LeetCode 904. Fruit Into Baskets - Full Explanation

Problem:

You are visiting a farm that has a single row of fruit trees represented by an integer array. Each tree produces a certain type of fruit, and you have two baskets. Each basket can hold only one type of fruit, but an unlimited number of them. Starting from any tree, you can pick fruits continuously until you encounter a third fruit type. The goal is to find the maximum number of fruits you can collect in one go.

Example:

```
Input: fruits = [1,2,3,2,2]
Output: 4
Explanation: We can pick from trees [2,3,2,2].
If we had started at the first tree, we would only pick from trees [1,2].
```

Brute Force Approach:

1. Iterate over each index as a starting point. 2. From that index, keep adding fruits to a set until you encounter a third fruit type. 3. Track the length of the current window and update the maximum length found. 4. Time Complexity: $O(n^2)$ since for each index, we may traverse the rest of the array. 5. Space Complexity: O(1) because the set can contain at most 3 elements.

Dry Run (Brute Force):

```
Input: fruits = [1,2,3,2,2]

i=0 → start with fruit 1
→ add 1 → set = {1}
→ add 2 → set = {1,2}
→ add 3 → set = {1,2,3} → invalid (3 types)
→ max = 2

i=1 → start with fruit 2
→ add 2 → {2}
→ add 3 → {2,3}
→ add 2 → {2,3}
→ add 2 → {2,3} → valid till end → max = 4

Answer = 4
```

Optimized Approach (Two Pointers + Sliding Window):

We use a sliding window with two pointers (left and right): 1. Move the right pointer to include new fruits in the window. 2. Store the count of each fruit in a hashmap. 3. If the hashmap size exceeds 2, move the left pointer to shrink the window. 4. Keep updating the maximum window size. Time Complexity: O(n) Space Complexity: O(1)

Dry Run (Sliding Window):

```
Input: fruits = [1,2,3,2,2]

left=0, right=0 → map={1:1}, max=1
right=1 → map={1:1,2:1}, max=2
right=2 → map={1:1,2:1,3:1} → 3 types → move left
```

```
→ remove fruits[left=0] → map={2:1,3:1}, left=1
right=3 → map={2:2,3:1}, max=3
right=4 → map={2:3,3:1}, max=4

Answer = 4
```

Solutions:

```
C++:
int totalFruit(vector<int>& fruits) {
    unordered_map<int,int> basket;
    int left = 0, maxFruits = 0;
    for (int right = 0; right < fruits.size(); ++right) {
        basket[fruits[right]]++;
        while (basket.size() > 2) {
            basket[fruits[left]]--;
            if (basket[fruits[left]] == 0)
                basket.erase(fruits[left]);
            left++;
        }
        maxFruits = max(maxFruits, right - left + 1);
    }
    return maxFruits;
}

Java:
class Solution {
    public int totalFruit(int[] fruits) {
        Map<Integer, Integer> basket = new HashMap<>();
        int left = 0, maxFruits = 0;
        for (int right = 0; right < fruits.length; right++) {
            basket.put(fruits[right], basket.getOrDefault(fruits[right], 0) + 1);
            while (basket.size() > 2) {
                basket.put(fruits[left], basket.get(fruits[left]) - 1);
                if (basket.get(fruits[left]) == 0) basket.remove(fruits[left]);
                left++;
            }
            maxFruits = Math.max(maxFruits, right - left + 1);
        }
        return maxFruits;
    }
}

Python:
def totalFruit(fruits):
    from collections import defaultdict
    count = defaultdict(int)
    left = 0
    max_len = 0

    for right, fruit in enumerate(fruits):
        count[fruit] += 1
        while len(count) > 2:
            count[fruits[left]] -= 1
            if count[fruits[left]] == 0:
                del count[fruits[left]]
            left += 1
        max_len = max(max_len, right - left + 1)

    return max_len
```