Bit Manipulation & Greedy Assignment

20BD1A1290-Jakkanagari Harika

Program 1:

You are given N fruits. The weight of the fruits is represented by an array A. All those fruits which have the same weight can be sliced in one step. Task: Your task is to determine the number of steps to slice all the fruits.

Solution:

import java.util.\*;

class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

System.out.println(method(a,n));

}

public static int method(int a[],int n){

ArrayList<Integer> al=new ArrayList<>();

for(int i=0;i<n;i++){

if(!al.contains(a[i])){

al.add(a[i]);

}

}

return al.size();

}

}

Example:

Input: N = 4, A = [3, 6, 7, 7]

Output: 3

Program 2:

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money. Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1. You may assume that you have an infinite number of each kind of coin.

Solution:

import java.util.\*;

class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

int amount=sc.nextInt();

System.out.println(method(a,amount,n));

}

public static int method(int a[], int amount,int n) {

if (amount == 0) {

return amount;

}

if (n == 1) {

if (amount % a[0] == 0) {

return amount / a[0];

} else {

return -1;

}

}

Arrays.sort(a);

int numberOfWays = 0;

for (int i =n - 1; i >= 0; i--) {

if (a[i] <= amount) {

numberOfWays += (amount / a[i]);

amount -= (a[i] \* (amount / a[i]));

}

}

if(numsberOfWays==0){

return -1;

}

return numberOfWays;

}

}

Example:

Input: coins = [1,2,5], amount = 11

Output: 3

Program 3:

There are n boxes with different quantities of candies in each of them. The 𝑖-th box has 𝑎𝑖 candies inside. You also have n friends that you want to give the candies to, so you decided to give each friend a box of candies. But, you don't want any friends to get upset so you decided to eat some (possibly none) candies from each box so that all boxes have the same quantity of candies in them. Note that you may eat a different number of candies from different boxes and you cannot add candies to any of the boxes. What's the minimum total number of candies you have to eat to satisfy the requirements?

Solution:

import java.util.\*;

public class Main{

public static int minTotalCandies(int[] boxes) {

int n = boxes.length;

int sum = 0;

int minCandies = Arrays.stream(boxes).min().getAsInt();

for (int i = 0; i < n; i++) {

int r=0;

int a=boxes[i];

while(a>minCandies){

a--;

r++;

}

sum+=r;

}

return sum;

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

int minCandies = minTotalCandies(a);

System.out.println(minCandies);

}

}

Example:

Input: n = 5, a = [1,2,3,4,5]

Output: 10

Program 5:

Given an integer array input = nums of unique elements, return all possible Subsets (the power set) The solution set must not contain duplicate subsets. Return the solution in any order.

Solution:

import java.io.\*;

import java.util.\*;

class Main{

public static void findSubsets(int j, int[] a, ArrayList<Integer> al,ArrayList<ArrayList<Integer>> res)

{

if (!al.isEmpty()) {

if (!res.contains(al)) {

res.add(new ArrayList<>(al));

}

}

res.add(new ArrayList<>(al));

for (int i =j; i <a.length; i++) {

if (i !=j&&a[i] ==a[i - 1])

continue;

al.add(a[i]);

findSubsets(i + 1,a, al, res);

al.remove(al.size() - 1);

}

}

public static ArrayList<ArrayList<Integer>> method(int a[], int n)

{

Arrays.sort(a);

ArrayList<ArrayList<Integer>> res= new ArrayList<>();

ArrayList<Integer> al=new ArrayList<>();

res.add(al);

findSubsets(0, a,al,res);

return res;

}

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] a =new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

System.out.println(method(a,n));

}

}

Example:

Input: nums = [1,2,3]

Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]

Program 6:

Given an array of integers, every element appears thrice except for one, which occurs once. Find that element that does not appear thrice. NOTE: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory? Problem Constraints 2 <= A <= 5\*10 6 0 <= A <= INTMAX Input Format First and only argument of input contains an integer array A. Output Format Return a single integer.

Solution:

import java.util.\*;

class Main{

static int method(int arr[], int n)

{

int ones = 0, twos = 0;

int cm;

for (int i = 0; i < n; i++) {

twos = twos | (ones & arr[i]);

ones = ones ^ arr[i];

cm = ~(ones & twos);

ones &= cm;

twos &= cm;

}

return ones;

}

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] a =new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

System.out.println(method(a,n));

}

}

Example:

Input 1:  A = [1, 2, 4, 3, 3, 2, 2, 3, 1, 1] Input 2:  A = [0, 0, 0, 1]

Output 1: 4 Output 2: 1

Program 7

Min XOR value Problem Description Given an integer array A of N integers, find the pair of integers in the array which have minimum XOR value. Report the minimum XOR value. Problem Constraints 2 <= length of the array <= 100000 0 <= A[i] <= 10 9 Input Format First and only argument of input contains an integer array A. Output Format Return a single integer denoting minimum xor value.

Solution:

import java.util.\*;

class Main{

static int method(int arr[], int n)

{

int min\_xor = Integer.MAX\_VALUE;

for (int i = 0; i < n; i++)

for (int j = i + 1; j < n; j++)

min\_xor = Math.min(min\_xor, arr[i] ^ arr[j]);

return min\_xor;

}

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] a =new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

System.out.println(method(a,n));

}

}

Example:

Input 1:  A = [0, 2, 5, 7] Input 2:  A = [0, 4, 7, 9]

Output 1: 2 Output 2: 3

Program 8:

Single Integer II

Problem Description

Given an array of integers A, every element appears twice except for one. Find that integer that occurs once. NOTE: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Problem Constraints 1 <= |A| <= 2000000 0 <= A[i] <= INTMAX

Input Format

The first and only argument of input contains an integer array A.

Output Format

Return a single integer denoting the single element.

Solution:  
import java.util.\*;

class Main{

static int method(int ar[], int ar\_size)

{

int res = ar[0];

for (int i = 1; i < ar\_size; i++)

res = res ^ ar[i];

return res;

}

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] a =new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

System.out.println(method(a,n));

}

}

Example Input

Input 1:  A = [1, 2, 2, 3, 1] Input 2:  A = [1, 2, 2]

Example Output

Output 1: 3 Output 2: 1

Program 9:

The Hamming distance between two integers is the number of positions at which the corresponding bits are different. Given two integers x and y, return the Hamming distance between them.

Solution:

import java.util.\*;

class Main{

static int method(int n,int a)

{

int res=n^a;

int c=0;

while(res>0){

c+=res&1;

res>>=1;

}

return c;

}

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a=sc.nextInt();

System.out.println(method(a,n));

}

}

Example:  
Input: x = 1, y = 4 Output: 2