# I Incomplete Implementation

Time limit: 2s

Merge sort is a sorting algorithm. It works by splitting an array in half, sorting both halves recursively and then merging those halves together to sort the entire array. Your friend is working on an implementation of the merge sort algorithm, but unfortunately he is not quite there yet: he can only sort half of the array! In great despair he turns to you for help: can you use his unfinished code to write an algorithm that sorts an array completely?

In its current state, your friend's code is a sorting function that can be run on arbitrary subarrays, as long as it is precisely half as long as the original array. It then correctly sorts this subarray. Note that a subarray does not have to be contiguous, it can be any subset of the original array!

You decide to play around with this function. You start with a jumbled array and try to sort it (see Figure I.1). After choosing 3 subarrays and using them as input for the sorting function, you end up with a sorted array. Interestingly, it seems that no matter what the original array is, you can always sort it completely by invoking your friend's sorting function only 3 times. You decide that this makes for a good challenge: you want to extend the code to work for a full array, making at most three calls to the sorting function.

Now you need to figure out which subarrays to sort! Given an array of length $n$, output at most three subarrays of length $\frac{1}{2}n$ so that sorting these subarrays in order will result in a sorted array. It is guaranteed that this is always possible.

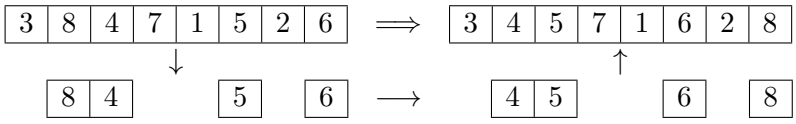


Figure I.1: First sorting step of Sample output 1

## Input

The input consists of:

- One line containing a single integer $n$ ($4 \leq n \leq 10^5$) divisible by 4, the length of the array.
- One line containing $n$ unique integers $a_i$ ($1 \leq a_i \leq n$), the array to be sorted.

## Output

The output consists of:

- One line containing the number of function calls $f$ ($0 \leq f \leq 3$).

- $f$ lines, each containing $\frac{1}{2}n$ unique integers $b_i$ ($1 \leq b_i \leq n$), the indices determining the subarray to be sorted at each of the function calls.

If there are multiple valid solutions, you may output any one of them. You do not have to minimize $f$.

**Sample Input 1**

```
8
3 8 4 7 1 5 2 6
```

**Sample Output 1**

```
3
2 3 6 8
1 3 4 5
2 4 5 7
```

**Sample Input 2**

```
4
1 4 3 2
```

**Sample Output 2**

```
3
3 4
2 3
3 4
```

**Sample Input 3**

```
8
1 4 8 7 5 6 3 2
```

**Sample Output 3**

```
2
6 5 3 8
4 3 7 2
```