# Assignment 11

Done By: Nayan Man Singh Pradhan

---

## Problem 11.1

**Longest ordered subarray**

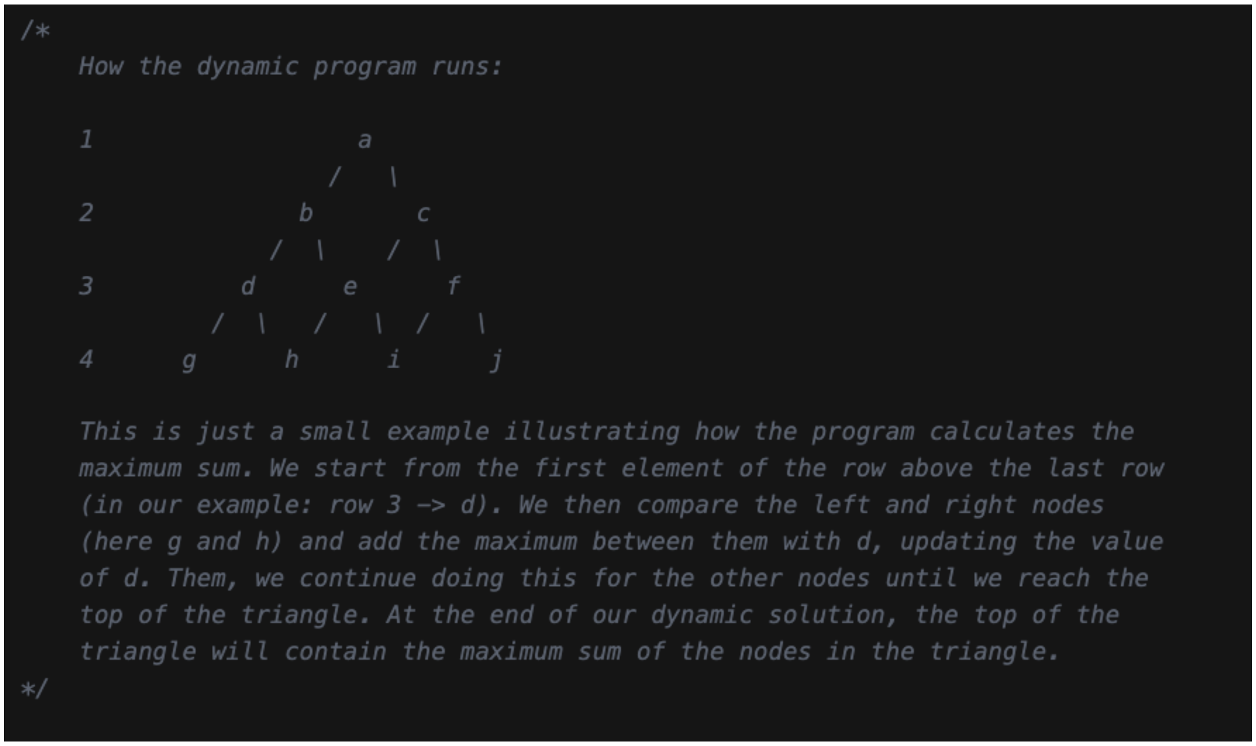
A *.cpp* file named “LOS.cpp” is created. Execute make to run.

## Problem 11.2

**Sum in triangles**

**a.**

A *.cpp* file named “triangle.cpp” is created. Execute make to run. I have added a screenshot from the *.cpp* file that explains how the dynamic program runs.



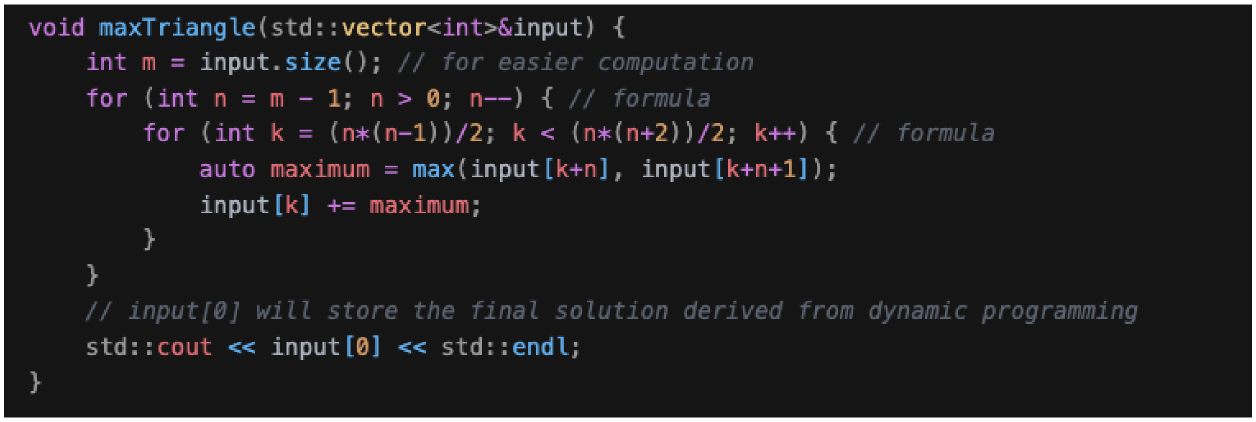
```
/*
    How the dynamic program runs:

    1                  a
                     /   \
    2              b       c
                 /  \    /  \
    3          d      e      f
             /  \   /   \  /   \
    4      g      h      i      j

    This is just a small example illustrating how the program calculates the
    maximum sum. We start from the first element of the row above the last row
    (in our example: row 3 -> d). We then compare the left and right nodes
    (here g and h) and add the maximum between them with d, updating the value
    of d. Them, we continue doing this for the other nodes until we reach the
    top of the triangle. At the end of our dynamic solution, the top of the
    triangle will contain the maximum sum of the nodes in the triangle.
*/
```

b.

My program:



```
void maxTriangle(std::vector<int>&input) {
    int m = input.size(); // for easier computation
    for (int n = m - 1; n > 0; n--) { // formula
        for (int k = (n*(n-1))/2; k < (n*(n+2))/2; k++) { // formula
            auto maximum = max(input[k+n], input[k+n+1]);
            input[k] += maximum;
        }
    }
    // input[0] will store the final solution derived from dynamic programming
    std::cout << input[0] << std::endl;
}
```

I use two *"for"* loops in my program that iterates through all the nodes/elements (n) of the triangle. **Therefore, the runtime of my solution will be approximately $\approx \boldsymbol{O(n^2)}$.**

The brute force approach will create different arrays for all possible paths and pick the path that gives the largest sum. While this approach may be comparable with the dynamic programming approach for very small values of n, as n -> 10000 the number of arrays will increase drastically as every node (except the nodes in the bottom row) will have two solutions, causing the runtime to increase drastically too. The runtime of the brute force solution $= 2^{k-1}$, where k is the number of rows. **Therefore, the runtime of the brute force solution is approximately $\approx \boldsymbol{O(2^k)}$, where k is the number of rows.**

c.

The greedy algorithm does not work for this problem as the greedy algorithm always picks the locally optimal solution. The greedy problem would start off by picking the largest number in the triangle from the top giving 7, 8, 1, 7, and finally 5, which equals to 28 (which is not the largest sum). The dynamic programming approach solves all the sub-problems and pricks the globally optimal solution, which makes it suitable for this problem, while the greedy algorithm picks the locally optimal solution, which might not lead to the correct solution, making the greedy unsuitable for this problem.