Binary Search functions in C++ STL (binary_search, lower_bound and upper_bound)

Binary search is an important component in competitive programming or any algorithmic competition, having knowledge of shorthand functions reduces the time to code them. This searching only works when container is **sorted**. Related functions are discussed below.

1.binary_search(start_ptr, end_ptr, num) : This function returns boolean **true if the element is present** in the container, else returns false.

```
// C++ code to demonstrate the working of binary_search()

#include<bits/stdc++.h>
using namespace std;

int main()
{
    // initializing vector of integers
    vector<int> arr = {10, 15, 20, 25, 30, 35};

    // using binary_search to check if 15 exists
    if (binary_search(arr.begin(), arr.end(), 15))
       cout << "15 exists in vector";
    else
       cout << "15 does not exist";

    cout << endl;

    // using binary_search to check if 23 exists
    if (binary_search(arr.begin(), arr.end(), 23))
          cout << "23 exists in vector";
    else
          cout << "23 does not exist";

    cout << endl;
}
```

Output:

```
15 exists in vector
23 does not exist
```

2. lower_bound(start_ptr, end_ptr, num) : Returns pointer to "**position of num**" if container contains **1** occurrence of num. Returns pointer to "**first position of num**" if container contains **multiple occurrence** of num. Returns pointer to "**position of next higher number than num**" if container **does not contain occurrence** of num. Subtracting the pointer to 1st position i.e "**vect.begin()**" returns the actual index.

```
// C++ code to demonstrate the working of lower_bound()
#include<bits/stdc++.h>
using namespace std;

int main()
{
    // initializing vector of integers
    // for single occurrence
    vector<int> arr1 = {10, 15, 20, 25, 30, 35};

    // initializing vector of integers
    // for multiple occurrences
    vector<int> arr2 = {10, 15, 20, 20, 25, 30, 35};

    // initializing vector of integers
    // for no occurrence
    vector<int> arr3 = {10, 15, 25, 30, 35};

    // using lower_bound() to check if 20 exists
    // single occurrence
    // prints 2
    cout << "The position of 20 using lower_bound "
            " (in single occurrence case) : ";
    cout << lower_bound(arr1.begin(), arr1.end(), 20)
            - arr1.begin();

    cout << endl;

    // using lower_bound() to check if 20 exists
    // multiple occurrence
    // prints 2
    cout << "The position of 20 using lower_bound "
             "(in multiple occurrence case) : ";
    cout << lower_bound(arr2.begin(), arr2.end(), 20)
            - arr2.begin();

    cout << endl;

    // using lower_bound() to check if 20 exists
    // no occurrence
```

```
    // prints 2 ( index of next higher)
    cout << "The position of 20 using lower_bound "
             "(in no occurrence case) : ";
    cout << lower_bound(arr3.begin(), arr3.end(), 20)
            - arr3.begin();

    cout << endl;
}
```

Output:

```
The position of 20 using lower_bound (in single occurrence case) : 2
The position of 20 using lower_bound (in multiple occurrence case) : 2
The position of 20 using lower_bound (in no occurrence case) : 2
```

3. upper_bound(start_ptr, end_ptr, num) : Returns pointer to "**position of next higher number than num**" if container contains **1** occurrence of num. Returns pointer to "**first position of next higher number than last occurrence of num**" if container contains **multiple occurrence** of num. Returns pointer to "**position of next higher number than num**" if container **does not contain occurrence** of num. Subtracting the pointer to 1st position i.e "**vect.begin()**" returns the actual index.

```
// C++ code to demonstrate the working of upper_bound()
#include<bits/stdc++.h>
using namespace std;

int main()
{
    // initializing vector of integers
    // for single occurrence
    vector<int> arr1 = {10, 15, 20, 25, 30, 35};

    // initializing vector of integers
    // for multiple occurrences
    vector<int> arr2 = {10, 15, 20, 20, 25, 30, 35};

    // initializing vector of integers
    // for no occurrence
    vector<int> arr3 = {10, 15, 25, 30, 35};


    // using lower_bound() to check if 20 exists
    // single occurrence
    // prints 3
    cout << "The position of 20 using upper_bound"
           " (in single occurrence case) : ";
    cout << upper_bound(arr1.begin(), arr1.end(), 20)
            - arr1.begin();

    cout << endl;
```

```
    // using lower_bound() to check if 20 exists
    // multiple occurrence
    // prints 4
    cout << "The position of 20 using upper_bound "
            "(in multiple occurrence case) : ";
    cout << upper_bound(arr2.begin(), arr2.end(), 20)
            - arr2.begin();

    cout << endl;

    // using lower_bound() to check if 20 exists
    // no occurrence
    // prints 2 ( index of next higher)
    cout << "The position of 20 using upper_bound"
            " (in no occurrence case) : ";
    cout << upper_bound(arr3.begin(), arr3.end(), 20)
           - arr3.begin();

    cout << endl;
}
```

Output:

```
The position of 20 using upper_bound (in single occurrence case) : 3
The position of 20 using upper_bound (in multiple occurrence case) : 4
The position of 20 using upper_bound (in no occurrence case) : 2
```

This article is contributed by **Manjeet Singh(HBD.N)**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Meta Binary Search | One-Sided Binary Search

Interpolation search vs Binary search

Linear Search vs Binary Search

Binary Search in PHP

Binary Search

Variants of Binary Search

Binary Search a String

The Ubiquitous Binary Search | Set 1

Binary Search in Java

Binary Search In JavaScript

Binary Search using pthread

Binary Search (bisect) in Python

Randomized Binary Search Algorithm

A Problem in Many Binary Search Implementations

Binary Search in C++ Standard Template Library (STL)

Article Tags : C++ Binary Search cpp-algorithm-library cpp-binary-search STL

Practice Tags : STL Binary Search CPP

2

To-do Done

2.1

Based on **21** vote(s)

Feedback/ Suggest Improvement Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!