***Search in a Sorted Rotated Array***

Given an infinite sorted array consisting 0s and 1s. The problem is to find the index of first ‘1’ in that array. As the array is infinite, therefore it is guaranteed that number ‘1’ will be present in the array.

**Examples:**

Input : arr[] = {0, 0, 1, 1, 1, 1}

Output : 2

Input : arr[] = {1, 1, 1, 1,, 1, 1}

Output : 0

**Approach:** The problem is closely related to the problem of finding position of an element in a sorted array of infinite numbers. As the array is infinite, therefore we do not know the upper and lower bounds between which we have to find the occurrence of first ‘1’.

Below is an algorithm to find the upper and lower bounds.

**Algorithm:**

**posOfFirstOne(arr)**

Declare **l** = 0, **h** = 1

while **arr[h]** == 0

l = h

h = 2\*h;

return **indexOfFirstOne(arr, l, h)**

}

Here **h** and **l** are the required upper and lower bounds. **indexOfFirstOne(arr, l, h)** is used to find the index of occurrence of first ‘1’ between these two bounds.

Python

# Python 3 implementation to find the

# index of first 1 in an infinite

# sorted array of 0's and 1's

# function to find the index of first

# '1' binary search technique is applied

def indexOfFirstOne(arr, low, high) :

while (low <= high) :

mid = (low + high) // 2

# if true, then 'mid' is the index

# of first '1'

if (arr[mid] == 1 and (mid == 0 or

arr[mid - 1] == 0)) :

break

# first '1' lies to the left of 'mid'

elif (arr[mid] == 1) :

high = mid - 1

# first '1' lies to the right of 'mid'

else :

low = mid + 1

# required index

return mid

# function to find the index of first

# 1 in an infinite sorted array of 0's

# and 1's

def posOfFirstOne(arr) :

# find the upper and lower bounds between

# which the first '1' would be present

l = 0

h = 1

# as the array is being considered infinite

# therefore 'h' index will always exist in

# the array

while (arr[h] == 0) :

# lower bound

l = h

# upper bound

h = 2 \* h

# required index of first '1'

return indexOfFirstOne(arr, l, h)

# Driver program

arr = [ 0, 0, 1, 1, 1, 1 ]

print( "Index = ", posOfFirstOne(arr))

**Output**

Index = 2

Let p be the position of element to be searched. Number of steps for finding high index ‘h’ is O(Log p). The value of ‘h’ must be less than 2\*p. The number of elements between h/2 and h must be O(p). Therefore, time complexity of Binary Search step is also O(Log p) and overall time complexity is 2\*O(Log p) which is O(Log p).