SRISHA S

B.E COMPUTER SCIENCE & DESIGN

241701056

WEEK 15

Problem Statement 1:

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 ≤ n ≤ 100

0 < arr[i] ≤ 100

Input Format For Custom Testing

The rst line contains an integer, n, the number of elements in arr. Each line i of the n subsequent lines (where 0 ≤ i < n) contains an integer, arr[i]

Sample Input For Custom Testing

5

1

3

2

4

5

<u>Sample Output</u>

5 4 2 3 1

<u>Explanation</u>

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

Reset answer

```
1    /*
2     * Complete the 'reverseArray' function below.
3     *
4     * The function is expected to return an INTEGER_ARRAY.
5     * The function accepts INTEGER_ARRAY arr as parameter.
6     */
7
8    /*
9     * To return the integer array from the function, you should:
10    *     - Store the size of the array to be returned in the result_count variable
11    *     - Allocate the array statically or dynamically
12    *
13    * For example,
14    * int* return_integer_array_using_static_allocation(int* result_count) {
15    *     *result_count = 5;
16    *
17    *     static int a[5] = {1, 2, 3, 4, 5};
18    *
19    *     return a;
20    * }
21    *
22    * int* return_integer_array_using_dynamic_allocation(int* result_count) {
23    *     *result_count = 5;
24    *
25    *     int *a = malloc(5 * sizeof(int));
26    *
27    *     for (int i = 0; i < 5; i++) {
28    *         *(a + i) = i + 1;
29    *     }
30    *
31    *     return a;
32    * }
33    *
34    */
35   int* reverseArray(int arr_count, int *arr, int *result_count) {
36       *result_count=arr_count;
37       for(int i=0;i<arr_count/2;i++){
38           int temp=arr[i];
39           arr[i]=arr[arr_count-i-1];
40           arr[arr_count-i-1]=temp;
41
42       }
43       return arr;

44
45   }
46
```

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

Passed all tests! ✓

PROBLEM STATEMENT 2:

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 sum(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 7 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 sum(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 cut ThemAll in the editor below.

cutThemAll has the following parameters):

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 ≤ n≤ 10^5

1 ≤ t ≤ 10^9

* 1 ≤ lengths/i] ≤ 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 lengths.

Each line i of the n subsequent lines (where 0 ≤ i < n) contains an integer, lengths[i].

The next line contains an integer, minLength, the minimum length accepted by the machine.

<u>Sample Case 0</u>

<u>Sample Input For Custom Testing</u>

STDIN   Function

-----   --------

- lengths] size n = 4

3 → lengths = [3, 5, 4, 3]

5

4

3

9 → minLength= 9

<u>Sample Output</u>

Possible

<u>Explanation</u>

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.

<u>Sample Case 1</u>

Sample Input For Custom Testing

STDIN Function

----- --------

3 → lengthsl 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.

Answer: (penalty regime: 0 %)

Reset answer

```c
/*
 * Complete the 'cutThemAll' function below.
 *
 * The function is expected to return a STRING.
 * The function accepts following parameters:
 *  1. LONG_INTEGER_ARRAY lengths
 *  2. LONG_INTEGER minLength
 */

/*
 * To return the string from the function, you should either do static allocation or dynamic allocation
 *
 * For example,
 * char* return_string_using_static_allocation() {
 *     static char s[] = "static allocation of string";
 *
 *     return s;
 * }
 *
 * char* return_string_using_dynamic_allocation() {
 *     char* s = malloc(100 * sizeof(char));
 *
 *     s = "dynamic allocation of string";
 *
 *     return s;
 * }
 *
 */
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";
}
```

	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! ✓