**Infinite Champions Programme – Day 5 (Assignment Sheet)**

**Instructions  
• Deadline: Submit your solutions by 1st October, 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 (452)**](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, input is the start and end coordinates of the horizontal diameter. Return the minimum number of arrows required to burst all balloons.  
   • Objective: Use sorting and greedy choice to minimize the number of arrows.  
   • YouTube Solution (Java): [Minimum Number of Arrows to Burst Balloons – Java Solution](https://www.youtube.com/watch?v=Z9wX3N9x2rA)

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=0;i<points.length;i++){

            if(points[i][0]>end){

                arrow++;

                end=points[i][1];

            }

        }

        return arrow;

    }

}

1. [**Non-overlapping Intervals (435)**](https://leetcode.com/problems/non-overlapping-intervals/)  
   • Problem: Given a collection of intervals, find the minimum number of intervals you need to remove to make the rest non-overlapping.  
   • Objective: Apply greedy interval scheduling by sorting intervals by end time.  
   • YouTube Solution (Java): [Non-overlapping Intervals – Java Solution](https://www.youtube.com/watch?v=BTObFnHbD4U)

class Solution {

    public int eraseOverlapIntervals(int[][] intervals) {

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

        int count = 0;

        int n=intervals.length;

        int last= Integer.MIN\_VALUE;

        for (int[] interval : intervals) {

            if (interval[0] >= last) {

                count++;

                last = interval[1];

            }

        }

        return n - count;

    }

}

1. [**Queue Reconstruction by Height (406)**](https://leetcode.com/problems/queue-reconstruction-by-height/)  
   • Problem: You are given an array of people represented by pairs (h, k), where h is the height and k is the number of people in front of this person with height greater or equal to h. Reconstruct the queue.  
   • Objective: Use greedy sorting by height and insert people into the correct positions.  
   • YouTube Solution (Java): [Queue Reconstruction by Height – Java Solution](https://www.youtube.com/watch?v=qUL3gj0QjIU)

class Solution {

    public int[][] reconstructQueue(int[][] people) {

        Arrays.sort(people, (a, b) -> {

            if (a[0] == b[0]){

                return a[1] - b[1];

            }

            else{

               return b[0] - a[0];

            }

        });

        List<int[]> result = new ArrayList<>();

        for (int[] person : people) {

            result.add(person[1], person);

        }

        return result.toArray(new int[people.length][]);

    }

}

1. [**Lemonade Change (860)**](https://leetcode.com/problems/lemonade-change/)  
   • Problem: At a lemonade stand, each lemonade costs $5. Customers pay with $5, $10, or $20 bills. Determine if you can provide correct change to every customer.  
   • Objective: Use greedy coin change strategy to handle payments.  
   • YouTube Solution (Java): [Lemonade Change – Java Solution](https://www.youtube.com/watch?v=0yGf7mS9iQ0)

class Solution {

    public boolean lemonadeChange(int[] bills) {

        int five = 0, ten = 0;

        for (int bill : bills) {

            if (bill == 5) {

                five++;

            }

            else if (bill == 10) {

                if (five > 0) {

                    five--;

                    ten++;

                } else {

                    return false;

                }

            }

            else {

                if (ten > 0 && five > 0) {

                    ten--;

                    five--;

                } else if (five >= 3) {

                    five -= 3;

                } else {

                    return false;

                }

            }

        }

        return true;

    }

}

1. [**Boats to Save People (881)**](https://leetcode.com/problems/boats-to-save-people/)  
   • Problem: You are given an array people where people[i] is the weight of the ith person, and each boat can carry at most two people with a weight limit. Return the minimum number of boats required.  
   • Objective: Use two pointers and greedy pairing to minimize boats.  
   • YouTube Solution (Java): [Boats to Save People – Java Solution](https://www.youtube.com/watch?v=3VdgjTUdUuA)

class Solution {

    public int numRescueBoats(int[] people, int limit) {

         Arrays.sort(people);

        int l= 0, r= people.length - 1;

        int count = 0;

        while (l<= r) {

            if (people[l] + people[r] <= limit) {

                l++;

            }

            r--;

            count++;

        }

        return count;

    }

}

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