27、分积木，考点 or 实现——位运算

题目描述

Solo和koko是两兄弟，妈妈给了他们一大堆积木，每块积木上都有自己的重量。现在他们想要将这些积木分成两堆。哥哥Solo负责分配，弟弟koko要求两个人获得的积木总重量"相等"（根据Koko的逻辑），个数可以不同，不然就会哭，但koko只会先将两个数转成二进制再进行加法，而且总会忘记进位（每个进位都忘记）。如当25（11101）加11（01011）时，koko得到的计算结果是18（10010）：

```
 11001
+01011
--------
 10010
```

Solo想要尽可能使自己得到的积木总重量最大，且不让koko哭。

输入描述

第一行是一个整数N(2≤N≤100)，表示有多少块积木；

第二行为空格分开的N个整数Ci(1≤Ci≤106)，表示第i块积木的重量。

输出描述

如果能让koko不哭，输出Solo所能获得积木的最大总重量；否则输出"NO"。

用例

输入	3 3 5 6
输出	11
说明	无

题目解析

此题中Koko的计算逻辑其实就是 按位异或，即两个相应的二进制位值不同则为1，否则为0。

因此，如果我们想按Koko的求和逻辑平分总重量的话，必然要生成两份相同的二进制数重量，而两个相同二进制数按位异或的结果就是0。

因此我们只要按位异或所有重量，最终结果为0的话，才能按照Koko的逻辑平分总重量。

而一旦可以平分总重量，则减去任意一个重量，都可以分成两份相同的二进制数，而为了使Solo能分得最大重量，则必然减去一个最轻的给Koko。

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const n = parseInt(lines[0]);
    const weights = lines[1].split(" ").map(Number).slice(0, n);

    console.log(getSoloMaxWeight(weights));

    lines.length = 0;
  }
});

function getSoloMaxWeight(weights) {
  // 升序排序
  weights.sort((a, b) => a - b);

  const min = weights[0];

  // correctSum记录Solo计算的正确的总重量
  let correctSum = min;
  // faultSum记录Koko计算的错误的总重量
  let faultSum = min;

  for (let i = 1; i < weights.length; i++) {
    correctSum += weights[i];
    // Koko的计算方法其实就是二进制按位异或运算，即如果两个相应的二进制位值不同则为1，否则为0。
    faultSum ^= weights[i];
  }

  // 如果按照Koko计算方法，若想按重量平分，必然会生成两份相同的二进制数，而两个相同二进制数，按位异或的结果必然是0
  if (faultSum === 0) {
    return correctSum - min; // faultSum=0表示可以平分，因此任意减去一个重量，都可以得到两个相同的二进制数，因此就减去最小的
  } else {
    return "NO"; // faultSum != 0 表示无法按照Koko的逻辑平分
  }
}
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();

    int[] weights = new int[n];
    for (int i = 0; i < n; i++) weights[i] = sc.nextInt();

    System.out.println(getResult(n, weights));
  }

  public static String getResult(int n, int[] weights) {
    // 升序
    Arrays.sort(weights);

    int min = weights[0];

    // correctSum记录Solo计算的正确的总重量
    int correctSum = min;
    // faultSum记录Koko计算的错误的总重量
    int faultSum = min;

    for (int i = 1; i < weights.length; i++) {
      correctSum += weights[i];
      // Koko的计算方法其实就是二进制按位异或运算，即如果两个相应的二进制位值不同则为1，否则为0。
      faultSum ^= weights[i];
    }

    // 如果按照Koko计算方法，若想按重量平分，必然会生成两份相同的二进制数，而两个相同二进制数，按位异或的结果必然是0
    if (faultSum == 0) {
      // faultSum=0表示可以平分，因此任意减去一个重量，都可以得到两个相同的二进制数，因此就减去最小的，这样Solo就可以分得最重的
      return correctSum - min + "";
    } else {
      // faultSum != 0 表示无法按照Koko的逻辑平分
      return "NO";
    }
  }
}
```

Python算法源码

```python
# 输入获取
n = int(input())
weights = list(map(int, input().split()))


# 算法入口
def getResult(n, weights):
    weights.sort()

    minV = weights[0]

    correctSum = minV
    faultSum = minV

    for i in range(1, n):
        correctSum += weights[i]
        faultSum ^= weights[i]

    if faultSum == 0:
        return str(correctSum - minV)
    else:
        return "NO"


# 算法调用
print(getResult(n, weights))
```