Find the minimum element in a sorted and rotated array

A sorted array is rotated at some unknown point, find the minimum element in it.

Following solution assumes that all elements are distinct.

Examples

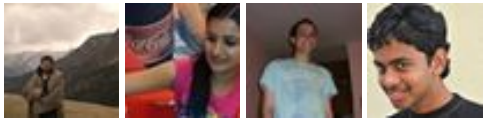
```
Input: {5, 6, 1, 2, 3, 4}
Output: 1

Input: {1, 2, 3, 4}
Output: 1

Input: {2, 1}
Output: 1
```

A simple solution is to traverse the complete array and find minimum. This solution requires $\Theta(n)$ time.
We can do it in O(Logn) using Binary Search. If we take a closer look at above examples, we can easily figure out following pattern: The minimum element is the only element whose previous element is greater than it. If there is no such element, then there is no rotation and first element is the minimum element. Therefore, we do binary search for an element which is smaller than the previous element. We strongly recommend you to try it yourself before seeing the following C implementation.

```
// C program to find minimum element in a sorted and rotated array
#include <stdio.h>
```

```
int findMin(int arr[], int low, int high)
{
    // This condition is needed to handle the case when array is not
    // rotated at all
    if (high < low)  return arr[0];

    // If there is only one element left
    if (high == low) return arr[low];

    // Find mid
    int mid = low + (high - low)/2; /*(low + high)/2;*/

    // Check if element (mid+1) is minimum element. Consider
    // the cases like {3, 4, 5, 1, 2}
    if (mid < high && arr[mid+1] < arr[mid])
       return arr[mid+1];

    // Check if mid itself is minimum element
    if (mid > low && arr[mid] < arr[mid - 1])
       return arr[mid];

    // Decide whether we need to go to left half or right half
    if (arr[high] > arr[mid])
       return findMin(arr, low, mid-1);
    return findMin(arr, mid+1, high);
}

// Driver program to test above functions
int main()
{
    int arr1[] =  {5, 6, 1, 2, 3, 4};
    int n1 = sizeof(arr1)/sizeof(arr1[0]);
    printf("The minimum element is %d\n", findMin(arr1, 0, n1-1));

    int arr2[] =  {1, 2, 3, 4};
    int n2 = sizeof(arr2)/sizeof(arr2[0]);
    printf("The minimum element is %d\n", findMin(arr2, 0, n2-1));

    int arr3[] =  {1};
    int n3 = sizeof(arr3)/sizeof(arr3[0]);
    printf("The minimum element is %d\n", findMin(arr3, 0, n3-1));

    int arr4[] =  {1, 2};
    int n4 = sizeof(arr4)/sizeof(arr4[0]);
    printf("The minimum element is %d\n", findMin(arr4, 0, n4-1));
```


Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

```
    int arr5[] =  {2, 1};
    int n5 = sizeof(arr5)/sizeof(arr5[0]);
    printf("The minimum element is %d\n", findMin(arr5, 0, n5-1));

    int arr6[] =  {5, 6, 7, 1, 2, 3, 4};
    int n6 = sizeof(arr6)/sizeof(arr6[0]);
    printf("The minimum element is %d\n", findMin(arr6, 0, n6-1));

    int arr7[] =  {1, 2, 3, 4, 5, 6, 7};
    int n7 = sizeof(arr7)/sizeof(arr7[0]);
    printf("The minimum element is %d\n", findMin(arr7, 0, n7-1));

    int arr8[] =  {2, 3, 4, 5, 6, 7, 8, 1};
    int n8 = sizeof(arr8)/sizeof(arr8[0]);
    printf("The minimum element is %d\n", findMin(arr8, 0, n8-1));

    int arr9[] =  {3, 4, 5, 1, 2};
    int n9 = sizeof(arr9)/sizeof(arr9[0]);
    printf("The minimum element is %d\n", findMin(arr9, 0, n9-1));

    return 0;
}
```

Output:

```
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
```

How to handle duplicates?

It turned out that duplicates can't be handled in O(Logn) time in all cases. Thanks to Amit Jain for inputs. The special cases that cause problems are like {2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 2} and {2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2}. It doesn't look possible to go to left half or right half by doing constant number of comparisons at the middle. Following is an implementation that handles duplicates. It may become O(n) in worst case though.

```c
// C program to find minimum element in a sorted and rotated array
#include <stdio.h>

int min(int x, int y) { return (x < y)? x :y; }

// The function that handles duplicates.  It can be O(n) in worst case
int findMin(int arr[], int low, int high)
{
    // This condition is needed to handle the case when array is not
    // rotated at all
    if (high < low)  return arr[0];

    // If there is only one element left
    if (high == low) return arr[low];

    // Find mid
    int mid = low + (high - low)/2; /*(low + high)/2;*/

    // Check if element (mid+1) is minimum element. Consider
    // the cases like {1, 1, 0, 1}
    if (mid < high && arr[mid+1] < arr[mid])
       return arr[mid+1];

    // This case causes O(n) time
    if (arr[low] == arr[mid] && arr[high] == arr[mid])
        return min(findMin(arr, low, mid-1), findMin(arr, mid+1, high));

    // Check if mid itself is minimum element
    if (mid > low && arr[mid] < arr[mid - 1])
       return arr[mid];

    // Decide whether we need to go to left half or right half
    if (arr[high] > arr[mid])
       return findMin(arr, low, mid-1);
    return findMin(arr, mid+1, high);
}

// Driver program to test above functions
int main()
{
    int arr1[] =  {5, 6, 1, 2, 3, 4};
    int n1 = sizeof(arr1)/sizeof(arr1[0]);
    printf("The minimum element is %d\n", findMin(arr1, 0, n1-1));

    int arr2[] =  {1, 1, 0, 1};
    int n2 = sizeof(arr2)/sizeof(arr2[0]);
    printf("The minimum element is %d\n", findMin(arr2, 0, n2-1));
```


Recent Comments

```
    int arr3[] =  {1, 1, 2, 2, 3};
    int n3 = sizeof(arr3)/sizeof(arr3[0]);
    printf("The minimum element is %d\n", findMin(arr3, 0, n3-1));

    int arr4[] =  {3, 3, 3, 4, 4, 4, 4, 5, 3, 3};
    int n4 = sizeof(arr4)/sizeof(arr4[0]);
    printf("The minimum element is %d\n", findMin(arr4, 0, n4-1));

    int arr5[] =  {2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 2};
    int n5 = sizeof(arr5)/sizeof(arr5[0]);
    printf("The minimum element is %d\n", findMin(arr5, 0, n5-1));

    int arr6[] =  {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1};
    int n6 = sizeof(arr6)/sizeof(arr6[0]);
    printf("The minimum element is %d\n", findMin(arr6, 0, n6-1));

    int arr7[] =  {2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2};
    int n7 = sizeof(arr7)/sizeof(arr7[0]);
    printf("The minimum element is %d\n", findMin(arr7, 0, n7-1));

    return 0;
}
```

Output:

```
The minimum element is 1
The minimum element is 0
The minimum element is 1
The minimum element is 3
The minimum element is 0
The minimum element is 1
The minimum element is 0
```

This article is contributed by **Abhay Rathi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Tpoics:

- Remove minimum elements from either side such that 2*min becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3

Writing code in comment? Please use **ideone.com** and share the link here.