Climbing the Leaderboard editorial

Problem link: https://www.hackerrank.com/challenges/climbing-the-leaderboard/editorial

First, let’s build the leader board.
Example:
for the list of score below:

scores:	100	100	50	40	40	20	10
	0	*1*	*2*	*3*	*4*	*5*	*6*

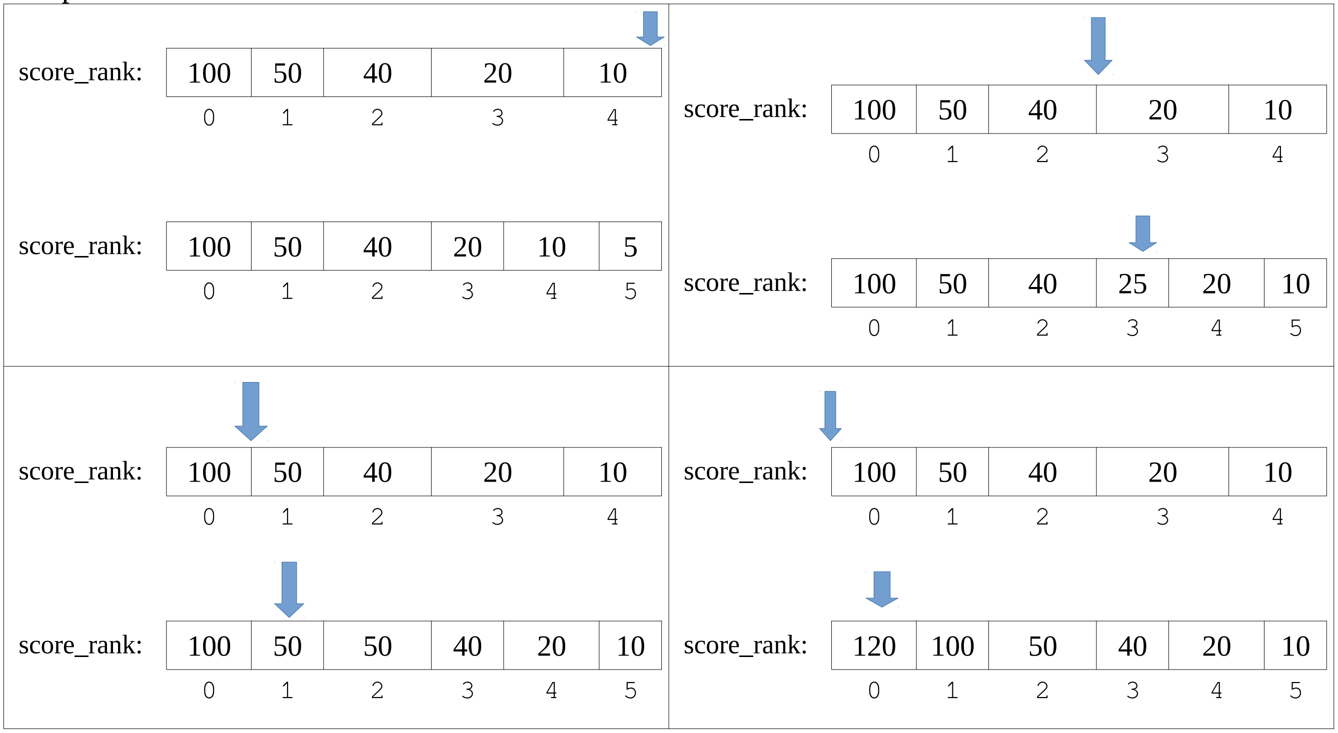
The leader board will be:

score_rank:	100	50	40	20	10
	0	*1*	*2*	*3*	*4*

let’s take this scores for Alice:

alice:	5	25	50	120
	0	*1*	*2*	*3*

The process will be:



score_rank:	100	50	40	20	10
	0	1	2	3	4

score_rank:	100	50	40	20	10	5
	0	1	2	3	4	5

score_rank:	100	50	40	20	10
	0	1	2	3	4

score_rank:	100	50	40	25	20	10
	0	1	2	3	4	5

score_rank:	100	50	40	20	10
	0	1	2	3	4

score_rank:	100	50	50	40	20	10
	0	1	2	3	4	5

score_rank:	100	50	40	20	10
	0	1	2	3	4

score_rank:	120	100	50	40	20	10
	0	1	2	3	4	5

As you see, for each Alice’s score, we lool for the last insertion position in the sorted by descending order array.

a is the size of the Alice’s score list

r is the size of the rank list

O(a x r) approach

For each Alice's score, compute the new score by pairwise comparisons.

We gonna use an algorithm similar to the insertion sort.

let’s take an example:
Scores for other players:

scores:	100	100	50	40	40	20	10
	0	1	2	3	4	5	6

Scores of Alice:

alice:	5	25	50	120
	0	1	2	3

The initial rank:

score_rank:	100	50	40	20	10
	0	1	2	3	4

In the code, we will not really insert the score in the $score_{rank}$ array, we gonna just compute the rank and add it to the *result* array.
For Alice’s score 5 , it will be inserted at the 6th position (index 5):

score_rank:	100	50	40	20	10	5
	0	1	2	3	4	5

result:	6
	0

For Alice’s score 25 , it will be inserted at the 4th position (index 3):

score_rank:	100	50	40	25	20	10
	0	1	2	3	4	5

result:	6	4
	0	1

For Alice’s score 50 , it exists (no need to insert) at the 2nd position (index 1):

score_rank:	100	50	40	20	10
	0	1	2	3	4

result:	6	4	2
	0	1	2

For Alice’s score 120 , it will be inserted at the 1st position (index 0):

score_rank:	120	100	50	40	20	10
	0	1	2	3	4	5

result:	6	4	2	1
	0	1	2	3

Whole c++ code:

```cpp
/*
Terminated due to timeout
O(alice_count x rank_list_size) -> quadratic
*/

#include <bits/stdc++.h>

using namespace std;

vector<int> build_rank(vector<int> scores){
  vector<int> score_rank;
  int scores_size = scores.size();
  for (int i = 0 ; i < scores_size - 1 ; ++i)
    if (scores[i] != scores[i+1]) score_rank.push_back(scores[i]);
  score_rank.push_back(scores[scores_size-1]);
  return score_rank;
}

// Complete the climbingLeaderboard function below.
vector<int> climbing_leaderboard(vector<int> scores, vector<int> alice) {
  // (1): Compute initial rank
  vector<int> score_rank = build_rank(scores);

  // (2): For each alice's score compute her rank
  // Complexity: O(alice_count x rank_list_size)
  vector<int> result;
  int rank_size = score_rank.size();
  for (auto alice_score: alice){
    int j = 0;
    while (j < rank_size && alice_score < score_rank[j])
      j++;

    result.push_back(j + 1);
  }

  return result;
}
```

```
int main(){
    int scores_count;
    cin >> scores_count;


    vector<int> scores(scores_count);

    for (int i = 0; i < scores_count; i++)
      cin >> scores[i];


    int alice_count;
    cin >> alice_count;

    vector<int> alice(alice_count);

    for (int i = 0; i < alice_count; i++)
      cin >> alice[i];


    vector<int> result = climbing_leaderboard(scores, alice);

   for (int i = 0; i < result.size(); i++) {
        cout << result[i];

        if (i != result.size() - 1) {
            cout << "\n";
        }
    }

    cout << "\n";

    return 0;
}
```

O(a x log r) approach

Search the score that is immediately greater or equal to the Alice’s score.

let’s take an example:

Scores for other players:

scores:	100	100	50	40	40	20	10
	0	1	2	3	4	5	6

Scores of Alice:

alice:	5	25	50	120
	0	1	2	3

The initial rank:

score_rank:	100	50	40	20	10
	0	1	2	3	4

For Alice’s score 5 , the score immediately greater or equal to 5 is 10:
index(5) = index(10) + 1 = 5, so the rank of 5 is 6.

result:	6
	0

For Alice’s score 25 , the score immediately greater or equal to 25 is 40, so the rank of 25 is:
index(25) = index of (40) + 1 = 2 + 1 = 3, so score 25 will be at rank 4.

result:	6	4
	0	1

For Alice’s score 50 , the score immediately greater or equal to 50 is 50, so the rank of 50 is:
index(50) = 1, so score 50 will be at rank 2

result:	6	4	2
	0	1	2

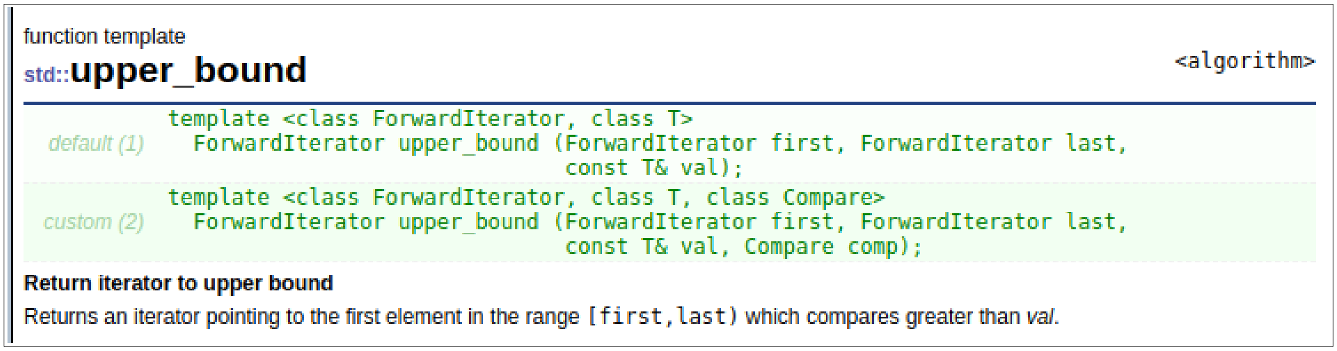
For Alice’s score 120 , the score immediately greater or equal to 120 doesn’t exist:
index(120) = 0 ; so 120 will be placed at rank 1.

result:	6	4	2	1
	0	1	2	3

The arrays are sorted by descending order, so in C++, we will use the STL function ***upper_bound*** combined to the binary function object class ***greater_equal<int>()*** to compute the score immediately greater or equal to Alice's score.

upper_bound

Greetz to *cbreak* ***and*** *Svitkona* ***on ##c++ channel on irc freenode***



function template

std::**upper_bound** <algorithm>

```
default (1)  template <class ForwardIterator, class T>
               ForwardIterator upper_bound (ForwardIterator first, ForwardIterator last,
                                            const T& val);
custom (2)   template <class ForwardIterator, class T, class Compare>
               ForwardIterator upper_bound (ForwardIterator first, ForwardIterator last,
                                            const T& val, Compare comp);
```

Return iterator to upper bound

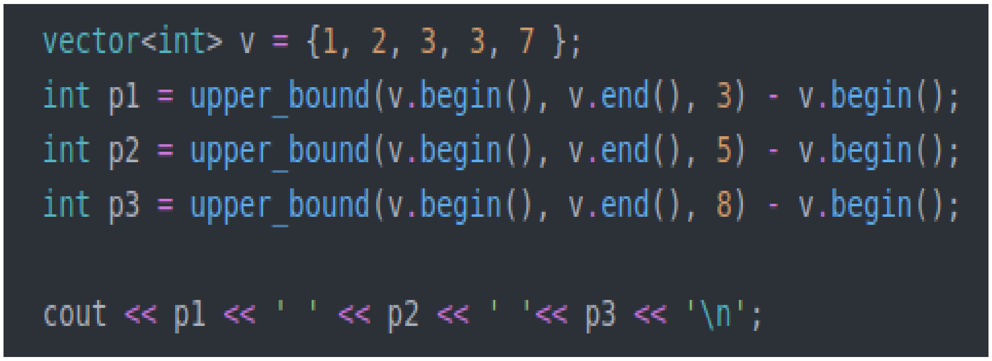
Returns an iterator pointing to the first element in the range `[first,last)` which compares greater than *val*.

http://www.cplusplus.com/reference/algorithm/upper_bound/

With *upper_bound* and *lower_bound*, without comparator function, the array must be sorted in ascending order.

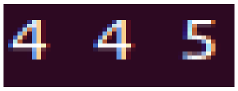
Is like to find the first insertion position.

example:



```
vector<int> v = {1, 2, 3, 3, 7 };
int p1 = upper_bound(v.begin(), v.end(), 3) - v.begin();
int p2 = upper_bound(v.begin(), v.end(), 5) - v.begin();
int p3 = upper_bound(v.begin(), v.end(), 8) - v.begin();

cout << p1 << ' ' << p2 << ' '<< p3 << '\n';
```

output



```
4 4 5
```

Is like to find the first insertion position.

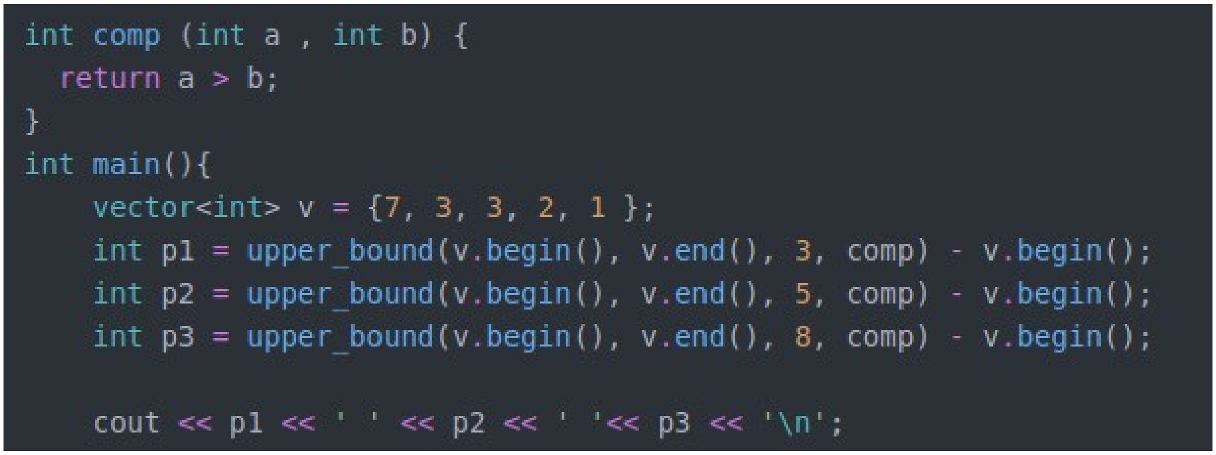
The 1st insertion position of 3 is 4.

The 1st insertion position of 5 is 4.

The 1st insertion position of 8 is 5.

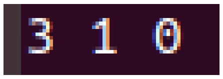
if the array is sorted by descending order, we must use the adequate comparator (because elements of the array are compared by >)

Is like to find the first insertion position.



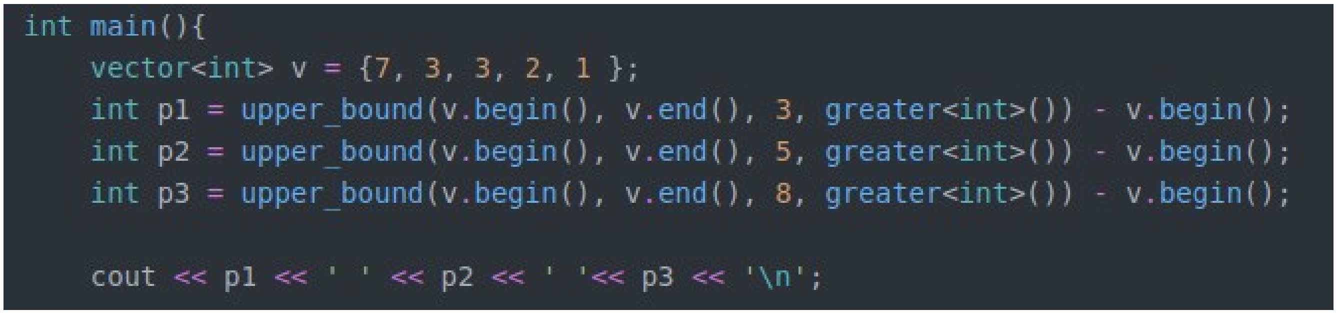
```
int comp (int a , int b) {
  return a > b;
}
int main(){
    vector<int> v = {7, 3, 3, 2, 1 };
    int p1 = upper_bound(v.begin(), v.end(), 3, comp) - v.begin();
    int p2 = upper_bound(v.begin(), v.end(), 5, comp) - v.begin();
    int p3 = upper_bound(v.begin(), v.end(), 8, comp) - v.begin();

    cout << p1 << ' ' << p2 << ' '<< p3 << '\n';
```

output



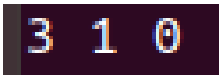
```
3 1 0
```

Instead using a personal function, you can use a built-in functions called: ***binary functions objects classes***



```
int main(){
    vector<int> v = {7, 3, 3, 2, 1 };
    int p1 = upper_bound(v.begin(), v.end(), 3, greater<int>()) - v.begin();
    int p2 = upper_bound(v.begin(), v.end(), 5, greater<int>()) - v.begin();
    int p3 = upper_bound(v.begin(), v.end(), 8, greater<int>()) - v.begin();

    cout << p1 << ' ' << p2 << ' '<< p3 << '\n';
```

output



```
3 1 0
```

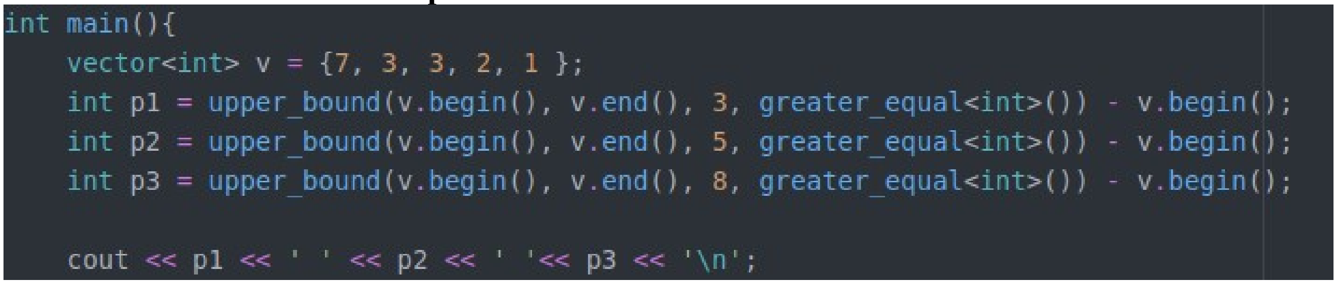
Is like to find the first insertion position.

The 1st insertion position of 3 is 4.

The 1st insertion position of 5 is 4.
The 1st insertion position of 8 is 5.

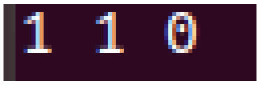
greater_equal<int>()

Is like to find the last insertion position.



```
int main(){
    vector<int> v = {7, 3, 3, 2, 1 };
    int p1 = upper_bound(v.begin(), v.end(), 3, greater_equal<int>()) - v.begin();
    int p2 = upper_bound(v.begin(), v.end(), 5, greater_equal<int>()) - v.begin();
    int p3 = upper_bound(v.begin(), v.end(), 8, greater_equal<int>()) - v.begin();

    cout << p1 << ' ' << p2 << ' '<< p3 << '\n';
```

out put



```
1 1 0
```

The last insertion position of 3 is 1.

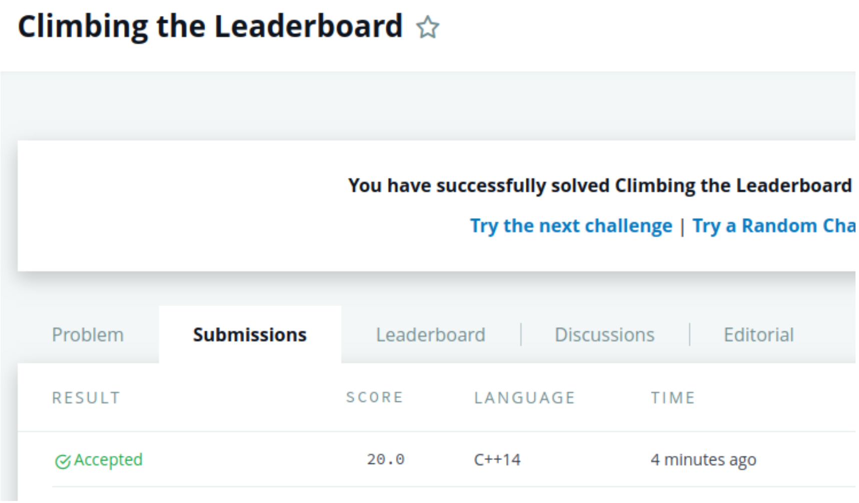
The last insertion position of 5 is 1.
The last insertion position of 8 is 0.

This what we need to compute the rank of the Alice’s score ins the list of scores

Whole c++ code:

```
/*
Successful
O(alice_count * log rank_size)
*/

#include <bits/stdc++.h>

using namespace std;

vector<int> build_rank(vector<int> scores){
  vector<int> score_rank;
  int scores_size = scores.size();
  for (int i = 0 ; i < scores_size - 1 ; ++i)
    if (scores[i] != scores[i+1]) score_rank.push_back(scores[i]);
  score_rank.push_back(scores[scores_size-1]);
  return score_rank;
}

vector<int> climbing_leaderboard(vector<int> scores, vector<int> alice) {
  // (1): Build the rank
  vector<int> score_rank = build_rank(scores);

  // (2): For each alice's score compute her rank:
      // By searching the last value in the rank array, which is greater or eqaul
to the alice's score.
      // To do that: we use:
            // upper_bound function (http://www.cplusplus.com/reference/algorithm/
upper_bound/)
            //  greater_equal<int>() binary function object class
(http://www.cplusplus.com/reference/functional/greater_equal/)
  // Complexity: O(alice_count * log rank_size)
  vector<int> result;
  int rank_size = score_rank.size();
  for (auto alice_score: alice)
    result.push_back(upper_bound(score_rank.begin(),score_rank.end(),alice_score,
greater_equal<int>()) - score_rank.begin() + 1);


  return result;
}
```

```
int main(){
    int scores_count;
    cin >> scores_count;


    vector<int> scores(scores_count);

    for (int i = 0; i < scores_count; i++)
      cin >> scores[i];


    int alice_count;
    cin >> alice_count;

    vector<int> alice(alice_count);

    for (int i = 0; i < alice_count; i++)
      cin >> alice[i];


    vector<int> result = climbing_leaderboard(scores, alice);

   for (int i = 0; i < result.size(); i++) {
        cout << result[i];

        if (i != result.size() - 1) {
            cout << "\n";
        }
    }

    cout << "\n";

    return 0;
}
```



Climbing the Leaderboard ☆

You have successfully solved Climbing the Leaderboard

Try the next challenge | Try a Random Cha

RESULT	SCORE	LANGUAGE	TIME
✓ Accepted	20.0	C++14	4 minutes ago