Algorithms

The library defines a large number of functions that are specially suited to be used on a large number of elements at a time or say a range. Now let's straightway take a look at these functions.

Non-modifying sequence operations:

Function	Description
all_of	The following function tests a condition to all the elements of the range.
any_of	The following function tests a condition to some or any of the elements of the range
none_of	The following function checks if none of the elements follow the condition or not.
for_each	The function applies an operation to all the elements of the range.
find	The function finds a value in the range.
find_if	The function finds for an element in the range.
find_if_not	The function finds an element in the range but in the opposite way as the above one.
find_end	The function is used to return the last element of the range.
find_first_of	The function finds for the element that satisfies a condition and occurs at the first.
adjacent_find	The function makes a search for finding the equal and adjacent elements in a range.
count	The function returns the count of a value in the range.
count_if	The function returns the count of values that satisfies a condition.
mismatch	The function returns the value in sequence which is the first mismatch.
equal	The function is used to check if the two ranges have all elements equal.
is_permutation	The function checks whether the range in reference is a permutation of some other range.
search	The function searches for the subsequence in a range.
search_n	The function searches the range for the occurrence of an element.

Modifying sequence operations

Function	Description
copy	The function copies the range of elements.
copy_n	The function copies n elements of the range
copy_if	The function copies the elements of the range if a certain condition is fulfilled.
copy_backward	The function copies the elements in a backward order
move	The function moves the ranges of elements.
move_backward	The function moves the range of elements in the backward order
swap	The function swaps the value of two objects.
swap_ranges	The function swaps the value of two ranges.
iter_swap	The function swaps the values of two iterators under reference.
transform	The function transforms all the values in a range.
replace	The function replaces the values in the range with a specific value.

replace_if	The function replaces the value of the range if a certain condition is fulfilled.
replace_copy	The function copies the range of values by replacing with an element.
replace_copy_if	The function copies the range of values by replacing with an element if a certain condition is fulfilled.
fill	The function fills the values in the range with a value.
fill_n	The function fills the values in the sequence.
generate	The function is used for the generation of values of the range.
generate_n	The function is used for the generation of values of the sequence.
remove	The function removes the values from the range.
remove_if	The function removes the values of the range if a condition is fulfilled.
remove_copy	The function copies the values of the range by removing them.
remove_copy_if	The function copies the values of the range by removing them if a condition is fulfilled.
unique	The function identifies the unique element of the range.
unique_copy	The function copies the unique elements of the range.
reverse	The function reverses the range.
reverse_copy	The function copies the range by reversing values.
rotate	The function rotates the elements of the range in left direction.
rotate_copy	The function copies the elements of the range which is rotated left.
random_shuffle	The function shuffles the range randomly.
shuffle	The function shuffles the range randomly with the help of a generator.

Partitions

Function	Description
is_partitioned	The function is used to deduce whether the range is partitioned or not.
partition	The function is used to partition the range.
stable_partition	The function partitions the range in two stable halves.
partition_copy	The function copies the range after partition.
partition_point	The function returns the partition point for a range.

Sorting

Function	Description
sort	The function sorts all the elements in a range.
stable_sort	The function sorts the elements in the range maintaining the relative equivalent order.
partial_sort	The function partially sorts the elements of the range.
partial_sort_copy	The function copies the elements of the range after sorting it.
is_sorted	The function checks whether the range is sorted or not.
is_sorted_until	The function checks till which element a range is sorted.
nth_element	The functions sorts the elements in the range.

Binary search

Function	Description
lower_bound	Returns the lower bound element of the range.
upper_bound	Returns the upper bound element of the range.
equal_range	The function returns the subrange for the equal elements.
binary_search	The function tests if the values in the range exists in a sorted sequence or not.

Merge

Function	Description
merge	The function merges two ranges that are in a sorted order.
inplace_merge	The function merges two consecutive ranges that are sorted.
includes	The function searches whether the sorted range includes another range or not.
set_union	The function returns the union of two ranges that is sorted.
set_intersection	The function returns the intersection of two ranges that is sorted.
set_difference	The function returns the difference of two ranges that is sorted.
set_symmetric_difference	The function returns the symmetric difference of two ranges that is sorted.

Heap

Function	Description
push_heap	The function pushes new elements in the heap.
pop_heap	The function pops new elements in the heap.
make_heap	The function is used for the creation of a heap.
sort_heap	The function sorts the heap.
is_heap	The function checks whether the range is a heap.
is_heap_until	The function checks till which position a range is a heap.

Min/Max

Function	Description
min	Returns the smallest element of the range.
max	Returns the largest element of the range.
minmax	Returns the smallest and largest element of the range.
min_element	Returns the smallest element of the range.
max_element	Returns the largest element of the range.
minmax_element	Returns the smallest and largest element of the range.

Other functions

Function	Description
lexicographical_comapre	The function performs the lexicographical less-than comparison.
next_permutation	The function is used for the transformation of range into the next permutation.
perv_permutation	The function is used for the transformation of range into the previous permutation.

Example1: 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;
}
```

Output

```
Vector is: 10 20 5 23 42 15
Vector after sorting is: 5 10 15 20 23 42
Vector after sorting in Descending order is: 42 23 20 15 10 5
Vector after reversing is: 5 10 15 20 23 42
Maximum element of vector is: 42
```

Minimum element of vector is: 5
The summation of vector elements is: 115

Example 2: 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;
}
```

Output

Occurrences of 20 in vector : 2
Element found

Example 3: 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;
}
```

Output

```
The lower bound is at position: 3
The upper bound is at position: 5
```

Example 4: 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;
}
```

Output

```
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

Example 5: 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;
}
```

Output

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

Example 6: 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;
}
```

Output

Distance between first to max element: 7