Assignment 11 Solution – Binary Search | DSA

Question 1

Given a non-negative integer `x`, return the square root of `x` rounded down to the nearest integer. The returned integer should be non-negative as well.
You must not use any built-in exponent function or operator.
- For example, do not use `pow(x, 0.5)` in c++ or `x  0.5` in python.
Example 1:
Input: x = 4
Output: 2
Explanation: The square root of 4 is 2, so we return 2.

Example 2:
Input: x = 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.

Solution Code:

```
package in.ineuron.pptAssignment11;

public class SquareRoot {
   public static int sqrt(int x) {
      if (x == 0 || x == 1) {
         return x;
      }

      long left = 1;
      long right = x;
      long result = 0;

      while (left <= right) {
         long mid = left + (right - left) / 2;

         if (mid * mid <= x) {
            result = mid;
            left = mid + 1;
         } else {
            right = mid - 1;
         }
      }

      return (int) result;
   }
```

```
    public static void main(String[] args) {
        int x = 4;
        int sqrtValue = sqrt(x);
        System.out.println(sqrtValue);
    }
}
```

Question 2

A peak element is an element that is strictly greater than its neighbors.
Given a 0-indexed integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.
You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.
You must write an algorithm that runs in `O(log n)` time.

Example 1:
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.

Example 2:
Input: nums = [1,2,1,3,5,6,4]
Output: 5
Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

Solution Code:

```
package in.ineuron.pptAssignment11;

public class PeakElement {
    public static int findPeakElement(int[] nums) {
        int left = 0;
        int right = nums.length - 1;

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

            if (nums[mid] < nums[mid + 1]) {
                // The peak is on the right side
                left = mid + 1;
            } else {
                // The peak is on the left side, or mid is the peak
                right = mid;
            }
```

```
        }

        return left;
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 1};
        int peakIndex = findPeakElement(nums);
        System.out.println(peakIndex);
    }
}
```

Question 3

Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return the only number in the range that is missing from the array.

Example 1:
Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Example 2:
Input: nums = [0,1]
Output: 2
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

Example 3:
Input: nums = [9,6,4,2,3,5,7,0,1]
Output: 8
Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.

Solution Code:

```
package in.ineuron.pptAssignment11;

public class MissingNumber {
    public static int missingNumber(int[] nums) {
        int n = nums.length;

        // Initialize the result with n because n is missing from the array
        int missing = n;
```

```
        // XOR all the numbers from 0 to n and the numbers in the array
        for (int i = 0; i < n; i++) {
            missing ^= i;
            missing ^= nums[i];
        }

        return missing;
    }

    public static void main(String[] args) {
        int[] nums = {3, 0, 1};
        int missingNumber = missingNumber(nums);
        System.out.println(missingNumber);
    }
}
```

Question 4

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.
There is only one repeated number in `nums`, return this repeated number.
You must solve the problem without modifying the array `nums` and uses only constant extra space.

Example 1:
Input: nums = [1,3,4,2,2]
Output: 2

Example 2:
Input: nums = [3,1,3,4,2]
Output: 3

Solution Code:

```
package in.ineuron.pptAssignment11;

public class FindDuplicate {
    public static int findDuplicate(int[] nums) {
        // Step 1: Find the intersection point of the two pointers
        int tortoise = nums[0];
        int hare = nums[0];
```

```
                do {
                        tortoise = nums[tortoise];
                        hare = nums[nums[hare]];
                } while (tortoise != hare);

                // Step 2: Find the entrance to the cycle
                tortoise = nums[0];
                while (tortoise != hare) {
                        tortoise = nums[tortoise];
                        hare = nums[hare];
                }

                return hare;
        }

        public static void main(String[] args) {
                int[] nums = { 1, 3, 4, 2, 2 };
                int duplicateNumber = findDuplicate(nums);
                System.out.println(duplicateNumber);
        }
}
```

Question 5

Given two integer arrays `nums1` and `nums2`, return an array of their intersection. Each element in the result must be unique and you may return the result in any order.

Example 1:
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]

Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]
Explanation: [4,9] is also accepted.

Solution Code:

```
package in.ineuron.pptAssignment11;
import java.util.*;

public class IntersectionArrays {
        public static int[] intersection(int[] nums1, int[] nums2) {
                Set<Integer> set = new HashSet<>();
                Set<Integer> intersectionSet = new HashSet<>();
```

```
        // Add unique elements from nums1 to the set
        for (int num : nums1) {
            set.add(num);
        }

        // Check if elements from nums2 are present in the set
        for (int num : nums2) {
            if (set.contains(num)) {
                intersectionSet.add(num);
            }
        }

        // Convert the set to an array
        int[] result = new int[intersectionSet.size()];
        int index = 0;
        for (int num : intersectionSet) {
            result[index++] = num;
        }

        return result;
    }

    public static void main(String[] args) {
        int[] nums1 = { 1, 2, 2, 1 };
        int[] nums2 = { 2, 2 };
        int[] intersectionArray = intersection(nums1, nums2);

        System.out.print("[");
        for (int i = 0; i < intersectionArray.length; i++) {
            System.out.print(intersectionArray[i]);
            if (i < intersectionArray.length - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
}
```

Question 6

Suppose an array of length `n` sorted in ascending order is rotated between `1` and `n` times. For example, the array `nums = [0,1,2,4,5,6,7]` might become:
- `[4,5,6,7,0,1,2]` if it was rotated `4` times.
- `[0,1,2,4,5,6,7]` if it was rotated `7` times.

Notice that rotating an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` of unique elements, return the minimum element of this array.

You must write an algorithm that runs in `O(log n) time.`

Example 1:
Input: nums = [3,4,5,1,2]
Output: 1
Explanation: The original array was [1,2,3,4,5] rotated 3 times.

Example 2:
Input: nums = [4,5,6,7,0,1,2]
Output: 0
Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.

Example 3:
Input: nums = [11,13,15,17]
Output: 11
Explanation: The original array was [11,13,15,17] and it was rotated 4 times.

Solution Code:

```
package in.ineuron.pptAssignment11;

public class MinimumRotatedArray {
	public static int findMin(int[] nums) {
		int left = 0;
		int right = nums.length - 1;

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

			if (nums[mid] > nums[right]) {
				// Minimum element is on the right side of mid
				left = mid + 1;
			} else {
				// Minimum element is on the left side of or equal to mid
				right = mid;
			}
		}
```

```
            return nums[left];
        }

        public static void main(String[] args) {
            int[] nums = { 11, 13, 15, 17 };
            int minElement = findMin(nums);
            System.out.println(minElement);
        }
    }
```

Question 7

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given `target` value.
If `target` is not found in the array, return `[-1, -1]`.
You must write an algorithm with `O(log n)` runtime complexity.

Example 1:
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]

Example 2:
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]

Example 3:
Input: nums = [], target = 0
Output: [-1,-1]

Solution Code:

```
package in.ineuron.pptAssignment11;

public class FindRange {
    public static int[] searchRange(int[] nums, int target) {
        int[] result = { -1, -1 };

        // Find the leftmost occurrence of the target
        int leftIndex = findIndex(nums, target, true);

        // If the target is not found, return [-1, -1]
        if (leftIndex == -1) {
            return result;
        }
```

```
        // Find the rightmost occurrence of the target
        int rightIndex = findIndex(nums, target, false);

        // Set the starting and ending positions of the target in the result array
        result[0] = leftIndex;
        result[1] = rightIndex;

        return result;
    }

    // Helper function to find the leftmost or rightmost occurrence of the target
    private static int findIndex(int[] nums, int target, boolean leftmost) {
        int index = -1;
        int left = 0;
        int right = nums.length - 1;

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

            if (nums[mid] == target) {
                index = mid;

                if (leftmost) {
                // Move the right pointer to search for the leftmost occurrence
                    right = mid - 1;
                } else {
                // Move the left pointer to search for the rightmost occurrence
                    left = mid + 1;
                }
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return index;
    }

    public static void main(String[] args) {
        int[] nums = { 5, 7, 7, 8, 8, 10 };
        int target = 6;
        int[] range = searchRange(nums, target);

        System.out.println("[" + range[0] + ", " + range[1] + "]");
    }
}
```

Question 8

Given two integer arrays `nums1` and `nums2`, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in any order.

Example 1:
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2,2]

Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [4,9]
Explanation: [9,4] is also accepted.

Solution Code:

```
package in.ineuron.pptAssignment11;

import java.util.*;

public class IntersectionArrays_8 {
	public static int[] intersect(int[] nums1, int[] nums2) {
		Map<Integer, Integer> map = new HashMap<>();
		List<Integer> intersectionList = new ArrayList<>();

		// Count the frequency of elements in nums1
		for (int num : nums1) {
			map.put(num, map.getOrDefault(num, 0) + 1);
		}

		// Check the frequency of elements in nums2
		for (int num : nums2) {
			if (map.containsKey(num) && map.get(num) > 0) {
				intersectionList.add(num);
				map.put(num, map.get(num) - 1);
			}
		}

		// Convert the list to an array
		int[] result = new int[intersectionList.size()];
		for (int i = 0; i < intersectionList.size(); i++) {
			result[i] = intersectionList.get(i);
		}

		return result;
	}
```

```
        public static void main(String[] args) {
                int[] nums1 = { 1, 2, 2, 1 };
                int[] nums2 = { 2, 2 };
                int[] intersectionArray = intersect(nums1, nums2);

                System.out.print("[");
                for (int i = 0; i < intersectionArray.length; i++) {
                        System.out.print(intersectionArray[i]);
                        if (i < intersectionArray.length - 1) {
                                System.out.print(", ");
                        }
                }
                System.out.println("]");
        }
}
```