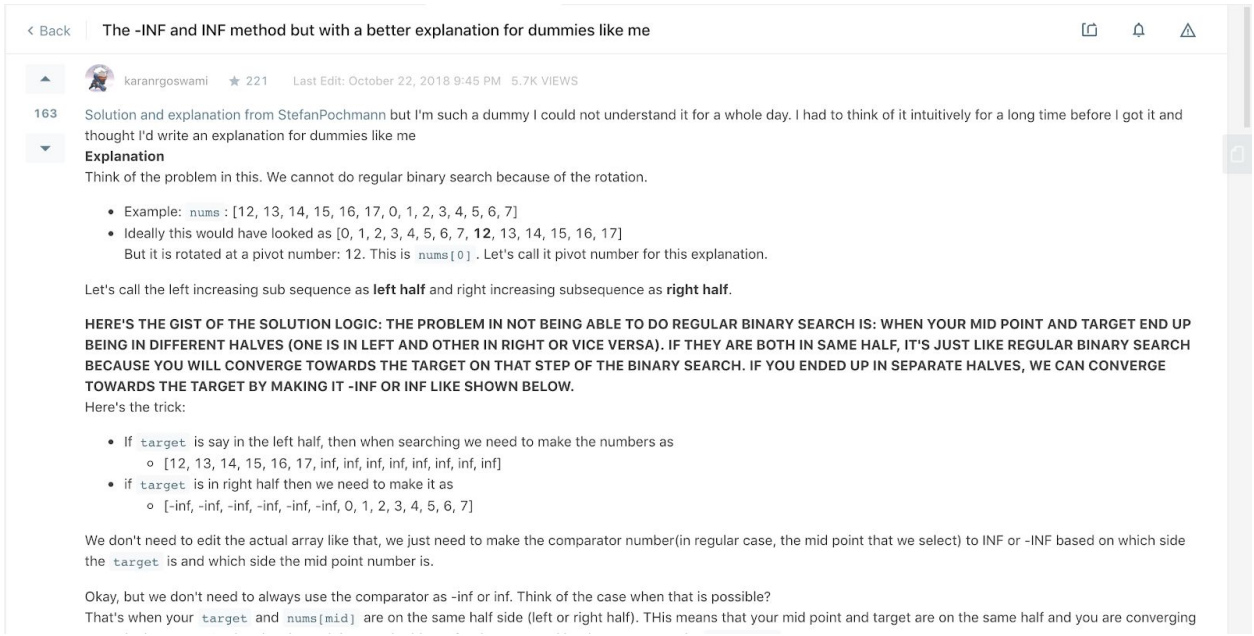
Other Approaches(1) - Explanation 1



‹ Back | The -INF and INF method but with a better explanation for dummies like me

karanrgoswami ★ 221 Last Edit: October 22, 2018 9:45 PM 5.7K VIEWS

163

Solution and explanation from StefanPochmann but I'm such a dummy I could not understand it for a whole day. I had to think of it intuitively for a long time before I got it and thought I'd write an explanation for dummies like me

Explanation

Think of the problem in this. We cannot do regular binary search because of the rotation.

- Example: `nums` : [12, 13, 14, 15, 16, 17, 0, 1, 2, 3, 4, 5, 6, 7]
- Ideally this would have looked as [0, 1, 2, 3, 4, 5, 6, 7, **12**, 13, 14, 15, 16, 17]
  But it is rotated at a pivot number: 12. This is `nums[0]` . Let's call it pivot number for this explanation.

Let's call the left increasing sub sequence as **left half** and right increasing subsequence as **right half**.

HERE'S THE GIST OF THE SOLUTION LOGIC: THE PROBLEM IN NOT BEING ABLE TO DO REGULAR BINARY SEARCH IS: WHEN YOUR MID POINT AND TARGET END UP BEING IN DIFFERENT HALVES (ONE IS IN LEFT AND OTHER IN RIGHT OR VICE VERSA). IF THEY ARE BOTH IN SAME HALF, IT'S JUST LIKE REGULAR BINARY SEARCH BECAUSE YOU WILL CONVERGE TOWARDS THE TARGET ON THAT STEP OF THE BINARY SEARCH. IF YOU ENDED UP IN SEPARATE HALVES, WE CAN CONVERGE TOWARDS THE TARGET BY MAKING IT -INF OR INF LIKE SHOWN BELOW.

Here's the trick:

- If `target` is say in the left half, then when searching we need to make the numbers as
  - [12, 13, 14, 15, 16, 17, inf, inf, inf, inf, inf, inf, inf, inf]
- if `target` is in right half then we need to make it as
  - [-inf, -inf, -inf, -inf, -inf, -inf, 0, 1, 2, 3, 4, 5, 6, 7]

We don't need to edit the actual array like that, we just need to make the comparator number(in regular case, the mid point that we select) to INF or -INF based on which side the `target` is and which side the mid point number is.

Okay, but we don't need to always use the comparator as -inf or inf. Think of the case when that is possible?
That's when your `target` and `nums[mid]` are on the same half side (left or right half). THis means that your mid point and target are on the same half and you are converging towards the target. So then just keep doing regular binary for that step and let the comparator be `nums[mid]` .

- How do we check if they both (`nums[mid]` and `target`) are on the same half? We have to check if they are both greater than the pivot number or both smaller than pivot number
  - `if (((nums[mid] > nums[0]) && (target > nums[0])) || ((nums[mid] <= nums[0]) && (target <= nums[0])))` .
  - This can also be done as `if ((target > nums[0]) == (nums[mid] > nums[0]))` .
  - Ok, so if they both are on the same half let our comparator be `nums[mid]` , because we are converging towards the target and are on the same half at the moment of comparision.
  - `comparator = nums[mid]`
  - proceed with comparing how you do regular binary search comparision.

But, what if `nums[mid]` and `target` are on different halves? Then we have to not use the `nums[mid]` as comparator. We have to use -INF or INF. How can we decide whether to make comparator as -INF or INF instead of `nums[mid]` ?

- target and `nums[mid]` are on different halves. we have to change `nums[mid]` to -INF or INF.
- Let's find out which side `nums[mid]` is. If we know which half (left or right) it is in, we know whether to select -INF or INF.
- Compare target to `nums[0]` . (Why compare `target` and not `nums[mid]` because we want to make the numbers on `nums[mid]` 's side as -INF or INF when comparing, not on `target` 's side, so we use target as the reference)
- if `target` is greater than `nums[0]` , target is on the **left half**. Look at the example above and you can see this.
  - For example: `target` is 14, it is greater than 12 so it belongs in **left half**
  - This means that `nums[mid]` is on the other half: **right half**. Make comparator as INF.
- if `target` is less than nums[0], target is on the **right half**. Look at the example above:
  - For example if `target` is 5, it is less than 12 so it belongs in **right half**
  - This means that `nums[mid]` is on the other half. **left half**. make comparator as -INF.

Now you can go ahead and do binary search

CODE:

```
class Solution {
public:
    int search(vector<int>& nums, int target)
    {
            int l = 0, r = nums.size()-1;
            while(l <= r)
            {
                int mid = (r - l)/2 + l;
                int comparator = nums[mid];
                // Checking if both target and nums[mid] are on same side.
                if((target < nums[0]) && (nums[mid] < nums[0]) || (target >= nums[0]) && (nums[mid] >= nums[0]))
                    comparator = nums[mid];
                else
                {
                    // Trying to figure out where nums[mid] is and making comparator as -INF or INF
                    if(target <nums[0])
                        comparator = -INFINITY;
                    else
                        comparator = INFINITY;

                }
                if(target == comparator) return mid;
                if(target > comparator)
                    l = mid+1;
                else
                    r = mid-1;

            }
            return -1;
    }
};
```

Other Approaches(1) - Explanation 2

< Back | Clever idea making it simple

StefanPochmann ★ 46097 Last Edit: October 24, 2018 2:51 PM 82.6K VIEWS

590

This very nice idea is from rantos22's solution who sadly only commented *"You are not expected to understand that :)"*, which I guess is the reason it's now it's hidden among the most downvoted solutions. I present an explanation and a more usual implementation.

Explanation

Let's say `nums` looks like this: [12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Because it's not fully sorted, we can't do normal binary search. But here comes the trick:

- If target is let's say 14, then we adjust `nums` to this, where "inf" means infinity:
  [12, 13, 14, 15, 16, 17, 18, 19, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
- If target is let's say 7, then we adjust `nums` to this:
  [-inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

And then we can simply do ordinary binary search.

Of course we don't actually adjust the whole array but instead adjust only on the fly only the elements we look at. And the adjustment is done by comparing both the target and the actual element against nums[0].

Code

If `nums[mid]` and `target` are *"on the same side"* of `nums[0]`, we just take `nums[mid]`. Otherwise we use -infinity or +infinity as needed.

```
int search(vector<int>& nums, int target) {
    int lo = 0, hi = nums.size();
    while (lo < hi) {
        int mid = (lo + hi) / 2;
```

```
        int mid = (lo + hi) / 2;

        double num = (nums[mid] < nums[0]) == (target < nums[0])
                   ? nums[mid]
                   : target < nums[0] ? -INFINITY : INFINITY;

        if (num < target)
            lo = mid + 1;
        else if (num > target)
            hi = mid;
        else
            return mid;
    }
    return -1;
}
```

idea