Queries with Fixed Length

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'solve' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts following parameters:
 *  1. INTEGER_ARRAY arr
 *  2. INTEGER_ARRAY queries
 */

vector<int> solve(vector<int> arr, vector<int> queries) {
    int n=arr.size();
    vector<int> left(n),right(n);
    stack<int> st;
    for(int i=0;i<n;i++){
        while(!st.empty()&&arr[st.top()]<=arr[i]) st.pop();
        left[i]=st.empty()?-1:st.top();
        st.push(i);
    }
    while(!st.empty()) st.pop();
    for(int i=n-1;i>=0;i--){
        while(!st.empty()&&arr[st.top()]<arr[i]) st.pop();
        right[i]=st.empty()?n:st.top();
        st.push(i);
    }
    vector<int> ans(n+1,INT_MAX);
    for(int i=0;i<n;i++){
        int len=right[i]-left[i]-1;
        ans[len]=min(ans[len],arr[i]);
    }
    for(int i=n-1;i>=1;i--) ans[i]=min(ans[i],ans[i+1]);
    vector<int> res;
    for(int q:queries) res.push_back(ans[q]);
    return res;
}
```

```
int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input =
split(rtrim(first_multiple_input_temp));

    int n = stoi(first_multiple_input[0]);

    int q = stoi(first_multiple_input[1]);

    string arr_temp_temp;
    getline(cin, arr_temp_temp);

    vector<string> arr_temp = split(rtrim(arr_temp_temp));

    vector<int> arr(n);

    for (int i = 0; i < n; i++) {
        int arr_item = stoi(arr_temp[i]);

        arr[i] = arr_item;
    }

    vector<int> queries(q);

    for (int i = 0; i < q; i++) {
        string queries_item_temp;
        getline(cin, queries_item_temp);

        int queries_item = stoi(ltrim(rtrim(queries_item_temp)));

        queries[i] = queries_item;
    }

    vector<int> result = solve(arr, queries);

    for (size_t i = 0; i < result.size(); i++) {
        fout << result[i];

        if (i != result.size() - 1) {
            fout << "\n";
        }
    }

    fout << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int,
int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int,
int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));
    return tokens;
}
```