Maximum Width Ramp

solved by	Senan
Platform	LeetCode
difficulty	Medium
Serial	962
tags	Stack
language	C++
solved on	@10/10/2024
link	https://leetcode.com/problems/maximum-width-ramp/description/
Completion	☑

Intuition

The task is to find the maximum width ramp in the array `nums`. A "ramp" is defined as a pair of indices `(i, j)` such that `i < j` and `nums[i] <= nums[j]`. The goal is to maximize `j - i`.

The intuition behind this approach is to use a **monotonic decreasing stack** to track potential starting indices `i` where `nums[i]` could be a valid left end of a ramp. Then, we try to maximize `j - i` by checking each `j` starting from the right side of the array and finding valid `i` values from the stack.

Approach

1. **Building the stack**:

   We iterate through the array from left to right and push indices onto the stack if they could be a valid `i`. The stack maintains indices of the array in decreasing order of values of `nums[i]`, which ensures that for any `i` in the stack, `nums[i]` is a potential valid left end of a ramp for future `j` values.

2. **Checking for ramps**:

   Starting from the rightmost index `j` (i.e., `n - 1`), we pop elements from the stack while `nums[st.top()] <= nums[j]`. Each time we pop from the stack, it means we found a valid ramp with `i = st.top()` and `j = current index`. We update `ans` with the maximum `j - i`.

Complexity

Time Complexity:

- **Stack Construction (First loop)**:

  The first loop runs for all `n` elements, and each element is pushed onto the stack once. Hence, this part is **O(n)**.

- **Checking ramps (Second loop)**:

  In the second loop, for each `j` (from right to left), we pop from the stack only once for each element. So the second loop is also **O(n)**.

Thus, the overall time complexity is **O(n)**.

Space Complexity:

We use a stack to store indices, which in the worst case can hold all `n` elements. Hence, the space complexity is **O(n)**.

Code

```
int maxWidthRamp(vector<int>& nums) {
    // decreasing stack

    int n = nums.size();
    stack<int> st;

    // Build the decreasing stack
    for (int i = 0; i < n; i++) {
        if (st.empty() || nums[st.top()] > nums[i]) {
            st.push(i);
        }
    }

    int ans = 0;

    // Check ramps from the right to the left
    for (int i = n - 1; i >= 0; i--) {
        while (!st.empty() && nums[st.top()] <= nums[i]) {
            ans = max(ans, i - st.top());
            st.pop();
        }
    }

    return ans;
}
```