Find median in a stream

Given an input stream of N integers. The task is to insert these numbers into a new stream and find the median of the stream formed by each insertion of X to the new stream.

Input:
The first line of input contains an integer N denoting the number of elements in the stream. Then the next N lines contains integer x denoting the number to be inserted into the stream.
Output:
For each element added to the stream print the floor of the new median in a new line.

Constraints:
1 <= N <= 10^6
1 <= x <= 10^6

Example:
Input:
4
5
15
1
3
Output:
5
10
5
4

Explanation:
Testcase 1:
Flow in stream : 5, 15, 1, 3
5 goes to stream --> median 5 (5)
15 goes to stream --> median 10 (5, 15)
1 goes to stream --> median 5 (5, 15, 1)
3 goes to stream --> median 4 (5, 15, 1, 3)

Heap Sort

Given an array of size N. The task is to sort the array elements by completing functions heapify() and buildHeap() which are used to implement Heap Sort.

Example 1:

```
Input:
N = 5
arr[] = {4,1,3,9,7}
Output:1 3 4 7 9
Explanation:After sorting elements
using heap sort, elements will be
in order as 1,3,4,7,9.
```

Example 2:

```
Input:
N = 10
arr[] = {10,9,8,7,6,5,4,3,2,1}
Output:1 2 3 4 5 6 7 8 9 10
Explanation:After sorting elements
using heap sort, elements will be
in order as 1, 2,3,4,5,6,7,8,9,10.
```

Your Task :

Complete the functions *heapify() and* buildheap().

Expected Time Complexity: O(N * Log(N)).
Expected Auxiliary Space: O(1).

Constraints:

$1 <= N <= 10^6$

$1 <= arr[i] <= 10^6$

Binary Heap Operations

A binary heap is a Binary Tree with the following properties:
1) It's a *complete tree* (All levels are completely filled except possibly the last level and

the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

2) A Binary Heap is either **Min Heap** or **Max Heap**. In a *Min Binary Heap*, the key at the *root* must be *minimum* among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.

You are given an empty Binary Min Heap and some queries and your task is to implement the three methods **insertKey, deleteKey,** and **extractMin** on the Binary Min Heap and call them as per the query given below:
1) *1 x* (a query of this type means to insert an element in the min-heap with value x)
2) *2 x* (a query of this type means to remove an element at position x from the min-heap)
3) *3* (a query like this removes the min element from the min-heap and prints it).

Example 1:

```
Input:
Q = 7
Queries:
insertKey(4)
insertKey(2)
extractMin()
insertKey(6)
deleteKey(0)
extractMin()
extractMin()
Output: 2 6 - 1
Explanation: In the first test case for
query
insertKey(4) the heap will have  {4}
insertKey(2) the heap will be {2 4}
extractMin() removes min element from
          heap ie 2 and prints it
          now heap is {4}
insertKey(6) inserts 6 to heap now heap
          is {4 6}
deleteKey(0) delete element at position 0
          of the heap,now heap is {6}
extractMin() remove min element from heap
          ie 6 and prints it  now the
          heap is empty
extractMin() since the heap is empty thus
          no min element exist so -1
          is printed.
```

Example 2:

```
Input:
Q = 5
Queries:
insertKey(8)
insertKey(9)
deleteKey(1)
extractMin()
extractMin()
Output: 8 -1
```

Your Task:
You are required to complete the 3 methods **insertKey()** which take one argument the value to be inserted, **deleteKey()** which takes one argument the position from where the element is to be deleted and **extractMin()** which returns the minimum element in the heap(-1 if the heap is empty)

Expected Time Complexity: O(Q*Log(size of Heap)).
Expected Auxiliary Space: O(1).

Constraints:
1 <= **Q** <= 10^4
1 <= **x** <= 10^4

Rearrange characters

Given a string S with repeated characters (only lowercase). The task is to rearrange characters in a string such that no two adjacent characters are same.

Note : It may be assumed that the string has only lowercase English alphabets.

Input:
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains a single line containing a string of lowercase english alphabets.

Output:
For each test case in a new line print "1" (without quotes) if the generated string doesn't

contains any same adjacent characters, else if no such string is possible to be made print "0" (without quotes).

Constraints:
1 <= T <= 100
1 <= length of string <= 10^4

Example:
Input:
3
geeksforgeeks
bbbabaaacd
bbbbb

Output:
1
1
0

Explanation:
Testcase 1: All the repeated characters of the given string can be rearranged so that no adjacent characters in the string is equal.
Testcase 3: Repeated characters in the string cannot be rearranged such that there should not be any adjacent repeated character.

Kth largest element in a stream

Given an input stream of n integers, find the kth largest element for each element in the stream.

Input:
The first line of input contains an integer **T** denoting the number of test cases. Then T test cases follow. Each test case contains two lines. The first line of each test case contains two space separated integers **k** and **n** . Then in the next line are **n** space separated values of the array.

Output:
For each test case, in a new line, print the space separated values denoting the **kth** largest

element at each insertion, if the **kth** largest element at a particular insertion in the stream doesn't exist print **-1**.

Constraints:
1 <= T <= 100
1 <= K <= n
1 <= n <= 10^6
1 <= stream[] <= 10^5

Example:
Input:
2
4 6
1 2 3 4 5 6
1 2
3 4

Output:
-1 -1 -1 1 2 3
3 4

Explanation:
Testcase1:
k = 4
For 1, the 4th largest element doesn't exist so we print -1.
For 2, the 4th largest element doesn't exist so we print -1.
For 3, the 4th largest element doesn't exist so we print -1.
For 4, the 4th largest element is 1 {1, 2, 3, 4}
For 5, the 4th largest element is 2 {2, 3, 4 ,5}
for 6, the 4th largest element is 3 {3, 4, 5}