***Count Occurrences in Sorted Array***

Given a sorted array arr[] and a number x, write a function that counts the occurrences of x in arr[]. Expected time complexity is O(Logn)

**Examples:**

**Input:** arr[] = {1, 1, 2, 2, 2, 2, 3,}, x = 2

**Output:** 4 // x (or 2) occurs 4 times in arr[]

**Input:** arr[] = {1, 1, 2, 2, 2, 2, 3,}, x = 3

**Output:** 1

**Input:** arr[] = {1, 1, 2, 2, 2, 2, 3,}, x = 1

**Output:** 2

**Input:** arr[] = {1, 1, 2, 2, 2, 2, 3,}, x = 4

**Output:** -1 // 4 doesn't occur in arr[]

**Method: Binary Search**

C++Java

import java.util.\*;

import java.io.\*;

import java.lang.\*;

class GFG

{

static int firstOcc(int arr[], int n, int x)

{

int low = 0, high = n - 1;

while(low <= high)

{

int mid = (low + high) / 2;

if(x > arr[mid])

low = mid + 1;

else if(x < arr[mid])

high = mid - 1;

else

{

if(mid == 0 || arr[mid - 1] != arr[mid])

return mid;

else

high = mid - 1;

}

}

return -1;

}

static int lastOcc(int arr[], int n, int x)

{

int low = 0, high = n - 1;

while(low <= high)

{

int mid = (low + high) / 2;

if(x > arr[mid])

low = mid + 1;

else if(x < arr[mid])

high = mid - 1;

else

{

if(mid == n - 1 || arr[mid + 1] != arr[mid])

return mid;

else

low = mid + 1;

}

}

return -1;

}

static int countOcc(int arr[], int n, int x)

{

int first = firstOcc(arr, n, x);

if(first == -1)

return 0;

else

return lastOcc(arr, n, x) - first + 1;

}

public static void main(String args[])

{

int arr[] = {10, 20, 20, 20, 40, 40}, n = 6;

int x = 20;

System.out.println(countOcc(arr, n, x));

}

}

**Output :**

3

**Time Complexity :**O(Log n)

**Space Complexity:** O(1)