题目描述

为了压缩空间和时间，我们决定采用一种新的文件格式来存储任务信息。每个任务由一个唯一的ID和一个执行时间表示。例如，任务1需要3个时间单位，任务2需要5个时间单位，任务3需要1个时间单位。

我们的CPU一次只能处理一个任务，一个任务执行完毕，CPU才能执行下一个任务。现在给定一个任务列表，请计算CPU完成所有任务所需的最短时间。

输入描述

- 第一个数组为CPU一次最多可以处理的任务个数，取值范围[1, 1000]
- 第二个数组为任务数目长度，取值范围[1, 1000]
- 第三个数组为任务数组，数字范围[1, 1000]

输出描述

- 执行完成所有任务最少需要多少时。

例例

输入	3 5 1 2 3 4 5
输出	6
说明	一次最多执行3个任务，最小耗时6s

输入	4 5 1 4 5 1 1
输出	5
说明	一次最多执行4个任务，最小耗时5s

题目解析

题干中有一个很关键的词：“**每每完成两个任务后CPU冷却需要花费两个单位时间的时间**”，也就是说，假设冷却花费个数不是一个恒定的时间，而是每完成3个task一次，而这种情况在CPU调度时间中很常见。

示例

```
4
5
2 4 1 1 1
```

第1时，任务数组时间每一个元素-1。
因为CPU只能同时处理2个任务编号处理，所最多3个任务，CPU不能处理两个，因此第1个时间单位，还剩余个任务分别为两个。

第2时，任务数组又减去一个元素-1。
因为CPU只能同时处理2个任务编号处理，所最多一次只能处理2个任务，因此第1个时间单位，还剩余个任务分别为两个。

第3时，任务数组又减去一个元素-1。
因为CPU只能同时处理2个任务编号处理，所最多一次只能处理2个任务，因此第2个时间单位，还剩余个任务分别为两个。

第4时，任务数组又减去一个元素-1。
由于没有任务，因此可以与1个已处理单个，因此本轮不会进行冷却

第5时，任务数组又减去一个元素-1。
由于没有任务，因此可以与1个已处理单个，因此本轮不会进行冷却

因此CPU处理完任务长长的最短所有任务。

示例例例

```
3
5
1 2 3 4 5
```

看看这里心长长是所有可以点。

也想这样是要每2个完成任务要冷却，它所以**2个CPU空闲**每每时下，**最少**需要多长时间时间完成？

长长的每时中，

- 第1时有1个任务需要处理，但是CPU最多可以处理5个，因此会空置3个CPU空闲
- 第2时剩下4个任务需要处理，但是CPU最多可以处理3个，因此会空置2个CPU空闲
- 第3时剩下3个任务需要处理，但是CPU最多可以处理3个，此时不会空置3个CPU空闲
- 第4时剩下2个任务需要处理，但是CPU最多可以处理3个，此时不会空置3个CPU空闲，并且会重新1个任务到下一轮
- 第5时剩下1个任务需要处理，但是CPU最多可以处理3个，并且只有上一轮需要等待冷却，因此CPU会闲置2个，并且会重新3个任务到下一轮。

所以3次工作周期内，为确保充分了CPU空闲，那么每次有冷却时间时间CPU空闲呢？

最后我们分别给出两种正确解答的参考代码，以及解析思路。

在题目中，要解决的核心问题是：任务数组，我们不能把他的任务按顺序执行，而是，按照任务数量和需要冷却时间，来决定哪个任务先执行，哪个不能把他的任务按顺序执行，而是，按照任务数量和需要冷却时间，来决定哪个任务先执行，哪个不能把他的任务按顺序执行，这样才能使CPU空闲时间最少。

Java算法源码

```
import java.util.Scanner;

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

    int maxCount = sc.nextInt();

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

    System.out.println(getResult(maxCount, tasks));
  }

  public static int getResult(int maxCount, int[] tasks) {
    int time = 0;
    int remain = 0;

    for (int task : tasks) {
      if (task + remain > maxCount) {
        remain = task + remain - maxCount;
      } else {
        remain = 0;
      }
      time++;
    }

    while (remain > 0) {
      remain = maxCount;
      time++;
    }

    return time;
  }
}
```

JS算法源码

```
// 引入 readline 模块来处理输入
const readline = require("readline");

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

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

rl.on("close", () => {
  if (lines.length >= 3) {
    let maxCount = parseInt(lines[0]);
    let taskLength = parseInt(lines[1]);
    let tasks = lines[2]
      .split(" ")
      .slice(0, taskLine)
      .map(item => parseInt(item));

    console.log(getProfile(maxCount, tasks));

    lines.length = 0;
  }
});

function getProfile(maxCount, tasks) {
  let time = 0;
  let remain = 0;

  for (let i = 0; i < tasks.length; i++) {
    if (tasks[i] + remain > maxCount) {
      remain = tasks[i] + remain - maxCount;
    } else {
      remain = 0;
    }
    time++;
  }

  while (remain > 0) {
    remain = maxCount;
    time++;
  }

  return time;
}
```

Python算法源码

```
# 输入处理
max_count = int(input())
n = int(input())
tasks = list(map(int, input().split()))

# 计算函数
def get_min():
    time = 0
    remain = 0

    for task in tasks:
        if task + remain > max_count:
            remain = task + remain - max_count
        else:
            remain = 0
        time += 1

    while remain > 0:
        remain = max_count
        time += 1

    return time

# 输出结果
print(get_min())
```