Lab1 Submission

2019147029 조은기

Longest Mountain in Array

```
class Solution {
public:
    int longestMountain(vector<int> &arr) {
        int arrSize = arr.size();
        if (arrSize < 3) {
            return 0;
        }
        int ans = 0, i = 0;
        while (i < arrSize - 1) {
            bool increasing = true;
            int j = i;
            while (j < arrSize - 1) {
                if ((increasing && arr[j] < arr[j + 1]) || (
                        !increasing && arr[j] > arr[j + 1])) {
                    j++;
                } else if (increasing && i < j && arr[j] > arr[j + 1]) {
                    increasing = false;
                    j++;
                } else {
                    break;
                }
            }
            if (i < j && !increasing) {
                if (j - i + 1 > ans) {
                    ans = j - i + 1;
                }
                i = j;
            } else {
                i++;
            }
        }
        return ans;
    }
};
```

Q1: 배열의 길이가 3 미만일 때 0을 반환하는 이유는 무엇인가요?

A1: Mountain은 오르는 구간이 있고, 내리는 구간이 있어야 하기 때문에 최소 3개의 값이 필요합니다. 따라서 배열 길이가 2 이하면 0을 early return 해버리는 것입니다.

Q2: `increasing` 변수는 어떤 역할을 하나요?

A2: `increasing` 변수는 현재 구간이 산의 오르는 구간인지 내리는 구간인지를 기록하는 역할을 합니다. 처음에는 증가 구간이 시작될 때부터 측정해야 하므로 `true`로 설정되어 있고, 최소 1 부분 증가한 이후에 감소하기 시작하면 `false`로 변경해 감소하는 부분으로 인식합니다.

Q3: `j` 인덱스를 사용하는 이유와 `j`를 증가시키는 조건은 무엇인가요?

A3: `j` 인덱스는 배열의 시작 인덱스인 `i`에서 시작하여 오르는 부분을 찾고, 이후에 감소하다가 실제로 mountain 구간의 끝을 찾기 위한 변수입니다.

- 오르는 구간에서 `arr[j] < arr[j + 1]`일 때 `j`를 증가시킵니다.
- 내리는 구간에서 `arr[j] > arr[j + 1]`이고 `i`와 `j`가 다른 경우, 즉 이전에 오르는 구간이 이미 있었을 경우에 `increasing`을 `false`로 바꾸고 내리는 구간을 시작합니다.

Q4: 코드에서 `i`를 `j`로 업데이트하는 이유는 무엇인가요?

A4: `i`를 `j`로 업데이트하는 것은 다시 while문으로 돌아갔을 때, 이미 앞서 있던 산은 건너뛰고 이후에 나올 산만을 검토하기 위해서입니다.

Q5: 이 코드의 시간 복잡도는 어떻게 되나요?

A5: 이 코드의 시간 복잡도는 `O(n)`입니다. 전체 배열을 한 번만 순회하기 때문입니다. 각 원소를 한 번만 검토하고, `i`와 `j`를 적절히 업데이트해서 배열의 같은 부분을 중복하지 않고 쭉 한 번 스캔합니다.