题目描述

在星球争霸篮球赛对抗赛中，最大的宇宙战队希望每个人都能拿到MVP，MVP的条件是单场最高分得分获得者。
可以并列所以宇宙战队决定在比赛中尽可能让更多队员上场，并且让所有得分的选手得分都相同，
然而比赛过程中的每1分钟的得分都只能由某一个人包揽。

输入描述

输入第一行为一个数字 t，表示为有得分的分钟数 1 ≤ t ≤ 50
第二行为 t 个数字，代表每一分钟的得分 p， 1 ≤ p ≤ 50

输出描述

输出有得分的队员都是MVP时，最少得MVP得分。

用例

输入	9 5 2 1 5 2 1 5 2 1
输出	6
说明	样例解释 一共 4 人得分，分别都是 6 分 5 + 1， 5 + 1， 5 + 1， 2 + 2 + 2

题目解析

本题和

LeetCode - 698 划分为k个相等的子集_伏城之外的博客-CSDN博客

华为机试 - 叠积木_伏城之外的博客-CSDN博客_叠积木 算法

华为机试 - 等和子数组最小和_伏城之外的博客-CSDN博客_等和子数组

华为机试 - 最大平分数组_伏城之外的博客-CSDN博客

属于同一类问题，解法相同，题解请看上面博客：划分k个相等的子集

JavaScript算法源码

```
/* 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 = lines[0] - 0;
    const arr = lines[1].split(" ").map(Number);

    console.log(getResult(arr, n));
    lines.length = 0;
  }
});

function getResult(arr, n) {
  const sum = arr.sort((a, b) => b - a).reduce((p, c) => p + c);

  let count = n;
  while (count >= 1) {
    // 根据网友指正，由于canPartition方法中会删除arr元素，因此我们不能直接传递arr过去，需要传递arr备份，否则会影响下一次cou
    // if (canPartition(arr, sum, count)) {
    if (canPartition([...arr], sum, count)) {
      return sum / count;
    } else {
      count--;
```

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 = lines[0] - 0;
    const arr = lines[1].split(" ").map(Number);

    console.log(getResult(arr, n));
    lines.length = 0;
  }
});

function getResult(arr, n) {
  const sum = arr.sort((a, b) => b - a).reduce((p, c) => p + c);

  let count = n;
  while (count >= 1) {
    // 根据网友指正，由于canPartition方法中会删除arr元素，因此我们不能直接传递arr过去，需要传递arr备份，否则会影响下一次cou
    // if (canPartition(arr, sum, count)) {
    if (canPartition([...arr], sum, count)) {
      return sum / count;
    } else {
      count--;
    }
  }
}

function canPartition(arr, sum, count) {
  if (sum % count) return false;

  let subSum = sum / count;

  if (subSum < arr[0]) return false;

  while (arr[0] === subSum) {
    arr.shift();
    count--;
  }

  const buckets = new Array(count).fill(0);

  return partition(0, arr, subSum, buckets);
}

function partition(index, arr, subSum, buckets) {
  if (index === arr.length) {
    return true;
  }

  const select = arr[index];

  for (let i = 0; i < buckets.length; i++) {
    if (i > 0 && buckets[i] === buckets[i - 1]) continue;
    if (buckets[i] + select <= subSum) {
      buckets[i] += select;
      if (partition(index + 1, arr, subSum, buckets)) return true;
      buckets[i] -= select;
    }
  }

  return false;
}
```

Java算法源码

感谢m0_71826536指正41行错误，41行在用例

```
5
5 5 5 5 5
```

时会出现越界异常

```java
import java.util.LinkedList;
import java.util.Scanner;

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

    int m = sc.nextInt();
```

Java算法源码

感谢m0_71826536指正41行错误，41行在用例

```
5
5 5 5 5 5
```

时会出现越界异常

```java
import java.util.LinkedList;
import java.util.Scanner;

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

    int m = sc.nextInt();

    LinkedList<Integer> link = new LinkedList<>();
    for (int i = 0; i < m; i++) {
      link.add(sc.nextInt());
    }

    System.out.println(getResult(link, m));
  }

  public static int getResult(LinkedList<Integer> link, int m) {
    link.sort((a, b) -> b - a);

    int sum = 0;
    for (Integer ele : link) {
      sum += ele;
    }

    while (m >= 1) {
      // 根据网友指正，由于canPartition方法中会删除link元素，因此我们不能直接传递link过去，需要传递link备份，否则会影响下一
      //      if (canPartitionMSubsets(link, sum, m)) return sum / m;
      LinkedList<Integer> link_cp = new LinkedList<>(link);
      if (canPartitionMSubsets(link_cp, sum, m)) return sum / m;
      m--;
    }

    return sum;
  }

  public static boolean canPartitionMSubsets(LinkedList<Integer> link, int sum, int m) {
    if (sum % m != 0) return false;

    int subSum = sum / m;

    if (subSum < link.get(0)) return false;

    //    while (link.get(0) == subSum) { // 此段代码可能会出现越界
    while (link.size() > 0 && link.get(0) == subSum) {
      link.removeFirst();
      m--;
    }

    int[] buckets = new int[m];
    return partition(link, 0, buckets, subSum);
  }

  public static boolean partition(LinkedList<Integer> link, int index, int[] buckets, int subSum) {
    if (index == link.size()) return true;

    int select = link.get(index);

    for (int i = 0; i < buckets.length; i++) {
      if (i > 0 && buckets[i] == buckets[i - 1]) continue;
      if (select + buckets[i] <= subSum) {
        buckets[i] += select;
        if (partition(link, index + 1, buckets, subSum)) return true;
        buckets[i] -= select;
      }
    }

    return false;
  }
}
```

Python算法源码

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


# 算法入口
def getResult(link, m):
    link.sort(reverse=True)
```

Python算法源码

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


# 算法入口
def getResult(link, m):
    link.sort(reverse=True)

    sumV = 0
    for ele in link:
        sumV += ele

    while m >= 1:
        if canPartitionMSubsets(link[:], sumV, m):
            return int(sumV / m)
        m -= 1

    return sumV


def canPartitionMSubsets(link, sumV, m):
    if sumV % m != 0:
        return False

    subSum = sumV / m

    if subSum < link[0]:
        return False

    while len(link) > 0 and link[0] == subSum:
        link.pop(0)
        m -= 1

    buckets = [0] * m

    return partition(link, 0, buckets, subSum)


def partition(link, index, buckets, subSum):
    if index == len(link):
        return True

    select = link[index]

    for i in range(len(buckets)):
        if i > 0 and buckets[i] == buckets[i - 1]:
            continue

        if select + buckets[i] <= subSum:
            buckets[i] += select
            if partition(link, index + 1, buckets, subSum):
                return True
            buckets[i] -= select

    return False


# 算法调用
print(getResult(link, m))
```