

# 【华为OD机考 统一考试机试C卷】最多购买宝石数目（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

橱窗里有一排宝石，不同的宝石对应不同的价格，宝石的价格标记为  $\text{gems}[i]$

$0 \leq i < n$

$n = \text{gems.length}$

宝石可同时出售0个或多个，如果同时出售多个，则要求出售的宝石编号连续；

例如客户最大购买宝石个数为 $m$ ，购买的宝石编号必须为： $\text{gems}[i], \text{gems}[i+1], \dots, \text{gems}[i+m-1]$

$0 \leq i < n$

$m \leq n$

假设你当前拥有总面值为  $\text{value}$  的钱，请问最多能购买到多少个宝石，如无法购买宝石，则返回0。

## 输入描述

第一行输入 $n$ ，参数类型为int，取值范围： $[0, 10^6]$ ，表示橱窗中宝石的总数量。

之后  $n$  行分别表示从第0个到第 $n-1$ 个宝石的价格，即  $\text{gems}[0]$  到  $\text{gems}[n-1]$  的价格，类型为int，取值范围： $(0, 1000]$ 。

之后一行输入 $v$ ，类型为int，取值范围： $[0, 10^9]$ ，表示你拥有的钱。

输出描述

输出int类型的返回值，表示最大可购买的宝石数量。

用例1

输入

1	7
2	8
3	4
4	6
5	3
6	1
7	6
8	7
9	10

输出

1	3
---	---

说明

gems = [8,4,6,3,1,6,7], value = 10  
最多购买的宝石为gems[2]至gems[4]或者gems[3]至gems[5]

用例2

输入

1	0
2	1

输出

1	0
---	---

说明

```
gems = [], value = 1
因为没有宝石，所以返回0
```

用例3

输入

1	9
2	6
3	1
4	3
5	1
6	8
7	9
8	3
9	2
10	4
11	15

输出

1	4
---	---

说明

```
gems = [6, 1, 3, 1, 8, 9, 3, 2, 4], value = 15
最多购买的宝石为gems[0]至gems[3]
```

用例4

输入

1	9
2	1
3	

4		1
5		1
6		1
7		1
8		1
9		1
10		1
11		1
		10

输出

1		9
---	--	---

说明

`gems = [1, 1, 1, 1, 1, 1, 1, 1, 1], value = 10`  
最多购买的宝石为`gems[0]`至`gems[8]`，即全部购买

## 解题思路

以下是详细的解题思路：

1. 首先，代码从控制台读取宝石的数量，每颗宝石的价格，以及你拥有的钱的总面值。
2. 然后，初始化滑动窗口的左右边界（`left`和`right`）为0，窗口内宝石的总价（`sum`），以及最大可购买的宝石数量（`max`）。
3. 在`while`循环中，首先将右边界的宝石价格加到总价中。如果总价超过你拥有的钱，那么就将左边界的宝石价格从总价中减去，并将左边界向右移动，直到总价不超过你拥有的钱。
4. 然后，更新最大可购买的宝石数量。这是通过比较当前最大可购买的宝石数量和滑动窗口内的宝石数量（`right - left + 1`）来实现的。
5. 最后，将右边界向右移动，扩大滑动窗口。当滑动窗口的右边界超过宝石数量时，结束循环。
6. 在`while`循环结束后，输出最大可购买的宝石数量。

通过这种方式，代码能够找出在给定预算下，最多可以购买的宝石数量。

以例3中的宝石数量是9，宝石的价格分别是6、1、3、1、8、9、3、2、4，你拥有的钱的总面值是15。

以下是详细的计算步骤：

1. 初始化：left=0, right=0, sum=0, max=0。
2. 第一步：将右边界的宝石价格加到总价中，sum=6, right=1, left=0, max=1。
3. 第二步：将右边界的宝石价格加到总价中，sum=7, right=2, left=0, max=2。
4. 第三步：将右边界的宝石价格加到总价中，sum=10, right=3, left=0, max=3。
5. 第四步：将右边界的宝石价格加到总价中，sum=11, right=4, left=0, max=4。
6. 第五步：将右边界的宝石价格加到总价中，sum=19，总价超过了你拥有的钱，所以将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=13, right=4, left=1, max=4。
7. 第六步：将右边界的宝石价格加到总价中，sum=22，总价超过了你拥有的钱，所以将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=21, right=5, left=2, max=4。
8. 第七步：继续将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=18, right=5, left=3, max=4。
9. 第八步：继续将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=17, right=5, left=4, max=4。
10. 第九步：继续将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=9, right=5, left=5, max=4。
11. 第十步：将右边界的宝石价格加到总价中，sum=12, right=6, left=5, max=4。
12. 第十一步：将右边界的宝石价格加到总价中，sum=14, right=7, left=6, max=4。
13. 第十二步：将右边界的宝石价格加到总价中，sum=18，总价超过了你拥有的钱，所以将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=9, right=8, left=7, max=4。
14. 第十四步：滑动窗口的右边界已经超过了宝石数量，结束循环。

因此，最大可购买的宝石数量是4。

0	1	2	3	4	5	6	7	8
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4

## C++

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5
6  int main() {
7      // 创建一个对象来读取输入
8      int n;
9      cin >> n;
10     // 创建一个数组来存储每个宝石的价格
11     vector<int> gems(n);
12     // 读取每个宝石的价格
13     for (int i = 0; i < n; i++) {
14         cin >> gems[i];
15     }
16     // 读取你拥有的钱的总面值
17     int value;
18     cin >> value;
19
20     // 初始化滑动窗口的左右边界和窗口内宝石的总价
21     int left = 0, right = 0, sum = 0;
22     // 初始化最大可购买的宝石数量

```

```

23     int maxT = 0;
24     // 当滑动窗口的右边界没有超过宝石数量时, 继续循环
25     while (right < gems.size()) {
26         // 将右边界的宝石价格加到总价中
27         sum += gems[right];
28         // 当总价超过你拥有的钱时, 将左边界的宝石价格从总价中减去, 并将左边界向右移动
29         while (sum > value) {
30             sum -= gems[left];
31             left++;
32         }
33         // 更新最大可购买的宝石数量
34         maxT = max(maxT, right - left + 1);
35         // 将右边界向右移动
36         right++;
37     }
38     // 输出最大可购买的宝石数量
39     cout << maxT << endl;
40
41     return 0;
42 }

```

## Java

```

1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] args) {
5          // 创建一个 Scanner 对象来读取输入
6          Scanner scanner = new Scanner(System.in);
7          // 读取宝石的数量
8          int n = scanner.nextInt();
9          // 创建一个数组来存储每个宝石的价格
10         int[] gems = new int[n];
11         // 读取每个宝石的价格
12         for (int i = 0; i < n; i++) {
13             gems[i] = scanner.nextInt();
14         }
15         // 读取你拥有的钱的总面值
16         int value = scanner.nextInt();
17
18

```

```

18 // 初始化滑动窗口的左右边界和窗口内宝石的总价
19 int left = 0, right = 0, sum = 0;
20 // 初始化最大可购买的宝石数量
21 int max = 0;
22 // 当滑动窗口的右边界没有超过宝石数量时, 继续循环
23 while (right < gems.length) {
24     // 将右边界的宝石价格加到总价中
25     sum += gems[right];
26     // 当总价超过你拥有的钱时, 将左边界的宝石价格从总价中减去, 并将左边界向右移动
27     while (sum > value) {
28         sum -= gems[left];
29         left++;
30     }
31     // 更新最大可购买的宝石数量
32     max = Math.max(max, right - left + 1);
33     // 将右边界向右移动
34     right++;
35 }
36 // 输出最大可购买的宝石数量
37 System.out.println(max);
38 }
39 }

```

## JavaScript

```

1 const readline = require('readline').createInterface({
2     input: process.stdin,
3     output: process.stdout
4 });
5
6 let input = [];
7
8 readline.on('line', (line) => {
9     input.push(line);
10 });
11
12 readline.on('close', () => {
13     // 读取宝石的数量
14     const n = parseInt(input[0]);
15     // 创建一个数组来存储每个宝石的价格
16

```



```

16     const gems = input.slice(1, n + 1).map(Number);
17     // 读取你拥有的钱的总面值
18     const value = parseInt(input[n + 1]);
19
20     // 初始化滑动窗口的左右边界和窗口内宝石的总价
21     let left = 0, right = 0, sum = 0;
22     // 初始化最大可购买的宝石数量
23     let maxT = 0;
24     // 当滑动窗口的右边界没有超过宝石数量时, 继续循环
25     while (right < gems.length) {
26         // 将右边界的宝石价格加到总价中
27         sum += gems[right];
28         // 当总价超过你拥有的钱时, 将左边界的宝石价格从总价中减去, 并将左边界向右移动
29         while (sum > value) {
30             sum -= gems[left];
31             left++;
32         }
33         // 更新最大可购买的宝石数量
34         maxT = Math.max(maxT, right - left + 1);
35         // 将右边界向右移动
36         right++;
37     }
38     // 输出最大可购买的宝石数量
39     console.log(maxT);
40 }
});

```

## Python

```

1  # 创建一个对象来读取输入
2  n = int(input())
3  # 创建一个列表来存储每个宝石的价格
4
5  # 创建一个列表来存储每个宝石的价格
6  gems = []
7  # 读取每个宝石的价格
8  for i in range(n):
9      gems.append(int(input()))
10 # 读取你拥有的钱的总面值
11 value = int(input())
12
13

```

```

13 # 初始化滑动窗口的左右边界和窗口内宝石的总价
14 left = 0
15 right = 0
16 sum = 0
17 # 初始化最大可购买的宝石数量
18 maxT = 0
19 # 当滑动窗口的右边界没有超过宝石数量时，继续循环
20 while right < len(gems):
21     # 将右边界的宝石价格加到总价中
22     sum += gems[right]
23     # 当总价超过你拥有的钱时，将左边界的宝石价格从总价中减去，并将左边界向右移动
24     while sum > value:
25         sum -= gems[left]
26         left += 1
27     # 更新最大可购买的宝石数量
28     maxT = max(maxT, right - left + 1)
29     # 将右边界向右移动
30     right += 1
31 # 输出最大可购买的宝石数量
32 print(maxT)

```

## C语言

```

1
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main() {
6     int n;
7     // 读取宝石的总数量
8     scanf("%d", &n);
9
10    // 分配一个动态数组来存储每个宝石的价格
11    int *gems = (int *)malloc(n * sizeof(int));
12
13
14    // 读取每个宝石的价格
15    for (int i = 0; i < n; i++) {
16        scanf("%d", &gems[i]);
17    }
18
19

```

```

18
19 // 读取用户拥有的钱的总面值
20 int value;
21 scanf("%d", &value);
22
23 // 初始化滑动窗口的左右边界和窗口内宝石的总价
24 int left = 0, right = 0, sum = 0;
25 // 初始化最大可购买的宝石数量
26 int maxT = 0;
27
28 // 当滑动窗口的右边界没有超过宝石数量时, 继续循环
29 while (right < n) {
30     // 将右边界的宝石价格加到总价中
31     sum += gems[right];
32     // 当总价超过你拥有的钱时, 将左边界的宝石价格从总价中减去, 并将左边界向右移动
33     while (sum > value) {
34         sum -= gems[left];
35         left++;
36     }
37     // 更新最大可购买的宝石数量
38     if (right - left + 1 > maxT) {
39         maxT = right - left + 1;
40     }
41     // 将右边界向右移动
42     right++;
43 }
44
45 // 输出最大可购买的宝石数量
46 printf("%d\n", maxT);
47
48 // 释放动态分配的内存
49 free(gems);
50
51 return 0;
52 }

```

## 完整用例

### 用例1

0

0

用例2

1

500

1000

用例3

1

1000

500

用例4

3

300

200

500

1000

用例5

3

300

200

500

700

用例6

3

300

200

500

200

用例7

3  
300  
200  
500  
100

用例8

5  
100  
200  
300  
400  
500  
900

用例9

5  
100  
200  
300  
400  
500  
600

用例10

5  
200  
200  
200  
200  
200  
1000

## 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

用例1

用例2

用例3

用例4

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