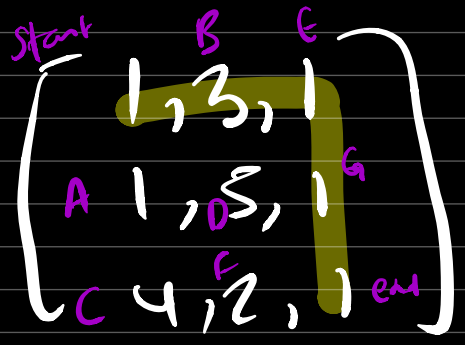
# Minimum Path Sum

Bruteforce ($2^n$)

mxn grid: find path from Top-left → Bottom Right




1 + 3 + 1 + 1 + 1 = 7

pathSum( start, end )

$1 + \min($ pathSum(A, end) , pathSum( B, end ) $)$

$1 + \min(ps(C, end), ps(D, end))$

...

baseCase →

pathSum( end, end ) = grid[end]

* Can improve with memorization

* Out-of-bounds, return INT_MAX;

↑ realized returning '0' for Out-of-bounds is incorrect.

```cpp
#include <climits>

class Solution {
public:

    int pathSum(vector<vector<int>>& grid, int row, int col, int maxRow, int maxCol) {
        //bounds check
        if (row >= maxRow || col >= maxCol) return INT_MAX;

        //base case reach end
        if (row == maxRow-1 && col == maxCol-1) {
            return grid[row][col];
        }
        int cur = grid[row][col];
        //recursive case
        return  cur + min(pathSum(grid, row+1, col, maxRow, maxCol), pathSum(grid, row, col+1, maxRow, maxCol));

    }



    int minPathSum(vector<vector<int>>& grid) {
        if (grid.empty() || grid[0].empty()) return 0;

        return pathSum(grid, 0, 0, grid.size(), grid[0].size());
    }
};
```

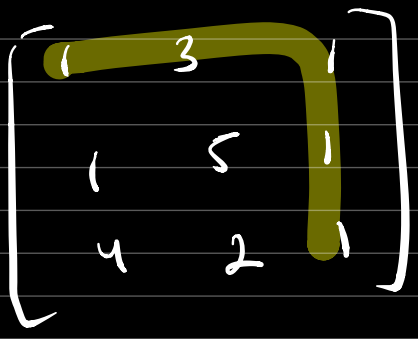
# Approach #2 → Memo-table (top-down DP)

```cpp
#include <climits>

class Solution {
public:

    int pathSum(vector<vector<int>>& grid, int row, int col, int maxRow, int maxCol, vector<vector<int>>& memo) {
        //bounds check
        if (row >= maxRow || col >= maxCol) return INT_MAX;

        //base case reach end
        if (row == maxRow-1 && col == maxCol-1) {
            return grid[row][col];
        }
        //if not in memo table, compute and insert into memo table before returning
        int cur = grid[row][col];
        if (memo[row][col] == 0) {
            memo[row][col] = cur + min(pathSum(grid, row+1, col, maxRow, maxCol, memo),
                                       pathSum(grid, row, col+1, maxRow, maxCol, memo));
        }
        //recursive case
        return  memo[row][col];

    }



    int minPathSum(vector<vector<int>>& grid) {
        if (grid.empty() || grid[0].empty()) return 0;
        vector<vector<int>> memo(grid.size(),vector<int>(grid[0].size(), 0));
        return pathSum(grid, 0, 0, grid.size(), grid[0].size(), memo);
    }
};
```

case Run Code Result Debugger

# Approach #3 → DP (Bottom-up)

1+3+1+1+1 = 7



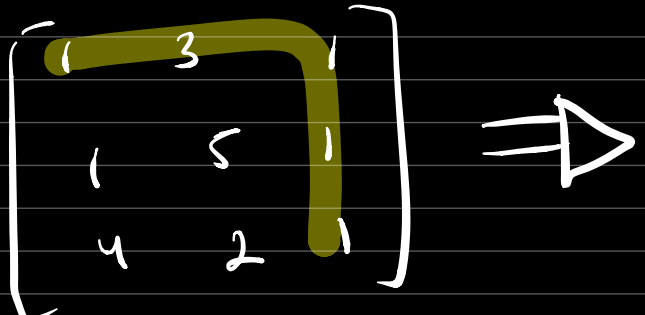

Start bottom right: output = grid(bottom right)

7 6 3
8 7 2
7 3 1

take min of right, down & add current square.

Also, can from top-left to bottom-right or same from bottom-right to top-left

DP table

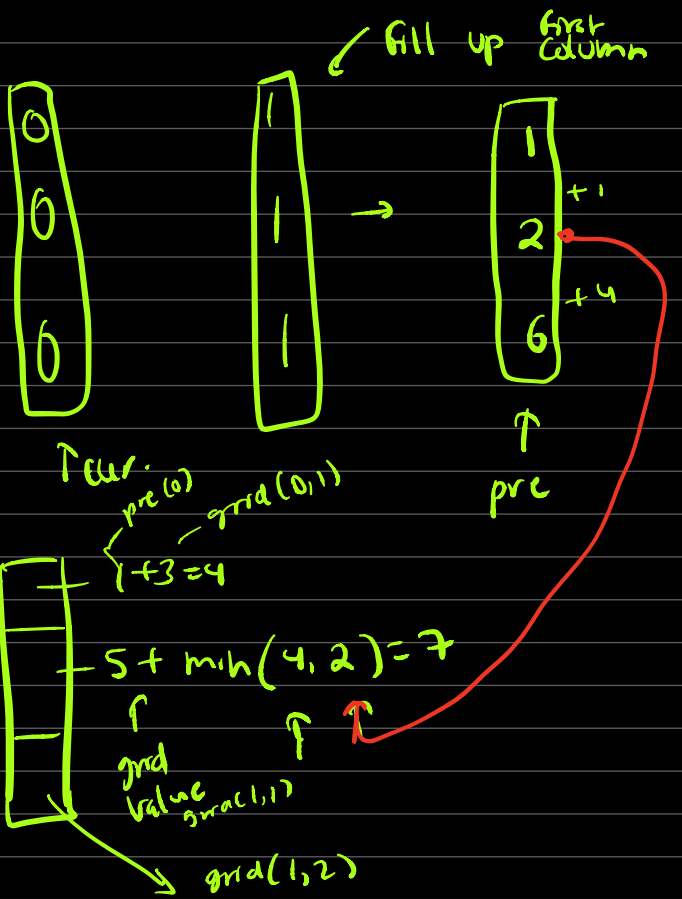


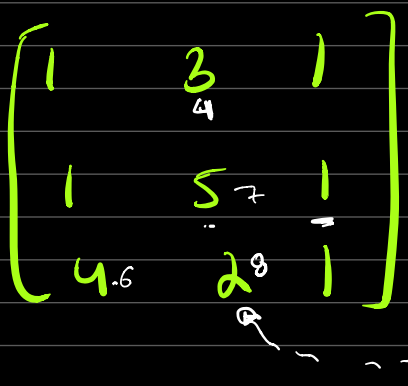

$$\begin{bmatrix} 1 & 4 & 5 \\ 2 & 7 & 6 \\ 6 & 8 & \underline{7} \end{bmatrix}$$

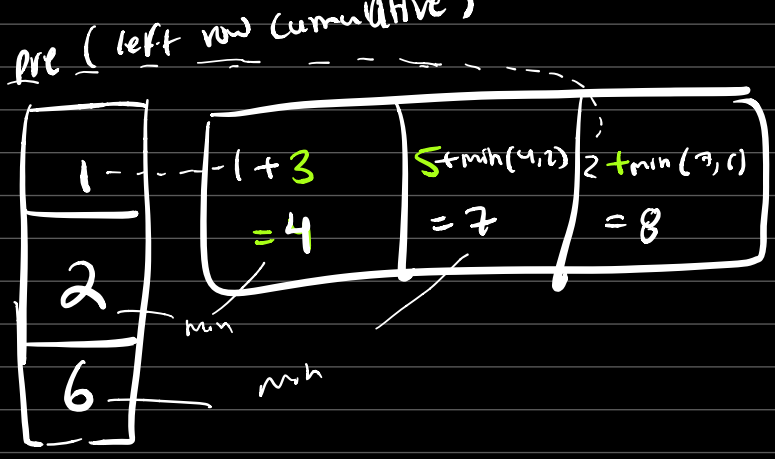
↑ ans.

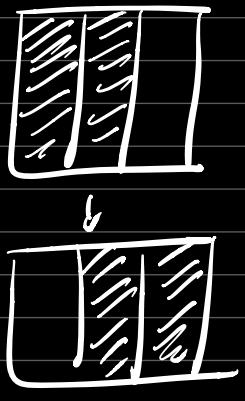
2-column approach

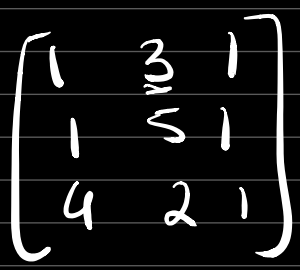

m
m x n
n

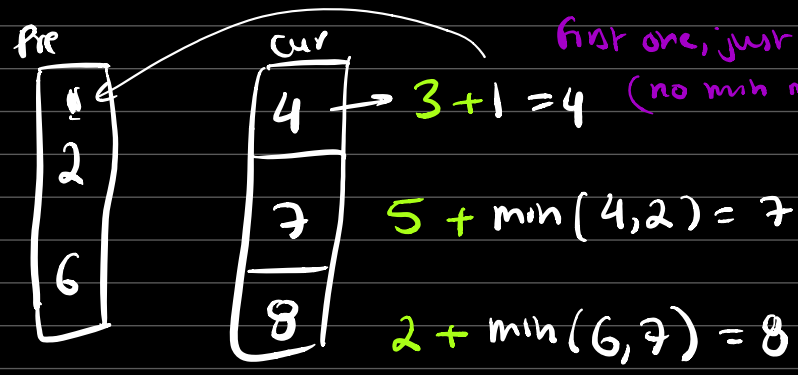

Fill up first column
0
0
0
1
1
1
1
2
6
+1
+4
↑ cur
↑ pre
pre(0)
grid(0,1)
cur
1+3=4
5+min(4,2)=7
grid value
grid(1,1)
grid(1,2)


1 3 1
1 5 1
4.6 2.8 1
4
7
8


pre (left row cumulative)
1
2
6
1+3 = 4
5+min(4,2) = 7
2+min(7,1) = 8
min
min








then pre = cur.

cur.clear()