

## 一、编程题

### ACM： 最多购买宝石数目

#### 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 $gems[i], 0 \leq i < n, n = gems.length$   
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 $gems[i], gems[i+1] \dots gems[i+m-1] (0 \leq i < n, m \leq n)$   
假设你当前拥有总面值为 $value$ 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 $gems[0]$ 到 $gems[n-1]$ 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： $[0, 10^6]$ 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。

#### 补充说明：

#### 示例1

输入：7

8

4

6

3

1

6

7

10

输出：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 $gems[5]$

示例2

输入：0

1

输出：0

说明：

```
gems = [], value = 1
```

因为没有宝石，所以返回0

示例3

输入：9

6

1

3

1

8

9

3

2

4

15

输出：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]

示例4

输入：9

1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
10

输出：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

代码:

```
import java.io.*;
```

```
import java.io.BufferedReader;
```

```
import java.io.InputStreamReader;
```

```
import java.io.IOException;
```

```
import java.util.*;
```

```
import java.util.Arrays;
```

```
public class Main {
```

```
    public static void main(String[] args) throws IOException {
```

```
        BufferedReader bufR = new BufferedReader(new InputStreamReader(System.in));
```

```
        String strIn;
```

```
        while ( (strIn = bufR.readLine()) != null ) {
```

```
            int n = Integer.parseInt(strIn);
```

```
            int[] array_prices = new int[n];
```

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

```
                int priceTemp = Integer.parseInt( bufR.readLine() );
```

```
                array_prices[i] = priceTemp;
```

```
            }
```

```
            int v = Integer.parseInt( bufR.readLine() );
```

```
            int[] array_max = new int[n];
```

```
            Arrays.fill(array_max, 0);
```

```

for (int i = 0; i < n; i++) {
    int vTemp = v;
    int lastIndex = -1;
    for (int j = i; j < n; j++) {
        if (vTemp >= array_prices[j]) {
            lastIndex = j;
            vTemp = vTemp - array_prices[j];
        } else {
            break;
        }
    }
    if (lastIndex > -1) {
        array_max[i] = lastIndex - i + 1;
    }
}

int max = 0;
for (int e : array_max) {
    if (e > max) {
        max = e;
    }
}

/*
if (n == 0) {
    System.out.println(0);
    return;
} else if (n == 1) {
    if (v >= array_prices[0]) {
        System.out.println(1);
        return;
    } else {
        System.out.println(0);
        return;
    }
}

int index_left = 0;
int index_right = n-1;
int priceAll = 0;
for (int e : array_prices) {
    priceAll = priceAll + e;
}

while (priceAll > v && index_left < index_right) {
    int price_left = array_prices[index_left];
    int price_right = array_prices[index_right];

```

```

        if (price_left > price_right) {
            priceAll = priceAll - array_prices[index_left];
            index_left++;
        } else {
            priceAll = priceAll - array_prices[index_right];
            index_right--;
        }
    }
    if (index_left == index_right) {
        if (v >= array_prices[index_left]) {
            System.out.println(1);
            return;
        } else {
            System.out.println(0);
            return;
        }
    }
    int max = index_right - index_left + 1;
    */

    System.out.println( max );
}

bufR.close();
}
}

```