HARSHAVARDHINI.M

CSD

241701018

Question 1:   Reverse a List

Problem Statement:

Given an array of integers, reverse the given array in place using an index and loop rather than a builtin 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 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].

```c
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
```

	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	

2

Question 2:   Maximize the Value

Rearrange an array of integers so that the calculated value U is maximized. Among the

arrangements that satisfy that test, choose the array with minimal ordering. The value of U for an array with

n elements is calculated as:

$U = arr[1] \times arr[2] \times (1 \div arr[3]) \times arr[4] \times ... \times arr[n-1] \times (1 \div arr[n])$ if n is odd (or)

$U = arr[1] \times arr[2] \times (1 \div arr[3]) \times arr[4] \times ... \times (1 \div arr[n-1]) \times arr[n]$ if n is even

The sequence of operations is the same in either case, but the length of the array, n, determines whether the calculation ends on arr[n] or (1÷arr[n]). Arrange the elements to maximize U and the items are in the numerically smallest possible order.

Example: arr = [5, 7, 9, 21, 34]

To maximize U and minimize the order, arrange the array as [9, 21, 5, 34, 7] so U = 9 × 21 × (1÷5) × 34 × (1÷7) = 183.6. The same U can be achieved using several other orders, e.g. [21, 9, 7, 34, 5] = 21 × 9 × (1÷7) × 34 × (1÷5) = 183.6, but they are not in the minimal order.

Function Description: Complete the function rearrange in the editor below. rearrange has the following parameter(s): int arr[n]: an array of integers Returns: int[n]: the elements of arr rearranged as described

Constraints: $1 \le n \le 105$, $1 \le n \le 109$

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 $1 \le i \le n$) contains an integer, arr[i].

Sample Input For Custom Testing

3

STDINFunction

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

4 → arr[] size n = 4 1

→ arr = [1, 2, 3, 4]

2   3

4

Sample Output 2

3   1

Explanation

U = 2×3×(1÷1)×4 = 24. All other arrangements where U = 24 are numerically higher than this array, e.g. [2, 3, 1, 4] < [3, 4, 1, 2].

```
29 ▾ char* cutThemAll(int lengths_count, long *lengths, long minLength) {
30       long t=0, i =1;
31 ▾     for(int i=0; i<=lengths_count-1; i++){
32           t += lengths[i];
33       }
34 ▾     do{
35 ▾         if(t-lengths[lengths_count-i-1] < minLength){
36               return "Impossible";
37           }
38           i++;
39
40       }while(i<lengths_count-1);
41       return "Possible";
42 }
43
```

	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	✓