Rotated Search

An Array with all the elements <b>sorted</b> across a <b>pivot</b> is given to you, You need to search the given element in the sorted rotated array.

If it's not found print -1 , otherwise print the index of that element.

An integer n - Size of the Array followed by elements of that array and an integer X to be searched.\

An integer index of that element or -1 , if not found.

5

5 6 1 2 3

1

2

1 <= n <= 10^6

1 <= arr[i] <= 10^9 , where arr[i] is any element in the array.

Index of element 1 is 2.

#include<bits/stdc++.h>

using namespace std;

int findPivot(const vector<int> &arr) {

int n = arr.size();

int high = n - 1;

int low = 0;

int ans = 0, mid;

while(low <= high) {

if(n==1) return arr[ans];

mid = low + ((high - low) / 2);

if( (arr[mid] < arr[(mid+1)%n]) && (arr[mid] < arr[(mid-1+n)%n]) ) {

ans = mid;

return ans;

}

else if(arr[mid] < arr[high]) {

high = mid;

}

else if(arr[mid] > arr[high]) {

low = mid + 1;

}

}

return ans;

}

int search(const vector<int> &A, int B) {

int n = A.size();

int m = findPivot(A);

auto it = std::next(A.begin(), m);

if( binary\_search( A.begin() , it, B)) {

return lower\_bound( A.begin() , it, B) - A.begin();

}

else if( binary\_search( it , A.end(), B)) {

return lower\_bound( it , A.end(), B) - A.begin();

}

else {

return -1;

}

}

int main() {

int n, k, r;

cin >> n;

vector<int>V;

for(int i = 0; i < n; i++) {

cin >> r;

V.push\_back(r);

}

cin >> k;

cout << search(V, k);

}

#include<bits/stdc++.h>

using namespace std;

int search(const vector<int> &A, int B) {

}

int main() {

int n, k, r;

cin >> n;

vector<int>V;

for(int i = 0; i < n; i++) {

cin >> r;

V.push\_back(r);

}

cin >> k;

cout << search(V, k);

}

Editorial:

Here,in this problem first of all we have to find the pivot element in the array,

- Pivot is the minimum element

- We need to find out the pivot element from the array.

- In sorted part we can apply binary search, we took initially high = n-1, low = 0.

- if element to the right and element to the left of the mid element is greater - the element is the pivot element.

- else :<br>

if :(element at mid is greater than the last) :<br>

a) Pivot is present in the right part

<br>else:

<br>b) Pivot is present in the left part.

- After that we can simply search in the two parts using binary search.

<br><br> Here is the code :

<pre><code class="lang-cpp">

#include&lt;bits/stdc++.h&gt;

using namespace std;

int findPivot(const vector<int> &arr) {

int n = arr.size();

int high = n - 1;

int low = 0;

int ans = 0, mid;

while(low <= high) {

if(n==1) return arr[ans];

mid = low + ((high - low) / 2);

if( (arr[mid] < arr[(mid+1)%n]) && (arr[mid] < arr[(mid-1+n)%n]) ) {

ans = mid;

return ans;

}

else if(arr[mid] < arr[high]) {

high = mid;

}

else if(arr[mid] > arr[high]) {

low = mid + 1;

}

}

return ans;

}

int search(const vector<int> &A, int B) {

int n = A.size();

int m = findPivot(A);

auto it = std::next(A.begin(), m);

if( binary \_ search( A.begin() , it, B)) {

return lower \_ bound( A.begin() , it, B) - A.begin();

}

else if( binary \_ search( it , A.end(), B)) {

return lower \_ bound( it , A.end(), B) - A.begin();

}

else {

return -1;

}

}

int main() {

int n, k, r;

cin >> n;

vector<int>V;

for(int i = 0; i < n; i++) {

cin >> r;

V.push\_back(r);

}

cin >> k;

cout << search(V, k);

}