1.Given a string, reverse it using stack. For example “CDAC KHARGHAR” should be converted to “RAHGRAHK CADC”.

2.Given an array, print the Next Greater Element (NGE) for every element. The Next greater Element for an element x is the first greater element on the right side of x in array. Elements for which no greater element exist, consider next greater element as -1.

Examples:

a) For any array, rightmost element always has next greater element as -1.

b) For an array which is sorted in decreasing order, all elements have next greater element as -1.

c) For the input array [4, 5, 2, 25}, the next greater elements for each element are as follows.

```
Element       NGE
   4      -->   5
   5      -->   25
   2      -->   25
   25     -->   -1
```

3.Given a singly linked list of size **N** of integers. The task is to check if the given linked list is palindrome or not.

Input:

First line of input contains number of test cases T. For each test case, first line of input contains length of linked list N and next line contains N integers as data of linked list.

Output:

For each test case output will be 1 if the linked list is a palindrome else 0.

User Task:

The task is to complete the function isPalindrome() which takes head as reference as the only parameter and returns true or false if linked list is palindrome or not respectively.

Example(To be used only for expected output):

Input:

2

3

1 2 1

4

1 2 3 4

Output:

1

0

Explanation:

Testcase 1: 1 2 1, linked list is palindrome.

4.Given a linked list of size **N** consisting of **0s**, **1s** and **2s**. The task os to sort this linked list such that all zeroes segregate to headside, 2s at the end and 1s in the mid of 0s and 2s.

Input:

First line of input contains number of test cases T. For each test case, first line of input contains length of linked list and next line contains N elements as the data in the linked list.

Output:

For each test case, segregate the 0s, 1s and 2s and display the linked list.

Constraints:

1 <= T <= 100

1 <= N <= 10^3

User Task:

The task is to complete the function **sortList**() which takes head reference as the argument and returns void.

Example:

Input:

1

8

1 2 2 1 2 0 2 2

Output:

0 1 1 2 2 2 2 2

Explanation:

Testcase 1: All the 0s are segregated to left end of the linked list, 2s to the right end of the list and 1s in between.

5. Given two linked lists of size **N** which is sorted in ascending order. Merge them in-place and return head of the merged list.
Note: It is strongly recommended to do merging in-place using **O(1)** extra space.

Input:

First line of input contains number of test cases T. For each test case, first line of input contains N and M, and next two line contains sorted lists of size N and M.

Output:

For each test case, print the merged list in sorted form.

User Task:

The task is to complete the function **SortedMerge**() which takes references to the heads of two linked lists as the arguments and returns the head of merged linked list.

Constraints:

1 <= T <= 200

1 <= N, M <= 10^3

1 <= Node's data <= 10^3

Example:

Input:

2

4 3

5 10 15 40

2 3 20

2 2

1 1

2 4

Output:

2 3 5 10 15 20 40

1 1 2 4

Explanation:

Testcase 1: After merging the two linked lists, we have merged list as 2, 3, 5, 10, 15, 20, 40.