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 have some built-in functions that can be used to 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 if a particular element is present in a sorted container or not. 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:

```
// 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 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 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 : " <
    cout << "\nlower_bound for element 55 is at position : " <

    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
```