std::sort() in C++ STL

Read Courses Practice Video

We have discussed qsort() in C. C++ STL provides a similar function sort that sorts a vector or array (items with random access)

It generally takes two parameters, the first one being the point of the array/vector from where the sorting needs to begin and the second parameter being the length up to which we want the array/vector to get sorted. The third parameter is optional and can be used in cases such as if we want to sort the elements lexicographically.

By default, the sort() function sorts the elements in ascending order.

Below is a simple program to show the working of sort().

CPP

```cpp
// C++ program to demonstrate default behaviour of
// sort() in STL.
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);

    /*Here we take two parameters, the beginning of the
    array and the length n upto which we want the array to
    be sorted*/
    sort(arr, arr + n);

    cout << "\nArray after sorting using "
            "default sort is : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```

Output

```
Array after sorting using default sort is :
0 1 2 3 4 5 6 7 8 9
```

Time Complexity: O(N log N)

Auxiliary Space: O(1)

How to sort in descending order?

sort() takes a third parameter that is used to specify the order in which elements are to be sorted. We can pass the “greater()” function to sort in descending order. This function does a comparison in a way that puts greater elements before.

CPP

```cpp
// C++ program to demonstrate descending order sort using
// greater<>().
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);

    sort(arr, arr + n, greater<int>());

    cout << "Array after sorting : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```

Output

```
Array after sorting :
9 8 7 6 5 4 3 2 1 0
```

Time Complexity: O(N log N)

Auxiliary Space: O(1)

Sort the array only in the given range: To deal with such types of problems we just have to mention the range inside the sort function.

Below is the implementation of above case:

C++

```cpp
// C++ program to demonstrate sort()
```

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

int main()
{
    int arr[] = { 0, 1, 5, 8, 9, 6, 7, 3, 4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // Sort the elements which lies in the range of 2 to
    // (n-1)
    sort(arr + 2, arr + n);

    cout << "Array after sorting : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
// This code is contributed by Suruchi Kumari
```

Output

```
Array after sorting :
0 1 2 3 4 5 6 7 8 9
```

Time Complexity: O(N log N)

Auxiliary Space: O(1)

How to sort in a particular order?
We can also write our own comparator function and pass it as a third parameter. This “comparator” function returns a value; convertible to bool, which basically tells us whether the passed “first” argument should be placed before the passed “second” argument or not.
For eg: In the code below, suppose intervals {6,8} and {1,9} are passed as arguments in the “compareInterval” function(comparator function). Now as i1.first (=6) < i2.first (=1), so our function returns “false”, which tells us that “first” argument should not be placed before “second” argument and so sorting will be done in order like {1,9} first and then {6,8} as next.

CPP

```
// A C++ program to demonstrate
// STL sort() using
// our own comparator
#include <bits/stdc++.h>
using namespace std;

// An interval has a start
// time and end time
struct Interval {
    int start, end;
};

// Compares two intervals
// according to starting times.
bool compareInterval(Interval i1, Interval i2)
{
    return (i1.start < i2.start);
}

int main()
{
    Interval arr[]
        = { { 6, 8 }, { 1, 9 }, { 2, 4 }, { 4, 7 } };
    int n = sizeof(arr) / sizeof(arr[0]);

    // sort the intervals in increasing order of
    // start time
    sort(arr, arr + n, compareInterval);

    cout << "Intervals sorted by start time : \n";
    for (int i = 0; i < n; i++)
        cout << "[" << arr[i].start << "," << arr[i].end
             << "] ";

    return 0;
}
```

Output

```
Intervals sorted by start time :
[1,9] [2,4] [4,7] [6,8]
```

The time complexity of std::sort() is:

1. Best Case – O(N log N)
2. Average Case – O(N log N)
3. Worst-Case – O(N log N)

Space Complexity: It may use O(log N) auxiliary space.

C++

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

template <class T>
```

```
class Comparator { // we pass an object of this class as
                   // third arg to sort function...
public:
    bool operator()(T x1, T x2)
    {
        return x1 < x2;
    }
};

template <class T> bool funComparator(T x1, T x2)
{ // return type is bool
    return x1 <= x2;
}

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

int main()
{
    int a[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int asize = sizeof(a) / sizeof(int);
    cout << "The array before sorting is : ";
    show(a, asize);
    cout << endl << "The array after sorting is(asc) :";
    sort(a, a + asize);
    show(a, asize);
    cout << endl << "The array after sorting is(desc) :";
    sort(a, a + asize, greater<int>());
    show(a, asize);
    cout << endl
         << "The array after sorting is(asc but our "
            "comparator class) :";
    sort(a, a + asize, Comparator<int>());
    show(a, asize);
    cout << endl
         << "The array after sorting is(asc but our "
            "comparator function) :";
    sort(a, a + asize, funComparator<int>);
    show(a, asize);

    return 0;
}
```

Output

```
The array before sorting is : 1 5 8 9 6 7 3 4 2 0
The array after sorting is(asc) :0 1 2 3 4 5 6 7 8 9
The array after sorting is(desc) :9 8 7 6 5 4 3 2 1 0
The array after sorting is(asc but our comparator class) :0 1 2 3 4 5 6 7 8 9
The array after sorting is(asc but our comparator function) :0 1 2 3 4 5 6 7 8 9
```

Time Complexity: O(N log N)
Auxiliary Space: O(1)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Last Updated : 23 Sep, 2023