LIS in C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm> // For std::max
using namespace std;

void LIS(const vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1); // dp[i] will store the length of LIS ending at index i
    int omax = 1; // To store the overall maximum length of LIS

    // Compute the length of the Longest Increasing Subsequence
    for (int i = 1; i < n; i++) {
        int max_len = 0;
        for (int j = 0; j < i; j++) {
            if (arr[i] > arr[j]) {
                if (dp[j] > max_len) {
                    max_len = dp[j];
                }
            }
        }
        dp[i] = max_len + 1;
        if (dp[i] > omax) {
            omax = dp[i];
        }
    }

    cout << omax << " "; // Print the length of the LIS

    // Printing the LIS length values (optional)
    for (int i = 0; i < n; i++) {
        cout << dp[i] << " ";
    }
    cout << endl;
}

int main() {
    vector<int> arr = {10, 22, 9, 33, 21, 50, 41, 60, 80, 3};

    LIS(arr);

    return 0;
}
```

Dry Run of the Program

Input:

- Array arr = {10, 22, 9, 33, 21, 50, 41, 60, 80, 3}

Initializations:

- n = arr.size() = 10
- dp is initialized to {1, 1, 1, 1, 1, 1, 1, 1, 1, 1} because each element starts with a subsequence length of 1.
- omax = 1, which stores the overall maximum length of the LIS.

Steps:

We iterate through the array and compute the LIS length for each index:

Iteration 1 (i = 1):

- arr[1] = 22
- For j = 0: arr[1] > arr[0] (22 > 10), so we check dp[0] = 1.
  - max_len = max(0, dp[0]) = 1
- dp[1] = max_len + 1 = 1 + 1 = 2
- omax = max(omax, dp[1]) = max(1, 2) = 2

Iteration 2 (i = 2):

- arr[2] = 9
- For j = 0: arr[2] > arr[0] (9 > 10) is false.
- For j = 1: arr[2] > arr[1] (9 > 22) is false.
- No update in dp[2], it remains 1.
- omax = max(omax, dp[2]) = max(2, 1) = 2

Iteration 3 (i = 3):

- arr[3] = 33
- For j = 0: arr[3] > arr[0] (33 > 10), so we check dp[0] = 1.
  - max_len = max(0, dp[0]) = 1
- For j = 1: arr[3] > arr[1] (33 > 22), so we check dp[1] = 2.
  - max_len = max(1, dp[1]) = 2
- For j = 2: arr[3] > arr[2] (33 > 9), so we check dp[2] = 1.
  - max_len = max(2, dp[2]) = 2
- dp[3] = max_len + 1 = 2 + 1 = 3
- omax = max(omax, dp[3]) = max(2, 3) = 3

Iteration 4 (i = 4):

- arr[4] = 21
- For j = 0: arr[4] > arr[0] (21 > 10), so we check dp[0] = 1.

- max_len = max(0, dp[0]) = 1
- For j = 1: arr[4] > arr[1] (21 > 22) is false.
- For j = 2: arr[4] > arr[2] (21 > 9), so we check dp[2] = 1.
  - max_len = max(1, dp[2]) = 1
- For j = 3: arr[4] > arr[3] (21 > 33) is false.
- dp[4] = max_len + 1 = 1 + 1 = 2
- omax = max(omax, dp[4]) = max(3, 2) = 3

Iteration 5 (i = 5):

- arr[5] = 50
- For j = 0: arr[5] > arr[0] (50 > 10), so we check dp[0] = 1.
  - max_len = max(0, dp[0]) = 1
- For j = 1: arr[5] > arr[1] (50 > 22), so we check dp[1] = 2.
  - max_len = max(1, dp[1]) = 2
- For j = 2: arr[5] > arr[2] (50 > 9), so we check dp[2] = 1.
  - max_len = max(2, dp[2]) = 2
- For j = 3: arr[5] > arr[3] (50 > 33), so we check dp[3] = 3.
  - max_len = max(2, dp[3]) = 3
- For j = 4: arr[5] > arr[4] (50 > 21), so we check dp[4] = 2.
  - max_len = max(3, dp[4]) = 3
- dp[5] = max_len + 1 = 3 + 1 = 4
- omax = max(omax, dp[5]) = max(3, 4) = 4

Iteration 6 (i = 6):

- arr[6] = 41
- For j = 0: arr[6] > arr[0] (41 > 10), so we check dp[0] = 1.
  - max_len = max(0, dp[0]) = 1
- For j = 1: arr[6] > arr[1] (41 > 22), so we check dp[1] = 2.
  - max_len = max(1, dp[1]) = 2
- For j = 2: arr[6] > arr[2] (41 > 9), so we check dp[2] = 1.
  - max_len = max(2, dp[2]) = 2
- For j = 3: arr[6] > arr[3] (41 > 33), so we check dp[3] = 3.
  - max_len = max(2, dp[3]) = 3
- For j = 4: arr[6] > arr[4] (41 > 21), so we check dp[4] = 2.
  - max_len = max(3, dp[4]) = 3
- For j = 5: arr[6] > arr[5] (41 > 50) is false.
- dp[6] = max_len + 1 = 3 + 1 = 4
- omax = max(omax, dp[6]) = max(4, 4) = 4

Iteration 7 (i = 7):

- arr[7] = 60
- For j = 0: arr[7] > arr[0] (60 > 10), so we check dp[0] = 1.
  - max_len = max(0, dp[0]) = 1
- For j = 1: arr[7] > arr[1] (60 > 22), so we check dp[1] = 2.

- max_len = max(1, dp[1]) = 2
- For j = 2: arr[7] > arr[2] (60 > 9), so we check dp[2] = 1.
  - max_len = max(2, dp[2]) = 2
- For j = 3: arr[7] > arr[3] (60 > 33), so we check dp[3] = 3.
  - max_len = max(2, dp[3]) = 3
- For j = 4: arr[7] > arr[4] (60 > 21), so we check dp[4] = 2.
  - max_len = max(3, dp[4]) = 3
- For j = 5: arr[7] > arr[5] (60 > 50), so we check dp[5] = 4.
  - max_len = max(3, dp[5]) = 4
- For j = 6: arr[7] > arr[6] (60 > 41), so we check dp[6] = 4.
  - max_len = max(4, dp[6]) = 4
- dp[7] = max_len + 1 = 4 + 1 = 5
- omax = max(omax, dp[7]) = max(4, 5) = 5

Further iterations will follow the same process, updating dp[i] and omax as needed.

Finally, after processing all elements, the length of the longest increasing subsequence will be omax = 6.

Output:-
6
1 2 1 2 4 4 5 6 1