Published in The Startup

Tuan Nhu Dinh Follow

Apr 2, 2020 · 5 min read · · Listen

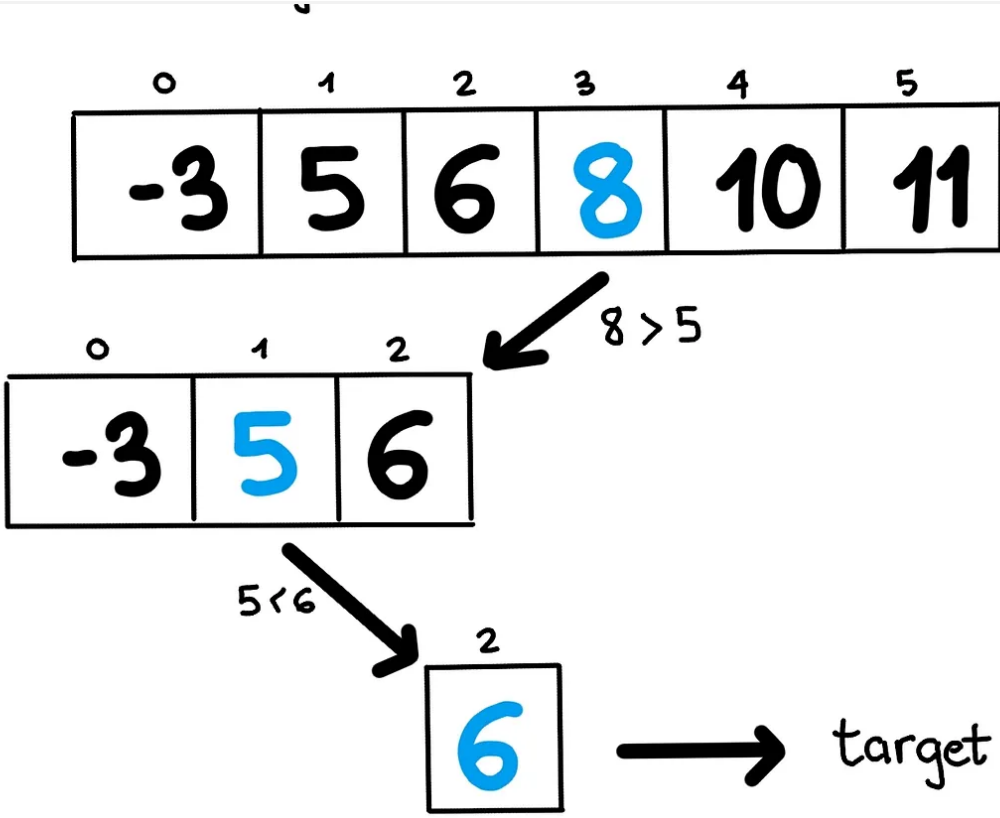
Binary search cheat sheet for coding interviews.

Image from www.jeffbullas.com

This blog is a part of my "15 days cheat sheet for hacking technical interviews at big tech companies". In this blog, we discuss the binary search in coding interviews.

Binary search algorithm

The basic idea of binary search is to **search** a given element x in a **sorted** array. The algorithm is based on divide and conquer technique. It repeatedly breaks down the array into two subarrays that may contain the item. It discards one of the subarray by utilising the fact that items are sorted. It continues halving the subarray until the value is found or the subarray is empty



The time complexity is **O(log n)**

Limitations

1. Binary search can only be applied when items are sorted. If the items are not sorted, we must sort them before applying the binary search. That will increase the total time complexity to O(n log n)

2. Binary search can only be applied to data structures which allow direct access to the items. If we do not have direct access then we can not apply binary search on it eg. we can not apply binary search to Linked List.

1. Basic problem

```
Given a sorted array and a target number.
Return the index of target number in the array if it exists, otherwise
return -1.
```


Examples:

```
Input: arr = [-3, 5, 6, 8, 10, 11, 15], target = 6
Output: 2

Input: arr = [-3, 5, 6, 8, 10, 11], target = 7
Output: -1
```

Why is the condition of while loop `left <= right` instead of `left < right` ?

- The searching should be terminated only when the array is empty. At each step, our searching array is `[left, right]`, when `left = right`, the array still contains 1 element and we need to check it.

Why do we have `left = mid + 1` and `right = mid – 1` instead of `left = mid` and `right = mid` ?

- The element at `mid` index has already been checked, so we don't need to include it in the next searching array.
- If we include `mid` index in the next searching array, we may end up with an infinite loop when `left = right`. For example:

```
Input: arr = [1], target = 3

We start with:
 left = 0, right = 0

We have:
 mid = 0 + (0-0)/ 2 = 0
 as arr[mid] = 1 < target, if we update left = mid = 0

We end up checking left = 0, right = 0 again ...
```

2. Find the first position of element

```
Given a sorted array and a target number.
Return the index of the first target number in the array if it exists,
otherwise return -1.

Examples:
```

```
Input: arr = [-3, 5, 6, 8, 10, 11, 15], target = 5
Output: 1

Input: arr = [-3, 5, 6, 6, 6, 10], target = 6
Output: 2

Input: arr = [-3, 5, 6, 6, 6, 10], target = 7
Output: -1
```

- We add a variable `first_position` to keep track of the latest valid `mid` index.

- We do not stop when finding out the target number, but continue searching on the left side.

3. Find the last position of the element

Given a sorted array and a **target** number.
Return the index of the last **target** number in the array if it exists, otherwise return -1.

Examples:

```
Input: arr = [-3, 5, 6, 8, 10, 11, 15], target = 5
Output: 1

Input: arr = [-3, 5, 6, 6, 6, 10], target = 6
Output: 4

Input: arr = [-3, 5, 6, 6, 6, 10], target = 7
Output: -1
```

- We add a variable `last_position` to keep track of the latest valid `mid` index.
- We do not stop when finding out the target number, but continue searching on the right side.

4. Find the left border element

```
Given a sorted array and a target number.
Return the index of the largest element which is smaller than target
if it exists, otherwise return -1.

Examples:

Input: arr = [-3, 5, 6, 8, 10, 11, 15], target = 5
Output: 0

Input: arr = [-3, 5, 6, 6, 7, 10], target = 6
Output: 1

Input: arr = [-3, 5, 6, 6, 7, 10], target = 7
Output: 3

Input: arr = [-3, 5, 6, 6, 7, 10], target = 1000
Output: 5

Input: arr = [-3, 5, 6, 6, 7, 10], target = -5
Output: -1
```

- We add a variable `left_border` to keep track of the latest number which is smaller than the target.
- We consider the case `arr[mid] = target` is the same as `arr[mid] > target`

4. Find the right border element

```
Given a sorted array and a target number.
Return the index of the smallest element which is larger than target
if it exists, otherwise return -1.

Examples:

Input: arr = [-3, 5, 6, 8, 10, 11, 15], target = 5
Output: 2

Input: arr = [-3, 5, 6, 6, 7, 10], target = 6
Output: 4

Input: arr = [-3, 5, 6, 6, 7, 10], target = 7
Output: 5
```

```
Input: arr = [-3, 5, 6, 6, 7, 10], target = 1000
Output: -1

Input: arr = [-3, 5, 6, 6, 7, 10], target = -5
Output: 0
```

- We add a variable `right_border` to keep track of the latest number which is larger than the target.
- We consider the case `arr[mid] == target` is the same as `arr[mid] < target`

Recommended questions

You can practice the binary search technique with the following questions:

1. Find First and Last Position of Element in Sorted Array
2. Search in Rotated Sorted Array
3. Find Minimum in Rotated Sorted Array
4. Find Minimum in Rotated Sorted Array II
5. Find K Closest Elements
6. Single Element in a Sorted Array

Binary Search Technical Interview Coding Interviews Algorithms Programming