14428 - 수열과 쿼리 16

난이도	골드1

문제

문제

길이가 N인 수열 A_1, A_2, ..., A_N이 주어진다. 이때, 다음 쿼리를 수행하는 프로그램을 작성하시오.

- `1 i v` : A_i를 v로 바꾼다. ($1 \le i \le N$, $1 \le v \le 10^9$)
- `2 i j` : A_i, A_{i+1}, ..., A_j에서 크기가 가장 작은 값의 인덱스를 출력한다. 그러한 값이 여러개인 경우에는 인덱스가 작은 것을 출력한다. ($1 \le i \le j \le N$, $1 \le v \le 10^9$)

수열의 인덱스는 1부터 시작한다.

입력

첫째 줄에 수열의 크기 N이 주어진다. ($1 \le N \le 100,000$)

둘째 줄에는 A_1, A_2, ..., A_N이 주어진다. ($1 \le A_i \le 10^9$)

셋째 줄에는 쿼리의 개수 M이 주어진다. ($1 \le M \le 100,000$)

넷째 줄부터 M개의 줄에는 쿼리가 주어진다.

출력

2번 쿼리에 대해서 정답을 한 줄에 하나씩 순서대로 출력한다.

예제 입력 1 복사

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

예제 출력 1 복사

```
3
4
4
3
```

풀이과정

전형적인 **인덱스 트리** 구현 문제였다. 다만 구간합 구하기 문제와는 달리 이 문제에서의 쿼리는 각 노드의 값의 합을 구하는 것이 아닌, **가장 작은 값의 인덱스**를 출력하는 것이었다. 단, 값이 여러 개인 경우 인덱스가 더 작은 것을 출력하도록 한다.

💡 주요 포인트

1. 가장 작은 값의 인덱스를 출력하라

위에서도 언급했다시피, 가장 작은 값의 인덱스를 출력하는 게 해당 문제의 쿼리문이었기 때문에, 현재 구간에서 가장 작은 값을 담는 인덱스 트리와 해당 인덱스를 담는 인덱스 트리를 각각 만들었다.

소스코드

```
package Tree.IndexedTree.BOJ14428;

/*
```

```
* < Indexed Tree >
* query & update by
* - Bottom up
* */

import java.io.*;
import java.util.Arrays;
import java.util.StringTokenizer;

public class Main {

    static int N, M, S;
    static long[] tree;
    static int[] minIndex;

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
        N = Integer.parseInt(br.readLine());

        S = 1;
        while(S < N)
            S *= 2;
        tree = new long[2*S];
        minIndex = new int[2*S];
        Arrays.fill(tree, Long.MAX_VALUE);
        Arrays.fill(minIndex, Integer.MIN_VALUE);

        // 1. Init Indexed tree
        StringTokenizer st = new StringTokenizer(br.readLine());
        for(int i=S ; i<S+N ; i++) {
            tree[i] = Long.parseLong(st.nextToken());
            minIndex[i] = i;
        }

        // 2. 내부 노드 순회
        for(int i=S-1 ; i>0 ; i--){
            setNode(i);
        }

        M = Integer.parseInt(br.readLine());

        // 2. Query & Update
        int command;
        for(int i=1 ; i<=M ; i++) {
            st = new StringTokenizer(br.readLine());
            command = Integer.parseInt(st.nextToken());
            int a = Integer.parseInt(st.nextToken());

            if(command == 1) {  // Update
                long b = Long.parseLong(st.nextToken());
                updateBU(a, b);
//                updateTD(1, S, 1, a, b-a);
            } else {    // Query - 크기가 가장 작은 값의 인덱스 출력
                int b = Integer.parseInt(st.nextToken());
                long answer = queryBU(a, b);
//                long answer = queryTD(1,S, 1, a,b);
                bw.write((answer - S + 1) + "\n");
```

```
            }
        }
        bw.flush();
        bw.close();
        br.close();
    }

    // < Bottom up >
    static void updateBU(int target, long value) {
        int node = S + target - 1;
        tree[node] = value;
        node /= 2;

        while(node >= 1) {
            setNode(node);
            node /= 2;
        }
    }

    private static void setNode(int node) {
        tree[node] = Math.min(tree[node*2], tree[node*2+1]);
        if(tree[node*2] <= tree[node*2+1]){
            minIndex[node] = minIndex[node*2];
        } else{
            minIndex[node] = minIndex[node*2+1];
        }
    }

    static int queryBU(int left, int right) {
        int resultIdx = 0;
        int leftNode = S + left - 1;
        int rightNode = S + right - 1;

        while(leftNode <= rightNode) {
            if(leftNode % 2 == 1){
                if(tree[leftNode] < tree[resultIdx]){
                    resultIdx = minIndex[leftNode];
                }
                if(tree[leftNode] == tree[resultIdx]){
                    resultIdx = Math.min(resultIdx, minIndex[leftNode]);
                }
                leftNode++;
            }
            if(rightNode % 2 == 0){
                if(tree[rightNode] < tree[resultIdx]){
                    resultIdx = minIndex[rightNode];
                }
                if(tree[rightNode] == tree[resultIdx]){
                    resultIdx = Math.min(resultIdx, minIndex[rightNode]);
                }
                rightNode--;
            }

            leftNode /= 2;
            rightNode /= 2;
        }

        return resultIdx;
```

```
    }
}
```