Print Lexicographic Next Permutation

|  |  |
| --- | --- |
| **PROBLEM STATEMENT** | Given an array of distinct integers representing a permutation, Generate and print next lexicographically higher permutation of the array.  If it is the last permutation of numbers in array, print - IMPOSSIBLE. |
|  |  |
| **INPUT FORMAT** | First line contains T - Number of test cases. Each test case has following two lines: Line 1:  N - Number of elements in array. Line 2:  N space separated array elements. |
| **OUTPUT FORMAT** | For each test case, print a separate line of output containing next lexicographically higher permutation. |
| **CONSTRAINTS** | 1 <= T <= 500 1 <= N <= 10000 1 <= A[i] <= 100000 |
| **NOTE** | InputandOutput can be huge.  **Use Fast Input as well as Fast Output Functions.** |

|  |  |  |
| --- | --- | --- |
| **SAMPLE INPUT** | **SAMPLE OUTPUT** | **EXPLANATION** |
| 2 3 1 3 2 5 9 6 4 2 1 | 2 1 3 IMPOSSIBLE | Permutations of 1 2 3 (lexicographical order wise) 1 2 3 **1 3 2** 2 1 3 2 3 1 3 1 2 3 2 1 That should explain first test case. |

### Example Input - Output

printNextPerm([1, 2, 3]) => 1 3 2

printNextPerm([2, 3, 1]) => 3 1 2

printNextPerm([2, 1, 8, 7, 6, 5]) => 2 5 1 6 7 8

### Code

#include <iostream>

#include <algorithm>

void display(int\* arr, int len)

{

for (auto i = 0; i < len; i++)

{

*std*::*cout* << arr[i] << " ";

}

*std*::*cout* << *std*::*endl*;

}

void printNextPerm(int\* arr, int len)

{

int index = len - 1;

while (index > 0 && arr[index - 1] >= arr[index])

index--;

if (index == 0)

{

*std*::*cout* << "IMPOSSIBLE" << *std*::*endl*;

return;

}

int j = len - 1;

while (arr[j] <= arr[index - 1])

j--;

*std*::*swap*(arr[index - 1], arr[j]);

*std*::*reverse*(&arr[index], arr + len);

display(arr, len);

}

int main()

{

int t; *std*::*cin* >> t;

for (auto i = 0; i < t; i++)

{

int numberOfElements = 0;

*std*::*cin* >> numberOfElements;

int\* myArray = new int[numberOfElements];

for (auto j = 0; j < numberOfElements; j++)

{

*std*::*cin* >> myArray[j];

}

printNextPerm(myArray, numberOfElements);

delete myArray;

}

return 0;

}