2. Implement Lower Bound and Upper Bound

04 June 2023 09:44 AM

Lower Bound: Smallest Index such that arr[ind] >= x.

arr[] = { 3, 5, 8, 15, 19 } n = 5

0 1 2 3 4

E.g.

if x = 8, then lower bound is now = 2 (index)

arr[ind] >= x

arr[2] >= 8

if x = 9, then lower bound is now = 3 (index)

arr[3] >= 9

15 >= 9

if x = 16, arr[4] >= 16

19 >= 16

x = 20, last hypothetically index = 5

now suppose,

arr[] = {3 5 8 15 19 19 19} n = 7

0 1 2 3 4 5 6

x = 19, then lb = 4 arr[4] >= 19

19 >= 19

arr[] = {1 2 3 3 7 8 9 9 9 11} x = 1

0 1 2 3 4 5 6 7 8 9

low ↑ mid ↑ high ↑

Even there is no element we will have hypothetical index which is last index.

ans = ∅, 4 mid = (0+9)/2 = 4 arr[4] >= x

7 >= 1

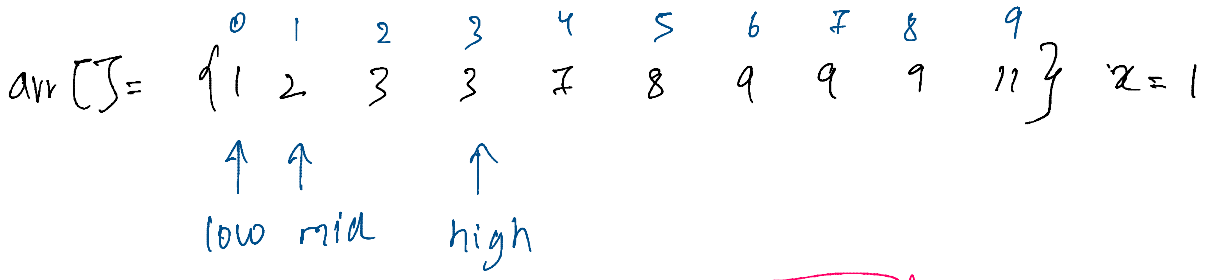
(you need to find the smallest) (probably can be ans so update it)

now eliminate the right space

high = mid - 1

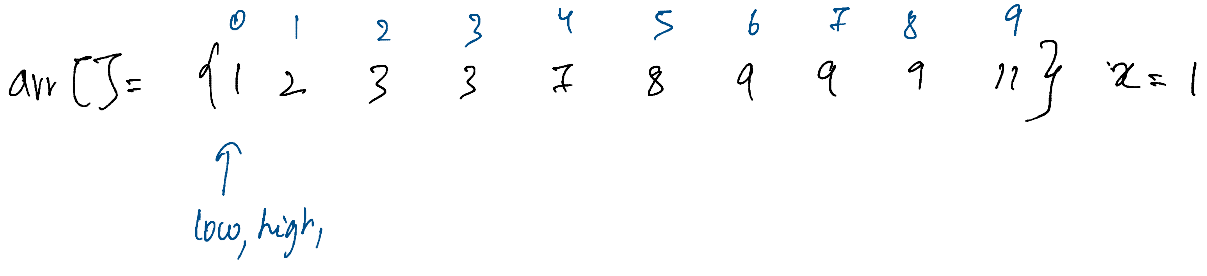
0 1 2 3 4 5 6 7 8 9

arr[] = {1 2 3 3 7 8 9 9 11 2} x = 1



arr[i] >= x

ans = 1, mid = 1, 2 >= 1 (can be ans so update it)



ans = 0, mid = 0, arr[0] >= 1

1 >= 1 (might be ans so update it)

now high will cross low, so you stop and return the ans, and the lower bound = 0

Lower Bound Implementation:

```
f(arr, target, n)
{ low=0, high = n-1;
```

```
ans = n;

while(low <= high)

{
    mid = (low + high) / 2;

    if (arr[mid] >= x)
        ans = mid; //may be an answer
        high = mid - 1;

    else
        low = mid + 1;

}

return ans;

}
```

```cpp
int lowerBound(vector<int> arr, int n, int x) {
  int low = 0, high = n - 1;
  int ans = n;

  while (low <= high) {
    int mid = (low + high) / 2;

    // Check if the element at the middle index is greater than or equal to the target
    if (arr[mid] >= x) {
      ans = mid;  // Update the potential lower bound.
      high = mid - 1;  // Adjust the search range to the left of the middle index.
    } else {
      low =
          mid + 1; // Adjust the search range to the right of the middle index.
    }
  }

  return ans;
}
```

Lower Bound C++ STL:

vector]

→ lb = lower_bound (arr.begin(), arr.end(), x) - arr.begin();

if it's arr

(arr , arr+n , x)

now if you asked to find out lb for particular index for eg from 2 to 7.

[2, 7]

lb = lower_bound (arr+2 , arr+7, x)

T.C ⇒ $O(\log_2 n)$

Upper Bound : Smallest index such that arr[ind] ≥ x

	0	1	2	3	4	5	6	7	8	9
arr[] =	2	3	6	7	8	8	11	11	11	12

x = 6 x = 11 x = 12 x = 13 x = 0

up = ind = 3 ⇒ 7 > 6 ind = 9 ind = 10 ind = 10 ind = 0

this is only change

```
if ( a[mid] >= x)
    ans = mid
    high = mid - 1
else
    low = mid + 1
```

Upper Bound C++ STL :

Vector]

→ ub = upper_bound (arr.begin(), arr.end(), x) − arr.begin();

if it's arr

(arr , arr+n , x)

```int upperBound(vector<int> &arr, int x, int n){
  int low = 0, high = n - 1;
  int ans = n;

  while (low <= high) {
    int mid = (low + high) / 2;

    // Check if the element at the middle index is greater than to the target.
    if (arr[mid] > x) {
      ans = mid;  // Update the potential lower bound.
      high = mid - 1;  // Adjust the search range to the left of the middle index.
    } else {
      low =
          mid + 1; // Adjust the search range to the right of the middle index.
    }
  }

  return ans;
}
```

Press Esc to exit full screen

Q) Search Insert Position :

Problem Statement Suggest Edit

You are given a sorted array *'arr'* of distinct values and a target value *'m'*. You need to search for the index of the target value in the array.

If the value is present in the array, then return its index.
If the value is not present, determine the index where it would be inserted in the array while maintaining the sorted order.

Example:

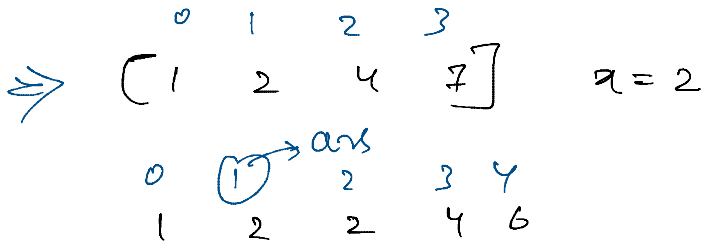
```
Input:  arr = [1, 2, 4, 7],  m = 6

Output: 3

Explanation: If the given array 'arr' is: [1, 2, 4, 7]
and m = 6. We insert m = 6 in the array and get 'arr'
as: [1, 2, 4, 6, 7]. The position of 6 is 3 (according
to 0-based indexing)
```

	0	1	2	3
arr[] =	1	2	4	7

x = 6

ans

0	1	2	3	4
1	2	4	6	7



It's similar to the lower bound. (arr[ind] >= x)

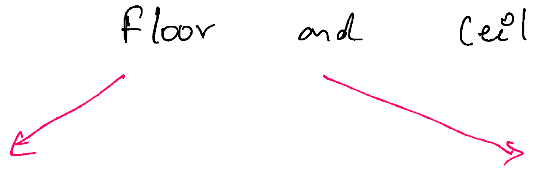
```cpp
int searchInsert(vector<int>& arr, int x)
{
  int n = arr.size();
  int low = 0, high = n - 1;
  int ans = n;

  while (low <= high) {
    int mid = (low + high) / 2;

    if (arr[mid] >= x) {
      ans = mid;
      high = mid - 1;
    } else {
      low =
          mid + 1;
    }
  }

  return ans;
}
```

View hints Last saved on 7:21:02 PM

Q) Floor and Ceil in Sorted Array

Problem Statement Suggest Edit

You're given an unsorted array A of N integers and an integer X. Find the floor and ceiling of X in Arr[0..N-1].
Floor of X is the largest element in the array which is smaller than or equal to X.
Ceiling of X is the smallest element in the array greater than or equal to X.



largest no in array <= x

smallest no in array >= x ⇒ lower_bound

largest no in array $\boxed{<= x}$ smallest no in array $\boxed{>= x}$ ⇒ lower_bound

arr[] = [10 20 30 40 50] x = 25

floor = 20 (20 <= 25)

ceil = 30 (30 >= 25)

arr[] = [10 20 25 30 40] x = 25

floor = 25 (25 <= 25)

ceil = 25 (25 >= 25)

Pseudo code for floor:

```
floor (arr, x)
{
  ans = -1;
  low = 0, high = n-1;
  while (low <= high)
  {
    mid = (low + high)/2;
    if (a[mid] <= mid)
      ans = a[mid]
      low = mid + 1
    else
      high = mid - 1;
```

```
    else
        high = mid - 1;
}
```

How is B.S making sure that we get the smallest index?

→ B.S eliminate left / right to find the smallest index

→ By default it does that.