Q1public class Main {

public static void main(String[] args) {

int x = 4;

int sqrt = mySqrt(x);

System.out.println("Square root of " + x + ": " + sqrt);

}

public static int mySqrt(int x) {

if (x == 0)

return 0;

int left = 1, right = x;

while (left <= right) {

int mid = left + (right - left) / 2;

if (mid <= x / mid && (mid + 1) > x / (mid + 1))

return mid;

else if (mid > x / mid)

right = mid - 1;

else

left = mid + 1;

}

return -1; // unreachable

}

}

Q2.public class Main {

public static void main(String[] args) {

int[] nums = {1, 2, 3, 1};

int peakIndex = findPeakElement(nums);

System.out.println("Peak element index: " + peakIndex);

}

public static int findPeakElement(int[] nums) {

int left = 0, right = nums.length - 1;

while (left < right) {

int mid = left + (right - left) / 2;

if (nums[mid] < nums[mid + 1]) {

left = mid + 1;

} else {

right = mid;

}

}

return left;

}

}

Q3.public class Main {

public static void main(String[] args) {

int[] nums = {3, 0, 1};

int missingNumber = findMissingNumber(nums);

System.out.println("Missing number: " + missingNumber);

}

public static int findMissingNumber(int[] nums) {

int n = nums.length;

int missingNumber = n; // Initialize missing number to n

// XOR all numbers from 0 to n

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

missingNumber ^= i;

}

// XOR all numbers in nums

for (int num : nums) {

missingNumber ^= num;

}

return missingNumber;

}

}

Q4.public class Main {

public static void main(String[] args) {

int[] nums = {1, 3, 4, 2, 2};

int repeatedNumber = findDuplicate(nums);

System.out.println("Repeated number: " + repeatedNumber);

}

public static int findDuplicate(int[] nums) {

int tortoise = nums[0];

int hare = nums[0];

// Move tortoise one step and hare two steps

do {

tortoise = nums[tortoise];

hare = nums[nums[hare]];

} while (tortoise != hare);

// Move one pointer to the start and the other from the intersection point

tortoise = nums[0];

while (tortoise != hare) {

tortoise = nums[tortoise];

hare = nums[hare];

}

return hare;

}

}

Q5.import java.util.HashSet;

import java.util.ArrayList;

import java.util.List;

public class Main {

public static void main(String[] args) {

int[] nums1 = {1, 2, 2, 1};

int[] nums2 = {2, 2};

int[] intersection = findIntersection(nums1, nums2);

System.out.print("Intersection: ");

for (int num : intersection) {

System.out.print(num + " ");

}

}

public static int[] findIntersection(int[] nums1, int[] nums2) {

HashSet<Integer> set = new HashSet<>();

HashSet<Integer> intersectionSet = new HashSet<>();

// Add elements from nums1 to the set

for (int num : nums1) {

set.add(num);

}

// Check for common elements in nums2 and add them to the intersectionSet

for (int num : nums2) {

if (set.contains(num)) {

intersectionSet.add(num);

}

}

// Convert the intersectionSet to an array

int[] intersection = new int[intersectionSet.size()];

int index = 0;

for (int num : intersectionSet) {

intersection[index++] = num;

}

return intersection;

}

}

Q6.public class Main {

public static void main(String[] args) {

int[] nums = {4, 5, 6, 7, 0, 1, 2};

int minElement = findMin(nums);

System.out.println("Minimum element: " + minElement);

}

public static int findMin(int[] nums) {

int left = 0;

int right = nums.length - 1;

// If the array is not rotated

if (nums[left] < nums[right]) {

return nums[left];

}

// Binary search to find the pivot element

while (left < right) {

int mid = left + (right - left) / 2;

// If the mid element is greater than its next element, it is the pivot element

if (nums[mid] > nums[mid + 1]) {

return nums[mid + 1];

}

// If the mid element is less than its previous element, it is the pivot element

if (nums[mid] < nums[mid - 1]) {

return nums[mid];

}

// If the mid element is greater than the left element, the pivot is on the right side

if (nums[mid] > nums[left]) {

left = mid + 1;

}

// If the mid element is less than the right element, the pivot is on the left side

else {

right = mid - 1;

}

}

return -1; // The code should never reach this point, it's added to satisfy Java syntax

}

}

Q7.public class Main {

public static void main(String[] args) {

int[] nums = {5, 7, 7, 8, 8, 10};

int target = 8;

int[] positions = searchRange(nums, target);

System.out.println("Starting position: " + positions[0]);

System.out.println("Ending position: " + positions[1]);

}

public static int[] searchRange(int[] nums, int target) {

int[] result = {-1, -1};

// Find the starting position

int left = 0;

int right = nums.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (nums[mid] >= target) {

right = mid - 1;

} else {

left = mid + 1;

}

}

if (left < nums.length && nums[left] == target) {

result[0] = left;

} else {

return result;

}

// Find the ending position

right = nums.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (nums[mid] <= target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

result[1] = right;

return result;

}

}

Q8.import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class Main {

public static void main(String[] args) {

int[] nums1 = {1, 2, 2, 1};

int[] nums2 = {2, 2};

int[] intersection = findIntersection(nums1, nums2);

System.out.println("Intersection: " + Arrays.toString(intersection));

}

public static int[] findIntersection(int[] nums1, int[] nums2) {

Map<Integer, Integer> countMap = new HashMap<>();

// Count the occurrences of each element in nums1

for (int num : nums1) {

countMap.put(num, countMap.getOrDefault(num, 0) + 1);

}

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

// Find the intersection elements and decrement their counts

for (int num : nums2) {

if (countMap.containsKey(num) && countMap.get(num) > 0) {

intersectionList.add(num);

countMap.put(num, countMap.get(num) - 1);

}

}

int[] intersection = new int[intersectionList.size()];

// Convert the intersection list to an array

for (int i = 0; i < intersectionList.size(); i++) {

intersection[i] = intersectionList.get(i);

}

return intersection;

}

}