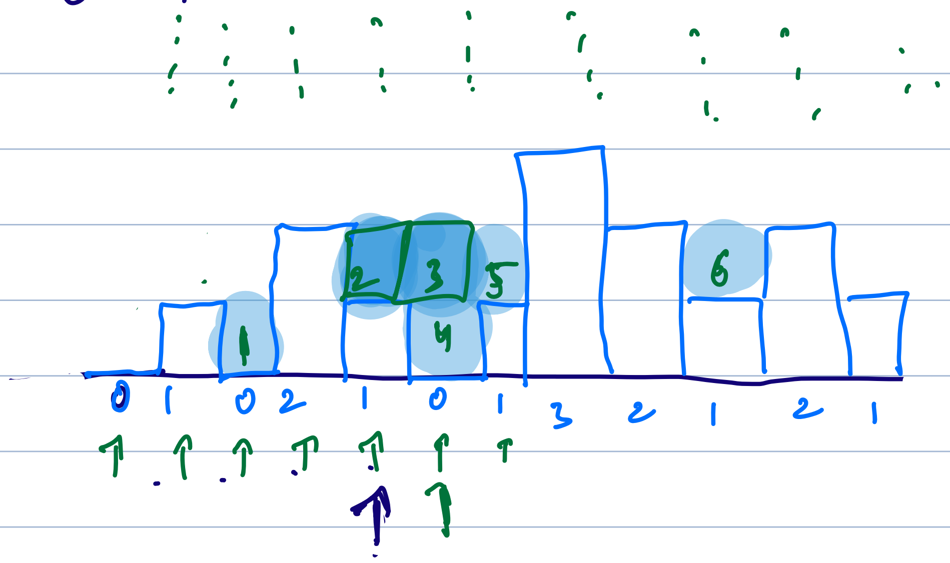
Question: Rain Water Trapping

Given an array of size N representing heights of buildings, compute how much water is trapped after it rains

A : 0 1 0 2 1 0 1 3 2 1 2 1

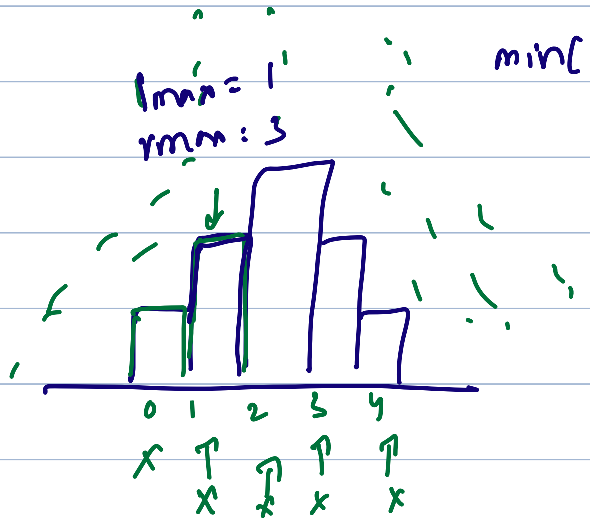


= 6 units

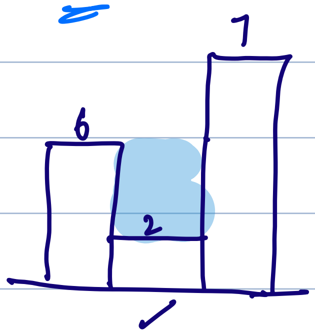
W_i : min(left_max, right_max) - building

min(1, 3) - 2) : -1

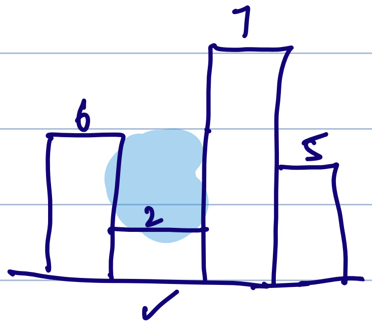
A =



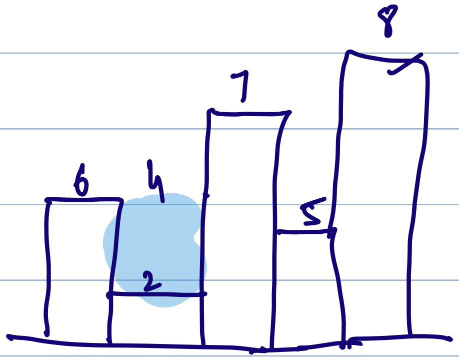
Observations



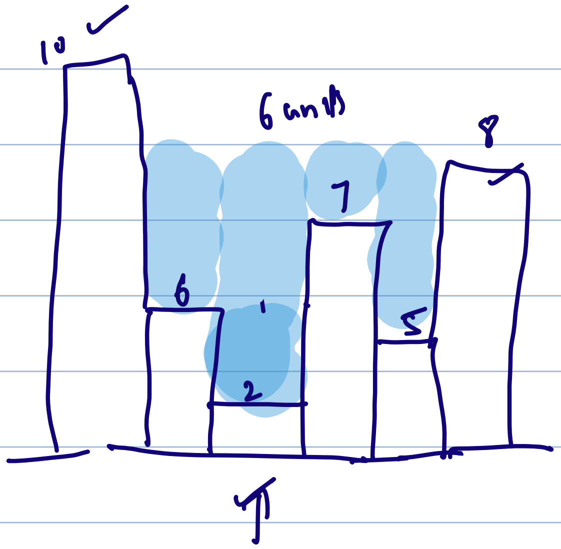
=> min(7,6) -2 = 4



=> 4



=> 4 units



min(10,8) -2

= 8 - 2 = 6

water[i] = min(leftMax, rightMax) - height

Brute Force

```
int totalWater = 0;
for (i=0; i<N; i++){
        int lmax =      max [0, i-1)        //  O(N)
        int rmax =      max (i+1, N-1)      //  O(N)

        water =     min( lmax, rmax)  - A[i];

        if(water > 0) {
            totalwater +=   water;
        }
}
```

N

T.C: $O(N^2)$

S.C: $O(1)$

Optimized Approach

d

	0	1	2	2	4	5	6	7	8	9	10
A :	4	2	5	7	4	2	3	6	8	2	3
lmax :	0	4	4	5	7	7	7	7	7	8	8
rmax :	8	8	8	8	8	8	8	8	3	3	0

lmax[i] = Max (0.....i-1)

lmax[i] = max[lmax[i-1] , A[i-1])

```
     lmax[0] = 0;
     for(i=1; i<N; i++){
O(N)     lmax[i] = max( lmax[i-1], A[i-1] );
     }
      T.C: O(N)
```

```
     rMax[N-1] = 0;
O(N) for(i=N-2; i>=0; i--){
         rmax[i] = max( rmax[i+1], A[i+1]);
     }
```

```
     int totalWater = 0;  ⇙                 left → right
     for(i=0; i<N; i++){
O(N)       water = min( lmax[i], rmax[i] ) - A[i];
           if(water >0){
               totalwater += water;
           }
     }
```

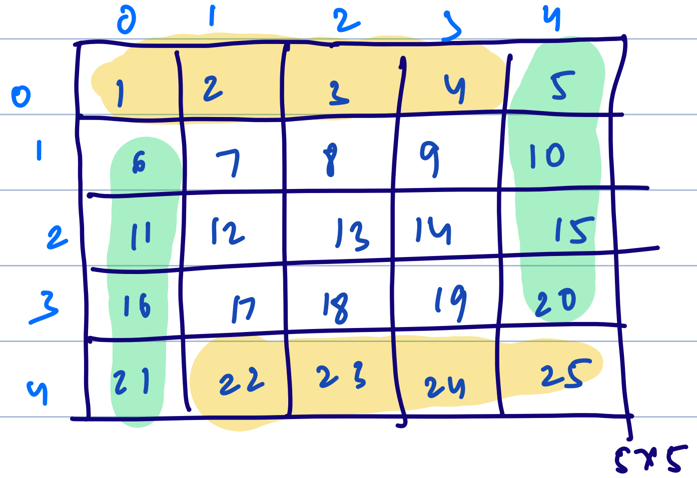
T.C: O(N)

S.C: O(N)

↳ 2 arrays

optimization: We can only use 1 array by carrying forward lmax.

Question: Given a square matrix N×N, print boundary elements in clockwise



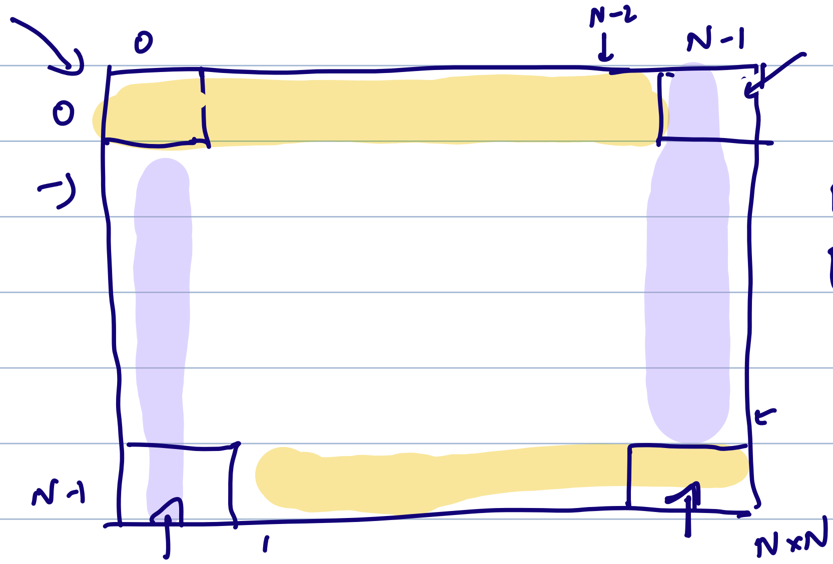
	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

5×5

1 2 3 4 5 10 15 20
25 24 23 22 21
16 11 6

Boundaries:

1) 0th row
2) (N-1)th col
3) (N-1)th row in reverse
4) 0th col in reverse



[0,0] → [0, N-2] : N-1

[0,N-1] → [N-2,N-1] : N-1

[N-1,N-1] → [N-1,1] : N-1

[N-1,0] → [1,0] : N-1

```
print Boundaries ( mat[][]){
        int row = 0, col = 0 ;

        for(k=0; k<N-1; k++){
                print( A[row][col] ;
                col++;
        }
        // row = 0       col = N-1
        for(k=0; k<N-1; k++){
                print( A[row][col]);
                row++;
        }
        // row=N-1 ,   col = N-1
        for (k=0; k<N-1; k++){
                print( A[row][col]);
                col--;
        }
        // row = N-1  col = 0

        foo(k=0; k<N-1; k++){
                print(A[row][col]);
                row--;
        }
        // row = 0 , col = 0
}
```

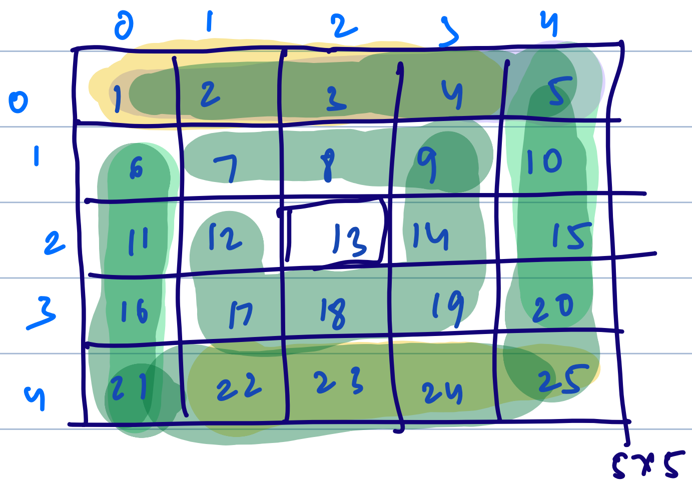
Question: Lawn Mowing Challenge

Scenerio

You need to program an automated grass-mowing robot for "**GreenTech Robotics**" to navigate a square lawn (**N x N**) represented as a grid. The lawn is designed as a **square grid**, where each cell in the grid shows the grass height in that area...

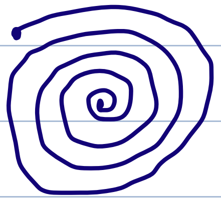
Problem Statement

Your challenge is to find out the heights of grass patches that the robot needs to cut (in the order they are cut), with the robot's path following a **spiral pattern** from the **outer** edge towards the **center** of the lawn.

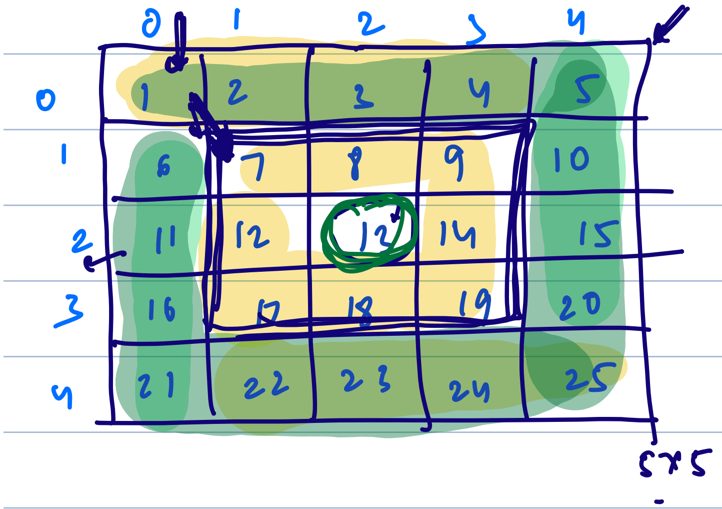


	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

5x5



1 2 3 4 5 10 15 20 25 24 23 22
21 16 11 6 7 8 9 14 19 18 17 12
13



	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	12	14	15
3	16	17	18	19	20
4	21	22	23	24	25

5x5

Iter 1:	(0,0)	N = 5 x 5
Iter 2:	(1,1)	N = 3
Iter 3:	(2,2)	N = 1

```
spiral Order ( mat[][] ){

        int row = 0, col = 0 ;
    while( N > 1 ){
        for(k=0; k<N-1; k++){
            print( A[row][col] ;
            col++;
        }
        // row = 0      col = N-1
        for(k=0; k<N-1; k++){
            print( A[row][col]);
            row++;
        }
        // row=N-1 ,   col = N-1
        foo (k=0; k<N-1; k++){
            print( A[row][col]);
            col--;
        }

        foo(k=0; k<N-1; k++){
            print(A[row][col]);
            row--;
        }

        row ++;   col++;
        N = N-2;
    }
}   // Edge case
if( N == 1)    print( A[row][col] );
```

T·C: $O(N^2)$ $\boxed{8:17}$

S·C: $O(1)$

Permutations:

=) [1 2 3] [3 2 1] [1 3 2] [2 1 3]

[2 3 1] [3 1 2]

=) [1 2 3 4] → [1 2 4 3] → [1 3 2 4] →

[1 3 4 2] → [1 4 2 3] → [1 4 3 2]

→ [2 1 3 4] → [2 1 4 3] → [2 3 1 4]

→ [2 3 4 1] → [2 4 1 3] → . . . .

Problem Statement

Implement the next permutation, which rearranges numbers into the numerically next greater permutation of numbers for a given array A of size N.

If such arrangement is not possible, it must be rearranged as the lowest possible order, i.e., sorted in ascending order.

NOTE:

The replacement must be in-place, do not allocate extra memory.
DO NOT USE LIBRARY FUNCTION

[1 2 3] → [1 3 2]

[3 2 1] → [1 2 3]

[1 2 6 3 4 5] → 1 2 6 3 5 4

1 2 4 3 5 6 → 1 2 4 3 6 5

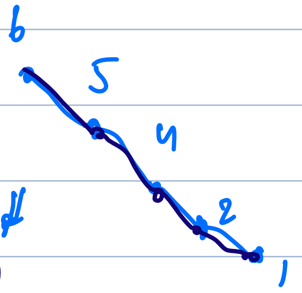
⇓

Ex: $7 \quad 8 \quad 4 \quad \overset{1}{\cancel{6}} \quad \overset{2}{\cancel{5}} \quad \overset{3}{\cancel{3}} \quad \overset{5}{\cancel{2}} \quad \overset{6}{\cancel{1}}$

0 1 2 3 4 5 6 7

3→4

7 8 3 6 5 4 2 1

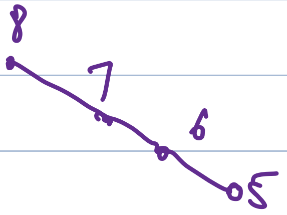


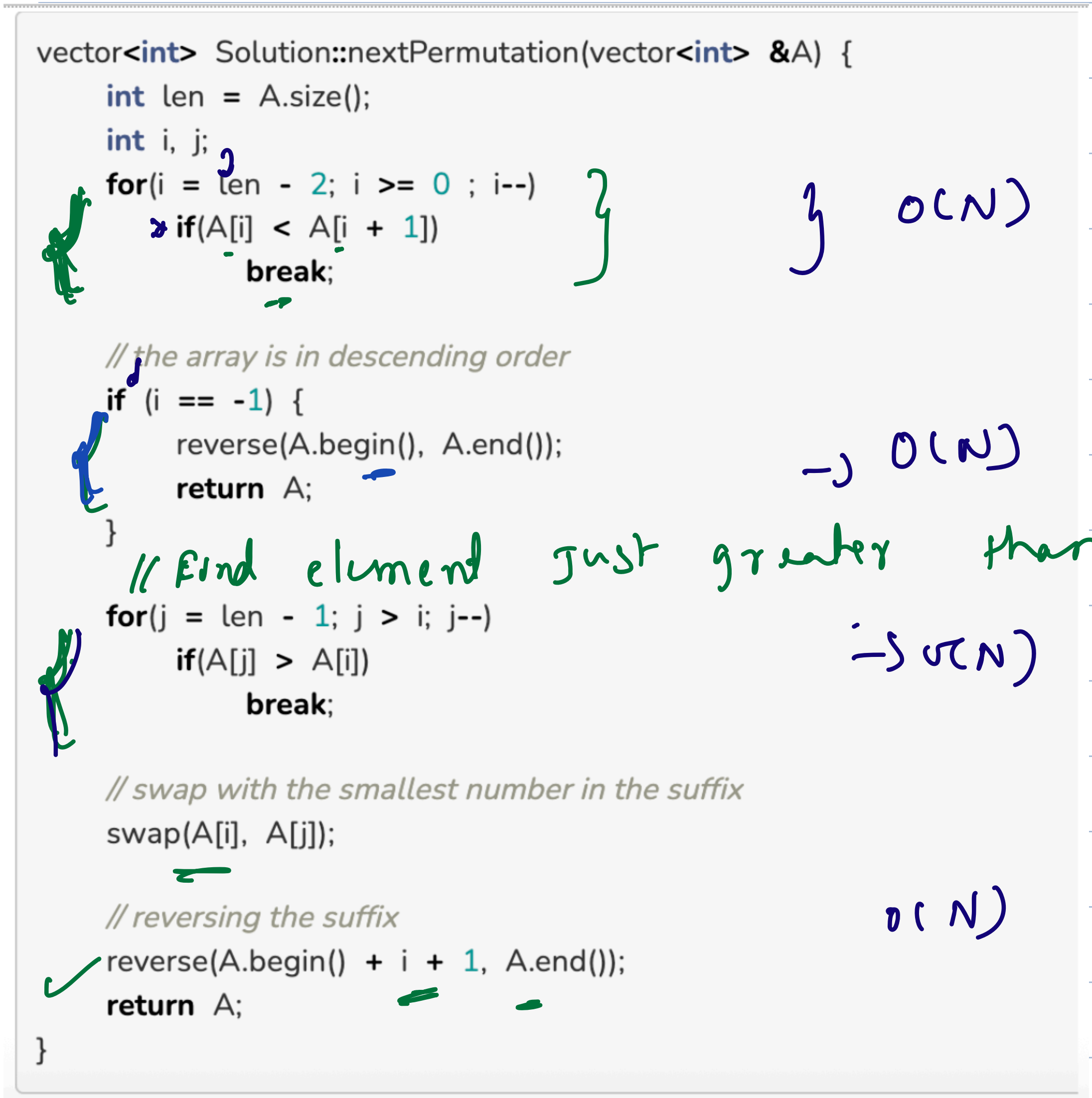
A :	4	3	1	5	8	7	6	2
	0	1	2	3	4	5	6	7
A' :	4	3	1	5	2	6	7	8

i=-1

i

2 1





```
vector<int> Solution::nextPermutation(vector<int> &A) {
    int len = A.size();
    int i, j;
    for(i = len - 2; i >= 0 ; i--)
        if(A[i] < A[i + 1])
            break;

    // the array is in descending order
    if (i == -1) {
        reverse(A.begin(), A.end());
        return A;
    }
    for(j = len - 1; j > i; j--)
        if(A[j] > A[i])
            break;

    // swap with the smallest number in the suffix
    swap(A[i], A[j]);

    // reversing the suffix
    reverse(A.begin() + i + 1, A.end());
    return A;
}
```

} O(N)

→ O(N)

// Find element just greater than A[i];

→ O(N)

O(N)

T.C: O(N)

S.C: O(1)

i=-1 i i i

6 5 3 2

1 2 9 8 7

```
A[i] < A[i+1] {
        break;
}
```

i = -1

9 8 7 6