Date : 03rd June
Mentor : Gladden Rumao
Topic : Divide And Conquer Problems

Q1. Given a sorted and rotated array A of N distinct elements which is rotated at some point, and given an element key. The task is to find the index of the given element key in the array A. The whole array A is given as the range to search.
Example 1:

Input:

N = 9
A[] = {5, 6, 7, 8, 9, 10, 1, 2, 3}
key = 10
l = 0 , h = 8
Output:
5
Explanation: 10 is found at index 5.

Example 2:

Input:
N = 4
A[] = {3, 5, 1, 2}
key = 6
l = 0 , h = 3
Output:
-1
Explanation: There is no element that has value 6.

Approach :

The idea is to find the pivot point, divide the array into two sub-arrays and perform a binary search.

The main idea for finding a pivot is –

- For a sorted (in increasing order) and rotated array, the pivot element is the only element for which the next element to it is smaller than it.
- Using binary search based on the above idea, pivot can be found.
    - It can be observed that for a search space of indices in range [l, r] where the middle index is mid,
        - If rotation has happened in the left half, then obviously the element at l will be greater than the one at mid.
        - Otherwise the left half will be sorted but the element at mid will be greater than the one at r.
- After the pivot is found, divide the array into two sub-arrays.
- Now the individual sub-arrays are sorted so the element can be searched using Binary Search.

Follow the steps mentioned below to implement the idea:

- Find out the pivot point using binary search. We will set the low pointer as the first array index and high with the last array index.
    - From the high and low we will calculate the mid value.
    - If the value at mid-1 is greater than the one at mid, return that value as the pivot.
    - Else if the value at the mid+1 is less than mid, return mid value as the pivot.
    - Otherwise, if the value at low position is greater than mid position, consider the left half. Otherwise, consider the right half.

- Divide the array into two sub-arrays based on the pivot that was found.
- Now call binary search for one of the two sub-arrays.
    - If the element is greater than the 0^{th} element then search in the left array
    - Else search in the right array.
- If the element is found in the selected sub-array then return the index
- Else return -1.

Follow the below illustration for a better understanding

Illustration:

Consider arr[] = {3, 4, 5, 1, 2}, key = 1

Pivot finding:

low = 0, high = 4:
=> mid = 2
=> arr[mid] = 5, arr[mid + 1] = 1
=> arr[mid] > arr[mid +1],
=> Therefore the pivot = mid = 2

Array is divided into two parts {3, 4, 5}, {1, 2}
Now according to the conditions and the key, we need to find in the part {1, 2}

Key Finding:

We will apply Binary search on {1, 2}.
low = 3 , high = 4.
=> mid = 3

=> arr[mid] = 1 , key = 1, hence arr[mid] = key matches.
=> The required index = mid = 3

So the element is found at index 3.

Solution :

```
class Solution
{
    public static int binarySearch(int[] A , int l
, int h, int key){
        if(h<l)
            return -1;
        int m = (l+h)/2;
        if(key==A[m])
            return m;
        else if(key>A[m])
            return binarySearch(A,m+1,h,key);
        else
            return binarySearch(A,l,m-1,key);
    }
    public static int findPivot(int[] A,int l,int
h){
        if(h<l)
            return -1;
        int m = (l+h)/2;
        if(m<h && A[m]>A[m+1])
            return m;
```

```
        if(m>l && A[m-1]>A[m])
            return m-1;
        if(A[l]>=A[m])
            return findPivot(A,l,m-1);
        else
            return findPivot(A,m+1,h);
    }
    int search(int A[], int l, int h, int key)
    {
        int pivot = findPivot(A,l,h);
        if(pivot==-1)
            return binarySearch(A,l,h,key);
        if(A[pivot]==key)
            return pivot;
        if(key>=A[l])
            return binarySearch(A,l,pivot-1,key);
        else
            return binarySearch(A,pivot+1,h,key);

    }
}
```

Time Complexity: O(log N).
Auxiliary Space: O(1)

Q2. An element is called a peak element if its value is not smaller than the value of its adjacent elements(if they exist).
Given an array arr[] of size N, Return the index of any one of its peak elements.

Example 1:

Input:
N = 3
arr[] = {1,2,3}
Possible Answer: 2
Generated Output: 1
Explanation: index 2 is 3.
It is the peak element as it is
greater than its neighbour 2.
If 2 is returned then the generated output will be 1 else 0.

Example 2:

Input:
N = 3
arr[] = {3,4,2}
Possible Answer: 1
Output: 1
Explanation: 4 (at index 1) is the
peak element as it is greater than
it's neighbor elements 3 and 2.
If 1 is returned then the generated output will be 1 else 0.

Approach :

- Create two variables, l and r, initialize l = 0 and r = n-1
- Recursively perform the below steps till l <= r, i.e. lower bound is less than the upper bound
- Check if the mid value or index mid =(low + high) / 2, is the peak element or not, if yes then print the element and terminate.

- Else if the element on the left side of the middle element is greater then check for peak element on the left side, i.e. update r = mid – 1
- Else if the element on the right side of the middle element is greater then check for peak element on the right side, i.e. update l = mid + 1

Solution :

```
class Solution

{

    public static int findPeak(int[] arr,int l,int
h , int n){

        int m = (l+h)/2;

        if( (m==0 || arr[m]>=arr[m-1]) && (m==n-1
|| arr[m]>=arr[m+1]) )

            return m;

        else if(m>0 && arr[m-1]>arr[m])

            return findPeak(arr,l,m-1,n);

        else
```

```
            return findPeak(arr,m+1,h,n);

    }

    public int peakElement(int[] arr,int n)

    {

       return findPeak(arr,0,n-1,n);

    }

}
```

Time Complexity: O(log N).
Auxiliary Space: O(1)