1299. Replace Elements with Greatest Element on Right Side

Easy | Topics | Companies | Hint

Given an array `arr`, replace every element in that array with the greatest element among the elements to its right, and replace the last element with `-1`.

After doing so, return the array.

Example 1:

```
Input: arr = [17,18,5,4,6,1]
Output: [18,6,6,6,1,-1]
Explanation:
- index 0 --> the greatest element to the right of index 0 is index 1 (18).
- index 1 --> the greatest element to the right of index 1 is index 4 (6).
- index 2 --> the greatest element to the right of index 2 is index 4 (6).
- index 3 --> the greatest element to the right of index 3 is index 4 (6).
- index 4 --> the greatest element to the right of index 4 is index 5 (1).
- index 5 --> there are no elements to the right of index 5, so we put -1.
```

Example 2:

```
Input: arr = [400]
Output: [-1]
Explanation: There are no elements to the right of index 0.
```

Constraints:

- `1 <= arr.length <= 10^4`
- `1 <= arr[i] <= 10^5`

```cpp
//https://leetcode
    .com/problems/replace-elements-with-greatest-element-on-right-side/description/
// Leetcode: Replace Elements with Greatest Element on Right Side


/*
    Iterate from the right to the left,
    We initilize max_right = -1, where max_right represent the max on the right.
    Each round, we set arr[i] = max_right, where max_right is maximum element on
        the right.
    Also we update max_right = max(max_right, arr[i]).
*/
```

```cpp
class Solution {
public:
    vector<int> replaceElements(vector<int>& arr) {
        int n = arr.size();
        vector<int> ans(n);
        int max_right = -1;
        for (int i = n - 1; i >= 0; i--) {
            ans[i] = max_right;
            if (max_right < arr[i]) {
                max_right = arr[i];
            }
        }
        return ans;
    }
};
```

1207. Unique Number of Occurrences

Easy | Topics | Companies | Hint

Given an array of integers `arr`, return `true` *if the number of occurrences of each value in the array is* ***unique*** *or* `false` *otherwise.*

Example 1:

```
Input: arr = [1,2,2,1,1,3]
Output: true
Explanation: The value 1 has 3 occurrences, 2 has 2 and 3 has 1. No two values have the same number of occurrences.
```

Example 2:

```
Input: arr = [1,2]
Output: false
```

Example 3:

```
Input: arr = [-3,0,1,-3,1,1,1,-3,10,0]
Output: true
```

Constraints:

- `1 <= arr.length <= 1000`
- `-1000 <= arr[i] <= 1000`

```
// https://leetcode.com/problems/unique-number-of-occurrences/description/
// Leetcode: Unique Number of Occurrences


/*
The problem asks whether the number of occurrences of each unique value in the array is unique. So for each distinct element in the array, we need to count how many times it appears, and then check if the counts themselves are all unique.
Iterate through the array and count the occurrences of each unique value. You can use a data structure for hasing  to keep track of these counts.
After counting the occurrences, we need to check if the counts themselves are unique.
We can do this with another data structure, like a set, to keep track of the unique counts.
*/
```

```cpp
class Solution {
public:
    bool uniqueOccurrences(vector<int>& arr) {
        unordered_map<int, int>mp;
        for (int i = 0;i < arr.size();i++) {
            mp[arr[i]]++;
        }
        unordered_set<int>st;
        for (auto i : mp) {
            st.insert(i.second);
        }
        if (mp.size() == st.size())return true;
        return false;
    }
};
```

2824. Count Pairs Whose Sum is Less than Target

Easy | Topics | Companies | Hint

Given a **0-indexed** integer array `nums` of length `n` and an integer `target`, return *the number of pairs* `(i, j)` *where* `0 <= i < j < n` *and* `nums[i] + nums[j] < target`.

Example 1:

```
Input: nums = [-1,1,2,3,1], target = 2
Output: 3
Explanation: There are 3 pairs of indices that satisfy the conditions in the statement:
- (0, 1) since 0 < 1 and nums[0] + nums[1] = 0 < target
- (0, 2) since 0 < 2 and nums[0] + nums[2] = 1 < target
- (0, 4) since 0 < 4 and nums[0] + nums[4] = 0 < target
Note that (0, 3) is not counted since nums[0] + nums[3] is not strictly less than the target.
```

Example 2:

```
Input: nums = [-6,2,5,-2,-7,-1,3], target = -2
Output: 10
Explanation: There are 10 pairs of indices that satisfy the conditions in the statement:
- (0, 1) since 0 < 1 and nums[0] + nums[1] = -4 < target
- (0, 3) since 0 < 3 and nums[0] + nums[3] = -8 < target
- (0, 4) since 0 < 4 and nums[0] + nums[4] = -13 < target
- (0, 5) since 0 < 5 and nums[0] + nums[5] = -7 < target
- (0, 6) since 0 < 6 and nums[0] + nums[6] = -3 < target
- (1, 4) since 1 < 4 and nums[1] + nums[4] = -5 < target
- (3, 4) since 3 < 4 and nums[3] + nums[4] = -9 < target
- (3, 5) since 3 < 5 and nums[3] + nums[5] = -3 < target
- (4, 5) since 4 < 5 and nums[4] + nums[5] = -8 < target
- (4, 6) since 4 < 6 and nums[4] + nums[6] = -4 < target
```

Constraints:

- `1 <= nums.length == n <= 50`
- `-50 <= nums[i], target <= 50`

```cpp
// https://leetcode.com/problems/count-pairs-whose-sum-is-less-than-target/description/
// LeetCode:Count Pairs Whose Sum is Less than Target


/*
 Simply run two loop check each pair that their sum is less than target or not.
*/
class Solution {
public:
    int countPairs(vector<int>& nums, int target) {
        int res = 0;
        for (int i = 0;i < nums.size();i++) {
            for (int j = i + 1;j < nums.size();j++) {
                if (nums[i] + nums[j] < target) {
                    res++;
                }
            }
        }
        return res;
    }
};
```

A. Good Pairs

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an array $a_1, a_2, \ldots, a_n$ of positive integers. A *good pair* is a pair of indices (i, j) with $1 \le i, j \le n$ such that, for all $1 \le k \le n$, the following equality holds:

$$|a_i - a_k| + |a_k - a_j| = |a_i - a_j|,$$

where $|x|$ denotes the absolute value of x.

Find a good pair. Note that i can be equal to j.

Input

The input consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. Description of the test cases follows.

The first line of each test case contains an integer n ($1 \le n \le 10^5$) — the length of the array.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) where a_i is the i-th element of the array.

The sum of n for all test cases is at most $2 \cdot 10^5$.

Output

For each test case, print a single line with two space-separated indices i and j which form a good pair of the array. The case $i = j$ is allowed. It can be shown that such a pair always exists. If there are multiple good pairs, print any of them.

Example

input

```
3
3
5 2 7
5
1 4 2 2 3
1
2
```

output

```
2 3
1 2
1 1
```

Note

In the first case, for $i = 2$ and $j = 3$ the equality holds true for all k:

- $k = 1$: $|a_2 - a_1| + |a_1 - a_3| = |2 - 5| + |5 - 7| = 5 = |2 - 7| = |a_2 - a_3|$,
- $k = 2$: $|a_2 - a_2| + |a_2 - a_3| = |2 - 2| + |2 - 7| = 5 = |2 - 7| = |a_2 - a_3|$,
- $k = 3$: $|a_2 - a_3| + |a_3 - a_3| = |2 - 7| + |7 - 7| = 5 = |2 - 7| = |a_2 - a_3|$.

```cpp
// https://codeforces.com/problemset/problem/1656/A
// Codeforces: Good Pairs

/*
Look at expression |ai-ak| + |ak-aj| = |ai-aj|
Let suppose ai is greatest element of array and aj is smaller element of
    array.
Now all values of array are in between aj to ai .
Now we can open mod directly as ai is largest element so
 |ai-ak|=(ai-ak) and aj is smallest element so, |ak-aj|=(ak-aj).
Now expression reduce to
 |ai-ak| +| ak-aj| = ai - ak + ak - aj = ai-aj = |ai-aj|.
So this problem reduce to finding index of max and min element.
*/

#include<bits/stdc++.h>
using namespace std;

 int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        int min_element = INT_MAX;
        int max_element = INT_MIN;
        int min_index;
        int max_index;
        for (int i = 0; i < n; ++i) {
            int a;
            cin >> a;
            if (a > max_element) {
                max_index = i + 1;
                max_element = a;
            }
            if (a < min_element) {
                min_index = i + 1;
                min_element = a;
            }
        }
        cout << min_index << " " << max_index << endl;
    }

 }
```

A. Array

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Vitaly has an array of n distinct integers. Vitaly wants to divide this array into three **non-empty** sets so as the following conditions hold:

1. The product of all numbers in the first set is less than zero (< 0).
2. The product of all numbers in the second set is greater than zero (> 0).
3. The product of all numbers in the third set is equal to zero.
4. Each number from the initial array must occur in exactly one set.

Help Vitaly. Divide the given array.

Input

The first line of the input contains integer n $(3 \le n \le 100)$. The second line contains n space-separated distinct integers $a_1, a_2, ..., a_n$ $(|a_i| \le 10^3)$ — the array elements.

Output

In the first line print integer n_1 $(n_1 > 0)$ — the number of elements in the first set. Then print n_1 numbers — the elements that got to the first set.

In the next line print integer n_2 $(n_2 > 0)$ — the number of elements in the second set. Then print n_2 numbers — the elements that got to the second set.

In the next line print integer n_3 $(n_3 > 0)$ — the number of elements in the third set. Then print n_3 numbers — the elements that got to the third set.

The printed sets must meet the described conditions. It is guaranteed that the solution exists. If there are several solutions, you are allowed to print any of them.

Examples

input
3 -1 2 0
output
1 -1 1 2 1 0

input
4 -1 -2 -3 0
output
1 -1 2 -3 -2 1 0

```
// https://codeforces.com/problemset/problem/300/A
// CodeForces: Array

/*
In this problem it is given that solution always exits so,
For negative product we can just print a single negative element so its product is always negative.
For positive element we can  print all positive element so its product is always positive and if
 there is no positive element then we can print two negative element ,product of two negative element is
  positive.
For zero procduct print rest of element.
*/
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0;i < n;i++) {
        cin >> arr[i];
    }
    vector<int>neg;
    vector<int>pos;
    vector<int>zero;
    for (int i = 0;i < n;i++) {
        if (arr[i] > 0) {
            pos.push_back(arr[i]);
        }
        else if (arr[i] < 0) {
            neg.push_back(arr[i]);
        }
        else {
            zero.push_back(arr[i]);
        }
    }
    if (pos.size() == 0) {
        pos.push_back(neg.back());
        neg.pop_back();
        pos.push_back(neg.back());
        neg.pop_back();
    }
    cout << 1 << " " << neg[0] << endl;

    cout << pos.size() << " ";
    for (int i = 0;i < pos.size();i++) {
        cout << pos[i] << " ";
    }
    cout << endl;
    cout << (neg.size() - 1) + zero.size() << " ";
    for (int i = 1;i < neg.size();i++) {
        cout << neg[i] << " ";
    }

    for (int i = 0;i < zero.size();i++) {
        cout << zero[i] << " ";
    }
    return 0;
}
```