702. Search in a Sorted Array of Unknown Size

Sept. 20, 2019 | 8.8K views

Given an integer array sorted in ascending order, write a function to search `target` in `nums`. If `target` exists, then return its index, otherwise return `-1`. **However, the array size is unknown to you**. You may only access the array using an `ArrayReader` interface, where `ArrayReader.get(k)` returns the element of the array at index `k` (0-indexed).

You may assume all integers in the array are less than `10000`, and if you access the array out of bounds, `ArrayReader.get` will return `2147483647`.

Example 1:

```
Input: array = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: array = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

Note:

1. You may assume that all elements in the array are unique.
2. The value of each element in the array will be in the range `[-9999, 9999]`.

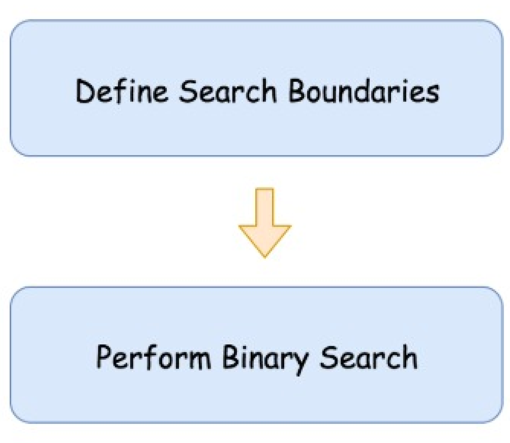
Solution

Approach 1: Binary Search

Split into Two Subproblems

The array is sorted, i.e. one could try to fit into a logarithmic time complexity. That means two subproblems, and both should be done in a logarithmic time:

- Define search limits, i.e. left and right boundaries for the search.
- Perform binary search in the defined boundaries.

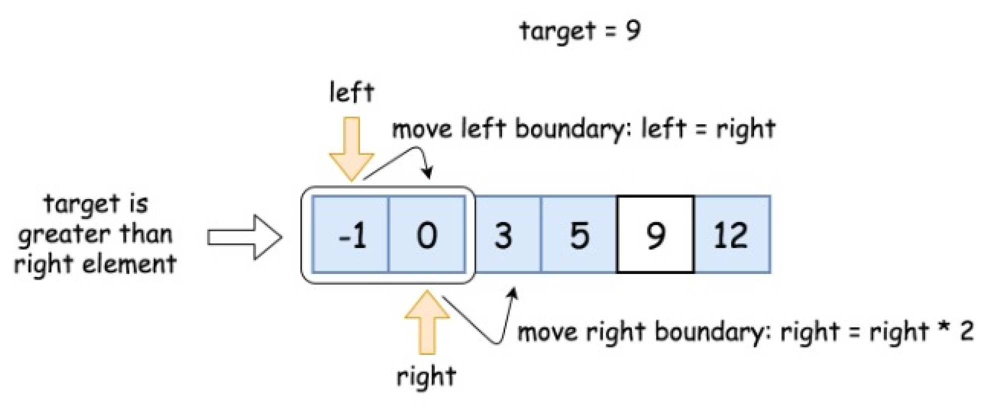


Define Search Boundaries

This is a key subproblem here.

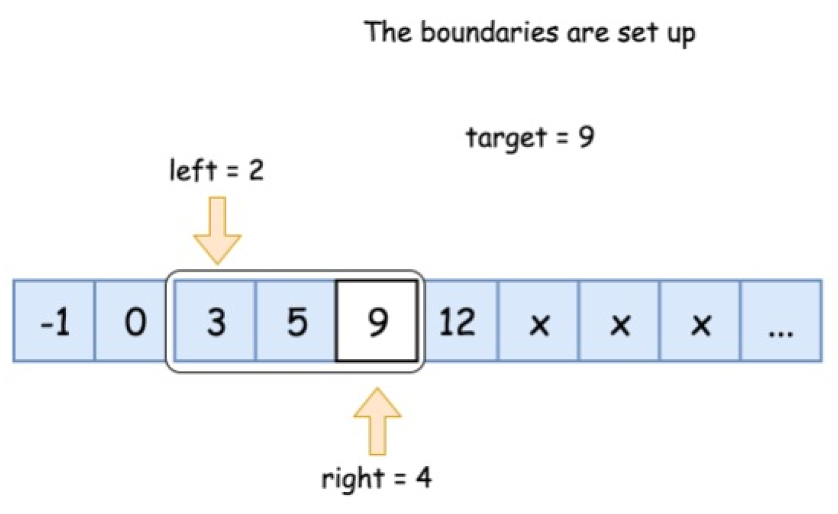
The idea is quite simple. Let's take two first indexes, 0 and 1, as left and right boundaries. If the target value is not among these zeroth and the first element, then it's outside the boundaries, on the right.

That means that the left boundary could moved to the right, and the right boundary should be extended. To keep logarithmic time complexity, let's extend it twice as far: `right = right * 2`.



If the target now is less than the right element, we're done, the boundaries are set. If not, repeat these two steps till the boundaries are established:

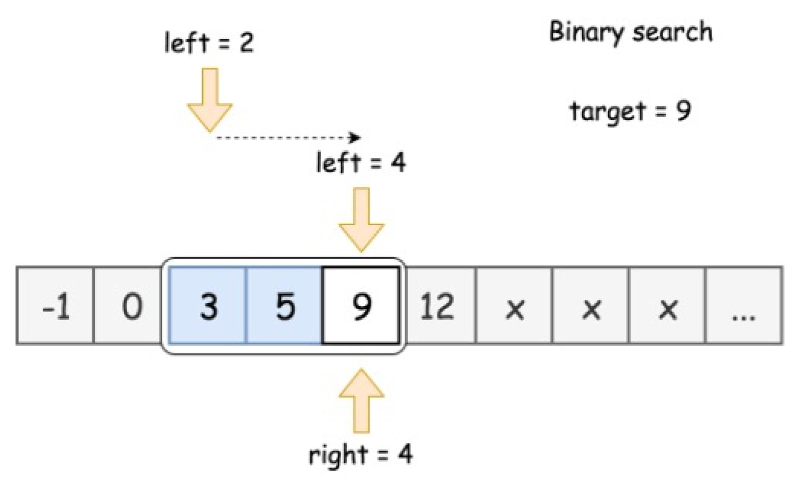
- Move the left boundary to the right: `left = right`.
- Extend the right boundary: `right = right * 2`.



Binary Search

Binary search is a textbook algorithm with a logarithmic time complexity. It's based on the idea to compare the target value to the middle element of the array.

- If the target value is equal to the middle element - we're done.
- If the target value is smaller - continue to search on the left.
- If the target value is larger - continue to search on the right.



Prerequisites: left and right shifts

To speed up, one could use here bitwise shifts:

- Left shift: `x << 1`. The same as multiplying by 2: `x * 2`.
- Right shift: `x >> 1`. The same as dividing by 2: `x / 2`.

Algorithm

Define boundaries:

- Initiate `left = 0` and `right = 1`.
- While target is on the right to the right boundary: `reader.get(right) < target`:
  - Set left boundary equal to the right one: `left = right`.
  - Extend right boundary: `right *= 2`. To speed up, use right shift instead of multiplication: `right <<= 1`.
- Now the target is between left and right boundaries.

Binary Search:

- While `left <= right`:
  - Pick a pivot index in the middle: `pivot = (left + right) / 2`. To avoid overflow, use the form `pivot = left + ((right - left) >> 1)` instead of straightforward expression above.
  - Retrieve the element at this index: `num = reader.get(pivot)`.
  - Compare middle element `num` to the target value.
    - If the middle element *is* the target `num == target`: return `pivot`.
    - If the target is not yet found:
      - If `num > target`, continue to search on the left `right = pivot - 1`.
      - Else continue to search on the right `left = pivot + 1`.
- We're here because target is not found. Return -1.

Implementation

C++ Java Python Copy

```python
class Solution:
    def search(self, reader, target):
        if reader.get(0) == target:
            return 0

        # search boundaries
        left, right = 0, 1
        while reader.get(right) < target:
            left = right
            right <<= 1

        # binary search
        while left <= right:
            pivot = left + ((right - left) >> 1)
            num = reader.get(pivot)

            if num == target:
                return pivot
            if num > target:
                right = pivot - 1
            else:
                left = pivot + 1

        # there is no target element
        return -1
```

Complexity Analysis

- Time complexity : $\mathcal{O}(\log T)$, where T is an index of target value.

  There are two operations here: to define search boundaries and to perform binary search.

  Let's first find the number of steps k to setup the boundaries. On the first step, the boundaries are $2^0..2^{0+1}$, on the second step $2^1..2^{1+1}$, etc. When everything is done, the boundaries are $2^k..2^{k+1}$ and $2^k < T \le 2^{k+1}$. That means one needs $k = \log T$ steps to setup the boundaries, that means $\mathcal{O}(\log T)$ time complexity.

  Now let's discuss the complexity of the binary search. There are $2^{k+1} - 2^k = 2^k$ elements in the boundaries, i.e. $2^{\log T} = T$ elements. As discussed, binary search has logarithmic complexity, that results in $\mathcal{O}(\log T)$ time complexity.

- Space complexity : $\mathcal{O}(1)$ since it's a constant space solution.

Analysis written by @liaison and @andvary

Comments: 4 Sort By

Type comment here... (Markdown is supported)

Preview Post

just_hands13 12 September 23, 2019 8:03 PM

as all the elements in the array are unique there can be max 2*10^5 elements (according to constraints). Also if we go out of bound of array we get 2147483647 in return
So we can simply apply binary search in start=0 and end=2 *10^5

4 Share Reply

SHOW 4 REPLIES

dsalkamoses 3 September 21, 2019 10:41 AM

nice article! however, left shift & right shift is wrongly represented in the definition.

3 Share Reply

SHOW 1 REPLY

ady_sn 20 April 24, 2020 7:46 PM

We don't even need to find upper boundary in log(n) time. We can find the upper boundary in O(1) time

```
Explanation: Given all the elements in the Array are unique. If first value is n
```

Read More

0 Share Reply

fjiang91 1 March 11, 2020 10:45 AM

See a better approach and the explanation in my post here:
https://leetcode.com/problems/search-in-a-sorted-array-of-unknown-size/discuss/535941/Clean-binary-search-with-explanation-python-solution-no-need-doubling

No need to search for the upper bound like this here.

0 Share Reply