Assalammualaikum

Mohammad Hanif Furqan Aufa Putra

5025221161

TUGAS 5 PBO C

Interesting fun of Yura

 Time limit 2 seconds

 Memory limit 128 MiB

At the lesson of computer science, Yura became very sad, so he invented a fun for himself.

In the beginning, he has an empty set. At each next turn, he comes up with a number and checks if it belongs to the set. If it belongs, then Yura screams "Yes". If not, he screams "No", and adds it to the set. Before making a new number, Yura screams the number of elements in the set.

The teacher was tired of Yura's screams, so he made him write a program that screams instead of a boy. But Yura does not know how to program, so he asked for help from you.

Input data

First line contains integer n (1 ≤ n ≤ 10^5). Each of the next n lines contains an integer, picked by Yura. Yura can think of only numbers in the interval from -10^9 to 10^9.

Output data

Print in a separate line what Yura shouts for each request.

Examples

Input example #1

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

Output example #1

```
No 1
No 2
No 3
No 4
Yes 4
```

Soal ini melibatkan Yura yang bermain dengan menyebutkan angka-angka dan memeriksa apakah angka tersebut sudah ada dalam himpunan. Jika angka itu sudah ada, dia katakan "Ya" dan sebelum menciptakan angka baru, Yura juga menyebutkan jumlah elemen dalam himpunan. Jika tidak ada, dia katakan "Tidak" dan tambahkan ke dalam himpunan.

- Yura mulai dengan himpunan kosong.

- Dia menyebutkan angka, jika angka itu sudah ada dalam himpunan, dia katakan "Ya" Yura menyebutkan jumlah elemen dalam himpunan.

- Jika tidak, dia katakan "Tidak" dan tambahkan angka ke dalam himpunan.

Input melibatkan bilangan bulat 'n' sebagai jumlah tindakan Yura, diikuti dengan 'n' bilangan bulat yang mewakili tindakan Yura.

Tujuan dari masalah ini adalah untuk menentukan apa yang Yura katakan untuk setiap tindakan dan jumlah elemen dalam himpunan saat itu. Output harus mencakup satu baris untuk setiap tindakan dengan respons yang sesuai.Untuk menyelesaikan masalah ini, kita dapat menggunakan struktur data `std::set` dalam bahasa pemrograman C++. `std::set` adalah wadah yang memungkinkan kita untuk menyimpan elemen-elemen unik dalam urutan tertentu. Ini sangat cocok untuk permainan yang Yura mainkan karena memastikan bahwa tidak ada duplikat dalam himpunan, dan juga mengurutkan elemen-elemen tersebut secara otomatis.

Pendekatan dalam kode adalah sebagai berikut:

- Menggunakan `std::set` untuk menyimpan angka-angka yang telah dipilih oleh Yura.

- Untuk setiap angka baru yang dipilih, kita memeriksa apakah angka tersebut sudah ada dalam `std::set`. Jika sudah ada, kita mencetak "Yes" bersama dengan jumlah elemen dalam `std::set`.

- Jika angka tersebut belum ada dalam set, kita mencetak "No", kemudian menambahkannya ke dalam set, dan mencetak jumlah elemen dalam `std::set`.

Contoh pengerjaan untuk input sample:

- Yura memilih angka pertama, yaitu 1. Karena himpunan awal kosong, kita mencetak "No 1". Set sekarang berisi 1.

- Yura memilih angka kedua, yaitu 2. Karena angka ini belum ada dalam set, kita mencetak "No 2". Set sekarang berisi 1 dan 2.

- Yura memilih angka ketiga, yaitu 3. Karena angka ini belum ada dalam set, kita mencetak "No 3". Set sekarang berisi 1, 2, dan 3.

- Yura memilih angka keempat, yaitu 4. Karena angka ini belum ada dalam set, kita mencetak "No 4". Set sekarang berisi 1, 2, 3, dan 4.

- Yura memilih angka kelima, yaitu 1. Karena angka ini sudah ada dalam set, kita mencetak "Yes" bersama dengan jumlah elemen dalam set, yaitu 4.

Dengan menggunakan `std::set`, kita dapat dengan mudah menyimpan elemen-elemen unik dan memeriksa keberadaan elemen dalam waktu efisien. Ini memungkinkan kita untuk menyelesaikan permainan yang ditemukan oleh Yura dengan cepat dan efisien, menghindari duplikasi, dan mengurutkan elemen-elemen sesuai kebutuhan.

Dan inilah kode yang diberi

```cpp
#include <set>
#include <cstdio>
#include <algorithm>
using namespace std;
#define gc getchar_unlocked
set <int> set_int;

void Get(int &ret){
  ret = 0; char inp=gc(); int kl=1;
  while(inp<'0' || inp>'9'){
    if (inp=='-') kl=-1; inp=gc();}
  while('0'<=inp && inp<='9')
    ret=(ret<<3)+(ret<<1)+(int)(inp-'0'), inp=gc();
  if(kl<1) ret=-ret;
}
int main()
{
    int n;
    Get(n);
    set<int> set_int;
    for (int i = 0; i < n; i++)
    {
        int k;
        Get(k);
        if (set_int.find(k) != set_int.end())
        {
        printf("Yes ");
        printf("%d\n", set_int.size());
        }
        else
        {
        printf("No ");
        set_int.insert(k);
        printf("%d\n", set_int.size());
        }
    }
}
```

Dengan menjalan program tersebut diperoleh hasil seperti dibawah ini dengan menggunakan test case yang saya coba buat

Success #stdin #stdout 0.01s 5424KB

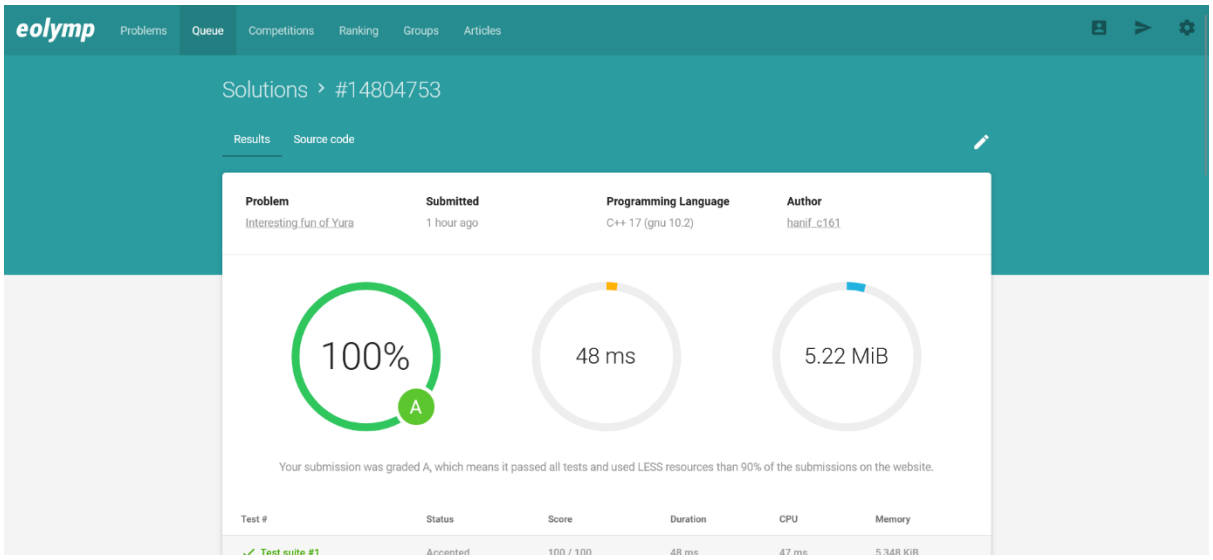
stdin

```
6
2
4
2
1
3
1
```

stdout

```
No 1
No 2
Yes 2
No 3
No 4
Yes 4
```

Dan dibawah bukti Verdict Accept



Replacement

Time limit **1** second

Memory limit **128** MiB

Given a sequence of n positive integers. You must replace each element with the next nearest one (with a larger index) that is strictly larger than its value. If there is no larger element, replace this element with zero.

Input data

First line contains the number of elements n ($1 \le n \le 10^5$). Second line contains n positive integers `a[i]` (`a[i]` $\le 10^9$) - the values of sequence elements.

Output data

Print the desired sequence, separating the neighboring elements with a single space.

Examples

Input example #1

```
6
1 2 3 1 1 5
```

Output example #1

```
2 3 5 5 5 0
```

Input example #2

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

Output example #2

```
2 3 4 5 0
```

Soal ini mengharuskan kita untuk menggantikan setiap elemen dalam sebuah rangkaian bilangan positif dengan elemen terdekat yang memiliki indeks lebih besar daripada elemen tersebut. Jika tidak ada elemen yang lebih besar, maka elemen tersebut harus digantikan dengan angka nol.

Pendekatan yang digunakan dalam kode ini melibatkan penggunaan dua struktur data, yaitu Class Stack dan Class Vector. Stack digunakan untuk melacak indeks-indeks elemen dalam rangkaian bilangan sementara Vector digunakan untuk menyimpan hasil akhir dari penggantian elemen-elemen. Stack memungkinkan kita untuk menyimpan indeks-indeks elemen yang belum memiliki pengganti yang lebih besar sementara Vector digunakan untuk menyimpan hasil akhir.

Dengan Class Utama Set sebagai pendekatan utama

Kode ini bekerja dengan mengiterasi melalui rangkaian bilangan dan menggunakan Stack untuk menyimpan indeks-indeks elemen yang belum memiliki pengganti yang lebih besar. Ketika elemen yang lebih besar ditemukan, penggantian dilakukan, dan hasilnya disimpan dalam Vector. Ini terus berlanjut hingga seluruh rangkaian bilangan diproses.

Contoh pengerjaan dengan sampel input adalah sebagai berikut:

Input:

6

1 2 3 1 1 5

Langkah 1:

- Menggunakan Stack dan Vector kosong.

- Iterasi pertama: elemen pertama adalah 1, tidak ada elemen lebih besar, maka diganti dengan 0. Hasil sementara: Vector = [0], Stack = [0].

Langkah 2:

- Iterasi kedua: elemen kedua adalah 2, lebih besar dari elemen pertama yang diwakili oleh indeks 0. Penggantian dilakukan, dan hasil sementara: Vector = [0, 2], Stack = kosong.

Langkah 3:

- Iterasi ketiga: elemen ketiga adalah 3, lebih besar dari elemen kedua yang diwakili oleh indeks 2. Penggantian dilakukan, dan hasil sementara: Vector = [0, 2, 3, 1], Stack = kosong.

Langkah 4:

- Iterasi keempat: elemen keempat adalah 1, tidak ada elemen lebih besar, maka diganti dengan 0. Hasil sementara: Vector = [0, 2, 3, 1, 0], Stack = [3].

Langkah 5:

- Iterasi kelima: elemen kelima adalah 1, lebih besar dari elemen keempat yang diwakili oleh indeks 3. Penggantian dilakukan, dan hasil sementara: Vector = [0, 2, 3, 1, 0, 5], Stack = kosong.

Langkah 6:

- Iterasi keenam: elemen keenam adalah 5, tidak ada elemen lebih besar, maka diganti dengan 0. Hasil akhir: Vector = [0, 2, 3, 1, 0, 5], Stack = kosong.

Kode ini menggunakan Class Stack untuk melacak indeks-indeks elemen yang belum memiliki pengganti yang lebih besar dan Class Vector untuk menyimpan hasil akhir dari penggantian elemen. Pendekatan ini memungkinkan penyelesaian masalah sesuai dengan deskripsi dalam soal, yaitu menggantikan setiap elemen dengan elemen terdekat yang lebih besar atau nol jika tidak ada elemen yang lebih besar.

Dengan Class Set

Dalam kode ini, pendekatan yang digunakan untuk menyelesaikan masalah ini melibatkan penggunaan dua kelas, yaitu `std::set` dan `std::vector`. `std::set` digunakan untuk menyimpan angka-angka dalam urutan yang akan digantikan. Kita memasukkan angka-angka ini ke dalam `std::set`, yang akan secara otomatis mengurutkannya dan menjaga agar tidak ada angka yang sama. Ini penting karena kita hanya ingin menggantikan angka dengan angka yang lebih besar, dan kita tidak ingin menggantikannya dengan angka yang sama.

Selanjutnya, `std::vector` digunakan untuk menyimpan urutan hasil akhir. Kami menginisialisasi vektor ini dengan nilai nol, yang akan kita gantikan nanti dengan angka yang lebih besar dari `std::set`.

Untuk memberikan contoh pengerjaan dari input sampel, pertama, kita memasukkan angka-angka ke dalam `std::set` dengan urutan terbalik dari indeks terakhir ke awal. Kemudian, kita mengambil elemen-elemen ini satu per satu dan mencari elemen pertama yang lebih besar daripadanya dalam `std::set`. Jika kita menemukan elemen tersebut, kita masukkan ke dalam `std::vector`, jika tidak, kita masukkan nol. Hasil akhirnya adalah urutan angka yang memenuhi syarat yang diberikan dalam soal.

Dengan demikian, penggunaan `std::set` dan `std::vector` memungkinkan kita untuk mengatasi masalah ini dengan efisien dan menghasilkan urutan angka yang sesuai dengan kriteria yang diberikan dalam soal.

Dan inilah kode Replacement Stack

```cpp
#include <stdio.h>
#include <stack>
#include <vector>
using namespace std;
#define gc getchar_unlocked

void Get(int &ret) {
    ret = 0; char inp = gc(); int kl = 1;
    while (inp < '0' || inp > '9') {
        if (inp == '-') kl = -1; inp = gc();}
    while ('0' <= inp && inp <= '9')
        ret = (ret << 3) + (ret << 1) + (int)(inp - '0'), inp = gc();
        if (kl < 1) ret = -ret;
}

int main()
{
    int n;
    Get(n);
    vector<int> a(n);
    for (int i = 0; i < n; i++)
    {
        Get(a[i]);
    }

    vector<int> result(n, 0);
    stack<int> st;

    for (int i = 0; i < n; i++)
    {
        while (!st.empty() && a[i] > a[st.top()])
        {
            result[st.top()] = a[i];
            st.pop();
        }
        st.push(i);
    }

    for (int i = 0; i < n; i++)
    {
        printf("%d", result[i]);
        if (i < n - 1)
        {
            printf(" ");
        }
    }

    printf("\n");

    return 0;
}

```

Dan inilah kode Replacement Set

```cpp
#include <stdio.h>
#include <set>
#include <vector>
#define MAX 100001
using namespace std;
#define gc getchar_unlocked

void Get(int &ret) {
    ret = 0; char inp = gc(); int kl = 1;
    while (inp < '0' || inp > '9') {
        if (inp == '-') kl = -1; inp = gc();}
    while ('0' <= inp && inp <= '9')
        ret = (ret << 3) + (ret << 1) + (int)(inp - '0'), inp = gc();
        if (kl < 1) ret = -ret;
}

set<int> s;
set<int>::iterator it;
int in[MAX], out[MAX], i, n;

int main() {
    Get(n);
    for (i = 0; i < n; i++) {
        Get(in[i]);
    }
    for (i = n - 1; i >= 0; i--) {
        s.insert(in[i]);
        it = s.find(in[i]);
        s.erase(s.begin(), it);
        it++;
        if (it == s.end()) out[i] = 0;
        else out[i] = *it;
    }

    for (i = 0; i < n; i++) {
        printf("%d ", out[i]);
    }

    printf("\n");

    return 0;
}

```

Dengan menjalan program tersebut diperoleh hasil seperti dibawah ini dengan menggunakan test case yang saya coba buat

Stack:

Success #stdin #stdout 0.01s 5548KB

stdin

```
9
1 3 2 4 6 5 3 2 1
```

stdout

```
3 4 4 6 0 0 0 0 0
```

Set:

Success #stdin #stdout 0.01s 5420KB

stdin

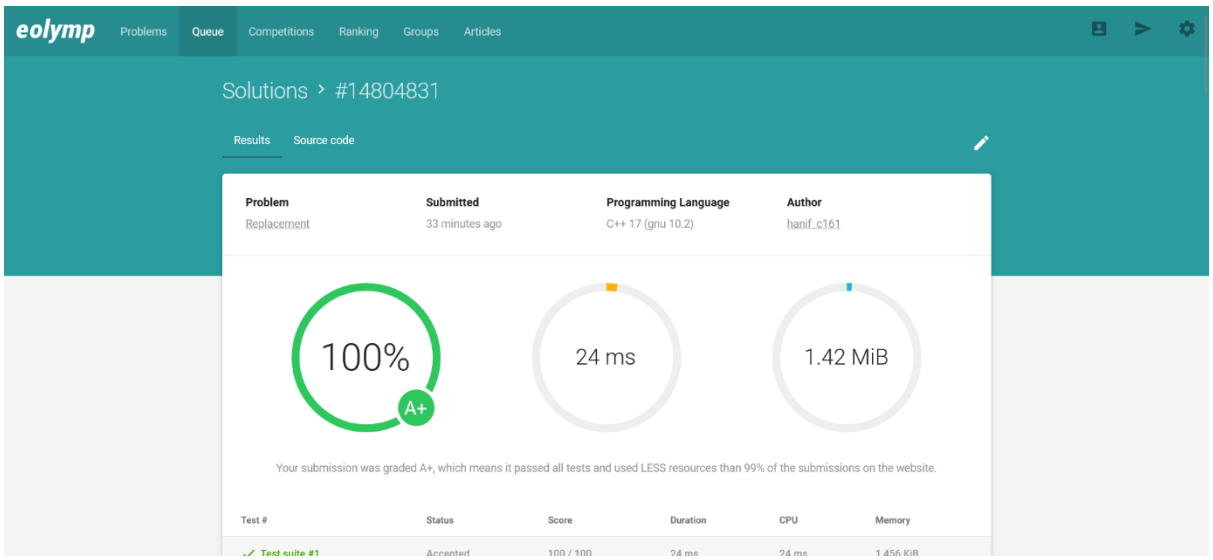
```
9
1 3 2 4 6 5 3 2 1
```

stdout

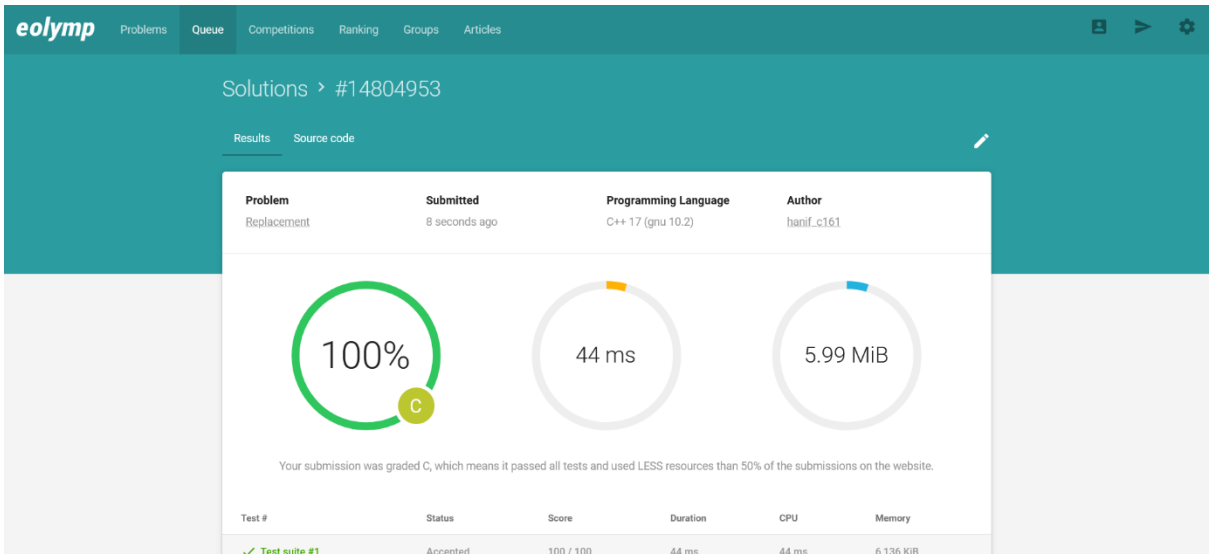
```
3 4 4 6 0 0 0 0 0
```

Dan dibawah bukti Verdict Accept

Stack:



Set:



Perbedaan antara dua kode tersebut cukup signifikan dalam hal waktu eksekusi dan penggunaan memori. Kode pertama berjalan dalam waktu 24ms dan menggunakan sekitar 1.42MiB memori, sementara kode kedua membutuhkan waktu 48ms dan mengonsumsi sekitar 5.68MiB memori.

Hal ini terjadi karena perbedaan dalam algoritma dan struktur data yang digunakan dalam kedua kode. Kode pertama menggunakan tumpukan (stack) untuk mencari elemen terbesar yang berada di sebelah kanan elemen saat ini, sementara kode kedua menggunakan struktur data set yang memerlukan alokasi memori tambahan.

Kode pertama secara efisien menemukan elemen yang lebih besar dan hanya mengalokasikan memori yang diperlukan untuk tumpukan, yang memungkinkan penggunaan memori yang lebih rendah. Sebaliknya, kode kedua memerlukan alokasi memori tambahan untuk set dan mencari elemen yang lebih besar dalam set, yang mengakibatkan penggunaan memori yang lebih tinggi.

Secara sederhana, kode pertama lebih cepat dan hemat memori karena penggunaan tumpukan yang efisien, sedangkan kode kedua membutuhkan lebih banyak memori dan waktu eksekusi karena penggunaan set.

Terima Kasih telah membaca, Waalaikumsalam wr wb.