# 1732. Find the Highest Altitude

Easy | Array | Prefix Sum

Leetcode Link

## Problem Description

The problem presents a scenario where a biker is going on a road trip across `n + 1` points that are situated at varying altitudes. He begins his journey from point 0, which is at sea level (altitude 0). The altitude changes as he moves from one point to the next, and these changes are represented by an array named `gain`, where each element `gain[i]` indicates the net altitude gain (or loss, if the value is negative) as the biker moves from point `i` to point `i + 1`. The objective is to figure out what the highest altitude the biker reaches during his trip is. The length of the `gain` array is `n`, with `i` ranging from 0 to `n - 1`.

## Intuition

To solve this problem, we need to keep track of the biker's altitude as he moves from one point to the next. We start at an altitude of 0 and add the net gain from the `gain` array consecutively to find the altitude at each subsequent point. While doing this, we keep an eye on the highest altitude reached thus far.

The solution approach involves two main steps:

1. Initialize a variable to keep track of the biker's altitude as he progresses on the trip (`h` in the given solution).
2. Initialize another variable to maintain the highest altitude (`ans` in the given solution) seen so far.

In a loop, we add each net gain to the current altitude `h`. After each addition, we compare the new altitude `h` with the current highest altitude `ans`. If `h` is greater than `ans`, it means we have found a new highest altitude, and we update `ans` to reflect this new value. We continue this process for all points to ensure we find the absolute highest altitude reached. Ultimately, we return the value of `ans` as the solution, which represents the highest altitude reached during the trip.

## Solution Approach

The given solution is a straightforward iterative approach. In terms of algorithms, data structures, or patterns used, this solution is very simple and doesn't employ complex data structures or algorithms. Here's a step-by-step walk-through of the implementation:

- First, we define a function `largestAltitude` that takes an `List[int]` named `gain` as input.
- Within the function, we initialize two variables, `ans` and `h`, to `0`. Variable `ans` is used to keep track of the highest altitude reached, while `h` keeps track of the current altitude.
- We iterate over each value `v` in the `gain` array using a for loop.
- On each iteration, we increment the current altitude `h` by the net gain value `v`. This represents the altitude at the current point `i + 1`.
- Next, we use the `max` function to update the `ans`. The `max` function takes two arguments, the current highest altitude `ans` and the new altitude `h`. If `h` is higher, `ans` is updated to `h`, otherwise it remains the same.
- After the loop completes, all points have been visited, and the `ans` contains the highest altitude the biker reaches.
- Finally, the function returns the value stored in `ans`.

To summarize, the solution iterates once through the list of altitude gains, keeps a running sum, and simultaneously tracks the maximum altitude encountered. No complex data structures are used, and the time complexity is O(n), where n is the size of the `gain` list, as it requires a single traversal of the list.

### Example Walkthrough

Let's take a small example to illustrate the solution approach with a gain array of `[4,-1,3,1,-5]`. This array represents the net altitude gain or loss as the biker moves from one point to another.

We start by initializing the current altitude, `h`, to 0, since the biker starts his journey at sea level. Likewise, we initialize the highest altitude reached, `ans`, also to 0.

Walkthrough of the steps:

1. The first element in `gain` is `4`, the biker moves from point `0` to point `1` and gains `4` units of altitude. The new current altitude `h` becomes `4`. Since `4 > 0`, we update `ans` to be `4`.
2. The next element is `-1`, representing a loss of altitude as the biker moves to the next point. Adjusting the current altitude `h` by `-1`, we get `h = 4 - 1 = 3`. The highest altitude `ans` remains `4`, as `3` is not greater than `4`.
3. The biker gains `3` units of altitude at the next step, which makes the new `h = 3 + 3 = 6`. This is greater than our current highest altitude `ans`, so we update `ans` to `6`.
4. Another gain is encountered, `1` unit, thus `h = 6 + 1 = 7`. Again, this is higher than the previous 'ans', so `ans` is updated to `7`.
5. The last element is `-5`, when the biker moves to the final point, losing `5` units. So `h` becomes `7 - 5 = 2`. Since `2` is not greater than the current `ans` which is `7`, we make no changes to `ans`.

After iterating through the entire `gain` array, we've kept track of the current and highest altitudes at each point. The final value of `ans` is `7`, which means the highest altitude reached by the biker during his trip is `7` units above sea level.

Finally, we return `7` as our answer.

## Python Solution

```python
from typing import List  # Import List from typing module to use for type hinting

class Solution:
    def largest_altitude(self, gain: List[int]) -> int:
        # Initialize variables:
        # max_altitude to track the highest altitude reached
        # current_altitude to keep the running sum of altitude changes
        max_altitude, current_altitude = 0, 0

        # Iterate through each altitude change in the list
        for altitude_change in gain:
            # Add the altitude change to the current altitude
            current_altitude += altitude_change

            # Update max_altitude if the current altitude is higher
            max_altitude = max(max_altitude, current_altitude)

        # Return the maximum altitude reached
        return max_altitude

```

## Java Solution

```java
class Solution {
    public int largestAltitude(int[] gain) {
        int maxAltitude = 0;   // Variable to store the highest altitude reached
        int currentAltitude = 0; // Variable to track the current altitude

        // Loop through all the altitude gains
        for (int altitudeGain : gain) {
            // Update the current altitude by adding the altitude gain
            currentAltitude += altitudeGain;
            // Update maxAltitude if the currentAltitude is greater than the maxAltitude seen so far
            maxAltitude = Math.max(maxAltitude, currentAltitude);
        }

        // Return the highest altitude reached
        return maxAltitude;
    }
}

```

## C++ Solution

```cpp
#include <vector>
#include <algorithm> // Include for the max() function

class Solution {
public:
    // This function calculates the largest altitude reached
    // based on changes in altitude represented by the 'gain' vector.
    int largestAltitude(vector<int>& gain) {
        int largestAltitudeReached = 0; // Will store the maximum altitude reached
        int currentAltitude = 0; // Will keep track of the current altitude

        // Iterate through each gain value
        for (int altitudeChange : gain) {
            currentAltitude += altitudeChange; // Update the current altitude by adding the altitude change
            largestAltitudeReached = max(largestAltitudeReached, currentAltitude); // Update maximum altitude if current altitude is
        }

        // Return the highest altitude reached during the hike
        return largestAltitudeReached;
    }
};

```

## Typescript Solution

```typescript
/**
 * This function calculates the highest altitude reached in a journey given the gain in altitude at each step.
 * @param {number[]} altitudeGain - An array of numbers representing the altitude gain at each step of the journey.
 * @return {number} - The highest altitude reached during the journey.
 */
const largestAltitude = (altitudeGain: number[]): number => {
    let highestAltitude: number = 0; // Initialize the highest altitude
    let currentAltitude: number = 0; // Initialize the current altitude

    // Loop through the altitude gain at each step
    for (const altitudeChange of altitudeGain) {
        currentAltitude += altitudeChange; // Update the current altitude
        highestAltitude = Math.max(highestAltitude, currentAltitude); // Update the highest altitude if current is greater
    }

    return highestAltitude; // Return the highest altitude reached
};

```

## Time and Space Complexity

### Time Complexity

The time complexity of the given code is `O(n)`, where `n` is the length of the `gain` list. This is because the code iterates through each element of the `gain` list exactly once.

### Space Complexity

The space complexity of the given code is `O(1)` (constant space complexity). This is because the space used does not grow with the size of the input; only a fixed amount of extra space is used for variables `ans` and `h` regardless of input size.