1. **[sort](https://www.geeksforgeeks.org/sort-c-stl/)(first\_iterator, last\_iterator)** – To sort the given vector.
2. **sort(first\_iterator, last\_iterator, greater<int>())** – To sort the given container/vector in descending order
3. **reverse(first\_iterator, last\_iterator)** – To reverse a vector. ( if ascending -> descending  OR  if descending -> ascending)
4. **\*max\_element (first\_iterator, last\_iterator)** – To find the maximum element of a vector.
5. **\*min\_element (first\_iterator, last\_iterator)**– To find the minimum element of a vector.
6. **accumulate(first\_iterator, last\_iterator, initial value of sum)** – Does the summation of vector elements

// A C++ program to demonstrate working of sort(),

// reverse()

#include <algorithm>

#include <iostream>

#include <vector>

#include <numeric> //For accumulate operation

using namespace std;

int main()

{

// Initializing vector with array values

int arr[] = {10, 20, 5, 23 ,42 , 15};

int n = sizeof(arr)/sizeof(arr[0]);

vector<int> vect(arr, arr+n);

cout << "Vector is: ";

for (int i=0; i<n; i++)

cout << vect[i] << " ";

// Sorting the Vector in Ascending order

sort(vect.begin(), vect.end());

cout << "\nVector after sorting is: ";

for (int i=0; i<n; i++)

cout << vect[i] << " ";

// Sorting the Vector in Descending order

sort(vect.begin(),vect.end(), greater<int>());

cout << "\nVector after sorting in Descending order is: ";

for (int i=0; i<n; i++)

cout << vect[i] << " ";

// Reversing the Vector (descending to ascending , ascending to descending)

reverse(vect.begin(), vect.end());

cout << "\nVector after reversing is: ";

for (int i=0; i<n; i++)

cout << vect[i] << " ";

cout << "\nMaximum element of vector is: ";

cout << \*max\_element(vect.begin(), vect.end());

cout << "\nMinimum element of vector is: ";

cout << \*min\_element(vect.begin(), vect.end());

// Starting the summation from 0

cout << "\nThe summation of vector elements is: ";

cout << accumulate(vect.begin(), vect.end(), 0);

return 0;

}

1. **find(first\_iterator, last\_iterator, x)** – Returns an iterator to the first occurrence of x in vector and points to last address of vector ((name\_of\_vector).end()) if element is not present in vector.

// C++ program to demonstrate working of count()

// and find()

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

int main()

{

// Initializing vector with array values

int arr[] = {10, 20, 5, 23 ,42, 20, 15};

int n = sizeof(arr)/sizeof(arr[0]);

vector<int> vect(arr, arr+n);

cout << "Occurrences of 20 in vector : ";

// Counts the occurrences of 20 from 1st to

// last element

**cout << count(vect.begin(), vect.end(), 20);**

// find() returns iterator to last address if

// element not present

find(vect.begin(), vect.end(),5) != vect.end()?cout << "\nElement found":cout << "\nElement not found";

return 0;

}

[**binary\_search**](https://www.geeksforgeeks.org/binary-search-algorithms-the-c-standard-template-library-stl/)**(first\_iterator, last\_iterator, x)** – Tests whether x exists in sorted vector or not.

**9. lower\_bound(first\_iterator, last\_iterator, x)** – returns an iterator pointing to the first element in the range [first,last) which         has a value not less than ‘x’.

**10. upper\_bound(first\_iterator, last\_iterator, x)** – returns an iterator pointing to the first element in the range [first,last)                  which has a value greater than ‘x’.

// C++ program to demonstrate working of lower\_bound()

// and upper\_bound().

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

int main()

{

// Initializing vector with array values

int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};

int n = sizeof(arr)/sizeof(arr[0]);

vector<int> vect(arr, arr+n);

// Sort the array to make sure that lower\_bound()

// and upper\_bound() work.

sort(vect.begin(), vect.end());

// Returns the first occurrence of 20

auto q = lower\_bound(vect.begin(), vect.end(), 20);

// Returns the last occurrence of 20

auto p = upper\_bound(vect.begin(), vect.end(), 20);

cout << "The lower bound is at position: ";

cout << q-vect.begin() << endl;

cout << "The upper bound is at position: ";

cout << p-vect.begin() << endl;

return 0;

}

The lower bound is at position: 3

The upper bound is at position: 5

1. **arr.erase(position to be deleted)** – This erases selected element in vector and shifts and resizes the vector elements accordingly.
2. **arr.erase(unique(arr.begin(),arr.end()),arr.end())** – This erases the duplicate occurrences in sorted vector in a single line.

// C++ program to demonstrate working

// of erase

#include <algorithm>

#include <bits/stdc++.h>

#include <iostream>

#include <vector>

using namespace std;

int main()

{

// Initializing vector with array values

int arr[] = { 5, 10, 15, 20, 20, 23, 42, 45 };

int n = sizeof(arr) / sizeof(arr[0]);

vector<int> vect(arr, arr + n);

cout << "Given Vector is:\n";

for (int i = 0; i < n; i++)

cout << vect[i] << " ";

vect.erase(find(vect.begin(),vect.end(),10));

cout << "\nVector after erasing element:\n";

for (int i = 0; i < vect.size(); i++)

cout << vect[i] << " ";

vect.erase(unique(vect.begin(), vect.end()),

vect.end());

cout << "\nVector after removing duplicates:\n";

for (int i = 0; i < vect.size(); i++)

cout << vect[i] << " ";

return 0;

}

Given Vector is:

5 10 15 20 20 23 42 45

Vector after erasing element:

5 15 20 20 23 42 45

Vector after removing duplicates:

5 15 20 23 42 45

**3. next\_permutation(first\_iterator, last\_iterator)**– This modified the vector to its next permutation.

**4. prev\_permutation(first\_iterator, last\_iterator)**– This modified the vector to its previous permutation.

// C++ program to demonstrate working

// of next\_permutation()

// and prev\_permutation()

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

int main()

{

// Initializing vector with array values

int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};

int n = sizeof(arr)/sizeof(arr[0]);

vector<int> vect(arr, arr+n);

cout << "Given Vector is:\n";

for (int i=0; i<n; i++)

cout << vect[i] << " ";

// modifies vector to its next permutation order

next\_permutation(vect.begin(), vect.end());

cout << "\nVector after performing next permutation:\n";

for (int i=0; i<n; i++)

cout << vect[i] << " ";

prev\_permutation(vect.begin(), vect.end());

cout << "\nVector after performing prev permutation:\n";

for (int i=0; i<n; i++)

cout << vect[i] << " ";

return 0;

}

Given Vector is:

5 10 15 20 20 23 42 45

Vector after performing next permutation:

5 10 15 20 20 23 45 42

Vector after performing prev permutation:

5 10 15 20 20 23 42 45

The **5. distance(first\_iterator,desired\_position)** – It returns the distance of desired position from the first iterator.This function               is very useful while finding the index.lower bound is at position:

// C++ program to demonstrate working of distance()

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

int main()

{

// Initializing vector with array values

int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};

int n = sizeof(arr)/sizeof(arr[0]);

vector<int> vect(arr, arr+n);

// Return distance of first to maximum element

cout << "Distance between first to max element: ";

cout << distance(vect.begin(),

max\_element(vect.begin(), vect.end()));

return 0;

}

Distance between first to max element: 7