# BOJ 2357 최솟값과 최댓값

# Step1 접근

## 문제

N(1 ≤ N ≤ 100,000)개의 정수들이 있을 때, a번째 정수부터 b번째 정수까지 중에서 제일 작은 정수, 또는 제일 큰 정수를 찾는 것은 어려운 일이 아니다. 하지만 이와 같은 a, b의 쌍이 M(1 ≤ M ≤ 100,000)개 주어졌을 때는 어려운 문제가 된다. 이 문제를 해결해 보자.

여기서 a번째라는 것은 입력되는 순서로 a번째라는 이야기이다. 예를 들어 a=1, b=3이라면 입력된 순서대로 1번, 2번, 3번 정수 중에서 최소, 최댓값을 찾아야 한다. 각각의 정수들은 1이상 1,000,000,000이하의 값을 갖는다.

## 입력

첫째 줄에 N, M이 주어진다. 다음 N개의 줄에는 N개의 정수가 주어진다. 다음 M개의 줄에는 a, b의 쌍이 주어진다.

## 출력

M개의 줄에 입력받은 순서대로 각 a, b에 대한 답을 최솟값, 최댓값 순서로 출력한다.

# Step1 접근

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 75 | 30 | 100 | 38 | 50 | 51 | 52 | 20 | 81 | 5 |

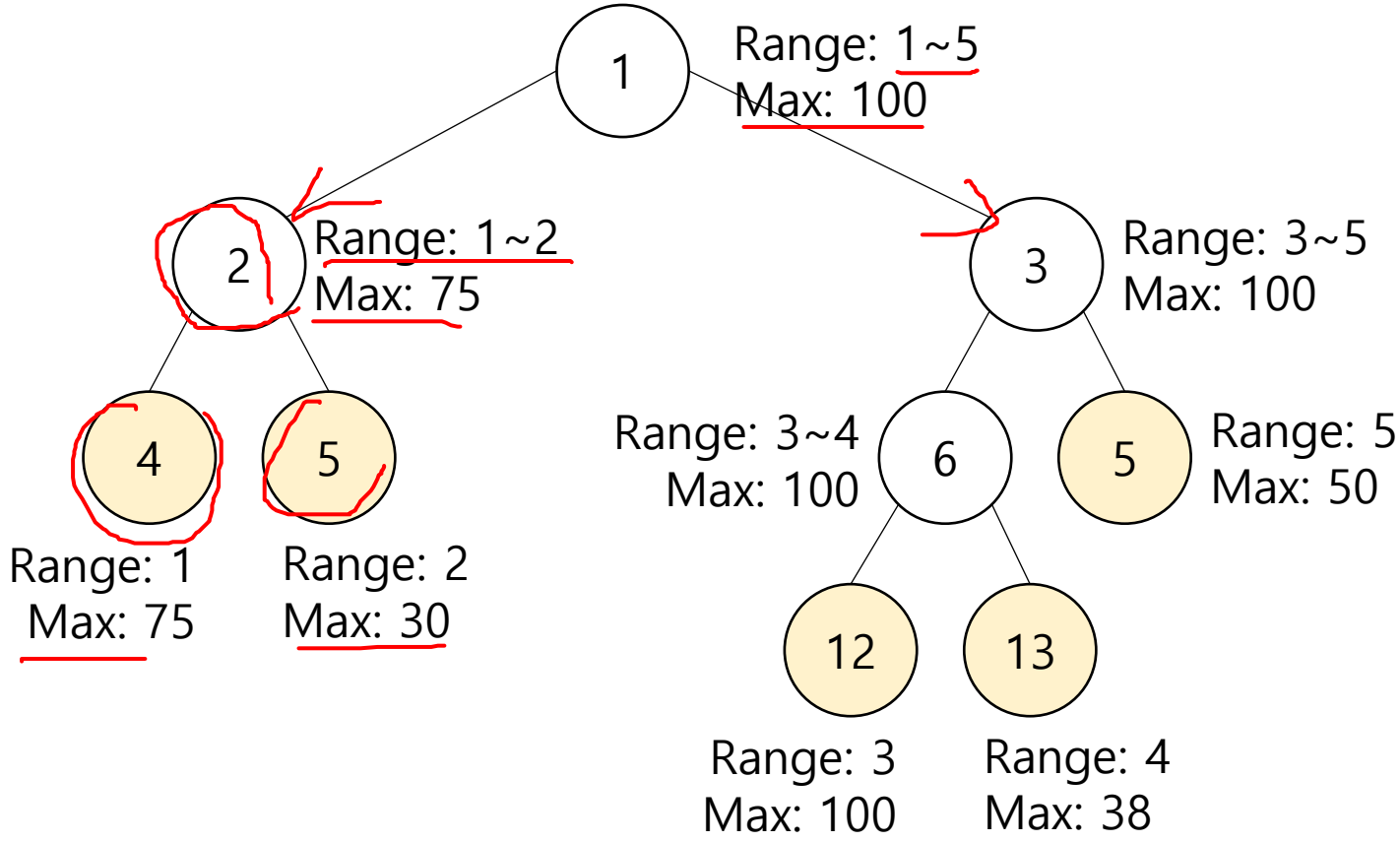
구간(부분)의 최소/최대 값을 구하는 문제
선형적으로 답을 구하면 평균적으로 O(N*M)의 시간복잡도를 갖게 됨
N*M = 10^10
시간초과

세그먼트 트리로 답을 구하면 O(M*logN)의 시간복잡도로 구할 수 있음
M*logN = 10^5 * 16.609640474437

따라서 입력된 배열에 대한 각 구간의 최소/최대 값을
1. 트리로 저장하고
2. 트리를 활용하여 구간 질의에 대한 답을 구해야 함

# Step1 접근 – 최댓값 세그먼트 트리

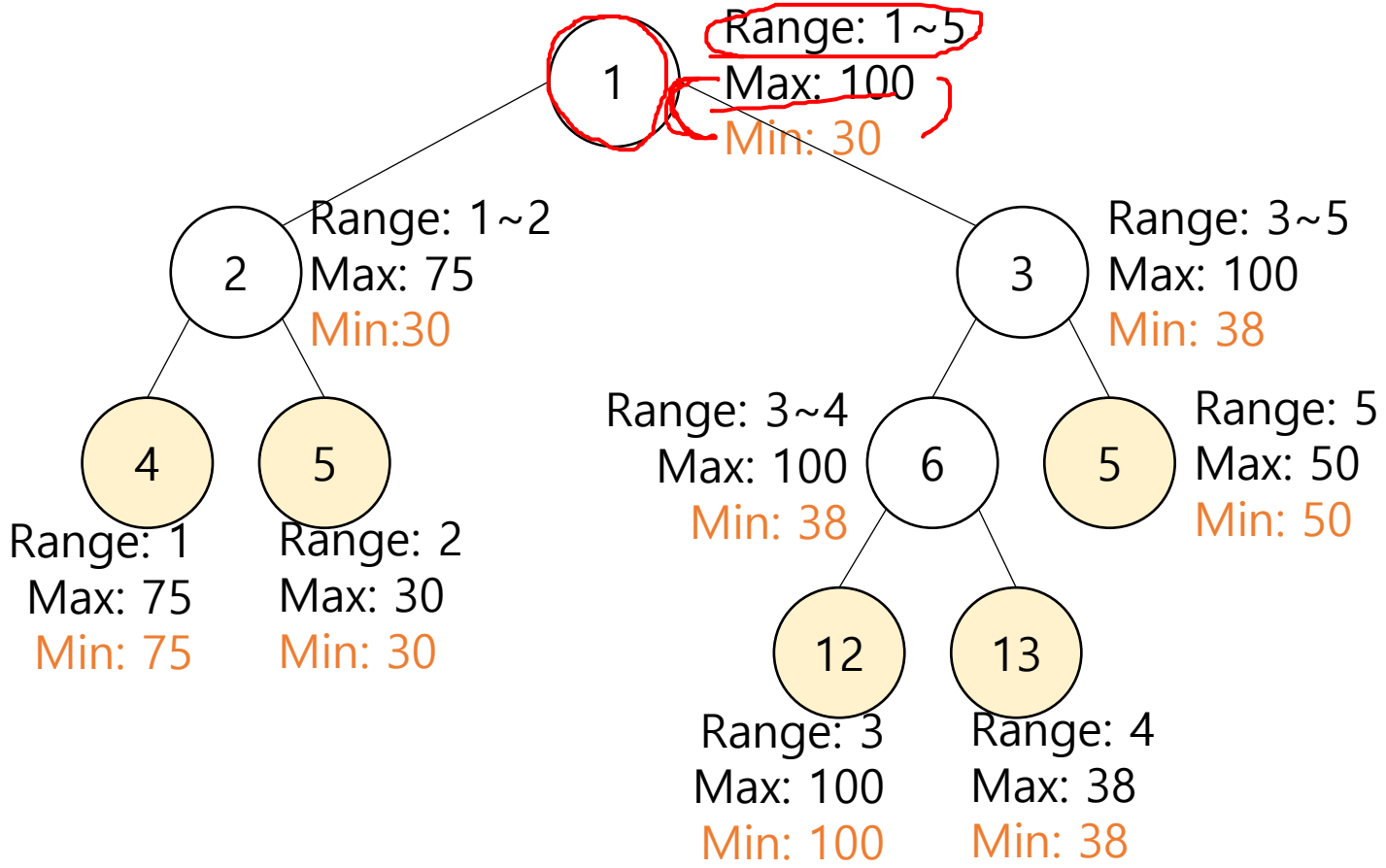
| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 75 | 30 | 100 | 38 | 50 |




<최대값 세그먼트 트리>

# Step1 접근

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 75 | 30 | 100 | 38 | 50 |




<최대, 최소값 세그먼트 트리>

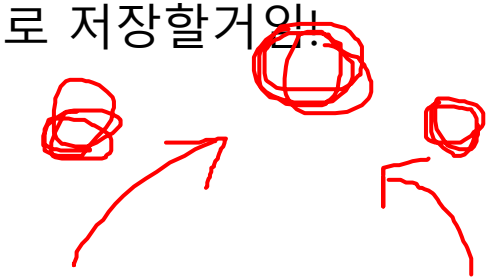
# Step1 접근 – 트리 만들기

트리의 루트는 배열 전체 범위에서의 최대/최소값이 저장됨

그의 자식 노드에는 각각 왼쪽 절반 범위와 오른쪽 절반 범위에서의 최대/최소값이 저장됨

이를 위해 재귀적으로 트리를 구성해 나가야 함!

seg_tree[node] = {max_value, min_value}; // pair<double, double> 형식으로 저장할거임!



즉 seg_tree[node].first : 해당 범위에서의 최대값
seg_tree[node].seconde : 해당 범위에서의 최소값

```
seg_tree[node].first = max(seg_tree[left_child].first, seg_tree[right_child].first)
seg_tree[node].second = min(seg_tree[left_child].second, seg_tree[right_child].second)
```

# Step1 접근 – 구간 질의 응답

주어진 인덱스 범위 내에서의 최대/최소값을 찾아야 함

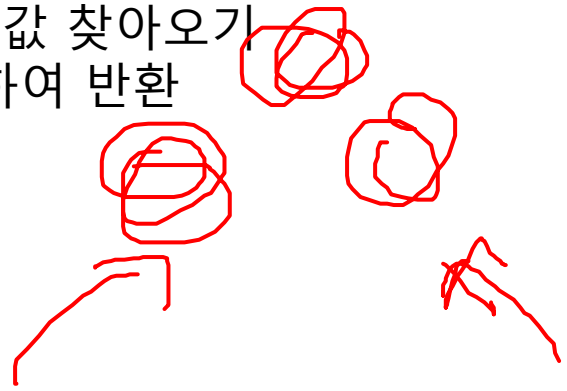
트리를 탐색하며
각 노드가 표현하는 범위가 질의 범위에 포함되는지 확인

완전히 포함되면 해당 구간의 최대/최소 값 가져오기
일부 포함되면 포함되는 구간을 찾아서 재귀적으로 구간의 최대/최소 값 찾아오기
포함되지 않으면 해당 구간의 최대값은 0, 최소값은 무한대라고 가정하여 반환

각 자식 노드로부터 얻게 된 부분 최대/최소 값을 비교하여
주어진 질의 구간에서의 최대/최소 값 반환

```
Max_value = max(left_child.first, right_child.first)
Min_value = min(left_child.second, right_child.second)

Return (max_value, min_value)
```



# Step2 구현

```
pii init(int start, int end, int node, vector<double> &tree, vector<pii> &seg_tree)
{
    if (start == end) {
        seg_tree[node] = { tree[start], tree[start] };
        return seg_tree[node];
    }
    else {
        int mid = (start + end) / 2;

        pii left_node = init(start, mid, node << 1, tree, seg_tree);
        pii right_node = init(mid + 1, end, (node << 1) + 1, tree, seg_tree);
        seg_tree[node].first = min(left_node.first, right_node.first);
        seg_tree[node].second = max(left_node.second, right_node.second);

        return seg_tree[node];
    }
}
```

# Step2 구현

```
pii getPartialMinMax(int start, int end, int node, int left, int right, vector<double>& tree, vector<pii>& seg_tree)
{
    if (end < left || start > right)
        return { 2147483647, 0 };
    else if (start >= left && end <= right) {
        return seg_tree[node];
    }
    else {
        int mid = (start + end) / 2;

        pii left_node = getPartialMinMax(start, mid, node << 1, left, right, tree, seg_tree);
        pii right_node = getPartialMinMax(mid + 1, end, (node << 1) + 1, left, right, tree, seg_tree);

        return { min(left_node.first, right_node.first), max(left_node.second, right_node.second) };
    }
}
```

# Step2 구현

```
cin.tie(NULL);
cout.tie(NULL);

int n, m;
cin >> n >> m;

vector<double> tree(n + 1);
vector<pii> seg_tree(n << 2);

for (int i = 1; i <= n; i++)
    cin >> tree[i];

init(1, n, 1, tree, seg_tree);

for (int i = 0; i < m; i++) {
    int a, b;
    cin >> a >> b;

    pii result = getPartialMinMax(1, n, 1, a, b, tree, seg_tree);

    cout << result.first << " " << result.second << "\n";
}
```

THX