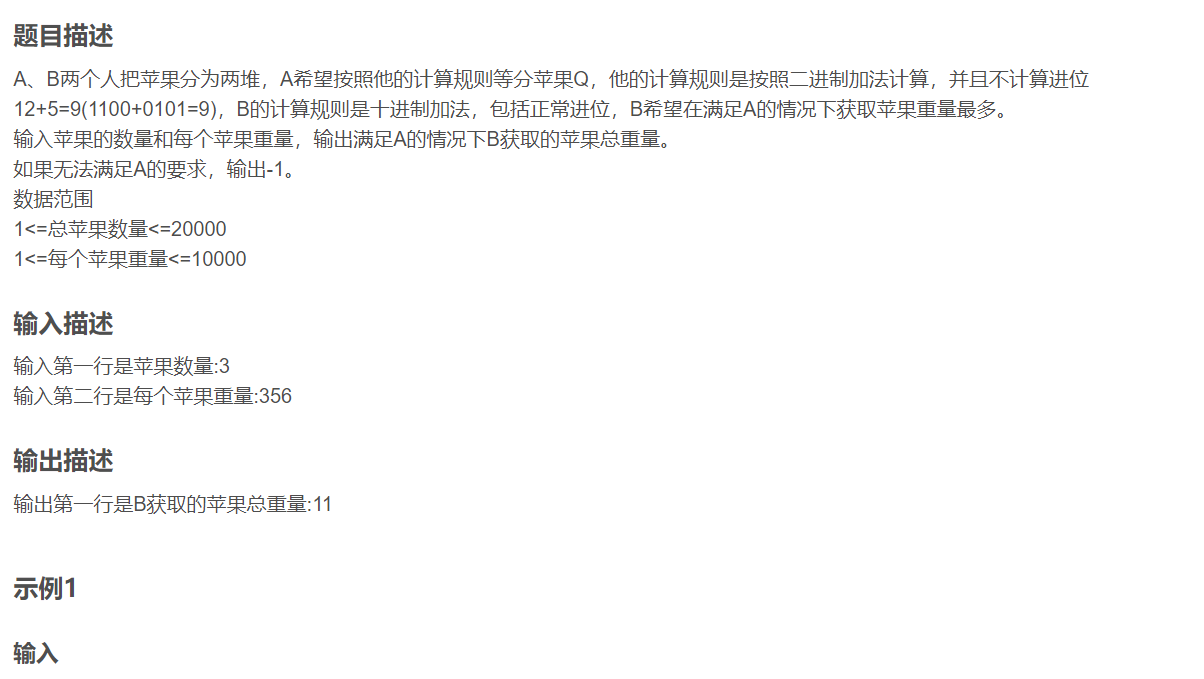
# **E卷-分苹果[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go ）**



3

3 5 6



11

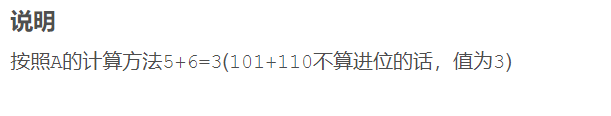


8

7258 6579 2602 6716 3050 3564 5396 1773



35165





import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt(); // 输入苹果数量

int[] arr = new int[n]; // 定义并分配数组存储每个苹果的重量

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

arr[i] = sc.nextInt(); // 输入每个苹果的重量

}

int weight = 0; // 初始化用于A的计算规则的累加值

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

weight ^= arr[i]; // 按照A的二进制规则（不计算进位）累加所有苹果重量

}

if (weight == 0) { // 如果按照A的规则计算，总重量为0

Arrays.sort(arr); // 对苹果重量进行排序，升序排列

int totalWeight = 0;

for (int i = 1; i < n; i++) { // 除去一个最重的苹果

totalWeight += arr[i]; // 计算剩下苹果的总重量

}

System.out.println(totalWeight); // 输出B获取的最大苹果总重量

} else {

System.out.println(-1); // 如果按A的规则，无法等分苹果，输出-1

}

}

}



n = int(input()) # 输入苹果数量

arr = [int(\_) for \_ in input().split()] # 输入每个苹果的重量，并存入数组

weight = 0 # 初始化用于A的计算规则的累加值

for i in range(0, n):

weight = weight ^ arr[i] # 按照A的二进制规则（不计算进位）累加所有苹果重量

if weight == 0: # 如果按照A的规则计算，总重量为0

arr.sort(reverse=True) # 按照降序排列苹果重量

arr.pop() # 弹出一个苹果，以保证B获取的重量最多

print(sum(arr)) # 输出B获取的最大苹果总重量

else:

print(-1) # 如果按A的规则，无法等分苹果，输出-1



#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

int n;

cin >> n; // 输入苹果数量

vector<int> arr(n); // 定义向量存储每个苹果的重量

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

cin >> arr[i]; // 输入每个苹果的重量

}

int weight = 0; // 初始化用于A的计算规则的累加值

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

weight ^= arr[i]; // 按照A的二进制规则（不计算进位）累加所有苹果重量

}

if (weight == 0) { // 如果按照A的规则计算，总重量为0

sort(arr.begin(), arr.end(),

greater<int>()); // 对苹果重量进行排序，降序排列

int totalWeight = 0;

for (int i = 0; i < n - 1; ++i) { // 除去一个最重的苹果

totalWeight += arr[i]; // 计算剩下苹果的总重量

}

cout << totalWeight << endl; // 输出B获取的最大苹果总重量

} else {

cout << -1 << endl; // 如果按A的规则，无法等分苹果，输出-1

}

return 0;

}



#include <stdio.h>

#include <stdlib.h>

// 比较函数用于qsort

int cmp(const void\* a, const void\* b) {

return (\*(int\*)b - \* (int\*)a); // 降序排列

}

int main() {

int n;

scanf("%d", &n); // 输入苹果数量

int arr[n]; // 定义数组存储每个苹果的重量

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

scanf("%d", &arr[i]); // 输入每个苹果的重量

}

int weight = 0; // 初始化用于A的计算规则的累加值

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

weight ^= arr[i]; // 按照A的二进制规则（不计算进位）累加所有苹果重量

}

if (weight == 0) { // 如果按照A的规则计算，总重量为0

qsort(arr, n, sizeof(int), cmp); // 对苹果重量进行排序，降序排列

int totalWeight = 0;

for (int i = 0; i < n - 1; i++) { // 除去一个最重的苹果

totalWeight += arr[i]; // 计算剩下苹果的总重量

}

printf("%d\n", totalWeight); // 输出B获取的最大苹果总重量

} else {

printf("-1\n"); // 如果按A的规则，无法等分苹果，输出-1

}

return 0;

}



// 引入读写IO模块

const readline = require("readline");

// 创建接口以便于读取输入

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

// 读取所有行输入

let input = [];

rl.on("line", (line) => {

input.push(line);

});

// 处理输入

rl.on("close", () => {

let n = parseInt(input[0]); // 输入苹果数量

let arr = input[1].split(" ").map(Number); // 输入每个苹果的重量，并转换为数组

let weight = 0; // 初始化用于A的计算规则的累加值

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

weight ^= arr[i]; // 按照A的二进制规则（不计算进位）累加所有苹果重量

}

if (weight === 0) {

// 如果按照A的规则计算，总重量为0

arr.sort((a, b) => a - b); // 对苹果重量进行排序，降序排列

arr.shift(); // 去掉最重的一个苹果

let totalWeight = arr.reduce((sum, value) => sum + value, 0); // 计算剩下苹果的总重量

console.log(totalWeight); // 输出B获取的最大苹果总重量

} else {

console.log(-1); // 如果按A的规则，无法等分苹果，输出-1

}

});



package main

import (

"bufio"

"fmt"

"os"

"sort"

"strconv"

"strings"

)

func main() {

reader := bufio.NewReader(os.Stdin)

// 读取第一行输入，苹果数量

line, \_ := reader.ReadString('\n')

n, \_ := strconv.Atoi(strings.TrimSpace(line))

// 读取第二行，苹果重量，并转换成整数数组

line, \_ = reader.ReadString('\n')

weightsStr := strings.Split(strings.TrimSpace(line), " ")

weights := make([]int, n)

for i := 0; i < n; i++ {

weights[i], \_ = strconv.Atoi(weightsStr[i])

}

// 初始化用于A的计算规则的累加值

weight := 0

for \_, w := range weights {

weight ^= w // 按照A的二进制规则（不计算进位）累加所有苹果重量

}

if weight == 0 { // 如果按照A的规则计算，总重量为0

sort.Sort(sort.Reverse(sort.IntSlice(weights))) // 对苹果重量进行排序，降序排列

totalWeight := 0

// 去掉最重的一个苹果

for i := 0; i < n - 1; i++ {

totalWeight += weights[i] // 计算剩下苹果的总重量

}

fmt.Println(totalWeight) // 输出B获取的最大苹果总重量

} else {

fmt.Println(-1) // 如果按A的规则，无法等分苹果，输出-1

}

}