***Binary Search Functions in C++ STL***

So far, we have discussed the Binary Search algorithm and its implementation by writing a function. The C++ standard template library has some built-in functions that can perform Binary Search operation directly on a sequential list or container.  
  
Some of these functions are:

* **binary\_search()**
* **upper\_bound()**
* **lower\_bound()**

**binary\_search()**

This function only checks whether a particular element is present in a sorted container. It accepts the starting iterator, ending iterator and the element to be checked as parameters and returns True if the element is present otherwise False.  
  
**Syntax**:

binary\_search(start\_ptr, end\_ptr, ele)

Below program illustrate the working of binary\_search() function with both Arrays and Vectors:  
CPP

// C++ code to demonstrate the working

// of binary\_search()

#include<bits/stdc++.h>

using namespace std;

int main()

{

/\*\*\* USING binary\_search() ON VECTOR \*\*\*/

// initializing vector of integers

vector<int> vec = {10, 15, 20, 25, 30, 35};

// using binary\_search to check if 15 exists

if (binary\_search(vec.begin(), vec.end(), 15))

cout << "15 exists in vector";

else

cout << "15 does not exist";

cout << endl;

/\*\*\* USING binary\_search() ON ARRAYS \*\*\*/

int arr[] = {10, 15, 20, 25, 30, 35};

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

// using binary\_search to check if 20 exists

if (binary\_search(arr, arr + n, 20))

cout << "20 exists in Array";

else

cout << "20 does not exist";

return 0;

}

**Output**:

15 exists in vector

20 exists in Array

**Note**: This function only checks if the element is present or not, it does not give any information about the location of the element if it exists.

**upper\_bound()**

The upper\_bound() function is used to find the upper bound of an element present in a container. That is it finds the location of an element just greater than a given element in a container. This function accepts the start iterator, end iterator and the element to be checked as parameters and returns the iterator pointing to the element just greater than the element passed as the parameter. If there does not exist any such element than the function returns an iterator pointing to the last element.  
  
**Syntax**:

upper\_bound(first\_itr, last\_itr, ele)

**Return Value**: Returns an iterator pointing to the element just greater than *ele*.  
  
**Examples**:

**Input** : 10 20 30 30 40 50

**Output** : upper\_bound for element 30 will return

an iterator pointing to the element 40.

**Input** : 10 20 30 40 50

**Output** : upper\_bound for element 45 will return

an iterator pointing to the element 50.

**Input** : 10 20 30 40 50

**Output** : upper\_bound for element 60 will

return end iterator.

**Note**: We can calculate the exact index position of the elements by subtracting the beginning iterator from the returned iterator.  
  
Below program illustrate the working of upper\_bound() function with both Vectors and Arrays:  
CPP

// CPP program to illustrate using upper\_bound()

// with vectors and arrays

#include <bits/stdc++.h>

using namespace std;

// Driver code

int main()

{

/\*\*\* USING upper\_bound() WITH VECTOR \*\*\*/

vector<int> v{ 10, 20, 30, 40, 50 };

// Print vector

cout << "Vector contains :";

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

cout << " " << v[i];

cout << "\n";

vector<int>::iterator upper1, upper2;

// std :: upper\_bound

upper1 = upper\_bound(v.begin(), v.end(), 35);

upper2 = upper\_bound(v.begin(), v.end(), 45);

cout << "\nUpper\_bound for element 35 is at position : "

<< (upper1 - v.begin());

cout << "\nUpper\_bound for element 45 is at position : "

<< (upper2 - v.begin())<<"\n\n";

/\*\*\* USING upper\_bound() WITH ARRAY \*\*\*/

int arr[] = { 10, 20, 30, 40, 50 };

// Print elements of array

cout << "Array contains :";

for (int i = 0; i < 5; i++)

cout << " " << arr[i];

cout << "\n";

// using upper\_bound

int up1 = upper\_bound(arr, arr+5, 35) - arr;

int up2 = upper\_bound(arr, arr+5, 45) - arr;

cout << "\nupper\_bound for element 35 is at position : "

<< (up1);

cout << "\nupper\_bound for element 45 is at position : "

<< (up2);

return 0;

}

**Output**:

Vector contains : 10 20 30 40 50

Upper\_bound for element 35 is at position : 3

Upper\_bound for element 45 is at position : 4

Array contains : 10 20 30 40 50

upper\_bound for element 35 is at position : 3

upper\_bound for element 45 is at position : 4

**lower\_bound()**

The lower\_bound() function is used to find the lower bound of an element present in a container. That is it finds the location of an element just smaller than a given element in a container. This function accepts the start iterator, end iterator and the element to be checked as parameters and returns the iterator pointing to the lower bound of the element passed as the parameter. If all elements of the container are smaller are less than the element passed, then it returns the last iterator.  
  
**Syntax**:

lower\_bound(first\_itr, last\_itr, ele)

**Return Value**: Returns an iterator pointing to the lower bound of the element *ele*. That is if *ele*exists in the container, it returns an iterator pointing to *ele*otherwise it returns an iterator pointing to the element just greater than ele.  
  
Below program illustrate the working of lower\_bound() function with both Vectors and Arrays:  
CPP

// CPP program to illustrate lower\_bound()

// for both vectors and array

#include <bits/stdc++.h>

using namespace std;

// Driver code

int main()

{

/\*\*\* USING lower\_bound() ON VECTORS \*\*\*/

vector<int> v{ 10, 20, 30, 40, 50 };

// Print vector

cout << "Vector contains :";

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

cout << " " << v[i];

cout << "\n";

vector<int>::iterator low1, low2;

// using lower\_bound()

low1 = lower\_bound(v.begin(), v.end(), 20);

low2 = lower\_bound(v.begin(), v.end(), 55);

cout << "\nlower\_bound for element 20 at position : "

<< (low1 - v.begin());

cout << "\nlower\_bound for element 55 at position : "

<< (low2 - v.begin());

/\*\*\* USING lower\_bound() ON ARRAYS \*\*\*/

int arr[] = { 10, 20, 30, 40, 50 };

// Print elements of array

cout << "\n\nArray contains :";

for (int i = 0; i < 5; i++)

cout << " " << arr[i];

cout << "\n";

// using lower\_bound()

int lb1 = lower\_bound(arr, arr + 5, 20) - arr;

int lb2 = lower\_bound(arr, arr + 5, 55) - arr;

cout << "\nlower\_bound for element 20 is at position : "

<< (lb1);

cout << "\nlower\_bound for element 55 is at position : "

<< (lb2);

return 0;

}

**Output**:

Vector contains : 10 20 30 40 50

lower\_bound for element 20 at position : 1

lower\_bound for element 55 at position : 5

Array contains : 10 20 30 40 50

lower\_bound for element 20 is at position : 1

lower\_bound for element 55 is at position : 5