Insertion Sort Algorithm

Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration. Insertion sort works similarly as we sort cards in our hand in a card game.

We assume that the first card is already sorted, then we select an unsorted card. If the unsorted card is greater than the card in hand, it is placed on the right, otherwise, to the left. In the same way, other unsorted cards are taken and put in their right place.

Questions:

1) 9, 5, 1, 4, 3
2) 12, 11,13, 5, 6
3) 12, 31, 25, 8, 32, 17
4) 14, 33, 27, 10, 35, 19, 42, 44

Algorithm

The simple steps of achieving the insertion sort are listed as follows -

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step2 - Pick the next element, and store it separately in a **key.**

Step3 - Now, compare the **key** with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted.

Algorithm

Now we have a bigger picture of how this sorting technique works, so we can derive simple steps by which we can achieve insertion sort.

Step 1 − If it is the first element, it is already sorted. return 1;
Step 2 − Pick next element
Step 3 − Compare with all elements in the sorted sub-list
Step 4 − Shift all the elements in the sorted sub-list that is greater than the

value to be sorted

Step 5 − Insert the value

Step 6 − Repeat until list 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
```

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
```

Insertion sort in C

```
#include <stdio.h>

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

void insertionSort(int array[], int size) {
  for (int step = 1; step < size; step++) {
    int key = array[step];
    int 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 (key < array[j] && j >= 0) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = key;
  }
}

// Driver code
int main() {
  int data[] = {9, 5, 1, 4, 3};
  int size = sizeof(data) / sizeof(data[0]);
  insertionSort(data, size);
  printf("Sorted array in ascending order:\n");
  printArray(data, size);
}
```

Insertion Sort Complexity

Time Complexity	
Best	O(n)
Worst	O(n^2)
Average	O(n^2)
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 has a small number of elements
- there are only a few elements left to be sorted

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.

Q5. When is the Insertion Sort algorithm used?

Insertion sort is used when number of elements is small. It can also be useful when the input array is almost sorted, and only a few elements are misplaced in a complete big array.