Insertion Sort

Consider you have 10 cards out of a deck of cards in your hand. And they are sorted, or arranged in the ascending order of their numbers.

If I give you another card, and ask you to insert the card in just the right position, so that the cards in your hand are still sorted. What will you do?

Well, you will have to go through each card from the starting or the back and find the right position for the new card, comparing it's value with each card. Once you find the right position, you will insert the card there.

Similarly, if more new cards are provided to you, you can easily repeat the same process and insert the new cards and keep the cards sorted too.

This is exactly how insertion sort works. It starts from the index 1(not 0), and each index starting from index 1 is like a new card, that you have to place at the right position in the sorted subarray on the left.

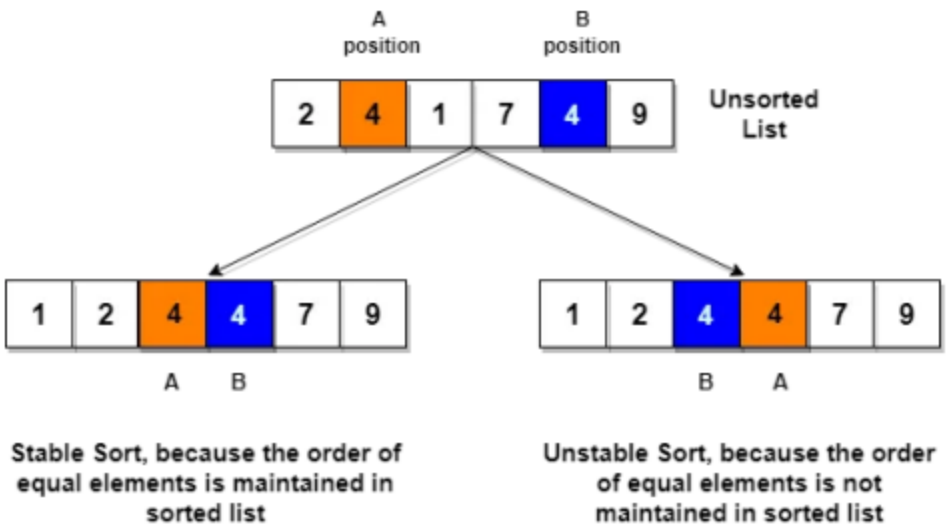
Following are some of the important characteristics of Insertion Sort:

1. It is efficient for smaller data sets, but very inefficient for larger lists.

2. Insertion Sort is adaptive, that means it reduces its total number of steps if a partially sorted array is provided as input, making it efficient.

3. It is better than Selection Sort and Bubble Sort algorithms.

4. It is a stable sorting technique, as it does not change the relative order of elements which are equal.



• How Insertion Sort Works?

Following are the steps involved in insertion sort:

We start by making the second element of the given array, i.e. element at index 1, the key. The key element here is the new card that we need to add to our existing sorted set of cards(remember the example with cards above).

We compare the key element with the element(s) before it, in this case, element at index 0:

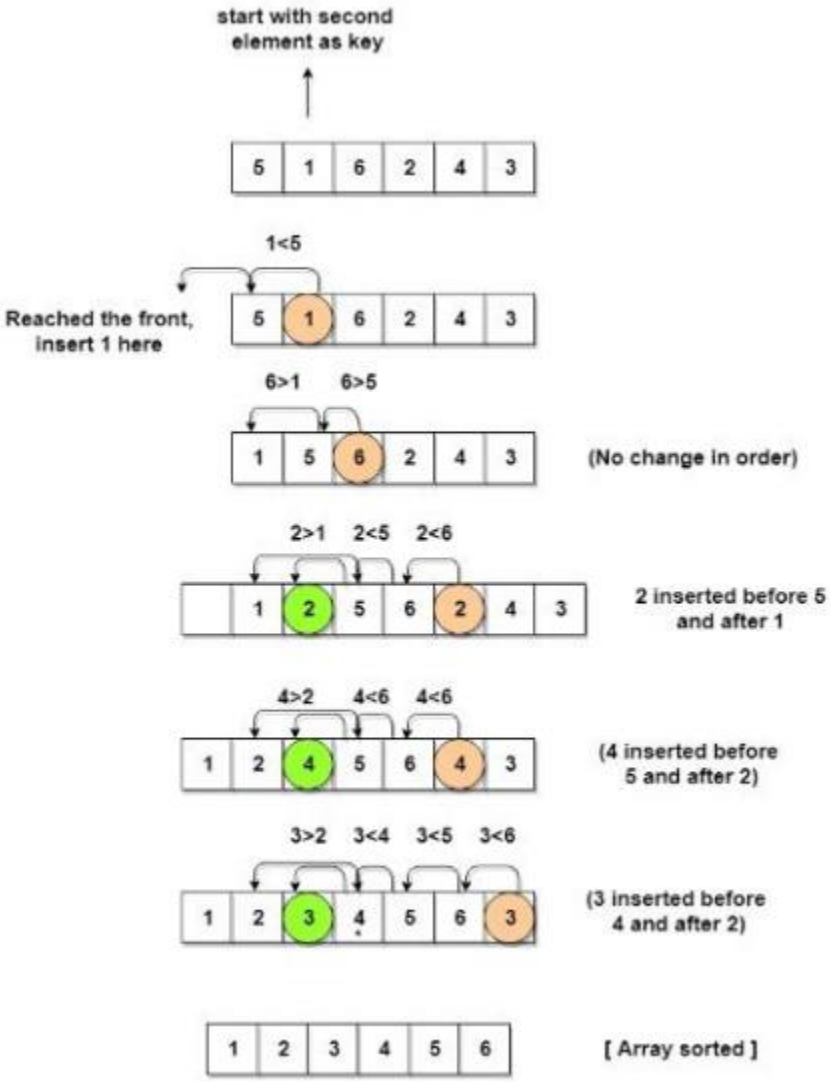
If the key element is less than the first element, we insert the key element before the first element.

If the key element is greater than the first element, then we insert it after the first element.

Then, we make the third element of the array as key and will compare it with elements to it's left and insert it at the right position.

And we go on repeating this, until the array is sorted.

Let's consider an array with values {5, 1, 6, 2, 4, 3}. Below, we have a pictorial representation of how insertion sort will sort the given array.



As you can see in the diagram above, after picking a key, we start iterating over the elements to the left of the key.

We continue to move towards left if the elements are greater than the key element and stop when we find the element which is less than the key element.

And, insert the key element after the element which is less than the key element.

- **Program for insertion sort:**

```
#include <iostream>
using namespace std;

class Insert {
public:
   void insert(int a[], int n) {
      int i, j, temp;
      for (i = 1; i < n; i++) {
         temp = a[i];
         j = i - 1;
         while (j >= 0 && temp <= a[j]) {
            a[j + 1] = a[j];
            j = j - 1;
         }
         a[j + 1] = temp;
      }
   }

   void printArr(int a[], int n) {
      for (int i = 0; i < n; i++)
         cout << a[i] << " ";
   }
};

int main() {
   int a[] = {5, 1, 6, 2, 4, 3};
   int n = sizeof(a) / sizeof(a[0]);

   Insert i1;
   cout << "Before sorting array elements are - " << endl;
   i1.printArr(a, n);

   i1.insert(a, n);

   cout << "\n\nAfter sorting array elements are - " << endl;
   i1.printArr(a, n);
```

```
   cout << endl;
   return 0;
}
```

We took an array with 6 integers. We took a variable key, in which we put each element of the array, during each pass, starting from the second element, that is a[1].

Then using the while loop, we iterate, until j becomes equal to zero or we find an element which is greater than key, and then we insert the key at that position.

We keep on doing this, until j becomes equal to zero, or we encounter an element which is smaller than the key, and then we stop. The current key is now at the right position.

We then make the next element as key and then repeat the same process.

In the above array, first we pick 1 as key, we compare it with 5(element before 1), 1 is smaller than 5, we insert 1 before 5. Then we pick 6 as key, and compare it with 5 and 1, no shifting in position this time. Then 2 becomes the key and is compared with 6 and 5, and then 2 is inserted after 1. And this goes on until the complete array gets sorted

- **Program of insertion sort (Strings):**

```
#include <iostream>
#include <string>
using namespace std;

int main(){
  string str;
  cout << "Enter the string: ";
  getline(cin, str);
  int len = str.length();

  for (int i = 1; i < len; i++){
```

```
    char temp = str[i];
    // Insert s[j] at its correct position.

    int j = i - 1;
    while (j >= 0 && str[j] > temp){
      str[j + 1] = str[j];
      j--;
    }
    str[j + 1] = temp;
  }

  cout << "\nSorted string: " << str << " \n";
  return 0;
}
```