July 5

LONELY PIXEL
Lang: Python
Idea: Single Pass Recursion, track top down left right, every row and column only need to be seen once.

```
def lonely_pixel(t,d,l,r,screen):
    if l > r or t > d or l < 0:
        return 0

    left = l
    while left <= r:
        if screen[t][left] == 1:
            break
        else:
            left+=1

    if left > r:
        return lonely_pixel(t+1,d,l,r,screen)
    else:
        if check_pixel(t,d,left,r,screen):
            return lonely_pixel(t + 1, d, l, left - 1,screen)
+
                   lonely_pixel(t+1, d, left+1, r, screen)
        else:
            return 1 + lonely_pixel(t + 1, d, l, left - 1,scre
en) +
                       lonely_pixel(t+1, d, left+1, r, screen)

def check_pixel(t,d,l,r,screen):
    if screen[t][l] == 0:
```

```
        return False
    else:
        left = l+1
        while left <= r:
            if screen[t][left] == 1:
                return True
            left += 1

        top = t + 1
        while top <= d:
            if screen[top][l] == 1:
                return True
            top += 1
        return False
```

DUP AND MISSING #
Lang: Python
Idea: Implicit hash table, no extra space 1 pass solution

```
    def findErrorNums(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """

        anticipated = (len(nums)*(len(nums) + 1))/2
        sums = 0
        dup = -1
        for x in nums:
            if nums[abs(x)-1] < 0:
                dup = abs(x)
            sums += abs(x)
```

```
            nums[abs(x)-1] = nums[abs(x)-1] * -1


        return [dup,anticipated-(sums-dup)]
```

RACE CAR
too sleepy to write, but here is the rationale, use DP to solve this question. 1D array is enough.
Each element in dp table represents optimal steps needed to reach that number.

```
Beginning DP table
___ | ___ | ___ | ___ | ___ | ___ | ___ | ___ | ___


Optimal path is always AAAAAAAAA ... etc, but finding the opti
mal path between the A's is the more difficutl path which we d
o with dp


Conceptually, fill in all the A's
    A     *     A'                      A''
___ | ___ | ___ | ___ | ___ | ___ | ___ | ___ | ___


How do I get to *? either A`RR`|A| or AA'`R`|A| right? RR twic
e so we can move forward at
speed 1 again at A or reverse from A' then we know just to add
A from our Dp table we see AARA or ARRA both work.


    A    ARA    A'          *           A''
___ | ___ | ___ | ___ | ___ | ___ | ___ | ___ | ___


Second example for reference same idea from A' and A'', either
use A' and two RR to move
forward or A'' and a single R to move backward.
```