<https://leetcode.com/problems/find-all-k-distant-indices-in-an-array/description/>

You are given a **0-indexed** integer array nums and two integers key and k. A **k-distant index** is an index i of nums for which there exists at least one index j such that |i - j| <= k and nums[j] == key.

Return **a list of all k-distant indices sorted in increasing order**.

**Example 1:**

**Input:** nums = [3,4,9,1,3,9,5], key = 9, k = 1

**Output:** [1,2,3,4,5,6]

**Explanation:** Here, nums[2] == key and nums[5] == key.

- For index 0, |0 - 2| > k and |0 - 5| > k, so there is no j where |0 - j| <= k and nums[j] == key. Thus, 0 is not a k-distant index.

- For index 1, |1 - 2| <= k and nums[2] == key, so 1 is a k-distant index.

- For index 2, |2 - 2| <= k and nums[2] == key, so 2 is a k-distant index.

- For index 3, |3 - 2| <= k and nums[2] == key, so 3 is a k-distant index.

- For index 4, |4 - 5| <= k and nums[5] == key, so 4 is a k-distant index.

- For index 5, |5 - 5| <= k and nums[5] == key, so 5 is a k-distant index.

- For index 6, |6 - 5| <= k and nums[5] == key, so 6 is a k-distant index.

Thus, we return [1,2,3,4,5,6] which is sorted in increasing order.

**Example 2:**

**Input:** nums = [2,2,2,2,2], key = 2, k = 2

**Output:** [0,1,2,3,4]

**Explanation:** For all indices i in nums, there exists some index j such that |i - j| <= k and nums[j] == key, so every index is a k-distant index. Hence, we return [0,1,2,3,4].

**Constraints:**

1 <= nums.length <= 1000

1 <= nums[i] <= 1000

key is an integer from the array nums.

1 <= k <= nums.length

**Attempt 1: 2024-01-24**

**Solution 1: Interval (10 min)**

class Solution {

    public List<Integer> findKDistantIndices(int[] nums, int key, int k) {

        List<Integer> result = new ArrayList<>();

        int n = nums.length;

        for(int i = 0; i < n; i++) {

            if(nums[i] == key) {

                int start = Math.max(0, i - k);

                int end = Math.min(n - 1, i + k);

                if(result.size() != 0 && result.get(result.size() - 1) >= start) {

                    start = result.get(result.size() - 1) + 1;

                }

                for(int j = start; j <= end; j++) {

                    result.add(j);

                }

            }

        }

        return result;

    }

}

Time Complexity: O(N)

Space Complexity: O(1)

**Refer to**

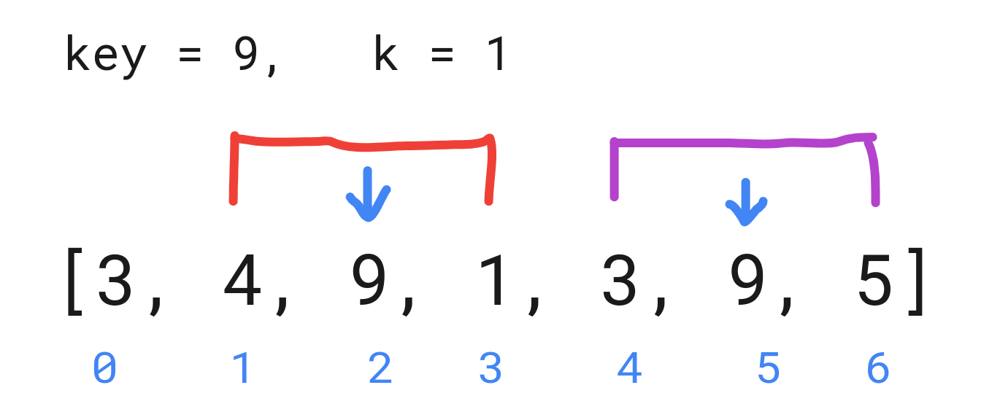
<https://leetcode.com/problems/find-all-k-distant-indices-in-an-array/solutions/1845912/o-n-solution-detailed-pictures-easy-to-understand-explaination-c/>

**Breaking down and understanding the problem**

The problem says to return indexes such that |i - j| <= k and nums[j] = key. What does it mean ?

See once we find the key at a particular index. We need to print those indexes which satisfy the condtion

|i - j| <= k. Right ? . Now that we understand it, let's look at a visual



From here we see that whenever the key is found. We can only move **AT MOST** k indexes to the left and **AT MOST** k indexes to the right. If we move any further along in any direction the |i - j| <= k condition breaks.

**So which basically means that the actual problem is whenever we find the key. we have to move k indexes to the left and k indexes to the right and that gives the answer.**

Now comes a few scenarios to keep in mind.

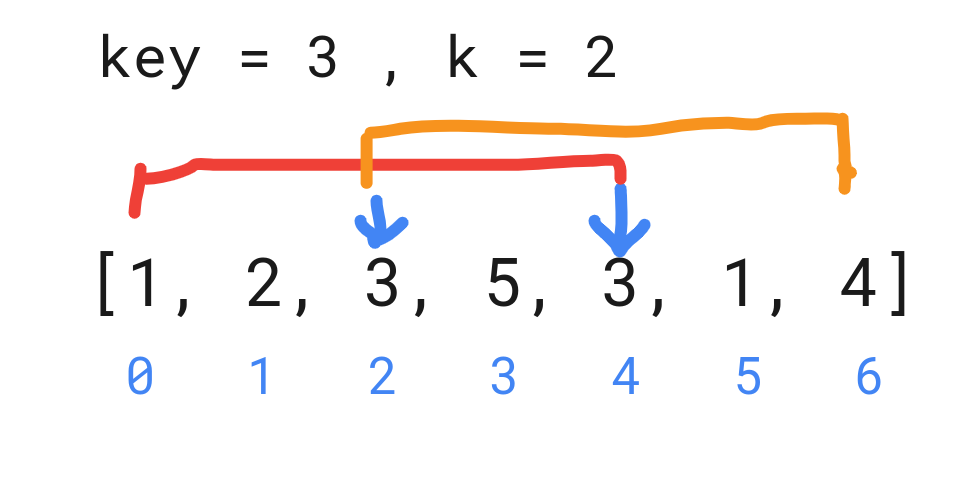
What if we find key on extreme left. We can't move to left of index 0, it gives us out of bounds.

In order to avoid this we can move to **max(0, i-k)** positions. So that we never fall below 0.

What if we find key on extreme right. We can't move beyond array limit.

In order to avoid this we can move to **min(n-1, i+k)** positions. So that we don't go beyond limit of array.

Avoiding overlapping in intervals



Here what we can do is, we check the last entry of index that we stored. We compare it with the start of the new interval. We take the max of last index entry + 1 & current start. In above example last entry for first 3 will be 4 **(See the end of red line)**. Starting index for 3 will be 2 **(See the start of orange line)**. So comparing

max(lastEntry of index + 1, startIndex) = max(5, 2) = 5. So we avoid overlapping interval and start from 5.

vector<int> findKDistantIndices(vector<int>& nums, int key, int k)

{

int n = nums.size();

vector<int> ans;

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

{

if(nums[i] == key) // if we find the key

{

int start = max(0, i-k); // initialize the start

int end = min(n-1, i+k); // intialize the end

if(ans.size()!=0) // check if any index is stored earlier

start = max(ans[ans.size() - 1] + 1, start); // avoid overlapping intervals

for(int j = start; j<=end; j++) // simply push every index from start till end

ans.push\_back(j);

}

}

return ans;

}

As we are able to avoid overlapping intervals. The time complexity is O(n).