Leaders in an Array

The leader in an array is defined as an element for which every element to its right is smaller.

> Definition: An element `x` in an array is a leader if for all elements `y` to the right of `x`, `x > y`.

Some important details:

- The rightmost element is always a leader because there are no elements to its right.
- The format of the answer may vary. You might need to return the leaders in the order they appear in the array or in sorted order.

Example

Consider the array `[10, 22, 12, 3, 6, 0]`.

- `22` is a leader because every element to its right (`12`, `3`, `6`, `0`) is smaller than `22`.
- `12` is a leader because every element to its right (`3`, `6`, `0`) is smaller than `12`.
- `3` is not a leader because `6` is to the right of `3` and `6 > 3`.
- `10` is not a leader because `22` and `12` are to the right of `10`, and they are greater than `10`.
- `6` is a leader because it is the last element in the array.
- `0` is not a leader because there are no elements to its right and it is by definition the last element.

Depending on the question, the returned answer can be `[22, 12, 6]` (in order of appearance) or `[6, 12, 22]` (sorted).

Brute Force Solution

The brute force solution involves checking each element to see if it's a leader by comparing it with all elements to its right.

Algorithm

1. Iterate through each element of the array.
2. For each element, iterate through all the elements to its right.
3. If any element to the right is greater than the current element, the current element is not a leader.
4. If no element to the right is greater, the current element is a leader.

Code

```
for (int i = 0; i < n; i++) {
    bool leader = true;
    for (int j = i + 1; j < n; j++) {
        if (arr[j] > arr[i]) {
            leader = false;
            break;
        }
    }
    if (leader) {
        // Store the leader in the answer
    }
}
```

Complexity Analysis

- Time Complexity: $O(n^2)$ because for every element, we do a linear search on the right portion of the array.
- Space Complexity: $O(n)$ in the worst case, if all elements are leaders (e.g., a descending sorted array). This space is used to store the answer.

Optimal Solution

The optimal solution involves iterating from the back of the array and keeping track of the maximum element seen so far on the right side.

Algorithm

1. Start iterating from the end of the array.
2. Keep track of the maximum element seen so far on the right side.
3. If the current element is greater than the maximum element on the right, it is a leader.
4. Update the maximum element with the current element if the current element is greater.
5. If the problem asks to return the leaders in the order of the array, reverse the answer. If it asks for sorted leaders, sort the collected leaders.

Example

Given the array `[10, 22, 12, 3, 6, 0]`:

1. Start from the end: `0`. The maximum on the right is negative infinity (or a very small number). `0` is a leader. Update the maximum to `0`.
2. Move to `6`. The maximum on the right is `0`. `6` is a leader. Update the maximum to `6`.
3. Move to `3`. The maximum on the right is `6`. `3` is not a leader. The maximum remains `6`.
4. Move to `12`. The maximum on the right is `6`. `12` is a leader. Update the maximum to `12`.
5. Move to `22`. The maximum on the right is `12`. `22` is a leader. Update the maximum to `22`.
6. Move to `10`. The maximum on the right is `22`. `10` is not a leader. The maximum remains `22`.

The leaders collected are `[0, 6, 12, 22]`. If the question asks for the original order, reverse it to get `[22, 12, 6, 0]`. If it asks for the sorted order, sort it to get `[0, 6, 12, 22]`.

Code

```cpp
vector<int> leaders(vector<int>& arr) {
    int n = arr.size();
    vector<int> ans;
    int max_right = INT_MIN; // Initialize to a very small number

    for (int i = n - 1; i >= 0; i--) {
        if (arr[i] > max_right) {
            ans.push_back(arr[i]);
            max_right = arr[i];
        }
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```

Complexity Analysis

- Time Complexity: $O(n)$ because we iterate through the array once. If sorting is required, it becomes $O(n \log n)$.
- Space Complexity: $O(n)$ in the worst case to store the answer.

Finding Leaders in an Array

The Algorithm

To identify leaders in an array, we iterate through the array from right to left, maintaining the maximum element encountered so far.

The process involves updating the `maxi` variable and checking if the current element is a leader.

Here's the breakdown:

1. Initialization: Start from the end of the array.
2. Iteration: Move from right to left.
3. Maximum Tracking: Update `maxi` if the current element is greater.
4. Leader Identification: Determine if the current element is a leader by comparing it to `maxi`.

Code Logic

Update `maxi` to `array[i]` if `array[i]` > `maxi`. This simple backward iteration continuously updates the maximum.

Compare each element to the running `maxi` to identify leaders.

Time Complexity

The time complexity to collect the leaders is $O(n)$.

If sorting is required (as per the problem statement), the time complexity becomes $n \log n$ in the worst-case scenario. Imagine if every element is a leader, it will take $O(n \log n)$ to sort.

Space Complexity

The space complexity is $O(n)$ in the worst case where every element is a leader.

Key Considerations

- The additional space used is primarily for returning the result, not for solving the problem itself.
- Make sure to explicitly state this to the interviewer.

Code Availability

Equivalent Java and Python code can be found in the link in the description (of the video), within the article.