Assignment Questions 11

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.

code:-

```
class Solution {
    public int mySqrt(int x) {
        if (x < 2) {
            return x;
        }
        int lo = 2, hi = x / 2;
        while (lo <= hi) {
            int mid = lo + (hi - lo) / 2;
            long sq = 1L * mid * mid;
            if (sq > x)
                hi = mid - 1;
            else if (sq < x)
                lo = mid + 1;
            else
                return mid;
        }
        return hi;
    }
}
```

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.

code:-

```
class Solution {
    public int findPeakElement(int[] arr) {
         int start=0,end=arr.length-1;
        while(start<end){
            int mid=start+(end-start)/2;
            if(arr[mid]<arr[mid+1]){
                  start=mid+1;

            }else{
                end=mid;
            }

        }
        return start;
    }
}
```

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.

code:-

```
class Solution {
    public int missingNumber(int[] nums) {
        int expectedSum = 0;
        int sum = 0;
        int len = nums.length;

        for (int i=0; i<=len; i++){
            expectedSum += i;
        }

        for (int num:nums){
            sum += num;
        }

        int missing = expectedSum-sum;

        return missing;
```

```
    }
}
```

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

code:-

```
public int findDuplicate(int[] nums) {
    int low = 1;
    int high = nums.length-1;
    int dup = -1;
    while(low <= high){
        int mid = (low+high)/2;
        int count = 0;
        for(int num : nums){
            if(num <= mid){
                count++;
            }
        }

        if(count > mid){
            dup = mid;
            high = mid-1;
        }
        else{
            low = mid+1;
        }
    }
    return dup;
}
```

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.

code:-

```
class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {

        Arrays.sort(nums1);
        Arrays.sort(nums2); // sort nums2[] to avoid duplicates

        ArrayList<Integer> res = new ArrayList<>();
        for (int i = 0; i < nums2.length; i++) {
            if (i >= 1 && nums2[i] == nums2[i - 1]) {
                continue;
            }
            if (binarysearch(nums1, nums2[i]) == true) {
                res.add(nums2[i]);
            }
        }

        return res.stream().mapToInt(i -> i).toArray(); // arraylist of Integer to
int[] array
    }
    private boolean binarysearch(int[] arr, int target) {
        int lo = 0;
        int hi = arr.length - 1;

        while (lo <= hi) {
            int mid = lo + (hi - lo) / 2;
            if (arr[mid] == target) {
                return true;
            } else if (arr[mid] < target) {
                lo = mid + 1;
            } else {
                hi = mid - 1;
            }
        }
        return false;
    }
}
```

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.

code:-

```
class Solution {
    public int findMin(int[] nums) {
        int l=0,h=nums.length-1;
        while(l<h)
        {
            int mid=l+(h-l)/2;
            if(nums[mid]>nums[h])
                l=mid+1;
            else
                h=mid;
        }
        return nums[h];
    }
}
```

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]

code:-

```
class Solution {
    public int[] searchRange(int[] nums, int target) {

        int[] result = new int[2];
        result[0] = searchFirst(nums, target);
        result[1] = searchLast(nums, target);

        return result;
    }

    // binary search to find the first occurrence of the target
    private int searchFirst(int[] nums, int target){
        int left = 0; // left index
        int right = nums.length - 1; // right index
        int index = -1; // index of the first occurrence

        while(left <= right){
```

```
            int mid = left + (right - left) / 2; // calculate the mid index

            if(nums[mid] == target){
                index = mid; // update index of the first occurrence
                right = mid - 1; // search in the left half
            }

            else if(nums[mid] < target){
                left = mid + 1; // search in the right half
            }
            else{
                right = mid - 1; // search in the left half
            }
        }

        return index;
    }

    // binary search to find the last occurrence of the target
    private int searchLast(int[] nums, int target){
        int left = 0; // left index
        int right = nums.length - 1; // right index
        int index = -1; // index of the last occurrence

        while(left <= right){

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

            if(nums[mid] == target){
                index = mid; // update index of the last occurrence
                left = mid + 1; // search in the right half
            }

            else if(nums[mid] < target){
                left = mid + 1; // search in the right half
            }
            else{
                right = mid - 1; // search in the left half
            }
        }

        return index;
    }
}
```

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.

code:-

```
 public int[] intersect(int[] nums1, int[] nums2) {

        Arrays.sort(nums1);
        Arrays.sort(nums2);

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

        int cur = nums1[0];
        int count = 1;

        for (int i = 1; i < nums1.length; i++) {

            if (cur == nums1[i]) {
                count++;
            }

            if (cur != nums1[i]) {

                int k = binarySearch(nums2, 0, nums2.length - 1, cur, count);

                if (k != -1) {
                    hm.put(cur, k);
                }

                cur = nums1[i];
                count = 1;
            }

        }


        int k = binarySearch(nums2, 0, nums2.length - 1, cur, count);

        if (k != -1) {
            hm.put(cur, k);
        }


        int size = 0;

        for (Integer s : hm.values()) {
            size += s;
        }


        int[] res = new int[size];


        int j = 0;
        for (Integer num : hm.keySet()) {
            for (int i = 0; i < hm.get(num); i++) {
                res[j] = num;
                j++;
            }
        }

        return res;
```

```
    }


    public int binarySearch(int[] arr, int l, int r, int x, int count) {
        if (l <= r) {

            int mid = l + (r - l) / 2;

            if (arr[mid] == x) {
                int k = 1;
                int left = mid - 1;
                int right = mid + 1;

                while (left >= 0 || right < arr.length) {

                    if (left >= 0 && arr[left] == x) {
                        k++;
                    }

                    if (right < arr.length && arr[right] == x) {
                        k++;
                    }


                    if((left >= 0 && arr[left] != x && right>=arr.length) ||
                       (right < arr.length && arr[right] != x && left<0) ||
                       (left >=0 && right <=arr.length && arr[left] != x &&
arr[right] != x)
                      ){
                        break;
                    }

                    left--;
                    right++;

                }


                return Math.min(k, count);
            }

            if (arr[mid] > x) {
                return binarySearch(arr, l, mid - 1, x, count);
            }
            return binarySearch(arr, mid + 1, r, x, count);
        }

        return -1;
    }
```