DP → pascal's triangle

learning curve

logic:

i \ j	0	1	2	3	4
0	1	1	1	1	1
1	/	1	2	3	4
2	/	/	1	3	6
3	/	/	/	1	4
4	/	/	/	/	1

Have 2 for-loops with two variables i and j.

① If $i = j$ then we are on the diagonal side of the dynamic programming where $dp[i][j] = 1$

② If $i = 0$ then $dp[i][j]$ where $i = 0$ then $dp[0][j] = 1$

③ when $0 < i < j$ then $dp[i][j] = dp[i][j-1] + dp[i-1][j-1]$

Relation Recurrence =

$$T(n) = \begin{cases} dp[i][j] = 1, & i == j \\ dp[i][j] = 1, & i == 0 \\ dp[i][j] = dp[i][j-1] + dp[i-1][j-1] & \end{cases}$$

after coding

j \ i	0	1	2	3	4
0	1	X	X	X	X
1	1	1	X	X	X
2	1	2	1	X	X
3	1	3	3	1	X
4	1	4	6	4	1

The first for loop will go from 0 to the numRows (the number that was provided by the user). The variable i would loop through 0 to numRows. The variable j would loop through 0 to i. This will ensure that the shape of the table is a triangle. The recurrence relation would be the same as above.

My time complexity would be about $O(n^2)$ since there are two for-loops that are occuring.

I also changed the orientation of my table.

`List<List<Integer>> ArrayResult = new ArrayList<>();` would store the entire triangle.

At each first for loop, a new ArrayList of row will be created. They will then be added to the massive ArrayResult after.