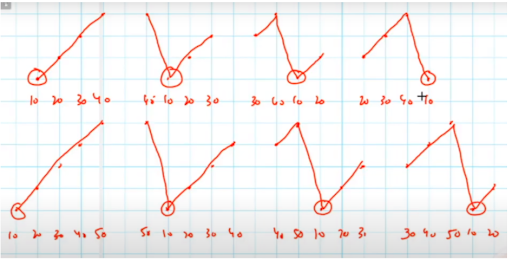
**1. Problem Discussion:**

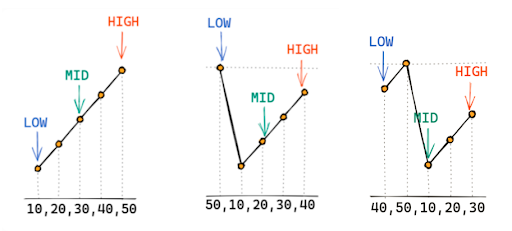
You will be given an array of distinct integers, which is sorted and rotated around an unknown point. You have to find the smallest element from the array, and you have to do this in O(logN) time complexity. How can you do this? Plot the number in the form of a graph. The dip/lowest point of the graph will be your answer.

****

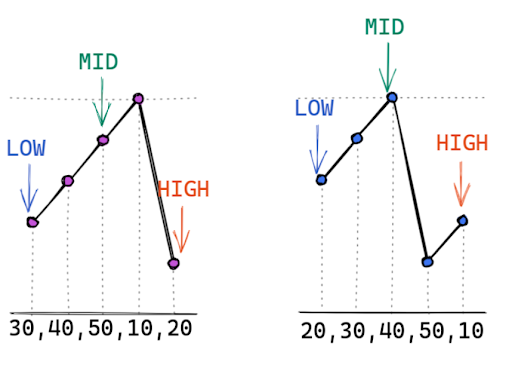
Now, try to think of a solution by yourself, and then only head over to the solution video.

**2. Approach:**

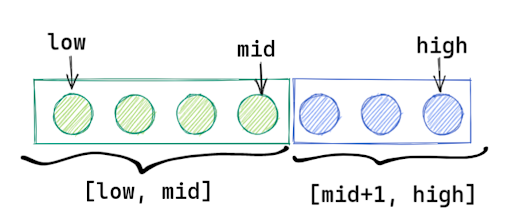
We'll use the array [10, 20, 30, 40, 50] as an example, but we'll analyse all of the array's possible shifts. We'll start with lo = 0 and hi = n-1, and the midpoint will be (lo+hi)/2.

****

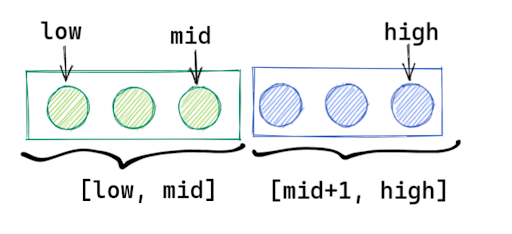
It can be seen in all of the previous examples that arr[mid] < arr[high]. And it's evident that the lowest value is always in the [low, mid] range, which includes both low and mid. On the other hand, if you plot the same situation for other scenarios, you will notice that there was a dip / lowest point between mid and high, as arr[mid] > arr[high], indicating that there was a dip / lowest point between mid and high. The lowest position is plainly between [mid+1, high] in this situation. What's the point of mid+1? Because arr[mid] is greater than arr[high], it is clear that this is not the lowest point.

****

As a result, we've divided the array into two parts based on one criteria.

****

Since n is odd in this case, one part is larger than the other.

****

Since n is an even number, both parts are the same length.

Pseudo Code:

lo = 0 hi = n-1 while lo is less than hi: mid = (lo + hi)/2 if arr[mid] > arr[hi] lo = mid + 1 // pivot lies in second half else hi = mid // pivot lies in first half now lo == hi so we can return arr[lo] or arr[hi]

**3. Code :**

Using the pseudocode as a reference, try to write the actual code on your own.

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static int findPivot(int[] arr) {

int lo = 0, hi = arr.length - 1;

while (lo < hi) {

int mid = (lo + hi) / 2;

if (arr[mid] > arr[hi]) {

lo = mid + 1;

} else {

hi = mid;

}

}

return arr[lo];

}

public static void main(String[] args) throws Exception {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int[] arr = new int[n];

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

arr[i] = scn.nextInt();

}

int pivot = findPivot(arr);

System.out.println(pivot);

}

}

**4. Analysis:**

Time Complexity: O(logN)

The length of the array is half every time. We divide the array every time we halve it, and the log is nothing more than repeated division. As a result, the number of times an array of length n can be divided is logn. So the time complexity is O. (logn).

Space Complexity: O(1)