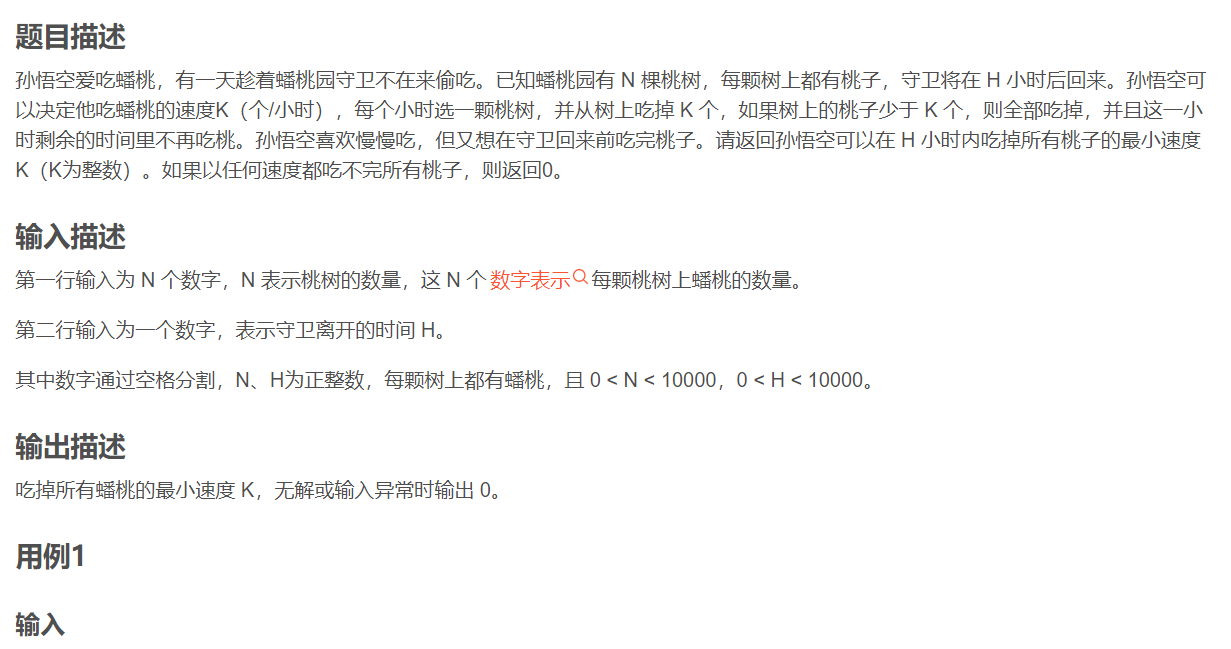
# **E卷-爱吃蟠桃的孙悟空[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go ）**

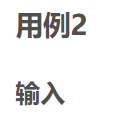


2 3 4 5

4



5

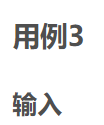


2 3 4 5

3



0



30 11 23 4 20

6



23



import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// 读取输入的桃子数量，并将其转化为整数数组

String[] input = scanner.nextLine().split(" ");

int[] arr = Arrays.stream(input).mapToInt(Integer::parseInt).toArray();

// 读取守卫离开的时间

int t = scanner.nextInt();

// 桃树数量

int n = arr.length;

// 初始化二分查找的下界lo为0

int lo = 1;

// 初始化二分查找的上界hi为桃树中最多的桃子数

int hi = Arrays.stream(arr).max().getAsInt();

// 初始化结果res为无穷大，表示目前还未找到合适的速度

int res = Integer.MAX\_VALUE;

// 辅助函数check，用于检查速度mid能否在规定时间内吃完所有桃子

while (lo <= hi) {

// 取中间值mid作为当前的尝试速度

int mid = (lo + hi) / 2;

// 计算所需的总时间

int totalTime = 0;

for (int p : arr) {

totalTime += (p + mid - 1) / mid; // 计算每棵树所需的时间

}

// 如果在守卫离开前能吃完所有桃子

if (totalTime <= t) {

res = mid; // 更新最小速度K为mid

hi = mid - 1; // 继续尝试更小的速度

} else {

lo = mid + 1; // 否则尝试更大的速度

}

}

// 如果res没有更新，说明无解，输出0

if (res == Integer.MAX\_VALUE) {

System.out.println(0);

} else {

System.out.println(res); // 输出最小速度K

}

}

}



# 读取输入的蟠桃数量，并将其转化为整数列表

arr = list(map(int, input().split()))

# 读取守卫离开的时间

t = int(input())

# 计算桃树的数量

n = len(arr)

# 初始化二分查找的下界lo为0

lo = 0

# 初始化二分查找的上界hi为桃树中最多的桃子数

hi = max(arr)

# 初始化结果res为无穷大，表示目前还未找到合适的速度

res = float("inf")

# 定义一个辅助函数check，用于检查以某个速度mid能否在规定时间内吃完桃子

def check(mid):

# tmp是一个生成器表达式，用于计算在速度为mid时，每棵树所需的时间

tmp = ((p - 1) // mid for p in arr)

# 计算出所有树所需的总时间

s = sum(tmp)

# 计算还剩余的时间x

x = t - n

# 判断在剩余时间x内是否可以吃完所有桃子

return s <= x

# 进行二分查找，确定最小速度K

while lo <= hi:

# 取中间值mid作为当前的尝试速度

mid = (lo + hi) >> 1

# 如果以速度mid能够在时间内吃完所有桃子

if check(mid):

# 更新最小速度K为当前的mid

res = mid

# 尝试更小的速度，缩小搜索范围

hi = mid - 1

else:

# 如果不能吃完，尝试更大的速度，增大搜索范围

lo = mid + 1

# 如果res仍然是无穷大，说明无解

if res == float("inf"):

print(0)

else:

# 输出最小速度K

print(res)



#include <bits/stdc++.h>

using namespace std;

int main() {

// 读取输入

vector<int> arr;

int t;

string input;

getline(cin, input);

stringstream ss(input);

int val;

while (ss >> val) {

arr.push\_back(val);

}

cin >> t;

// 桃树数量

int n = arr.size();

// 初始化二分查找的上下界

int lo = 1, hi = \*max\_element(arr.begin(), arr.end());

// 初始化结果为无穷大

int res = INT\_MAX;

// 二分查找，确定最小速度K

while (lo <= hi) {

// 取中间值mid作为当前的尝试速度

int mid = (lo + hi) / 2;

int totalTime = 0;

// 计算以mid速度吃完所有桃子所需的时间

for (int p : arr) {

totalTime += (p + mid - 1) / mid;

}

// 如果可以在规定时间内吃完

if (totalTime <= t) {

res = mid; // 更新最小速度K为mid

hi = mid - 1; // 尝试更小的速度

} else {

lo = mid + 1; // 否则尝试更大的速度

}

}

// 输出结果，如果没有找到合适的速度则输出0

if (res == INT\_MAX) {

cout << 0 << endl;

} else {

cout << res << endl;

}

return 0;

}



#include <stdio.h>

#include <limits.h>

int max(int arr[], int n) {

int max\_val = arr[0];

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

if (arr[i] > max\_val) {

max\_val = arr[i];

}

}

return max\_val;

}

int main() {

// 读取输入

int arr[10000], t, n = 0;

char c;

// 读取桃树上的桃子数量

while (scanf("%d%c", &arr[n], &c)) {

n++;

if (c == '\n') {

break;

}

}

// 读取守卫离开的时间

scanf("%d", &t);

// 初始化二分查找的下界lo为1

int lo = 1;

// 初始化二分查找的上界hi为桃树中最多的桃子数

int hi = max(arr, n);

// 初始化结果res为无穷大，表示目前还未找到合适的速度

int res = INT\_MAX;

// 二分查找，确定最小速度K

while (lo <= hi) {

// 取中间值mid作为当前的尝试速度

int mid = (lo + hi) / 2;

// 计算所需的总时间

int totalTime = 0;

// 计算以速度mid吃完所有桃子所需的时间

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

totalTime += (arr[i] + mid - 1) / mid;

}

// 如果可以在规定时间内吃完

if (totalTime <= t) {

res = mid; // 更新最小速度K为mid

hi = mid - 1; // 尝试更小的速度

} else {

lo = mid + 1; // 否则尝试更大的速度

}

}

// 输出结果，如果没有找到合适的速度则输出0

if (res == INT\_MAX) {

printf("0\n");

} else {

printf("%d\n", res);

}

return 0;

}



const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let arr = [];

let t = 0;

// 读取输入

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

if (arr.length === 0) {

arr = input.split(" ").map(Number); // 将桃子的数量转化为整数数组

} else {

t = parseInt(input); // 读取守卫离开的时间

solve(); // 调用函数求解问题

rl.close();

}

});

function solve() {

const n = arr.length;

let lo = 1;

let hi = Math.max(...arr); // 找到桃树中最多的桃子数

let res = Infinity;

// 二分查找确定最小速度K

while (lo <= hi) {

const mid = Math.floor((lo + hi) / 2); // 取中间值mid

let totalTime = 0;

// 计算以速度mid吃完所有桃子所需的时间

for (const p of arr) {

totalTime += Math.ceil(p / mid); // 计算每棵树所需的时间

}

// 如果可以在规定时间内吃完

if (totalTime <= t) {

res = mid; // 更新最小速度K为mid

hi = mid - 1; // 尝试更小的速度

} else {

lo = mid + 1; // 否则尝试更大的速度

}

}

// 输出结果

if (res === Infinity) {

console.log(0); // 如果没有找到合适的速度则输出0

} else {

console.log(res); // 输出最小速度K

}

}



package main

import (

"fmt" // 导入fmt包用于输入输出

)

// 定义一个函数，用于找到数组中最大的值

func max(arr []int) int {

maxVal := arr[0] // 假设数组中的第一个元素是最大值

for \_, val := range arr { // 遍历数组

if val > maxVal { // 如果当前元素比maxVal大，则更新maxVal

maxVal = val

}

}

return maxVal // 返回数组中的最大值

}

func main() {

var arr []int // 定义一个切片用于存储每棵树的桃子数量

var t, num int // 定义两个整数变量t为守卫离开的时间，num为每棵树上的桃子数

// 循环读取输入的桃子数量，直到输入结束

for {

if \_, err := fmt.Scanf("%d", &num); err == nil {

arr = append(arr, num) // 将读取的桃子数量添加到数组中

} else {

break // 如果没有更多输入，则退出循环

}

}

t = arr[len(arr) - 1]

arr = arr[:len(arr) - 1]

// 读取守卫离开的时间t

// 初始化二分查找的下界lo为1

lo := 1

// 初始化二分查找的上界hi为数组中最大的桃子数

hi := max(arr)

// 初始化结果res为无穷大，表示尚未找到合适的最小速度K

res := 1<<31-1

// 二分查找，确定最小速度K

for lo <= hi {

// 计算中间值mid作为当前的尝试速度

mid := (lo + hi) / 2

// 计算以速度mid吃完所有桃子所需的总时间

totalTime := 0

for \_, p := range arr { // 遍历每棵树上的桃子数量

totalTime += (p + mid - 1) / mid // 计算每棵树所需的时间并累加

}

// 如果总时间小于等于守卫离开的时间t

if totalTime <= t {

// 更新最小速度K为mid

res = mid

// 继续尝试更小的速度，缩小上界

hi = mid - 1

} else {

// 如果总时间大于t，则尝试更大的速度，增大下界

lo = mid + 1

}

}

// 如果结果res仍然是无穷大，说明无解，输出0

if res == 1<<31-1 {

fmt.Println(0)

} else {

// 输出找到的最小速度K

fmt.Println(res)

}

}