**Infinite Champions Programme – Day 1 (Assignment Sheet)**

**Instructions  
• Deadline: Submit your solutions by 26th September, 2025, EOD.  
• Platform: Test your solutions on LeetCode  
• Collaboration: Discussing concepts is encouraged, but all code must be your own.**

**1.** [**Minimum Number of Arrows to Burst Balloons**](https://leetcode.com/problems/minimum-number-of-arrows-to-burst-balloons/)

**• Problem: You are given a number of spherical balloons spread in a 2D space.  
For each balloon, the input is the start and end coordinates of the horizontal diameter.  
Find the minimum number of arrows that must be shot to burst all balloons.  
An arrow can be shot vertically from different points along the x-axis.  
• Objective: Implement a function that returns the minimum number of arrows needed to burst all balloons.  
• YouTube Solution (Java):** [**Minimum Number of Arrows to Burst Balloons - Java Solution**](https://www.youtube.com/watch?v=U6YQn6QKzvI)

class Solution {

    public int findMinArrowShots(int[][] points) {

        Arrays.sort(points, (a, b) -> Integer.compare(a[1], b[1]));

        int arrow=1;

        int end=points[0][1];

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

            if(points[i][0]>end){

                arrow++;

                end=points[i][1];

            }

        }

        return arrow;

    }

}

**2.** [**Sliding Window Maximum**](https://leetcode.com/problems/sliding-window-maximum/)

**• Problem: You are given an integer array nums. There is a sliding window of size k which is moving from the very left of the array to the very right.  
You can only see the k numbers in the window. Each time the sliding window moves right by one position.  
Return the maximum sliding window.  
• Objective: Implement a function that returns an array of maximum values for each window position.  
• YouTube Solution (Java):** [**Sliding Window Maximum - Java Solution**](https://www.youtube.com/watch?v=UjqMNv9K27s)

**import java.util.\*;**

**class Solution {**

**public int[] maxSlidingWindow(int[] nums, int k) {**

**int n = nums.length;**

**int[] arr = new int[n - k + 1];**

**PriorityQueue<int[]> p=new PriorityQueue<>((a,b)->b[0]-a[0]);**

**for(int i=0;i<k;i++){**

**p.offer(new int[]{nums[i],i});**

**}**

**int j=0;**

**arr[j++]=p.peek()[0];**

**for(int i=k;i<n;i++){**

**p.offer(new int[]{nums[i],i});**

**while(p.peek()[1]<=i-k){**

**p.poll();**

**}**

**arr[j++]=p.peek()[0];**

**}**

**return arr;**

**}**

**}**

**3.** [**Find First and Last Position of Element in Sorted Array**](https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/)

**• Problem: Given a sorted array of integers arr and a target value, find the start and end positions of the target value. If the target does not exist in the array, return [-1, -1].  
• Objective: Implement a function that returns the indices of the first and last occurrences of the target value.  
• YouTube Solution (Java):** [**Find First and Last Position of Element in Sorted Array - Java Solution**](https://www.youtube.com/watch?v=4sQL7R5ySUU)

**Submission Checklist  
• Time and space complexity analysis for each solution.  
• Test cases demonstrating the correctness of your solutions.**

**class Solution {**

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

**int first=firstBound(nums,target,true);**

**int last=firstBound(nums,target,false);**

**return new int[]{first, last};**

**}**

**public int firstBound(int[] nums,int target,boolean isfirst){**

**int left=0;**

**int right=nums.length-1;**

**int ans=-1;**

**while(left<=right){**

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

**if(nums[mid]==target){**

**ans=mid;**

**if(isfirst){**

**right=mid-1;**

**}**

**else{**

**left=mid+1;**

**}**

**}**

**else if(nums[mid]<target){**

**left=mid+1;**

**}**

**else{**

**right=mid-1;**

**}**

**}**

**return ans;**

**}**

**}**