

# 【华为OD机考 统一考试机试C卷】 机器人仓库搬砖（C++ Java JavaScript Python C语言）

## 华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，**C卷真题已基本整理完毕**

抽到原题的概率为2/3到3/3，**也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。**

另外订阅专栏还可以联系笔者开通在线 **OJ** 进行刷题，提高刷题效率。

**真题目录：**华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明

**专栏：**2023华为OD机试( B卷+C卷+D卷) (C++JavaJSPy)

**华为OD面试真题精选：**华为OD面试真题精选

**在线OJ：**点击立即刷题，模拟真实机考环境

## 题目描述

机器人搬砖，一共有N堆砖存放在N个不同的仓库中，第i堆砖中有bricks[i]块砖头，要求在8小时内搬完。机器人每小时能搬砖的数量取决于有多少能量格，机器人一个小时中只能在一个仓库中搬砖，机器人的能量格每小时补充一次且能量格只在这一个小时有效，为使得机器人损耗最小化尽量减小每次补充的能量格数 为了保障在8小时内能完成搬砖任务，请计算每小时给机器人充能的最小能量格数。

- 1、无需考虑机器人补充能量格的耗时，
- 2、无需考虑机器人搬砖的耗时；
- 3、机器人每小时补充能量格只在这一个小时中有效；

## 输入描述

第一行为一行数字，空格分隔

## 输出描述

机器人每小时最少需要充的能量格，若无法完成任务，输出 -1

## 示例1

|    |               |
|----|---------------|
| 输入 | 30 12 25 8 19 |
| 输出 | 15            |

示例2

|    |                                 |
|----|---------------------------------|
| 输入 | 10 12 25 8 19 8 6 4 17 19 20 30 |
| 输出 | -1                              |

解题思路

1. 首先，检查砖块的数量是否大于8，如果大于8，则返回-1。在本例中，砖块数量为5，因此不返回-1。
2. 初始化左右边界 `left` 和 `right`。`left` 设置为1（因为至少需要每小时1块能量），`right` 设置为数组中的最大值（即30）。
3. 进入 `while` 循环执行二分查找。这个循环将不断缩小搜索范围直到找到最小的每小时能量块数量。
4. 在每次循环中，计算中间值 `middle`，然后计算使用 `middle` 作为每小时能量块数量时，完成搬运所有砖块需要的总时间 `total_time`。
5. 如果 `total_time` 大于 `hours`，说明 `middle` 太小，不能在规定时间内完成任务，因此将 `left` 设置为 `middle + 1`。
6. 如果 `total_time` 小于等于 `hours`，说明 `middle` 可能太大或刚好合适，因此将 `right` 设置为 `middle` 以尝试找到更小的值。
7. 当 `left` 和 `right` 相遇时，循环结束，此时 `left` 即为我们寻找的最小能量块数量。
8. 最后，再次计算使用 `left` 作为每小时能量块数量时，完成搬运所有砖块需要的总时间 `sum`。如果 `sum` 大于 `hours`，说明即使是 `left` 也不能在规定时间内完成任务，因此返回-1。
9. 如果 `sum` 小于等于 `hours`，则 `left` 是可以在规定时间内完成任务的最小能量块数量，返回 `left`。

现在，我们分析给定的用例：

- 输入： 30 12 25 8 19
- 输出： 15

执行过程如下：

1. 初始化 `left = 1` 和 `right = 30`。
2. 进入循环, 计算 `middle`。
3. 第一次循环:  $middle = (1 + 30) / 2 = 15$ , 计算 `total_time` 为  $2 + 1 + 2 + 1 + 2 = 8$ 。因为 `total_time` 等于 `hours`, 所以 `right` 更新为 `15`。
4. 第二次循环:  $middle = (1 + 15) / 2 = 8$ , 计算 `total_time` 为  $4 + 2 + 4 + 1 + 3 = 14$ 。因为 `total_time` 大于 `hours`, 所以 `left` 更新为 `9`。
5. 第三次循环:  $middle = (9 + 15) / 2 = 12$ , 计算 `total_time` 为  $3 + 1 + 3 + 1 + 2 = 10$ 。因为 `total_time` 大于 `hours`, 所以 `left` 更新为 `13`。
6. 第四次循环:  $middle = (13 + 15) / 2 = 14$ , 计算 `total_time` 为  $3 + 1 + 2 + 1 + 2 = 9$ 。因为 `total_time` 大于 `hours`, 所以 `left` 更新为 `15`。
7. 第五次循环:  $middle = (15 + 15) / 2 = 15$ , 由于 `left` 已经等于 `right`, 循环结束。
8. 最后, `left` 为 `15`, 计算 `sum` 为  $2 + 1 + 2 + 1 + 2 = 8$ , 等于 `hours`, 因此返回 `15`。

所以, 对于给定的输入 `30 12 25 8 19`, 输出应该是 `15`, 这表明每小时至少需要充能15块能量格, 才能在8小时内完成搬砖任务。

## C++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <cmath>
5  using namespace std;
6
7  int minEnergyBlocks(vector<int>& bricks, int hours) {
8      if (bricks.size() > 8) { // 如果砖块数量大于8
9          return -1; // 返回-1
10     }
11     int left = 1, right = *max_element(bricks.begin(), bricks.end()); // 初始化左右边界
12     while (left < right) { // 二分查找
13         int middle = (left + right) / 2; // 取中间值
14         int total_time = 0; // 计算当前能量块数量下能够搬完所有砖的总时间
15         for (int i = 0; i < bricks.size(); i++) {
16             total_time += ceil((double)bricks[i] / middle);
17         }
18         if (total_time > hours) { // 如果当前能量块数量下搬完所有砖的总时间超过了限定时间, 缩小搜索范围
```

```

19         left = middle + 1;
20     } else { // 否则, 减小能量块数量
21         right = middle;
22     }
23 }
24 int sum = 0;
25 for (int i = 0; i < bricks.size(); i++) {
26     sum += ceil((double)bricks[i] / left);
27 }
28 if (sum > hours) { // 检查最终确定的能量块数量是否在规定时间内搬完所有砖
29     return -1; // 无法在规定时间内搬完所有砖
30 }
31 return left; // 返回最小能量块数量
32 }
33
34 int main() {
35     vector<int> bricks;
36     int brick;
37     while (cin >> brick) {
38         bricks.push_back(brick);
39     }
40     cout << minEnergyBlocks(bricks, 8); // 调用函数并输出结果
41     return 0;
42 }

```

## Java

```

1 import java.util.*;
2
3 public class Main {
4     public static int minEnergyBlocks(int[] bricks, int hours) {
5         if (bricks.length > 8) { // 如果砖块数量大于8
6             return -1; // 返回-1
7         }
8         int left = 1, right = Arrays.stream(bricks).max().getAsInt(); // 初始化左右边界
9         while (left < right) { // 二分查找
10             int middle = (left + right) / 2; // 取中间值
11             int total_time = 0; // 计算当前能量块数量下能够搬完所有砖的总时间
12             for (int i = 0; i < bricks.length; i++) {
13                 total_time += Math.ceil((double) bricks[i] / middle);
14             }
15             if (total_time > hours) {
16                 right = middle;
17             } else {
18                 left = middle + 1;
19             }
20         }
21         return left;
22     }
23 }

```

```

14     }
15     if (total_time > hours) { // 如果当前能量块数量下搬完所有砖的总时间超过了限定时间, 缩小搜索范围
16         left = middle + 1;
17     } else { // 否则, 减小能量块数量
18         right = middle;
19     }
20 }
21 int sum = 0;
22 for (int i = 0; i < bricks.length; i++) {
23     sum += Math.ceil((double) bricks[i] / left);
24 }
25 if (sum > hours) { // 检查最终确定的能量块数量是否能在规定时间内搬完所有砖
26     return -1; // 无法在规定时间内搬完所有砖
27 }
28 return left; // 返回最小能量块数量
29 }
30
31 public static void main(String[] args) {
32     Scanner scanner = new Scanner(System.in);
33     String[] input = scanner.nextLine().split(" ");
34     int[] bricks = new int[input.length];
35     for (int i = 0; i < input.length; i++) {
36         bricks[i] = Integer.parseInt(input[i]);
37     }
38     System.out.println(minEnergyBlocks(bricks, 8)); // 调用函数并输出结果
39 }
40 }

```

## javaScript

```

1 function minEnergyBlocks(bricks, hours) {
2     if (bricks.length > 8) { // 如果砖块数量大于8
3         return -1; // 返回-1
4     }
5     let left = 1, right = Math.max(...bricks); // 初始化左右边界
6     while (left < right) { // 二分查找
7         let middle = Math.floor((left + right) / 2); // 取中间值
8         let total_time = 0; // 计算当前能量块数量下能够搬完所有砖的总时间
9         for (let i = 0; i < bricks.length; i++) {
10             total_time += Math.ceil(bricks[i] / middle);
11         }
12         if (total_time > hours) {
13             left = middle + 1;
14         } else {
15             right = middle;
16         }
17     }
18     return left;
19 }

```

```

11     }
12     if (total_time > hours) { // 如果当前能量块数量下搬完所有砖的总时间超过了限定时间, 缩小搜索范围
13         left = middle + 1;
14     } else { // 否则, 减小能量块数量
15         right = middle;
16     }
17 }
18 let sum = 0;
19 for (let i = 0; i < bricks.length; i++) {
20     sum += Math.ceil(bricks[i] / left);
21 }
22 if (sum > hours) { // 检查最终确定的能量块数量是否能在规定时间内搬完所有砖
23     return -1; // 无法在规定时间内搬完所有砖
24 }
25 return left; // 返回最小能量块数量
26 }
27
28 const readline = require('readline');
29 const rl = readline.createInterface({
30     input: process.stdin,
31     output: process.stdout
32 });
33
34 rl.on('line', (input) => {
35     const bricks = input.split(' ').map(Number);
36     console.log(minEnergyBlocks(bricks, 8)); // 调用函数并输出结果
37     rl.close();
38 });

```

## Python

```

1 import math
2
3 def min_energy_blocks(bricks, hours):
4     if len(bricks) > 8:
5         return -1
6     # 初始化左右边界
7     left, right = 1, max(bricks)
8     # 二分查找
9     while left < right:
10

```

```

10     # 取中间值
11     middle = (left + right) // 2
12     # 计算当前能量块数量下能够搬完所有砖的总时间
13     total_time = sum(math.ceil(i/middle) for i in bricks)
14     if total_time > hours:
15         # 如果当前能量块数量下搬完所有砖的总时间超过了限定时间, 缩小搜索范围
16         left = middle + 1
17     else:
18         # 否则, 减小能量块数量
19         right = middle
20     # 检查最终确定的能量块数量是否能在规定时间内搬完所有砖
21     if sum(math.ceil(i/left) for i in bricks) > hours:
22         return -1 # 无法在规定时间内搬完所有砖
23     return left # 返回最小能量块数量
24
25 # 从输入中获取砖块数量列表
26 bricks = list(map(int, input().split()))
27 # 调用函数并输出结果
28 print(min_energy_blocks(bricks, 8))

```

## C语言

```

1  #include <stdio.h>
2  #include <math.h>
3
4  #define MAX_BRICKS 100 // 定义最大砖块堆数
5
6  // 函数: 计算机器人每小时最少需要充的能量格
7  int minEnergyBlocks(int bricks[], int n, int hours) {
8      if (n > hours) { // 如果砖块数量大于时间限制
9          return -1; // 无法完成, 返回-1
10     }
11
12     // 寻找数组中最大元素
13     int maxBrick = bricks[0];
14     for (int i = 1; i < n; i++) {
15         if (bricks[i] > maxBrick) {
16             maxBrick = bricks[i];
17         }
18     }
19
20     // 计算所需能量格
21     int totalEnergy = 0;
22     for (int i = 0; i < n; i++) {
23         totalEnergy += (bricks[i] + maxBrick - 1) / maxBrick;
24     }
25     return totalEnergy;
26 }
27
28 int main() {
29     int n, hours;
30     scanf("%d %d", &n, &hours);
31     int bricks[MAX_BRICKS];
32     for (int i = 0; i < n; i++) {
33         scanf("%d", &bricks[i]);
34     }
35     int result = minEnergyBlocks(bricks, n, hours);
36     if (result == -1) {
37         printf("无法在规定时间内搬完所有砖\n");
38     } else {
39         printf("最少需要充 %d 能量格\n", result);
40     }
41     return 0;
42 }

```

```
19
20 int left = 1, right = maxBrick; // 初始化左右边界
21 while (left < right) { // 二分查找
22     int middle = (left + right) / 2; // 取中间值
23     int total_time = 0; // 初始化总时间
24     // 计算在当前能量块数量下, 完成所有砖块的总时间
25     for (int i = 0; i < n; i++) {
26         total_time += ceil((double)bricks[i] / middle);
27     }
28
29     // 判断是否能在规定时间内完成
30     if (total_time > hours) {
31         left = middle + 1;
32     } else {
33         right = middle;
34     }
35 }
36
37 // 检查是否可以在规定时间内完成任务
38 int sum = 0;
39 for (int i = 0; i < n; i++) {
40     sum += ceil((double)bricks[i] / left);
41 }
42 if (sum > hours) {
43     return -1; // 无法完成
44 }
45 return left; // 返回最小能量格数
46 }
47
48 int main() {
49     int bricks[MAX_BRICKS]; // 定义砖块数组
50     int n = 0; // 数组中元素的数量
51     int brick;
52
53     // 从标准输入读取砖块数量
54     while (scanf("%d", &brick) != EOF) {
55         bricks[n] = brick;
56         n++;
57     }
58
59
```



```
60 | // 调用函数并输出结果
61 | printf("%d\n", minEnergyBlocks(bricks, n, 8));
62 | return 0;
    }
```

## 完整用例

### 用例1

5 5 5 5 5 5 5 5

### 用例2

1 1 1 1 1 1 1 100

### 用例3

10 10 10

### 用例4

10 20 30 40 50 60 70 80

### 用例5

8 8 8 8 8 8 8 8

### 用例6

150 120 150 120 150 150 120 150

### 用例7

45 30 60 20 55 35 50 40

### 用例8

7 13 19 25 31 37 43 49

### 用例9

用例10

文章目录

- 华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷
  - 题目描述
  - 输入描述
  - 输出描述
  - 示例1
  - 示例2
  - 解题思路
  - C++
  - Java
  - javaScript
  - Python
  - C语言
  - 完整用例
    - 用例1
    - 用例2
    - 用例3
    - 用例4
    - 用例5
    - 用例6
    - 用例7
    - 用例8
    - 用例9
    - 用例10

# 机考真题 华为OD



CSDN @算法大师