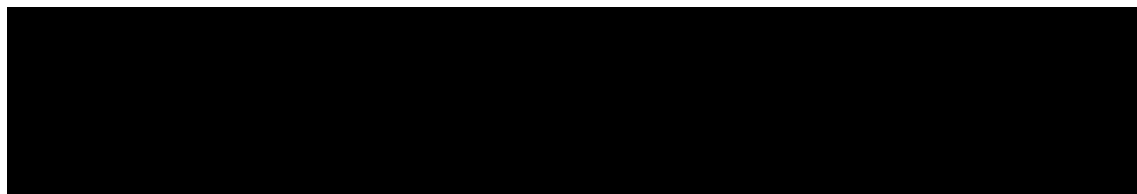
Even though insertion sort is efficient, still, if we provide an already sorted array to the insertion sort algorithm, it will still execute the outer for loop, thereby requiring n steps to sort an already sorted array of n elements, which makes its best case time complexity a linear function of n .

Wherein for an unsorted array, it takes for an element to compare with all the other elements which mean every n element compared with all other n elements. Thus, making it for $n \times n$, i.e., n^2 comparisons. One can also take a look at other sorting algorithms such as *Merge sort*, *Quick Sort*, *Selection Sort*, etc. and understand their complexities.

Worst Case Time Complexity [Big-O]: $O(n^2)$

Best Case Time Complexity [Big-omega]: $O(n)$

Average Time Complexity [Big-theta]: $O(n^2)$

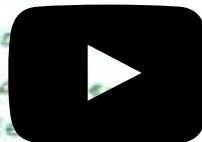


(<https://www.youtube.com/watch?v=i-SKeOcbwko>)

A	2	7	7	1	5	3
	0	1	2	3	4	5
i	Value	hole				
1	2	1				
1	2	0				
2	4	2				
2	4	1				

```

InsertionSort(A, n)
{
  for i ← 1 to n-1
  {
    value ← A[i]
    hole ← i
    while (hole > 0 & A[hole] > value)
    {
      A[hole] ← A[hole-1]
      hole ← hole-1
    }
    A[hole] ← value
  }
}
  
```





Blog (<https://blog.interviewbit.com>) | About Us (/pages/about_us/) | FAQ (</pages/faq/>) | Contact Us (/pages/contact_us/) |

Terms (</pages/terms/>) | Privacy Policy (</pages/privacy/>)

System Design Interview Questions (</courses/system-design/>) | Google Interview Questions (</google-interview-questions/>) |

Facebook Interview Questions (</facebook-interview-questions/>) | Amazon Interview Questions (</amazon-interview-questions/>) |

Microsoft Interview Questions (</microsoft-interview-questions/>) | SQL Interview Questions (</sql-interview-questions/>)

 Like Us (<https://www.facebook.com/interviewbit>)  Follow Us (https://twitter.com/interview_bit)

 Email (<mailto:hello@interviewbit.com>)

[Click here to start solving coding interview questions \(/\)](#)