Print all LIS In C++

```
#include <iostream>
#include <vector>
#include <deque>
using namespace std;

struct Pair {
    int l;    // length of the LIS
    int i;    // index in the array
    int v;    // value at index i in the array
    string psf; // path so far

    Pair(int l, int i, int v, string psf) {
        this->l = l;
        this->i = i;
        this->v = v;
        this->psf = psf;
    }
};

void printAllLIS(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1); // dp array to store the length of LIS ending at each index
    int omax = 0; // maximum length of LIS found
    int omi = 0; // index where the LIS with maximum length ends

    // Finding the length of LIS ending at each index
    for (int i = 0; i < n; i++) {
        int maxLen = 0;
        for (int j = 0; j < i; j++) {
            if (arr[i] > arr[j]) {
                if (dp[j] > maxLen) {
                    maxLen = dp[j];
                }
            }
        }
        dp[i] = maxLen + 1;

        if (dp[i] > omax) {
            omax = dp[i];
            omi = i;
        }
    }

    deque<Pair> q;
    q.push_back(Pair(omax, omi, arr[omi], to_string(arr[omi])));

    while (!q.empty()) {
```

Dry Run with Input Array {10, 22, 9, 33, 21, 50, 41, 60, 80, 3}

1. **Step 1: Calculate the LIS Lengths** (dp array):
   - We start with dp[i] = 1 for all i (since a single element is trivially a subsequence of length 1).
   - Iterating through each i and for each i, checking all previous j to update dp[i]:

   For each index:

   - **Index 0 (value 10)**: dp[0] = 1 (no previous elements).
   - **Index 1 (value 22)**: dp[1] = max(dp[0] + 1) = 2.
   - **Index 2 (value 9)**: dp[2] = 1 (no elements before it are smaller).
   - **Index 3 (value 33)**: dp[3] = max(dp[0] + 1, dp[1] + 1) = 3.
   - **Index 4 (value 21)**: dp[4] = max(dp[0] + 1) = 2.
   - **Index 5 (value 50)**: dp[5] = max(dp[0] + 1, dp[1] + 1, dp[3] + 1) = 4.
   - **Index 6 (value 41)**: dp[6] = max(dp[0] + 1, dp[1] + 1, dp[3] + 1) = 4.
   - **Index 7 (value 60)**: dp[7] = max(dp[0] + 1, dp[1] + 1, dp[3] + 1, dp[5] + 1) = 5.
   - **Index 8 (value 80)**: dp[8] = max(dp[0] + 1, dp[1] + 1, dp[3] + 1, dp[5] + 1, dp[7] + 1) = 6.
   - **Index 9 (value 3)**: dp[9] = 1.

   The dp array will look like this after processing:

   dp = {1, 2, 1, 3, 2, 4, 4, 5, 6, 1}

2. **Step 2: Find the Maximum LIS Length**:
   - The maximum LIS length omax = 6 and the index where it ends omi = 8 (corresponding to value 80).
3. **Step 3: Backtrack to Find All LIS**:
   - A deque q is initialized with the Pair containing the maximum LIS.
   - The initial Pair object in the deque:

     q = {Pair(6, 8, 80, "80")}

```
        Pair rem = q.front();
        q.pop_front();

        if (rem.l == 1) {
            cout << rem.psf << endl; // print the
path when the length of LIS is 1
        } else {
            for (int j = rem.i - 1; j >= 0; j--) {
                if (dp[j] == rem.l - 1 && arr[j] <=
rem.v) {
                    q.push_back(Pair(dp[j], j,
arr[j], to_string(arr[j]) + " -> " + rem.psf));
                }
            }
        }
    }
}

int main() {
    vector<int> arr = {10, 22, 9, 33, 21, 50,
41, 60, 80, 3};
    printAllLIS(arr);
    return 0;
}
```

- Now, backtrack and find all possible subsequences:
  - For Pair(6, 8, 80, "80"), we look for elements before index 8 that can form a LIS of length 5. We find:
    - dp[7] == 5 and arr[7] = 60 <= 80, so we push Pair(5, 7, 60, "60 -> 80").
    - Similarly, we continue for other indices, building the subsequences.

After backtracking, we find two possible LIS:

- 10 -> 22 -> 33 -> 41 -> 60 -> 80
- 10 -> 22 -> 33 -> 50 -> 60 -> 80

4. **Step 4: Output the Results**: The output is:

10 -> 22 -> 33 -> 41 -> 60 -> 80
10 -> 22 -> 33 -> 50 -> 60 -> 80

Output:-
10 -> 22 -> 33 -> 41 -> 60 -> 80
10 -> 22 -> 33 -> 50 -> 60 -> 80