## Largest Subarray With Contiguous Elements in C++

```
#include <iostream>
#include <unordered_set>
#include <vector>

using namespace std;

int solution(vector<int>& arr) {
    int ans = 0;

    for (int i = 0; i < arr.size() - 1; i++) {
        int min_val = arr[i];
        int max_val = arr[i];
        unordered_set<int> contiguous_set;
        contiguous_set.insert(arr[i]);

        for (int j = i + 1; j < arr.size(); j++) {
            if (contiguous_set.find(arr[j]) !=
contiguous_set.end()) {
                break; // If duplicate found, break the loop
            }

            contiguous_set.insert(arr[j]);
            min_val = min(min_val, arr[j]);
            max_val = max(max_val, arr[j]);

            if (max_val - min_val == j - i) {
                int len = j - i + 1;
                if (len > ans) {
                    ans = len;
                }
            }
        }
    }

    return ans;
}

int main() {
    vector<int> arr = {10, 12, 11};
    cout << solution(arr) << endl; // Output: 3

    return 0;
}
```

**Dry Run:**

**Input:**

arr = {10, 12, 11}

**Step 1 - Iterate Over the Array:**

- We start by iterating over the array with two nested loops.
- The outer loop runs from i = 0 to i = n - 2 (where n is the size of the array).
- For each value of i, we initialize min_val and max_val with the value of arr[i] and set up a unordered_set to keep track of the distinct elements in the current contiguous subarray.

**Step 2 - Inner Loop Iterations:**

The inner loop runs from j = i + 1 to j = n - 1. In each inner loop iteration:

- We check if the current element arr[j] already exists in the set contiguous_set. If it does, we break out of the loop (this handles duplicates).
- We update min_val and max_val with the current element.
- We check if the condition max_val - min_val == j - i holds. If it does, we calculate the length of the subarray as j - i + 1. If the length is greater than the previous maximum length (ans), we update ans.

**Detailed Execution:**

**First Outer Loop Iteration (i = 0):**

- Initialize: min_val = arr[0] = 10, max_val = arr[0] = 10, contiguous_set = {10}.

**Inner Loop Iterations for i = 0:**

1. **First Inner Loop (j = 1)**:
    - arr[1] = 12
    - Add 12 to contiguous_set, update min_val = 10, max_val = 12.
    - max_val - min_val = 12 - 10 = 2, j - i = 1, so the condition max_val - min_val == j - i holds.
    - Subarray length = 1 - 0 + 1 = 2.
    - ans is updated to 2.
2. **Second Inner Loop (j = 2)**:
    - arr[2] = 11
    - Add 11 to contiguous_set, update min_val = 10, max_val = 12.
    - max_val - min_val = 12 - 10 = 2, j -

| | |
|---|---|
| | i = 2, so the condition max_val - min_val == j - i holds.<br>o Subarray length = 2 - 0 + 1 = 3.<br>o ans is updated to 3.<br><br>**Second Outer Loop Iteration (i = 1):**<br><br>• Initialize: min_val = arr[1] = 12, max_val = arr[1] = 12, contiguous_set = {12}.<br><br>**Inner Loop Iterations for i = 1:**<br><br>1. **First Inner Loop (j = 2)**:<br>o arr[2] = 11<br>o Add 11 to contiguous_set, update min_val = 11, max_val = 12.<br>o max_val - min_val = 12 - 11 = 1, j - i = 1, so the condition max_val - min_val == j - i holds.<br>o Subarray length = 2 - 1 + 1 = 2.<br>o ans remains 3.<br><br>**Step 3 - Final Output:**<br><br>• The longest valid subarray has a length of 3, so the function returns 3. |
| Output:<br>3 | |