std::sort() in C++ STL

STL provides a sort() function which generally takes two parameters, the first 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 second 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.

```
// C++ program to demonstrate default behaviour of
// sort() in STL.

#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

int main() {
    vector<int> vect { 1, 5, 8, 9, 6, 7, 3, 4, 2, 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(vect.begin(), vect.end());

    cout << "\nVector after sorting using "
        "default sort is: [ ";
    for(int num:vect)
        cout << num << " ";
    cout << "]";

    return 0;
}

/*
Output:
Vector 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

Below is the implementation of above case:

```
// C++ program to demonstrate default behaviour of
// sort() in STL.

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

int main() {
    vector<int> vect { 1, 5, 8, 9, 6, 7, 3, 4, 2, 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(vect.begin(), vect.end(), greater<int>());

    cout << "\nVector after sorting using "
        "default sort is: [ ";
    for(int num:vect)
        cout << num << " ";
    cout << "]";

    return 0;
}

/*
Output:
Vector after sorting using default sort is: [ 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++ program to demonstrate default behaviour of
// sort() in STL.

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

int main() {
    vector<int> vect { 0, 1, 5, 8, 9, 6, 7, 3, 4, 2 };

    /*Here we take two parameters, the beginning of the
    array and the length n upto which we want the array to
    be sorted*/
    sort(vect.begin() + 2, vect.end());

    cout << "\nVector after sorting using "
        "default sort is: [ ";
    for(int num:vect)
        cout << num << " ";
    cout << "]";

    return 0;
}

/*
Output:
Vector 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 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. sort the vector according to the starting times of each pair using a lambda function as the comparator.

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

int main() {
    vector<pair<int, int>> nums{ { 6, 8 }, { 1, 9 }, { 2, 4 }, { 4, 7 } };
    int n = nums.size();

    // Sort the vector of pairs according to the starting times
    sort(nums.begin(), nums.end(), [](pair<int, int> a, pair<int, int> b) {
        return a.first < b.first;
    });

    cout << "Sorted nums vector according to starting times: { ";
    for (int i = 0; i < n; ++i) {
        cout << "{ " << nums[i].first << ", " << nums[i].second << " }";
        if (i != n - 1)
            cout << ", ";
    }
    cout << " }";

    return 0;
}


/*
Output:
Sorted nums vector according to starting times: { { 1, 9 }, { 2, 4 }, { 4, 7 }, { 6, 8 } }
*/
```

Time Complexity: O(N log N)

Auxiliary Space: O(1)