**PROBLEM – ROTATED LIST**

Problem Statement:

You are given list of numbers, obtained by rotating a sorted list an unknown number of times. Write a function to determine the minimum number of times the original sorted list was rotated to obtain the given list. Your function should have the worst-case complexity of O(log N), where N is the length of the list. You can assume that all the numbers in the list are unique.

Example: The list [5, 6, 9, 0, 2, 3, 4] was obtained by rotating the sorted list [0, 2, 3, 4, 5, 6, 9] 3 times.

We define "rotating a list" as removing the last element of the list and adding it before the first element. E.g. rotating the list [3, 2, 4, 1] produces [1, 3, 2, 4].

"Sorted list" refers to a list where the elements are arranged in the increasing order e.g. [1, 3, 5, 7].

**SOLUTION**:

**Step 1** – State the problem clearly. Identify input and output formats.

Problem: Given a rotated sorted list of numbers for some unknown number of times, we need to find the minimum number of rotations the original sorted list was rotated. (Note: All numbers are unique)

Input: nums – list of numbers after rotating the original sorted list.

Output: rotations – minimum number of rotations done to obtain the given list of numbers.

**Step 2** – Come up with some example inputs and outputs. Try to cover all edge cases.

1. A list of size 10 rotated 3 times.

2. A list of size rotated 5 times.

3. A list that wasn’t rotated at all.

4. A list that was rotated just once.

5. A list that was rotated n-1 times, where n is the size of the list.

6. A list that was rotated n times. (do you get back the original list here?)

7. An empty list.

8. A list containing just one element.

(Test cases are listed in the: bs\_rotated\_list\_test\_cases.py)

**Step 3** – Come up with a correct solution for the problem. State it in plain English.

If a list of sorted numbers was rotated k times, then the smallest number in the list ends up at position k (counting from 0). Further, it is the only number in the list that is smaller than the number that comes before it. Thus, we simply need to check for each number in the list which is smaller than the number that comes before it (if such a number exists).

Assume the example 1 from step 2:

nums = [19, 25, 29, 3, 5, 6, 7, 9, 11, 14]

In the nums list, we can observe that the number 3 is smaller than the number before it i.e. 29, thus counting the index from 0 to the position of number 3, since number 3 is at index 3, we can say that the list was rotated 3 times.

Using linear search algorithm as the first approach to solve the problem:

1. Declare a variable position = 0

2. Compare each element with the element after it based on the index value from the variable position.

3. If the check returns a true, then return the index of that number. (n=index, rotated n times)

4. If the check was false for all elements in the list return 0. (no rotations)

5. Repeat steps 2-4 for each element.

**Step 4** – Implement the solution and test it using example inputs. Fix bugs, if any.

(Code is implemented in the: bs\_rotated\_list.py)

**Step 5** – Analyze the algorithm’s complexity and identify inefficiencies, if any.

linear\_search\_complexity = O(N), N is the number of times the while loop is running. N = size of array.

This can be time consuming for inputs that are very large.

**Step 6** – Apply right technique to overcome the inefficiency. Repeat steps 3 to 6.

To overcome the inefficiency in the above step 5, we can make use of the binary search algorithm, to help reduce the algorithm’s complexity. Use the generic binary search algorithm to find the desired result required to solve the problem.

**Step 7** – Come up with a correct solution for the problem. State it in plain English.

If the middle element is smaller than its predecessor then it is the answer. However, if it is not the answer, then we can perform a check to find whether the answer lies to the left or the right of the middle element. To achieve this, we will compare the middle element with the last element of the list, if the middle element is smaller than the last element then the answer lies to the right, ortherwise the answer lies in the left.

Using binary search algorithm as an approach to solve the problem inefficiency:

1. Declare variable mid = size of list / 2

2. Check if the value of middle-1 index > value of middle index, if yes then, middle is the answer.

3. If middle index is not the answer, then check if value of middle index < value of last element in list, yes then the answer is lies in the left of middle element, or else to the right.

4. Repeat step 2&3 until the list elements are exhausted.

5. If no rotation, then return zero as the answer.

**Step 8** – Implement the solution and test it using example inputs. Fix bugs, if any.

(Code is implemented in the: bs\_rotated\_list.py)

**Step 9** – Analyze the algorithm’s complexity and identify inefficiencies, if any.

binary\_search\_complexity = O(log N). N is the number of times the while loop is running. N = size of array. This solves the inefficiency for computing very large inputs.