Binary search in rotated array

Given a sorted array rotate left. find the key element in the array using binary search

The approach is:

1. Divide the array into 2 halves
2. Check which half is sorted
3. Check if the key in sorted half range
4. Update array boundary
5. Repeat the process until reaching the key

Details:

First, define two indexes - **low** and **high**, that defines array length.

Begin a loop that continues as long as **low** <= **high**:

- Calculate the middle element (**high**+**low**)/2.
  This calculation is same as (**high**-**low**)/2 + **low**, because of:
  (**high**-**low**)/2 + 2***low**/2 leads to (**high**-**low**+2***low**)/2.

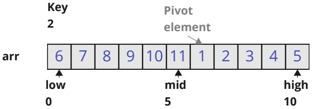
- Case1: If **arr**[**mid**] is the **key**, return mid.

- Look for array half without the pivot, the one that is sorted. Check if left half is ordered, **arr**[0] <= **arr**[**mid**].
  If the pivot in this range, **arr**[**low**] will be higher than **arr**[**mid**], what makes **arr**[**mid**] <= **arr**[**high**].
- Case2: If pivot on right side, check if the **key** in range **arr**[**low**] and **arr**[**mid**]
  - true - update **high** index to **mid**-1
  - false - update **low** index to **mid**+1

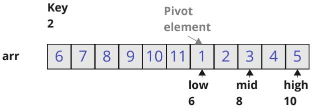
- Case3: If **pivot** not on right side, check if the **key** in range **arr**[**mid**] and **arr**[**high**]
  - true - update **low** index to **mid**+1
  - false - update **high** index to **mid**-1

Reaching end of the loop, means **key** not in the array

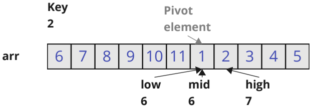
Example of binary search in rotated array



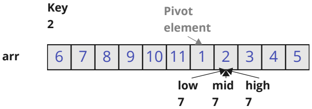
1. 6 < 11: left side sorted
2. 2 not between 6 and 11: continue search right side



1. 1 < 3: left side sorted
2. 2 between 1 and 3: continue search left side



1. 1 = 3: left side sorted
2. 2 not between 1 and 1: continue search right side



arr[mid] is the key: return 7