Count-Re-Count

ZIG-ZAG TRAVERSE SOLUTION

Tze Sien

FAKE PUBLICATION

British School, 19th Century

Index

1.0 Concepts

1	3	4	10
2	5	9	11
6	8	12	14
7	9	13	15

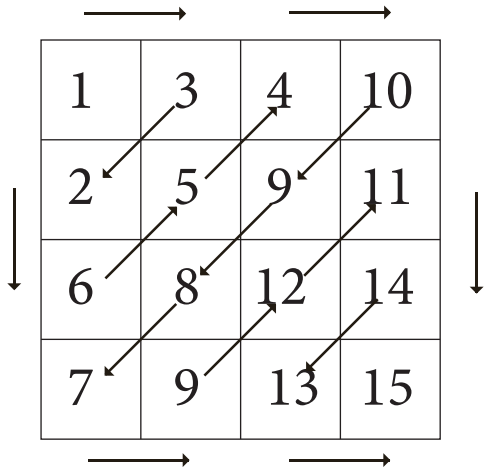
Fig. 1.1 Zig-Zag Data Structure

This time, we are given a Zig-Zag Matric.
Notice the pattern in those numbers? Let's decode it

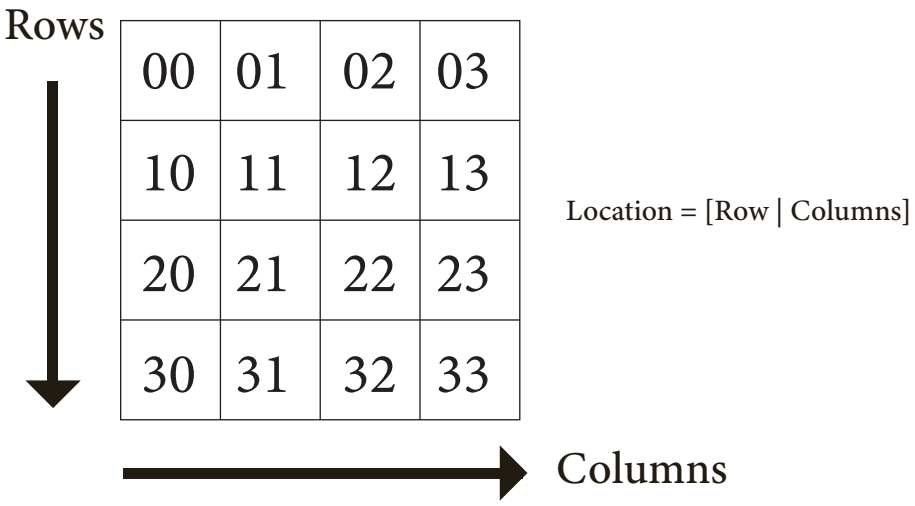
You can absolutely done the traverse with just a piece of paper and pen, using Count-Re-Count Algorithm.

Kids who can count able do it as well!

Sequence -> [Row, Column]



1. The very first step of solving this kind of traversing problem, always **make the path visible, in numeric form.**

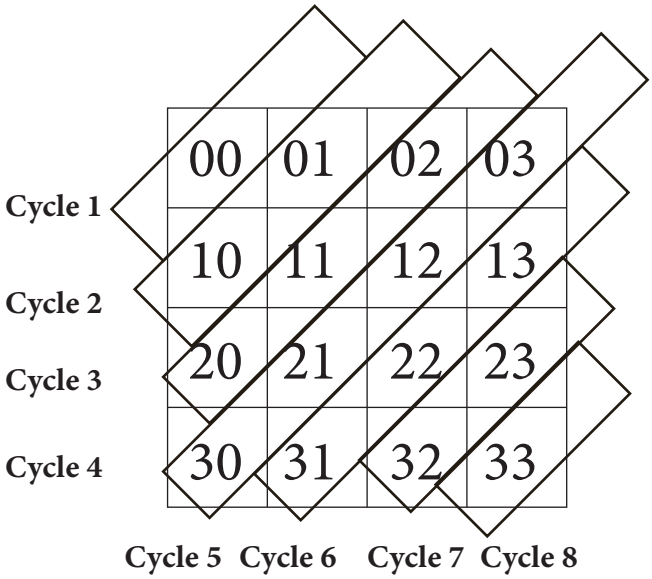


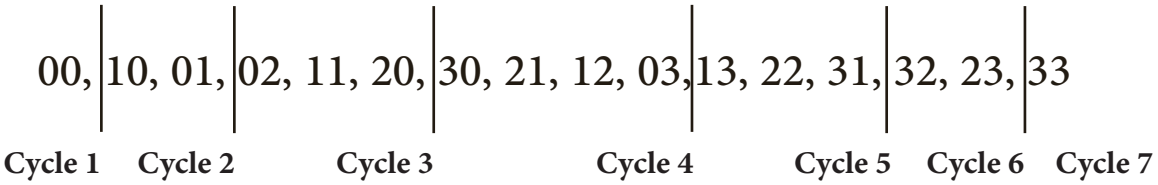
2. Let's see how the index of spiral sequence will look like

00, 10, 01, 02, 11, 20, 30, 21, 12, 03,13, 22, 31, 32, 23, 33

Do you notice any pattern? No, it's okay, let's chop it!

3. Chop! Chop! Chop, Make those sequence into cycles!





Now the pattern is clear! See it?

Element Count of Cycle

[1, 2, 3, 4, 3, 2, 1]

Formulae = floor(i/2) + (i%2), where i = cycle count

How about the pattern of numbers of each cycle?

Look at the Matrix's Shape!

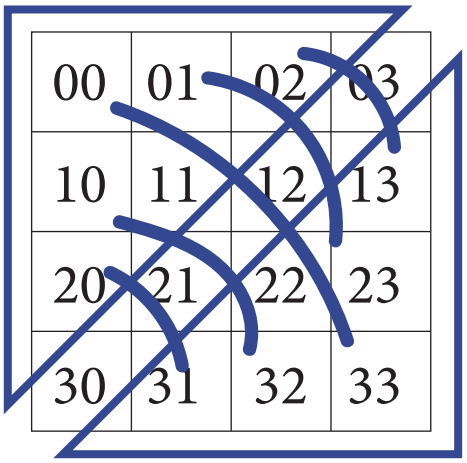
4x4 matrix,

so we just need to know how to count from

1 - 4 / 0 - 3,

I told you a kid can do this!

Left Triangle



Right Triangle

If you know Left Triangle, You know Right Triangle!

Our Matric Shape = 4 [0 - 3]

so we use N = 33

N - 00 = 33 | N - 01 =23 | N - 10 = 32

The "for Loop"

Left Triangle	Right Triangle	
00	33	Loop 1
10	23	
01	32	Loop 2
20	13	
11	22	
02	31	Loop 3
03		
12		
21		
30		Loop 4

Summary

Count 1 to N

1

2

3

4

Summary

Count to 1 to N

When pointer are pointing the same position, don't create inverse of it
Last loop just do for Left Triangle

1 (loop 0 to floor(1/2) + (1%2)) (1 loop)

Step 1 Add number to Holder
[0]
Step 2 Pointer Left & right

LeftTriangle
[0,0]

RightTriangle
[3,3] Middle

Step 3 Reverse
If(Left == Right)
  Middle [Left, right]
else
  Top[Left, right]
  Bottom [Right, Left]

2 (loop 0 to floor(2/2) + (2%2)) (1 loop)

[1, 0]

LeftTriangle
[1,0]
[0,1]

RightTriangle
[2,3] Top
[3,2] Bottom

3 (loop 0 to floor(3/2) + (3%2)) (2 loop)

[0, 1, 2]

LeftTriangle
[0,2]
[1,1]
[2,0]

RightTriangle
[3,1] Top
[2,2] Middle
[1,3] Bottom

4 (loop 0 to floor(1/2) + (4%2)) (2 loop)

[3, 2, 1, 0]

LeftTriangle
[3,0] Top
[2,1] Top
[1,2] Bottom
[0,3] Bottom

00, 10, 01, 02, 11, 20, 30, 21, 12, 03,13, 22, 31, 32, 23, 33

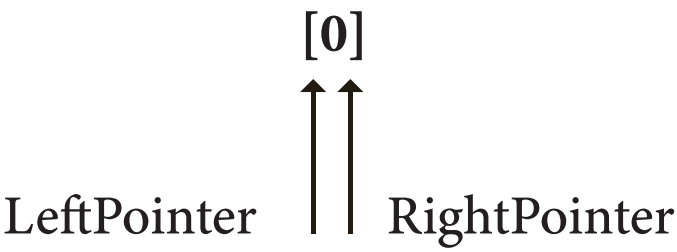
Construct Loop

N = Matric Shape - 1
N = 4 - 3 = 3

Holder = []
LeftPointer
Right Pointer
LeftTriangle = []
RightTriangle = []

Loop 0

1. Add 0 into the Holder
2. LeftPointer = 0
3. RightPointer = Holder.length - 1



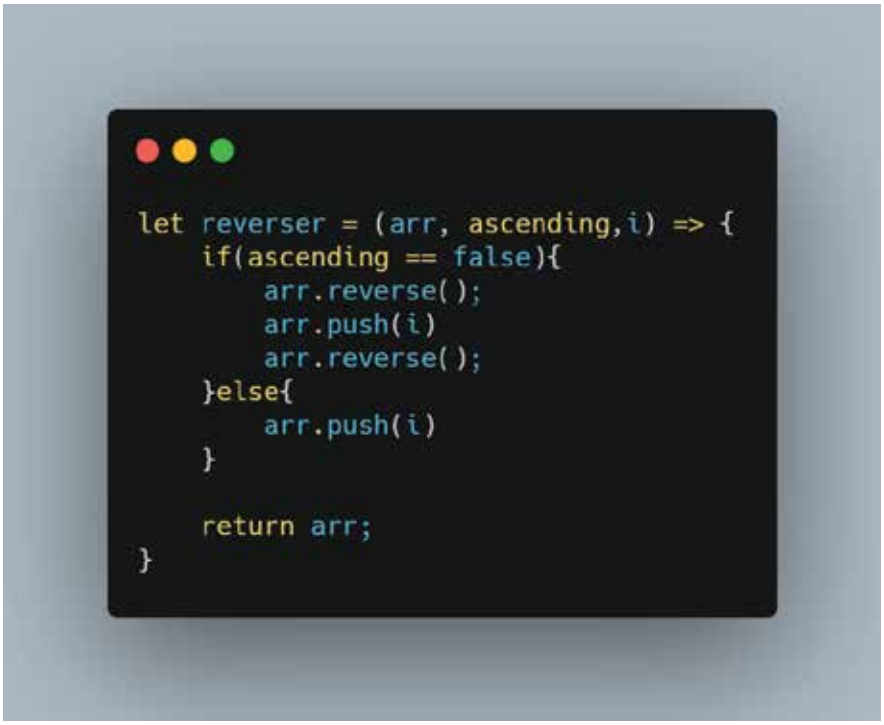
4. LeftPointer + RightPointer = [00]
5. RightTriangle = 33 - 00 = [33]

LeftTriangle = [1]
RightTriangle = [16]

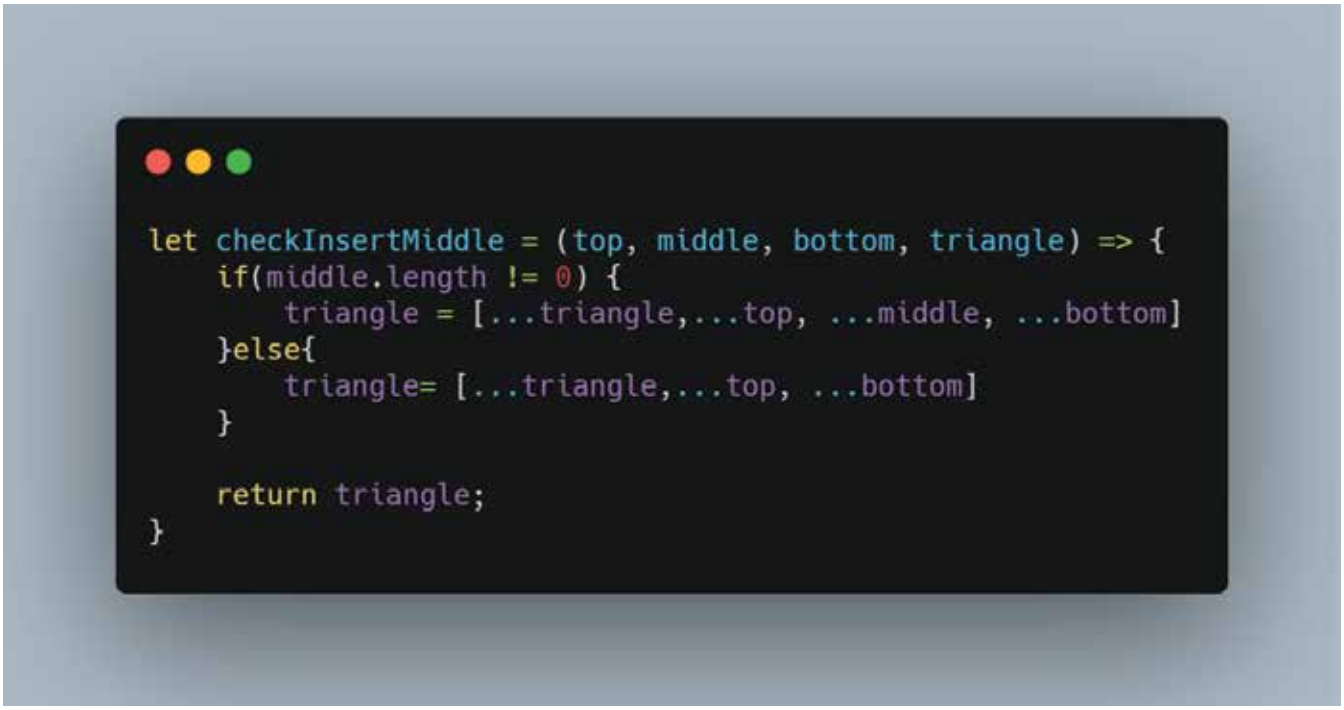
2.0 Implementations

Blur? Let's code it out!

Helper Functions



```
let reverser = (arr, ascending,i) => {
    if(ascending == false){
        arr.reverse();
        arr.push(i)
        arr.reverse();
    }else{
        arr.push(i)
    }

    return arr;
}
```



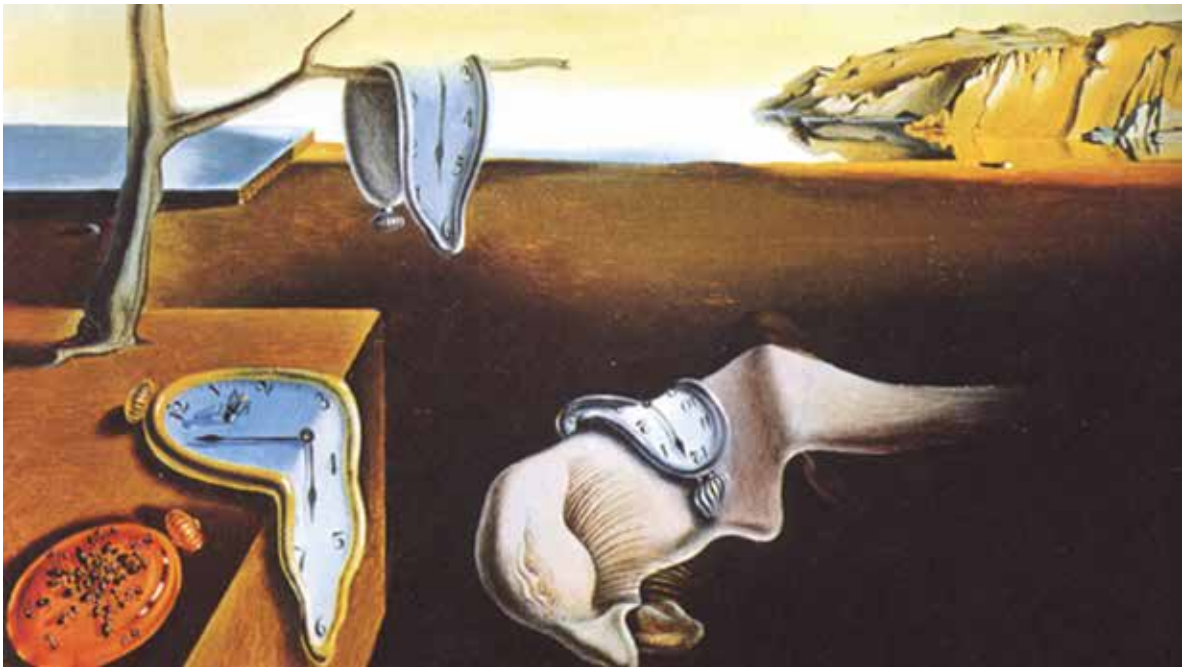
```
let checkInsertMiddle = (top, middle, bottom, triangle) => {
    if(middle.length != 0) {
        triangle = [...triangle,...top, ...middle, ...bottom]
    }else{
        triangle= [...triangle,...top, ...bottom]
    }

    return triangle;
}
```

```
// Time-Complexity = O(n/2(n/2 + 1))
let countAddReCount = (array) =>{

    let n = array.length;       // int                                  | O(1)
    let ascending = true;       // bool  true, false                    | O(1)
    let leftTriangle = [];      // array [0,1,2...,n(n-1)/2             | (n(n-1))/2 O(N(N-1)/2)
    let rightTriangle = [];     // array [inverse of LeftTriangle]      | (n(n-1))/2 O(N(N-1)/2)
    let count = [];             // array [0,1,2...,n-1]                 | O(N)

    for( let i = 0; i<n; i++ ){

        // Accumulator
        let top = []            // array [a, b]           | O(1)
        let middle = []         // array [c, d]           | O(1)
        let bottom = []         // array [e, f]           | O(1)
        let topRight = []       // array [inv(a), inv(b)] | O(1)
        let middleRight = []    // array [inv(c), inv(d)] | O(1)
        let bottomRight = []    // array [inv(e), inv(f)] | O(1)

        // Reverse the count array every cycle
        count = reverser(count, ascending, i)

        // Pointer to move around the count array [ 0,1,2,3,...n ]
        let locPtrLeft = 0;
        let locPtrRight = count.length - 1;

        // Formulae to know how many iteration we need in every cycle (Cycle/2, then we get ( Divider +
Remainder))
        let h = ( Math.floor((i+1)/2) + ((i+1)%2) )

        while(!(h < 1)){

            // Check if the number is [00,11,22,...,nn]
            if (locPtrLeft == locPtrRight){

                middle.push(array[count[locPtrLeft]][count[locPtrRight]])
                middleRight.push(array[n-1-count[locPtrLeft]][n-1-count[locPtrRight]])

            }else{

                top.push(array[count[locPtrLeft]][count[locPtrRight]])
                topRight.push(array[n-1-count[locPtrLeft]][n-1-count[locPtrRight]])

                bottom.unshift(array[count[locPtrRight]][count[locPtrLeft]])
                bottomRight.unshift(array[n-1-count[locPtrRight]][n-1-count[locPtrLeft]])

            }

            locPtrLeft++;
            locPtrRight--;
            h--;

        }

        // Only for Last Loop
        if(i < n-1) rightTriangle = checkInsertMiddle(topRight, middleRight, bottomRight,
rightTriangle);

        leftTriangle = checkInsertMiddle(top, middle, bottom, leftTriangle);
        count.reverse();
        ascending = !ascending;

    }

    return [...leftTriangle, ...rightTriangle.reverse()]

}
```

3.0 Time-Space Complexity

How good is the algorithm?

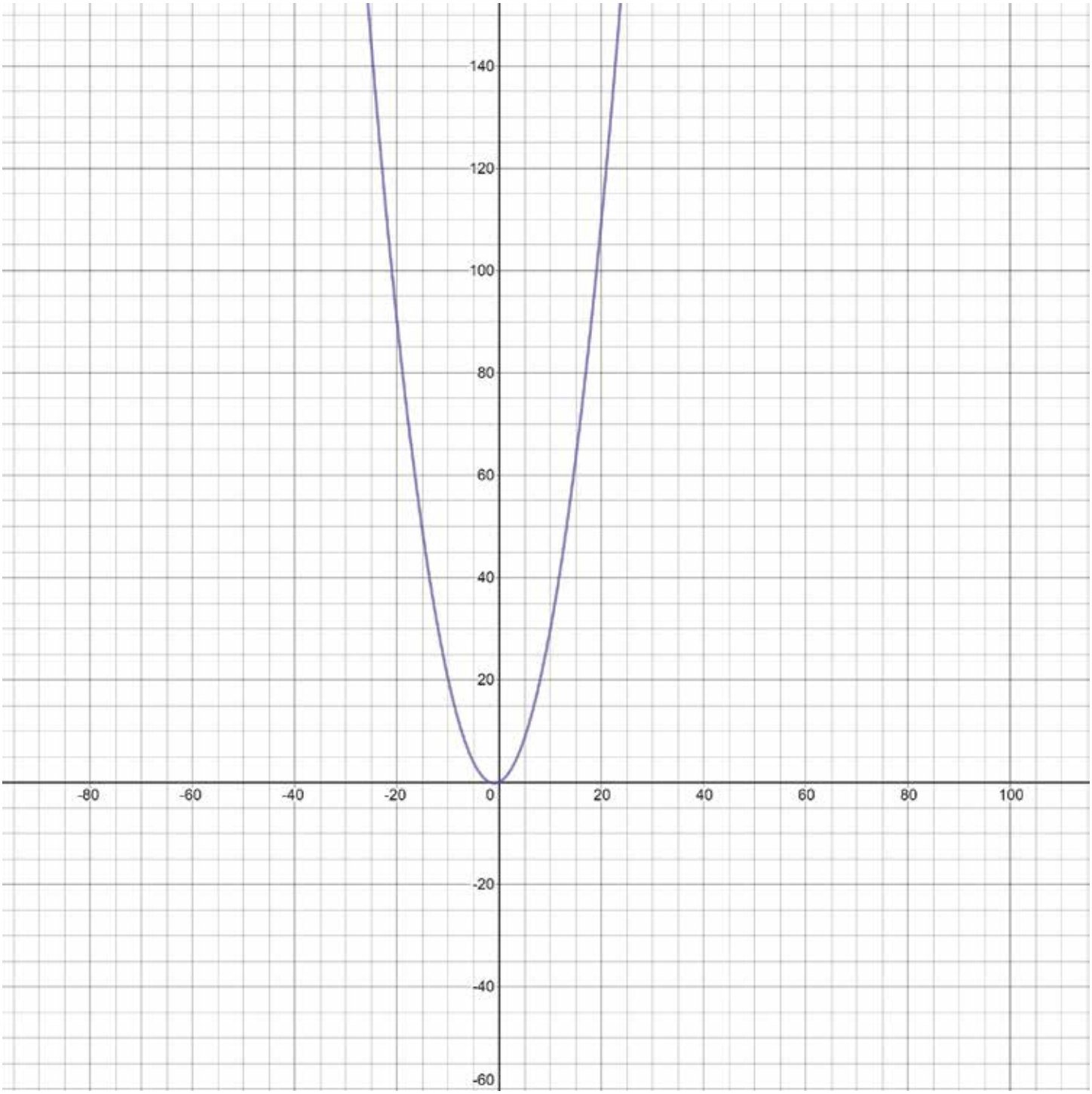
The excellent algorithm have low time space complexity.
Let's check with this Peel Traverse Algorithm!



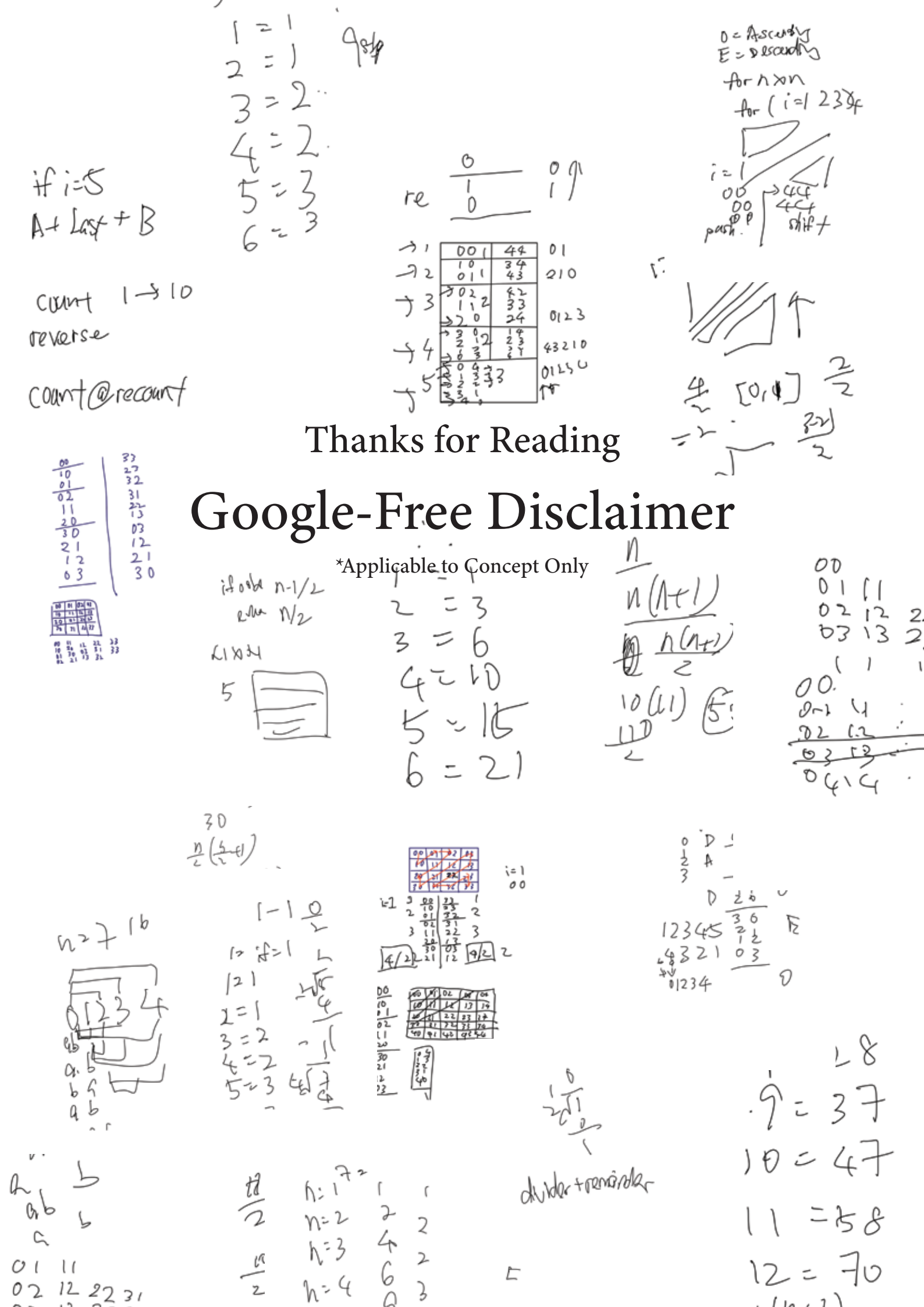
Space-Complexity

```
let n = array.length;    // int                                  | O(1)
let ascending = true;    // bool  true, false                    | O(1)
let leftTriangle = [];   // array [0,1,2...,n(n-1)/2             | (n(n-1))/2 O(N(N-1)/2)
let rightTriangle = [];  // array [inverse of LeftTriangle]      | (n(n-1))/2 O(N(N-1)/2)
let count = [];          // array [0,1,2...,n-1]                 | O(N)
let top = []             // array [a, b]                         | O(1)
let middle = []          // array [c, d]                         | O(1)
let bottom = []          // array [e, f]                         | O(1)
let topRight = []        // array [inv(a), inv(b)]               | O(1)
let middleRight = []     // array [inv(c), inv(d)]               | O(1)
let bottomRight = []     // array [inv(e), inv(f)]               | O(1)
```

Time-Complexity



$$O(n/2(n/2 + 1))$$



Thanks for Reading

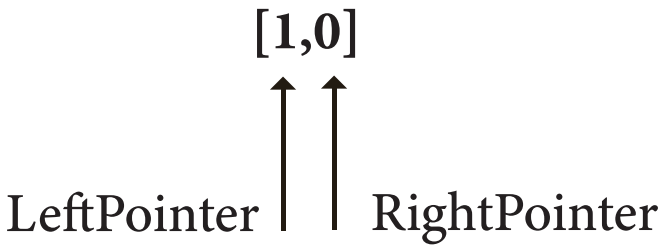
Google-Free Disclaimer

*Applicable to Concept Only

Loop 1

1. Check if the array is inversed (yes, inverse back)
2. Add 1 to the Holder [0, 1]
3. Inverse the array [1, 0]

4. LeftPointer = 0
5. RightPointer = Holder.length - 1



6. Iteration we need (floor(i/2) + (i%2)

7. Left Pointer + Right Pointer = [10]
8. Right Pointer + Left Pointer = [01]

9. Left Pointer ++;
10. RightPointer --;

LeftTriangle = [1, 2, 3]
RightTriangle = [14, 15,16]