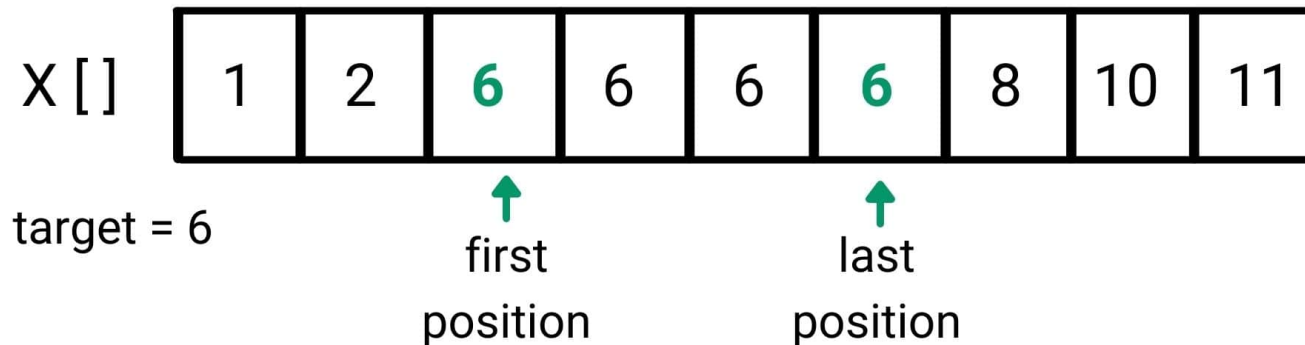




Find first and last positions of an element in a sorted array

Easy

First and **Last** positions of
an element in a sorted array



Asked in   

enjoyalgorithms.com

Difficulty: Easy, **Asked-in:** Google, Amazon, Microsoft

Key takeaway: This is a good interview problem to learn problem-solving using binary search. We are applying binary search twice to improve the time complexity over the brute force approach.

Let's understand the problem

Given an array of integers sorted in ascending order, write a code to find the first and last position of a given **target** value.

If the target is not found in the array, return [-1, -1].

We must design an algorithm with $O(\log n)$ time complexity.

Example 1

Input : $A[] = [-1, 1, 2, 2, 2, 5, 6, 12, 12]$, target = 2

Output : First Occurrence = 2, Last Occurrence = 4

Example 2

Input : $A[] = [21, 32, 51, 70, 71]$, target = 70

Output : First Occurrence = 3, Last Occurrence = 3

Discussed solution approaches

A brute force approach using linear search

An efficient approach by applying binary search twice

A brute force approach using a linear search

Solution Idea

The straightforward approach is to traverse the array and track the index of the first and last occurrence of the target. This approach is inefficient because we did not take advantage of the fact that the given array is sorted.

Solution Steps

1. Take an extra constant size array `firstLast[2]` to store the first and last occurrence of the target element. We will store the first occurrence at `firstLast[0]` and the last occurrence at `firstLast[1]`. At the start, we initialize both values with `-1`.
2. Now we traverse the array linearly from $i = 0$ to $n-1$ and update the `firstLast[0]` when we encounter the target value for the first time. To keep track of the last occurrence, we keep updating the value of the `firstLast[1]` every time we encounter the target value.
3. By the end of the loop, we return the `firstLast[]` array.

Solution Pseudocode

```
int[] firstLastOccurence(int A[], int n, int target)
{
    int firstLast[2] = [-1, -1]
    for (int i = 0; i < n; i = i + 1)
    {
        if (A[i] == target)
        {
            if (firstLast[0] == -1)
                firstLast[0] = i
            firstLast[1] = i
        }
    }
    return firstLast
}
```

Solution Analysis

We are running a single loop n time and doing an $O(1)$ operation at each iteration. So time complexity = $n * O(1)$ = $O(n)$. We are using constant extra space so that the space complexity would be $O(1)$.

An efficient approach using binary search

Solution Idea and Steps

Now critical questions are: can we improve the time complexity further? Can we use the sorted order property to enhance the efficiency of searching?

As we already know, searching any element in the sorted array, the idea of binary search works perfectly. It would take $O(\log n)$ time to search the target value in the sorted array. But the given problem is a little different from the binary search problem, where we need to search two occurrences of the target element. So how do we modify the standard binary search algorithm to solve it? Think.

The idea is simple: we can use binary search twice to solve the problem. The first binary search is to find the first occurrence of the target and the second binary search is to find the last occurrence of the target. Let's design an algorithm for both the steps separately.

Binary search for finding the first occurrence

Let's use the iterative binary search! We first initialize the $low = 0$ and $high = n - 1$ to track the left and right ends. Now we run a loop till $low \leq high$:

Calculate $mid = low + (high - low) / 2$

Now compare the value at mid with the target. The middle element will be the first occurrence in the two situations : 1) $target == A[mid]$ and $target > A[mid - 1]$ i.e. when first occurrence is present somewhere in the middle 2) $mid == 0$ and $A[mid] == target$ i.e. when the first occurrence is present at the first position. In both situations, we return mid -index as the first occurrence.

```
if ((mid == 0 || A[mid - 1] < target) && A[mid] == target)
    return mid
```

If ($\text{target} > A[\text{mid}]$), then we need to search the first occurrence in the right part because all the value in the left part is less than the target. Update $\text{low} = \text{mid} + 1$

But if both the above conditions are not satisfied, we need to search for the first occurrence in the left part (Think!). We update $\text{high} = \text{mid} - 1$

If we did not find the target value by the end of the loop, we return -1.

Pseudocode to find the first occurrence

```
int findFirstOccurrence(int A[], int n, int target)
{
    int low = 0, high = n - 1
    while (low <= high)
    {
        int mid = low + (high - low)/2
        if ((mid == 0 || A[mid - 1] < target) && A[mid] == target)
            return mid
        else if (target > A[mid])
            low = mid + 1
        else
            high = mid - 1
    }
}
```

```
    }  
    return -1  
}
```

Binary search for finding the last occurrence

Similarly, for finding the last occurrence, we also modify the binary search algorithm. We first initialize the low = 0 and high = n - 1 to track the left and right ends during the binary search loop. Now we run a loop till $low \leq high$:

Calculate $mid = low + (high - low)/2$

Now compare the value at mid with the target. The mid element will be the last occurrence in the two situations: 1) $target == A[mid]$ and $target < A[mid + 1]$ i.e. when the last occurrence is present somewhere in the middle 2) $mid == n-1$ and $A[mid] == target$ i.e. when the last occurrence is at the last position.

```
if ( (mid == n - 1 || A[mid + 1] > target) && A[mid] == target)  
    return mid
```

If ($target < A[mid]$), then we need to search the last occurrence in the left part because all the value in the right part is less than the target. Update $high = mid - 1$

But if both the above conditions are not satisfied, we need to search for the first occurrence in the right part (Think!). Update $low = mid + 1$

By the end of the loop, if we did not find the target value, return -1.

Pseudocode to find the last occurrence

```
int findLastOccurrence(int A[], int n, int target)
{
    int low = 0, high = n - 1
    while (low <= high)
    {
        int mid = low + (high - low)/2
        if ((mid == n - 1 || A[mid + 1] > target) && A[mid] == target)
            return mid
        else if (target < A[mid])
            high = mid - 1
        else
            low = mid + 1
    }
    return -1
}
```

Solution Pseudocode

```
int[] findFirstLastOccurrence(int A[], int n, int target)
{
```



```
int firstLast[2] = [-1, -1]
firstLast[0] = findFirstOccurrence(A, n, target)
firstLast[1] = findLastOccurrence(A, n, target)
return firstLast
}
```

Solution Analysis

Inside the function `firstLastOccurrence(A[], n, target)`, we are doing some constant extra operations and applying binary search twice. So time complexity = $O(1)$ + time complexity to search the first occurrence + time complexity to search the last occurrence = $O(1) + O(\log n) + O(\log n) = O(\log n)$.

We are using the implementation of iterative binary search, which takes $O(1)$ extra space. So space complexity = $O(1)$

Critical ideas to think!

Can we solve this problem using a single binary search?

How can we implement the above approach using a recursive binary search? What will be the space complexity?

Comparisons of time and space complexities

Using linear search: Time = $O(n)$, Space = $O(1)$

Using binary search: Time = $O(\log n)$, Space = $O(1)$

Coding problems to practice using binary search

Find Peak Element

Search a sorted 2D matrix

Find the square root of an integer

Find Smallest Letter Greater Than Target


Median of Two Sorted Arrays

Search in Rotated Sorted Array

Search a 2D Matrix II

Find Minimum in Rotated Sorted Array

Thanks to [Navtosh Kumar](#) for his contribution in creating the first version of the content. Please write comments if you find an error or you want to share more insights about the topic. Enjoy learning, Enjoy coding, Enjoy algorithms!

Author: Shubham Gautam 

Reviewer: EnjoyAlgorithms Team 

[binary-search](#)

[searching](#)

[amazon-interview-questions](#)

[microsoft-interview-questions](#)

[coding-interview-questions](#)



Share your Feedback

Email address

Message

Submit



More from EnjoyAlgorithms

Sort an array of 0s, 1s, and 2s —Dutch National Flag Problem

Given an array `a[]` consisting of 0s, 1s, and 2s. Write a program to sort the array of 0's, 1's, and 2's in ascending order. This is a famous coding interview...

Explore and Enjoy

Move Zeroes to End of an Array

Given an array `X[]` of `n` elements filled with several integers, some of them being zeroes, write a program to move all the zeroes to the end. This is an excellent...

Explore and Enjoy

Max Consecutive Ones

An input binary array `X[]` is given where all the elements are either 0 or 1. Write a program to find the maximum consecutive ones in this binary array.

Explore and Enjoy

Merge Sort Algorithm

Merge sort is one of the fastest comparison-based sorting algorithms, which works on the principle of the divide and conquer approach. The worst and best cas...

Explore and Enjoy

Minimum number of Jumps to reach End

An array of non-negative integers is given and the aim is to reach the last index in the minimum number of

Find whether an array is a subset of another array

We are given two integer arrays `X[]` and `Y[]`, write a

jumps. You are initially positioned at the first index of...

Explore and Enjoy

program to check whether array Y[] is a subset of array X[] or not. An array Y is a subset of another array X if...

Explore and Enjoy



Our weekly newsletter

Subscribe to get free weekly content on data structure and algorithms, machine learning, system design, oops and math.

Subscribe

Coding Interview

Machine Learning

System Design

OOPS Concepts

Latest Content

Success Stories

[Coding Interview Concepts](#)

[Coding Interview Guidance](#)

[Machine Learning Applications](#)

[System Design Concepts](#)

[Google Interview Questions](#)

[Microsoft Interview Questions](#)

[Coding Interview Questions](#)

[Machine Learning Concepts](#)

[Machine Learning Guidance](#)

[OOPS Design Concepts](#)

[Amazon Interview Questions](#)

[Facebook Interview Questions](#)

Follow us on:



© 2020 EnjoyAlgorithms, Inc. All rights reserved.