***Index of first Occurrence and last Occurrence in Sorted Array***

Given a sorted array **arr[]** with possibly duplicate elements, the task is to find indexes of the first and last occurrences of an element **x** in the given array.

**Examples:**

***Input :****arr[] = {1, 3, 5, 5, 5, 5, 67, 123, 125}, x = 5*  
***Output****: First Occurrence = 2*  
*Last Occurrence = 5*

***Input :****arr[] = {1, 3, 5, 5, 5, 5, 7, 123, 125 }, x = 7*

***Output :****First Occurrence = 6*  
*Last Occurrence = 6*

**A** **Naive Approach:**

*The idea to solve this problem is iterate on the elements of given array and check given elements in an array and keep track of****first****and****last****occurrence of the found element’s index.*

**Below are the steps to implement the above idea:**

* Run a for loop and for i = 0 to n-1
* Take first = -1 and last = -1
* When we find an element first time then we update first = i
* We always update last=i whenever we find the element.
* We print first and last.

Below is the implementation of the above approach:

C++Java

// Java program to find first and last occurrence of

// an elements in given sorted array

import java.io.\*;

class GFG {

// Function for finding first and last occurrence

// of an elements

public static void findFirstAndLast(int arr[], int x)

{

int n = arr.length;

int first = -1, last = -1;

for (int i = 0; i < n; i++) {

if (x != arr[i])

continue;

if (first == -1)

first = i;

last = i;

}

if (first != -1) {

System.out.println("First Occurrence = "

+ first);

System.out.println("Last Occurrence = " + last);

}

else

System.out.println("Not Found");

}

public static void main(String[] args)

{

int arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };

int x = 8;

findFirstAndLast(arr, x);

}

}

**Output**

First Occurrence = 8

Last Occurrence = 9

**Time Complexity:** O(n)   
**Auxiliary Space:**O(1)

**An efficient approach using binary search:**

**1. For the first occurrence of a number**

*a) If (high >= low)*  
*b) Calculate  mid = low + (high – low)/2;*  
*c) If ((mid == 0 || x > arr[mid-1]) && arr[mid] == x)*  
*return mid;*  
*d) Else if (x > arr[mid])*  
*return first(arr, (mid + 1), high, x, n);*  
*e) Else*  
*return first(arr, low, (mid -1), x, n);*  
*f) Otherwise return -1;*

**2. For the last occurrence of a number**

*a) if (high >= low)*  
*b) calculate mid = low + (high – low)/2;*  
*c)if( ( mid == n-1 || x < arr[mid+1]) && arr[mid] == x )*  
*return mid;*  
*d) else if(x < arr[mid])*  
*return last(arr, low, (mid -1), x, n);*  
*e) else*  
*return last(arr, (mid + 1), high, x, n);*  
*f) otherwise return -1;*

Below is the implementation of the above approach:

C++Java

// Java program to find first and last occurrence of

// an elements in given sorted array

import java.io.\*;

class GFG {

/\* if x is present in arr[] then returns the index of

FIRST occurrence of x in arr[0..n-1], otherwise

returns -1 \*/

public static int first(int arr[], int low, int high,

int x, int n)

{

if (high >= low) {

int mid = low + (high - low) / 2;

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

&& arr[mid] == x)

return mid;

else if (x > arr[mid])

return first(arr, (mid + 1), high, x, n);

else

return first(arr, low, (mid - 1), x, n);

}

return -1;

}

/\* if x is present in arr[] then returns the index of

LAST occurrence of x in arr[0..n-1], otherwise

returns -1 \*/

public static int last(int arr[], int low, int high,

int x, int n)

{

if (high >= low) {

int mid = low + (high - low) / 2;

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

&& arr[mid] == x)

return mid;

else if (x < arr[mid])

return last(arr, low, (mid - 1), x, n);

else

return last(arr, (mid + 1), high, x, n);

}

return -1;

}

public static void main(String[] args)

{

int arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };

int n = arr.length;

int x = 8;

System.out.println("First Occurrence = "

+ first(arr, 0, n - 1, x, n));

System.out.println("Last Occurrence = "

+ last(arr, 0, n - 1, x, n));

}

}

**Output**

First Occurrence = 8

Last Occurrence = 9

**Time Complexity:**O(log n)   
**Auxiliary Space:**O(1)

**An Iterative Implementation of Binary Search Solution :**

1. *For the****first occurrence****, we will first find the index of the number and then search again in the left subarray as long as we are finding the number.*
2. *For the****last occurrence****, we will first find the index of the number and then search again in the right subarray as long as we are finding the number*

**First occurrence:**

* Do while low <= high:
  + First, find the mid element
  + Check if the arr[mid] > x then the first element will occur on the left side of mid. So, bring the high pointer to mid – 1
  + Check if the arr[mid] < x then the first element will occur on the right side of mid. So, bring the low pointer to mid + 1
  + If arr[mid] == x then this may be the first element. So, update the result to mid and move the high pointer to mid – 1.
* Return the result.

**Last occurrence:**

* Do while low <= high:
  + First, find the mid element
  + Check if the arr[mid] > x then the last element will occur on the left side of mid. So, bring the low pointer to mid – 1
  + Check if the arr[mid] < x then the last element will occur on the right side of mid. So, bring the low pointer to mid + 1
  + If arr[mid] == x then this may be the last element. So, update the result to mid and move the low pointer to mid + 1.
* Finally, Return the result.

Below is the implementation of the above approach:

C++Java

// Java program to find first

// and last occurrences of a

// number in a given sorted array

import java.util.\*;

class GFG {

// if x is present in arr[] then

// returns the index of FIRST

// occurrence of x in arr[0..n-1],

// otherwise returns -1

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

{

int low = 0, high = n - 1, res = -1;

while (low <= high) {

// Normal Binary Search Logic

int mid = (low + high) / 2;

if (arr[mid] > x)

high = mid - 1;

else if (arr[mid] < x)

low = mid + 1;

// If arr[mid] is same as

// x, we update res and

// move to the left half.

else {

res = mid;

high = mid - 1;

}

}

return res;

}

// If x is present in arr[] then returns

// the index of LAST occurrence of x in

// arr[0..n-1], otherwise returns -1

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

{

int low = 0, high = n - 1, res = -1;

while (low <= high) {

// Normal Binary Search Logic

int mid = (low + high) / 2;

if (arr[mid] > x)

high = mid - 1;

else if (arr[mid] < x)

low = mid + 1;

// If arr[mid] is same as x,

// we update res and move to

// the right half.

else {

res = mid;

low = mid + 1;

}

}

return res;

}

// Driver program

public static void main(String[] args)

{

int arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };

int n = arr.length;

int x = 8;

System.out.println("First Occurrence = "

+ first(arr, x, n));

System.out.println("Last Occurrence = "

+ last(arr, x, n));

}

}

**Output**

First Occurrence = 8

Last Occurrence = 9

**Time Complexity**:O(log n)   
**Auxiliary Space:**O(1)