Task 1: Compute tree ordering

Compute the traversing ordering of the tree, store the corresponding (depth, index).

Sample input

- first line, an integer n: the number of nodes in tree
- follow $n-1$ lines, each line contains two integers u, v: the edge

```
7
0 1
0 2
1 3
1 4
2 5
2 6
```

the corresponding tree is

```
         0
      /     \
     1       2
   / \     / \
  3   4   5   6
```

Sample output

- print all elements in the ordering, each element per line - depth and index.

```
0 0
1 1
2 3
1 1
2 4
1 1
0 0
1 2
2 5
1 2
2 6
1 2
0 0
```

- the corresponding ordering is

```
[
  (dep: 0, idx: 0),
  (dep: 1, idx: 1),
  (dep: 2, idx: 3),
  (dep: 1, idx: 1),
  (dep: 2, idx: 4),
  (dep: 1, idx: 1),
  (dep: 0, idx: 0),
  (dep: 1, idx: 2),
  (dep: 2, idx: 5),
  (dep: 1, idx: 2),
  (dep: 2, idx: 6),
  (dep: 1, idx: 2),
  (dep: 0, idx: 0)
]
```

template

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 100;
struct Element {
  int dep, idx;
};
vector<Element> order;
vector<vector<int>> tree;
int n;

void read_tree() {
  cin >> n;
  tree.resize(n);
  for (int i=0; i<n-1; i++) {
    int u, v;
    cin >> u >> v;
    tree[u].push_back(v);
    tree[v].push_back(u);
  }
}

void dfs(int cur, int pa, int depth) {
  // figure out where to add "order.push_back({depth, index})"
  ?
  for (const auto& c: tree[cur]) if (c != pa) {
    ?
    dfs(c, cur, depth+1);
    ?
  }
  ?
}
```

```
int main() {
  read_tree();
  dfs(?, ?, ?);
  // print out elements
  for (auto& elem: order) {
    cout << elem.dep << " " << elem.idx << endl;
  }
  return 0;
}
```

Task 2: RMQ Sparse table

input:

- input tree (same as in previous task)
- then an integer m - the number of queries
- then follow m lines, each line contains two integers u, v - the index of node
- sample:

```
7
0 1
0 2
1 3
1 4
2 5
2 6
4
3 4
5 6
4 5
3 3
```

output:

- for each query, print one integer - the **depth** of LCA (not the index).
- sample:

```
1
1
0
2
```

template

```
#include <bits/stdc++.h>
#include <math.h>
using namespace std;
struct Element {
  int dep, idx;
};
vector<Element> order;
vector<vector<int>> tree;
vector<int> ts;   // timestamp of first visiting
vector<vector<int>> st;
int n;

void read_tree() {
  cin >> n;
  tree.resize(n);
  ts.resize(n);
  for (int i=0; i<n-1; i++) {
    int u, v;
    cin >> u >> v;
    tree[u].push_back(v);
    tree[v].push_back(u);
  }
}

void dfs(int cur, int pa, int depth) {
  // record timestamp
  ?

  order.push_back({depth, cur});
  for (const auto& c: tree[cur]) if (c != pa) {
    dfs(c, cur, depth+1);
    order.push_back({depth, cur});
  }
}

void init_st() {
  int L = order.size();
  st.resize(L);
  // we need 2^k < n and 2^(k+1) >= n
  int k = int(log2(L));
  // init space
  for (int i=0; i<L; i++) st[i].resize(k+1);
  // init value:
  //    basic case
  for (int i=0; i<L; i++) st[i][0] = ?
  //    recursion
  for (int i=1; i<=k; i++) {
    for (int j=0; j<L; j++) {
      int l = j, r = ?;
      st[j][i] = min(st[l][i-1], st[r][i-1]);
    }
  }
}

int LCA(int u, int v) {
  int i = ?, j = ?;
  int k = (int)log2(abs(i-j)+1);
  int res = min(?, ?);
  return res;
}

int main() {
  read_tree();
  dfs(0, -1, 0);
  init_st();
  int m;
  cin >> m;
  while (m--) {
    int u, v;
    cin >> u >> v;
    int res = LCA(u, v);
    cout << res << endl;
  }
  return 0;
}
```

Task 3: Segment tree

Let's forgot the LCA problem, we will only do RMQ in this task

Input

The first line contains one integer n - the size of array, then follow n lines, each line contains one integer - the elements in array;

Then follow by one integer m - the number of queries; Then follow by m lines, each line contains two integer l, r - the index of query

- sample

```
13
```

```
0
1
2
1
2
1
0
1
2
1
2
1
0
4
2 4
8 10
4 8
2 2
```

Output

For each query, print one line contains an integer - the min value between the query indexes.

- sample

```
1
1
0
2
```

Template

```
#include <bits/stdc++.h>
using namespace std;
vector<int> seg;
vector<int> arr;
int n;
const int INF = 1e8;

void read_arr() {
  cin >> n;
  arr.resize(n);
  for (int i=0; i<n; i++) cin >> arr[i];
}

void build_tree(int l, int r, int idx) {
  if (l == r) { // base case
    seg[idx] = arr[l];
    return;
  }
  int mid = (l + r) >> 1;
  build_tree(l, mid, idx << 1);
  build_tree(mid+1, r, idx << 1 | 1);
  // push up
  seg[idx] = min(seg[idx << 1], seg[idx << 1 | 1]);
}

int query(int ql, int qr, int L, int R, int idx) {
  if (?) return seg[idx];
  int mid = (L + R) >> 1;
  int resl = INF, resr = INF;
  if (?) resl = query(ql, qr, L, mid, idx << 1);
  if (?)  resr = query(ql, qr, mid+1, R, idx << 1|1);
  return min(resl, resr);
}

int main() {
  read_arr();

  // we will need about ? space
  seg.resize(?);
  build_tree(0, n-1, 1);
  int m;
  cin >> m;
  while (m--) {
    int l, r;
    cin >> l >> r;
    int res = query(l, r, 0, n-1, 1);
    cout << res << endl;
  }
  return 0;
}
```