Problem statement:

> You are given a triangular array/list 'TRIANGLE'. Your task is to return the minimum path sum to reach from the top to the bottom row.
> The triangle array will have N rows and the i-th row, where 0 ⩽ i < N will have i + 1 elements.
> You can move only to the adjacent number of row below each step. For example, if you are at index j in row i, then you can move to i or i + 1 index in row j + 1 in each step.

Example:

```
If the array given is 'TRIANGLE' = [[1], [2,3], [3,6,7], [8,9,6,1]]
the triangle array will look like:

1
2,3
3,6,7
8,9,6,10

For the given triangle array the minimum sum path would be 1->2->3->8.
Hence the answer would be 14.
```

Approach:

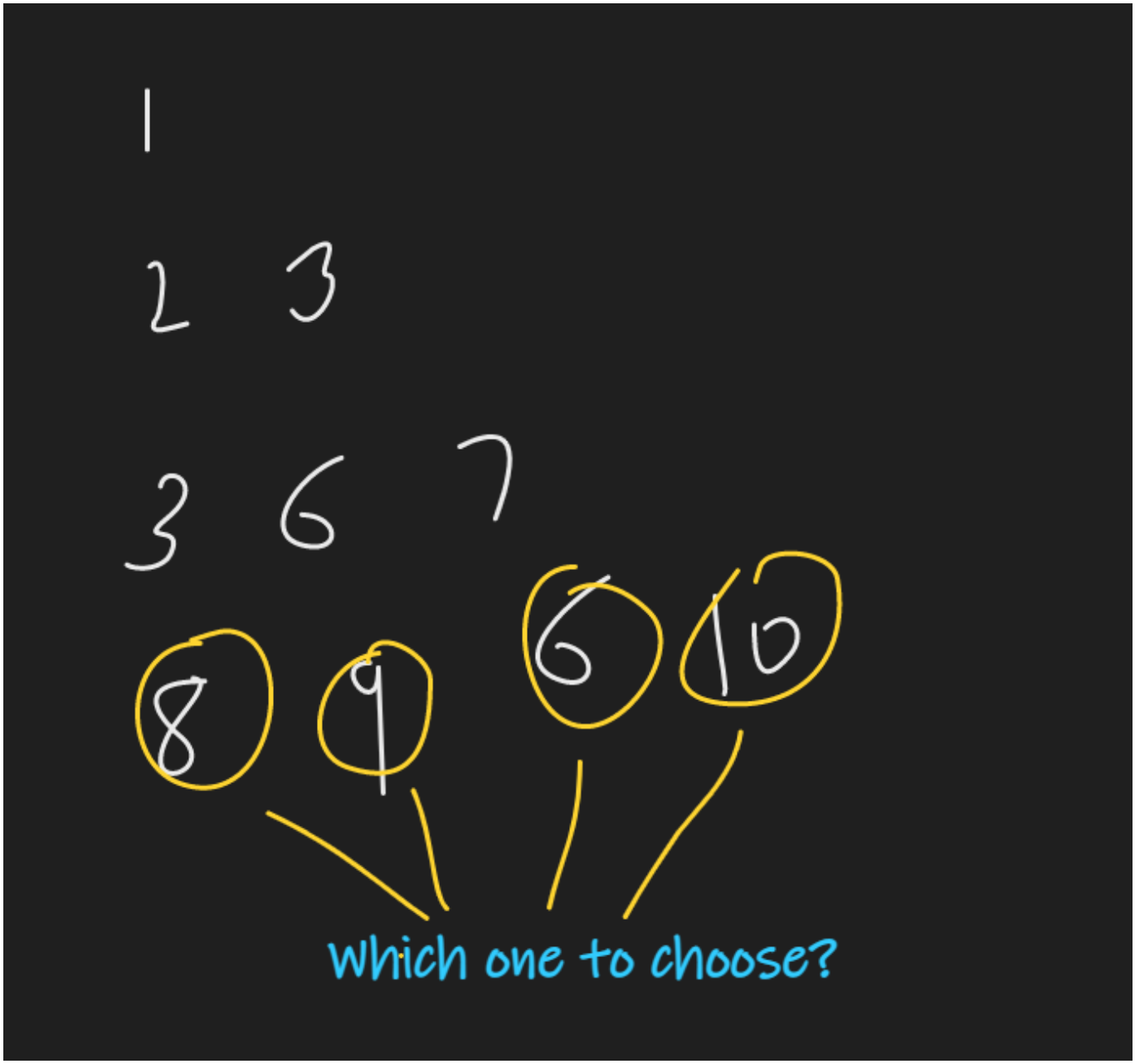
⟶ So here we can go in only 2 paths: 1)Down 2)Diagonal

So we will find all possible paths and take their minimum sum. And all possibilities means we have to do recursion here.

But can we start from (n-1,m-1)?

⟶ we can't start because in the matrix questions we knew that we have to reacth the end of the matrix and the destination was

fixed. but here we don't have fixed destination which we can see in diagram:



⟶ So that's why we will start from `(0,0)` and go till n-1 th index.

Base case:

⟶ Now there will be a base case when we reach at the last index. So here we will just return the element because we can't go further so we will take the current element in sum and return.

```
if(i == n-1){
        return arr[i][j];
}
```

Recursive code:

```
int solve(int i,int j,vector<int>&arr){
        if(i == n-1){
                return arr[i][j];
        }
        int down = arr[i][j] + solve(i+1,j,arr);
        int digonal = arr[i][j] + solve(i+1,j+1,arr);
        return min(down,diagonal);
}
int main(){
        cout<<solve(0,0,arr)<<endl;
}
```

memoized code

⟶ The max value for `i and j` will be n because we know that there are n rows and last row will have n elements so we have to take the matrix of nxn

```
int solve(int i,int j,vector<int>&arr,vector<vector<int>>&dp){
        if(i == n-1){
                return arr[i][j];
        }
        if(dp[i][j] != -1){
                return dp[i][j];
        }
        int down = arr[i][j] + solve(i+1,j,arr);
        int digonal = arr[i][j] + solve(i+1,j+1,arr);
        return dp[i][j] = min(down,diagonal);
}
```

```
int main(){
        vector<vector<int>>dp(n,vector<int>(n,-1));
        cout<<solve(0,0,arr,dp)<<endl;
}
```

Tabluation Code

⟶ Now we have written memoization code so we can convert it into tabulation code. But here we have started our recursion from (0,0) so our tabulation code will be opposite of it so it means we will start from `n-1`th row.

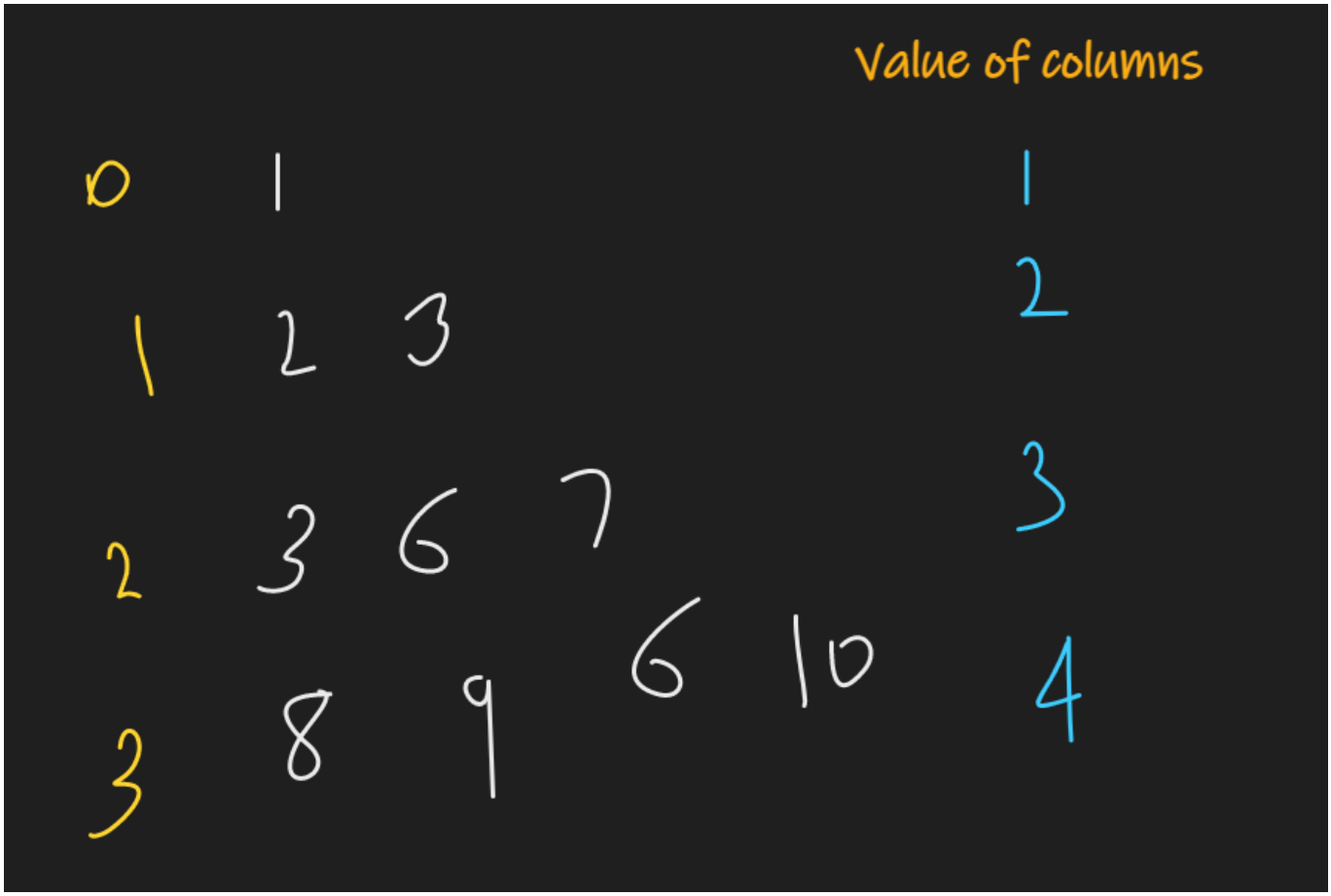
But here we have to consider every element at `n-1`th row. so we will add them in our dp array like this:

```
for(int j=0;j<n;j++){
        dp[n-1][j] = arr[n-1][j];
}
```

⟶ Now we will run our loop. So i will go from n-2 to 0

What will be value for j?

⟶ So here we can see that for each i the value of column is `i+1` so we just have to run j loop i+1 times from `i to 0`




```
// we are starting from n-2 because we have considered n-1th row
in base case
for(int i=n-2;i>=0;i--){
        for(int j=i;j>=0;j--){
                int down = arr[i][j] + dp[i+1][j];
                int diagonal = arr[i][j] + dp[i+1][j+1];
                dp[i][j] = min(down,diagonal);
        }
}
return dp[0][0];
```

Tabulation code

```
int solve(vector<int>&arr){
        vector<vector<int>>dp(n,vector<int>(n,0));
        for(int j=0;j<n;j++){
                dp[n-1][j] = arr[n-1][j];
```

```
        }
        for(int i=n-2;i>=0;i--){
                for(int j=i;j>=0;j--){
                        int down = arr[i][j] + dp[i+1][j];
                        int diagonal = arr[i][j] + dp[i+1][j+1];
                        dp[i][j] = min(down,diagonal);
                }
        }
        return dp[0][0];
}
int  main(){
        cout<<solve(arr)<<endl;
}
```

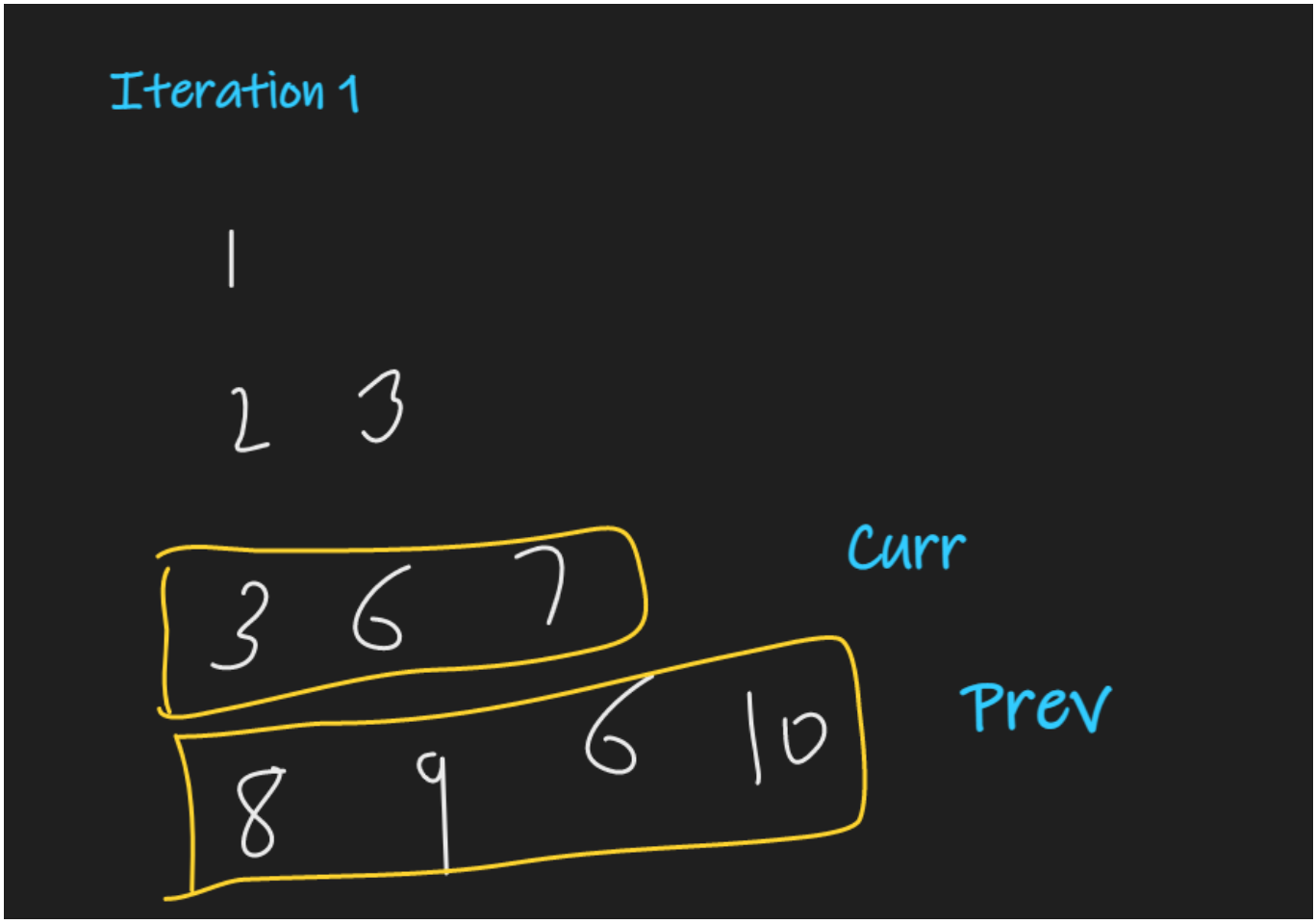
Complexity:

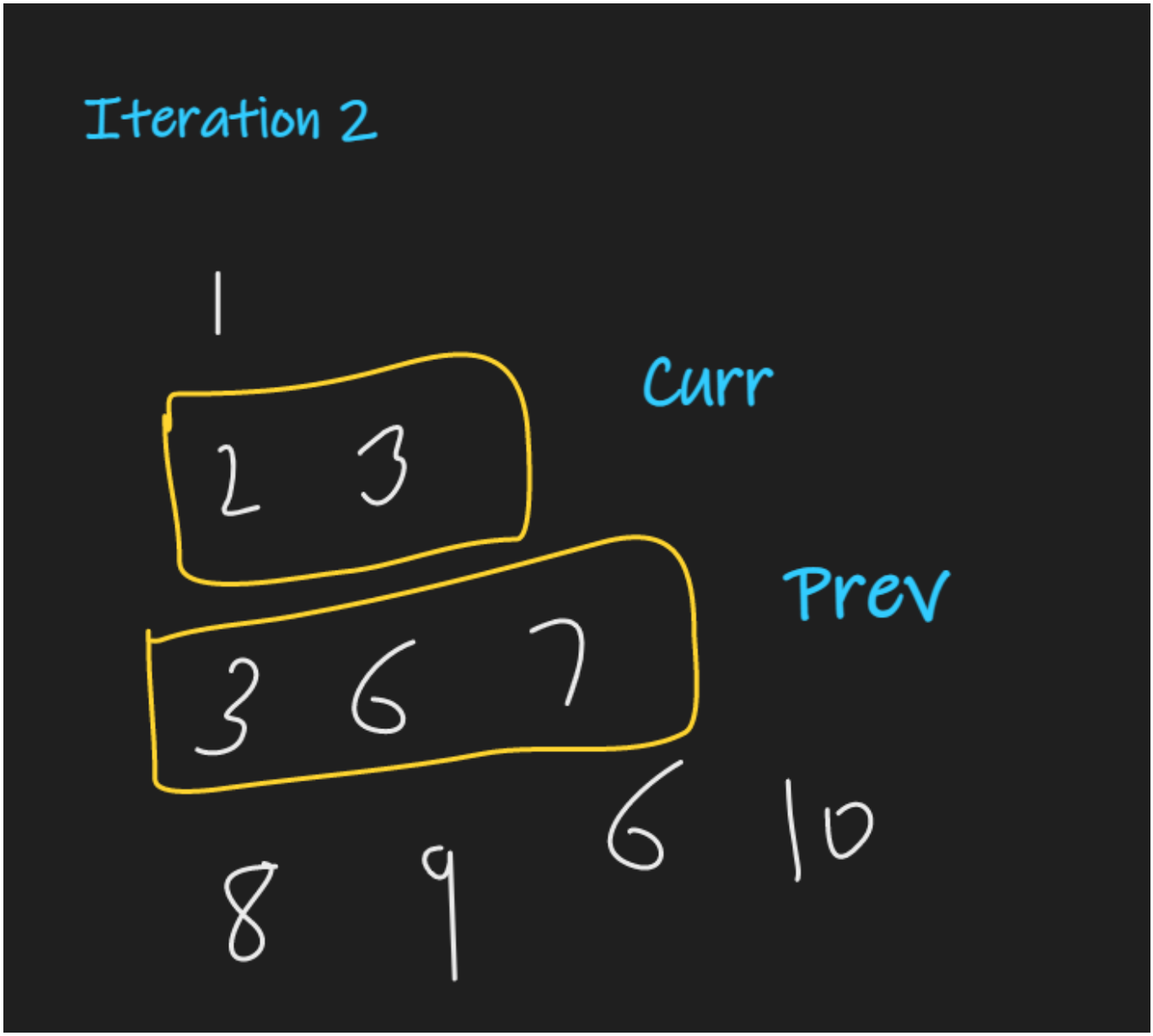
```
Time complexity: O(NxN)
Space complexity: O(NxN) // dp matrix space
```

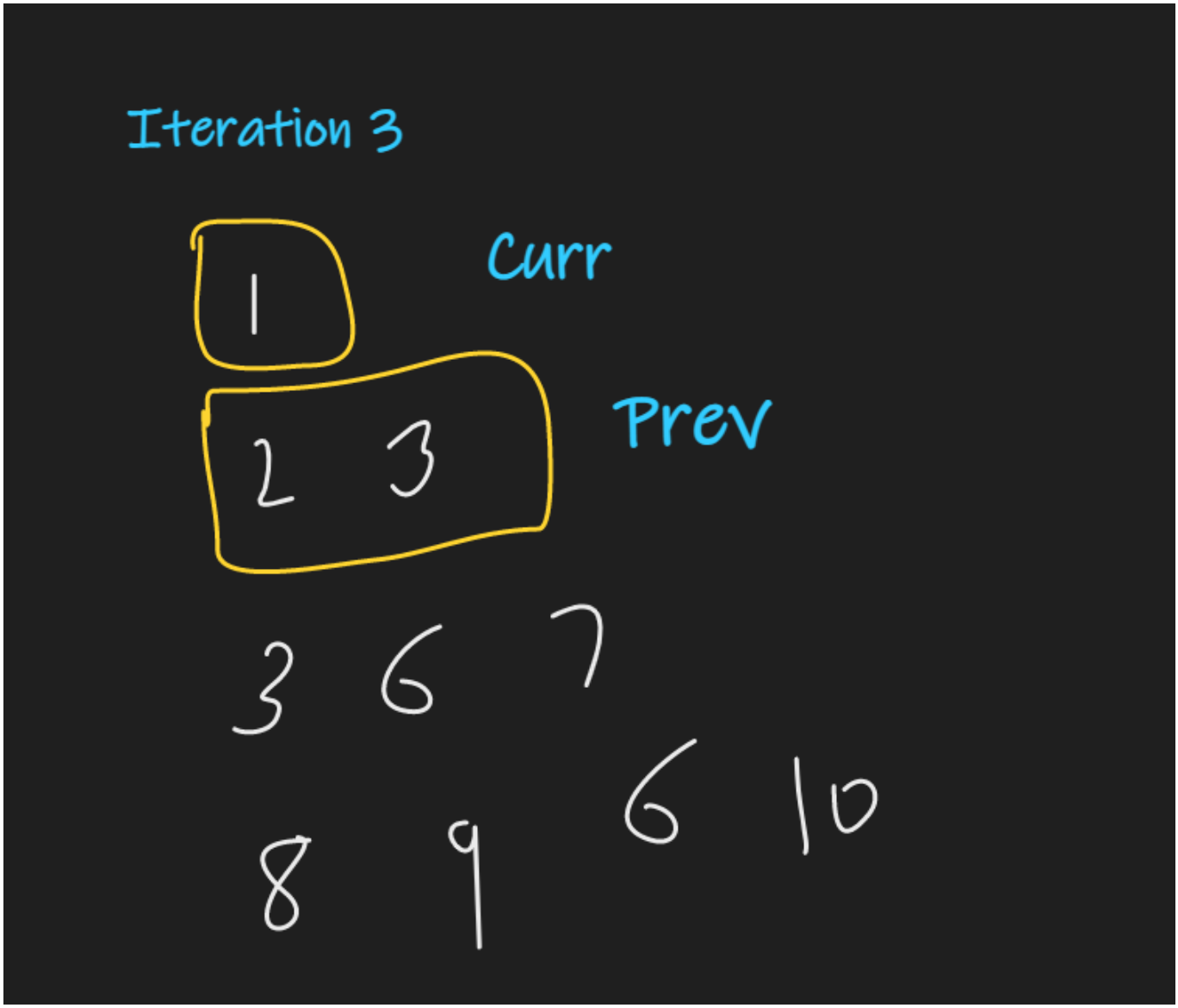
Space optimisation

⟶ Now here we can see that we just require the previous row to calculate the value for current row. so we just can take only 2 1D arrays called prev and curr to calculate the ans instead of taking whole triangle in DP array.

we can see the iterations in below diagram:


Iteration 1
1
2 3
3 6 7
Curr
8 9 6 10
Prev


Iteration 2
1
2 3
Curr
3 6 7
Prev
8 9 6 10



Code

```
int solve(vector<int>&arr){
        vector<int>prev(n);
        for(int j=0;j<n;j++){
                prev[j] = arr[n-1][j];
        }
        for(int i=n-2;i>=0;i--){
                vector<int>curr(n);
                for(int j=i;j>=0;j--){
                        int down = arr[i][j] + prev[j];
                        int diagonal = arr[i][j] + prev[j+1];
                        curr[j] = min(down,diagonal);
                }
```

```
                prev = curr;
        }
        return prev[0];
}
int main(){
        cout<<solve(arr)<<endl;
}
```

Complexity:

```
Time complexity: O(NxN)
Space complexity: O(N)
```