**19/05/25(Problem 1)**

**Problem Statement:**

A group of N friends plans a trip. Each friend suggests a destination along with the maximum amount they can spend on the trip. The cost of each destination is equal to the amount proposed by the person who suggested it.

Friends can pool their remaining money to help others afford a destination they couldn't otherwise. A destination is considered feasible if every friend can attend using their own money and/or the pooled money from others.

Write a program to determine the best quality destination(s) — defined as the destination(s) with the highest cost that the group can collectively afford.

**Inputs:**

* An integer n — the number of friends.
* An int[] amounts — array of maximum trip amounts corresponding to each destination.

**Output:**

* Return the highest possible cost that all friends can afford together.

**Constraints:**

* 1≤n≤105   
  (The number of friends can be as large as 100,000.)
* 0≤amounts[i]≤106  
  (Each friend’s maximum amount can range from 0 to 1,000,000.)
* The sum of all amounts could potentially be quite large, so the total amount can range from 0 to 106×105=1011.

**Source Code:-**

import java.util.Arrays;

import java.util.Scanner;

public class TripDestination {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Read the number of friends

        System.out.print("Enter the number of friends: ");

        int n = scanner.nextInt();

        int[] amounts = new int[n];

        System.out.println("Enter the maximum amount each friend can spend:");

        // Read each friend's maximum amount

        for (int i = 0; i < n; i++) {

            amounts[i] = scanner.nextInt();

        }

        int highestCost = findHighestAffordableDestination(n, amounts);

        System.out.println("The highest possible cost that all friends can afford together is: " + highestCost);

        scanner.close();

    }

    public static int findHighestAffordableDestination(int n, int[] amounts) {

        // Calculate the total amount pooled together

        int totalAmount = Arrays.stream(amounts).sum();

        // Calculate the average amount

        double average = (double) totalAmount / n;

        // Sort the amounts in descending order

        Arrays.sort(amounts);

        int highestCost = 0;

        // Check from the highest amount downwards

        for (int i = amounts.length - 1; i >= 0; i--) {

            // If the current amount is less than or equal to the average

            if (amounts[i] <= average) {

                highestCost = amounts[i];

                break; // We found the best affordable destination

            }

        }

        return highestCost;

    }

}

**Complexities:-**

* Time Complexity: O(n log n)
* Space Complexity: O(n)