

## Week-15-Pointers

**Question 1**

Correct

Marked out of  
1.00[Flag question](#)

Given an array of integers, reverse the given array in place using an index and loop rather than a built-in function.

**Example**

*arr = [1, 3, 2, 4, 5]*

Return the array *[5, 4, 2, 3, 1]* which is the reverse of the input array.

**Function Description**

Complete the function *reverseArray* in the editor below.

*reverseArray* has the following parameter(s):

*int arr[n]:* an array of integers

Return

*int[n]:* the array in reverse order

**Constraints**

$1 \leq n \leq 100$

$0 < arr[i] \leq 100$

**Input Format For Custom Testing**

The first line contains an integer, *n*, the number of elements in *arr*.

Each line *i* of the *n* subsequent lines (where  $0 \leq i < n$ ) contains an integer, *arr[i]*.

**Sample Case 0****Sample Input For Custom Testing**

5  
1  
3  
2  
4  
5

**Sample Output**

5  
4  
2  
3  
1

**Explanation**

The input array is *[1, 3, 2, 4, 5]*, so the reverse of the input array is *[5, 4, 2, 3, 1]*.

### Sample Case 1

#### Sample Input For Custom Testing

```
4  
17  
10  
21  
45
```

#### Sample Output

```
45  
21  
10  
17
```

#### Explanation

The input array is [17, 10, 21, 45], so the reverse of the input array is [45, 21, 10, 17].

## Source code

```
int* reverseArray(int arr_count, int *arr, int *result_count) {  
    *result_count=arr_count;  
    for (int i=0;i<arr_count/2;i++){  
        int temp=arr[i];  
        arr[i]=arr[arr_count-i-1];  
        arr[arr_count-i-1]=temp;  
    }  
    return arr;  
}
```

## Result

	Test	Expected	Got	
✓	int arr[] = {1, 3, 2, 4, 5}; int result_count; int* result = reverseArray(5, arr, &result_count); for (int i = 0; i < result_count; i++) printf("%d\n", *(result + i));	5 4 2 3 1	5 4 2 3 1	✓

Passed all tests! ✓

**Question 2**

Correct

Marked out of  
1.00[Flag question](#)

An automated cutting machine is used to cut rods into segments. The cutting machine can only hold a rod of *minLength* or more, and it can only make one cut at a time. Given the array *lengths*[] representing the desired lengths of each segment, determine if it is possible to make the necessary cuts using this machine. The rod is marked into lengths already, in the order given.

**Example***n* = 3*lengths* = [4, 3, 2]*minLength* = 7

The rod is initially  $\text{sum}(\text{lengths}) = 4 + 3 + 2 = 9$  units long. First cut off the segment of length  $4 + 3 = 7$  leaving a rod  $9 - 7 = 2$ . Then check that the length 2 rod can be cut into segments of lengths 4 and 3. Since 7 is greater than or equal to *minLength* = 7, the final cut can be made. Return "Possible".

**Example***n* = 3*lengths* = [4, 2, 3]*minLength* = 7

The rod is initially  $\text{sum}(\text{lengths}) = 4 + 2 + 3 = 9$  units long. In this case, the initial cut can be of length 4 or  $4 + 2 = 6$ . Regardless of the length of the first cut, the remaining piece will be shorter than *minLength*. Because  $n - 1 = 2$  cuts cannot be made, the answer is "Impossible".

**Function Description**

Complete the function *cutThemAll* in the editor below.

*cutThemAll* has the following parameter(s):

*int lengths[n]*: the lengths of the segments, in order

*int minLength*: the minimum length the machine can accept

**Returns**

string: "Possible" if all *n-1* cuts can be made. Otherwise, return the string "Impossible".

**Constraints**

- $2 \leq n \leq 10^5$
- $1 \leq t \leq 10^9$
- $1 \leq \text{lengths}[i] \leq 10^9$
- The sum of the elements of *lengths* equals the uncut rod length.

### **Input Format For Custom Testing**

The first line contains an integer,  $n$ , the number of elements in  $\text{lengths}$ .

Each line  $i$  of the  $n$  subsequent lines (where  $0 \leq i < n$ ) contains an integer,  $\text{lengths}[i]$ .

The next line contains an integer,  $\text{minLength}$ , the minimum length accepted by the machine.

### **Sample Case 0**

#### **Sample Input For Custom Testing**

STDIN Function

```
----  
4 → lengths[] size n = 4  
3 → lengths[] = [3, 5, 4, 3]  
5  
4  
3  
9 → minLength= 9
```

#### **Sample Output**

Possible

#### **Explanation**

The uncut rod is  $3 + 5 + 4 + 3 = 15$  units long. Cut the rod into lengths of  $3 + 5 + 4 = 12$  and 3. Then cut the 12 unit piece into lengths 3 and  $5 + 4 = 9$ . The remaining segment is  $5 + 4 = 9$  units and that is long enough to make the final cut.

### **Sample Case 1**

#### **Sample Input For Custom Testing**

STDIN Function

```
----  
3 → lengths[] size n = 3  
5 → lengths[] = [5, 6, 2]  
6  
2  
12 → minLength= 12
```

#### **Sample Output**

Impossible

#### **Explanation**

The uncut rod is  $5 + 6 + 2 = 13$  units long. After making either cut, the rod will be too short to make the second cut.

## Source code

```
char* cutThemAll(int lengths_count, long *lengths, long minLength) {
    long t=0,i=1;
    for(int i=0;i<=lengths_count-1;i++){
        t+=lengths[i];
    }
    do{
        if (t-lengths[lengths_count-i-1]< minLength){
            return "Impossible";
        }
        i++;
    }while(i<lengths_count-1);
    return "Possible";
}
```

## Result

	Test	Expected	Got	
✓	long lengths[] = {3, 5, 4, 3}; printf("%s", cutThemAll(4, lengths, 9))	Possible	Possible	✓
✓	long lengths[] = {5, 6, 2}; printf("%s", cutThemAll(3, lengths, 12))	Impossible	Impossible	✓

Passed all tests! ✓

