Find Minimum in Rotated Sorted Array

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).

Find the minimum element.

You may assume no duplicate exists in the array.

Solution 1

Classic binary search problem.

Looking at subarray with index [start,end]. We can find out that if the first member is less than the last member, there's no rotation in the array. So we could directly return the first element in this subarray.

If the first element is larger than the last one, then we compute the element in the middle, and compare it with the first element. If value of the element in the middle is larger than the first element, we know the rotation is at the second half of this array. Else, it is in the first half in the array.

Welcome to put your comments and suggestions.

```
int findMin(vector<int> &num) {
        int start=0,end=num.size()-1;

        while (start<end) {
            if (num[start]<num[end])
                return num[start];

            int mid = (start+end)/2;

            if (num[mid]>=num[start]) {
                start = mid+1;
            } else {
                end = mid;
            }
        }

        return num[start];
    }
```

Some corner cases will be discussed here

written by changhaz original link here

Solution 2

In this problem, we have only three cases.

Case 1. The leftmost value is less than the rightmost value in the list: This means that the list is not rotated. e.g> [1 2 3 4 5 6 7]

Case 2. The value in the middle of the list is greater than the leftmost and rightmost values in the list. e.g> [4 5 6 7 0 1 2 3]

Case 3. The value in the middle of the list is less than the leftmost and rightmost values in the list. e.g> [5 6 7 0 1 2 3 4]

As you see in the examples above, if we have case 1, we just return the leftmost value in the list. If we have case 2, we just move to the right side of the list. If we have case 3 we need to move to the left side of the list.

Following is the code that implements the concept described above.

```
int findMin(vector<int>& nums) {
    int left = 0,  right = nums.size() - 1;
    while(left < right) {
        if(nums[left] < nums[right])
            return nums[left];

        int mid = (left + right)/2;
        if(nums[mid] > nums[right])
            left = mid + 1;
        else
            right = mid;
    }

    return nums[left];
}
```

written by jaewoo original link here

Solution 3

Binary search: basically eliminate the impossible elements by half each time by exploiting the sorted property.

```
    int findMin(vector<int> &num) {
        int lo =0, hi = num.size()-1;
        while(lo<hi){
              int mid=(lo+hi)/2;
              if(num[mid]>num[hi]) lo=mid+1;
              else hi=mid;
        }
        return num[lo];
    }
```

written by lucastan original link here