sort() in C++ STL

Last Updated : 26 Nov, 2024

In C++, sort() is a built-in function used to sort the given range in desired order. It provides a simple and efficient way to sort the data in C++ but it only works on data structures that provide random access to its elements such as vectors and arrays.

Let's take a look at an example:

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    vector<int> v = {5, 3, 2, 1, 4};

    // Sort vector (by default in ascending order)
    sort(v.begin(), v.end());

    for (int i : v)
        cout << i << " ";
    return 0;
}
```

Output

```
1 2 3 4 5
```

This article covers the syntax, usage, and common examples of sort() method in C++:

Table of Content

Syntax of sort()

sort*(first, last, comp);*

Parameters:

- **first**: Iterator to the beginning of the range to be sorted.
- **last**: Iterator to the element just after the end of the range.
- **comp (optional):** Binary function, functor, or lambda expression that compares two elements in the range. By default, it is set as **< operator** so the sort() function sorts the data in ascending order.

Return Value:

- This function does not return any value.

Rules of Defining Comparator

To provide compatibility, a custom comparator function must follow these rules:

- It should take two arguments of the same type as the elements being sorted.
- It should return **true** if the first argument should come before the second; otherwise, it should return **false.**

To learn how to leverage sorting and other algorithms in C++, check out our **Complete C++ Course**, where you'll explore sorting techniques and how they apply in competitive programming and real-world applications.

Examples of sort()

The following examples demonstrate the use of sort() in C++ programs

Sort Array in Ascending Order

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

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

    // Sort array (by default in ascending order)
    sort(arr, arr + n);

    for (int i : arr)
        cout << i << " ";
    return 0;
}
```

Output

```
1 2 3 4 5
```

Sort Array in Descending Order

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

// Custom comparator for descending order
bool comp(int a, int b) {
      return a > b;
}

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

```
12
13      // Sort array in descending order
14      sort(arr, arr + n, comp);
15
16      for (int i : arr)
17          cout << i << " ";
18      return 0;
19  }
```

Output

```
5 4 3 2 1
```

Sort Vector of User Defined Type

C++

```
#include <bits/stdc++.h>
using namespace std;

// Custom data type
class A {
      public:
      int a;
      A(int x = 0): a(x) {}
};


// Custom comparator for A type
bool comp(A x, A y) {
    return x.a < y.a;
}

int main() {
    vector<A> v = {5, 3, 2, 1, 4};

    // Sort by absolute values
    sort(v.begin(), v.end(), comp);
```

```
        for (auto i : v)
            cout << i.a << " ";
        return 0;
    }
```

Output

```
1 2 3 4 5
```

Frequently Asked Questions – FAQs

Is sort() stable?

sort() is not stable. It may change the relative order of elements that are equal. Use stable_sort() for stable sorting.

Can we use sort with non-random access containers like list, forward_list, etc?

We cannot use sort with non-random access containers because it uses quick sort and heap sort which requires the random access to the container elements. But STL provides the specialized sort() methods for the non-random access containers like for list, list sort() is present.

How sort() Works?

The sort() function is implemented using the Intro Sort Algorithm. It is the combination of three standard sorting algorithms: insertion sort, quick sort and heap sort. It is chooses the best algorithm that fits the given case. Refer to this article to know more – Internal Working of STL sort() Function

Can we instruct sort() to use any particular algorithm?