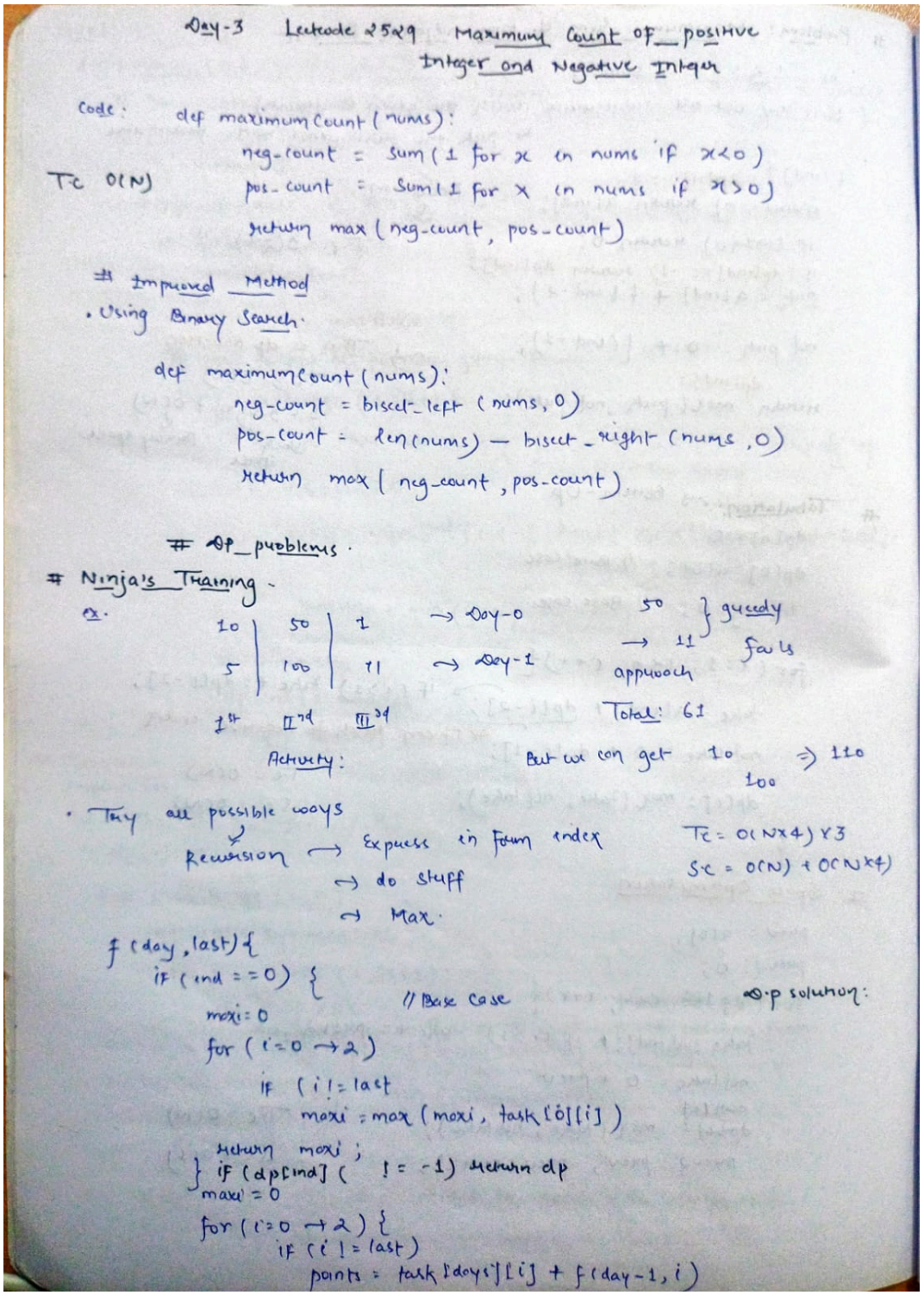
Day_3: DSA



Day-3 Leetcode 2529 Maximum Count of positive Integer and Negative Integer

Code:

```
def maximumCount(nums):
    neg_count = sum(1 for x in nums if x<0)
    pos_count = sum(1 for x in nums if x>0)
    return max(neg_count, pos_count)
```

TC O(N)

Improved Method

- Using Binary Search.

```
def maximumCount(nums):
    neg_count = bisect_left(nums, 0)
    pos_count = len(nums) - bisect_right(nums, 0)
    return max(neg_count, pos_count)
```

Dp_problems.

Ninja's Training.

Ex.

10	50	1	→ Day-0
5	100	11	→ Day-1
I^st	II^nd	III^rd	

50 } greedy → 11 fails approach

Total: 61

Activity:

But we can get 10 100 ⇒ 110

- Try all possible ways

Recursion → Express in form index
→ do stuff
→ Max.

TC = O(N×4)×3
SC = O(N) + O(N×4)

```
f(day, last){
  if(ind==0){                    // Base case
    maxi=0
    for(i=0→2)
      if(i!=last
        maxi=max(maxi, task[0][i])
    return maxi;
  } if(dp[ind]( != -1) return dp
  maxi=0
  for(i=0→2){
    if(i!=last)
      points = task[days][i] + f(day-1, i)
```

D.p solution:

```
        maxi = max ( maxi, point );
    }  dp[ind] =
return maxi ;
```

Tabulation → declare dp array int dp[n][4];

// Base case

```
dp[0][0] = max(arr[0][1], arr[0][2])
dp[0][1] = max(arr[0][0], arr[0][2])
dp[0][2] = max(arr[0][0], arr[0][1])
dp[0][3] = max(arr[0][0], arr[0][1], arr[0][2])
```

T.C = O(N x 4 x 3)
S.C = O(N x 4)

```
for ( day = 1 → n-1 )
  { for ( last task = 0 → 3 ) {
        for ( i = 0 → 2)
    }   if ( task != last ) {
            int point = point [day][task] + dp[day-1][task] :
            dp[day][last] = max (dp[day][last], point );
                                                  } } }
return dp[n-1][3]
```

// copy recursion code.

Space Optimization.

```
for ( i = 0 → 2 ) {
  if ( task != last ) {
      temp[last] = max( temp[last] , point [day] [task] + prev[task] );
  }
  prev = temp
return prev.
```

Maze problem:

0 0 0
0 -1 0
0 0 0

dead cell ⇒ -1
You can not go from there

for unique path →

```
f (i,j) {
  if (i==0 && j==0) return 1;
  if (i<0 || j<0) return 0;
  up = f(i-1, j)
  left = f ( i, j-1)
  return up + left ; }
```

Addition of one more base case:

```
f(i,j){
    IF (i>=0 && j>=0 && a[i][j] == -1) return 0;
    if (i==0 && j==0) return 1;
    IF (i<0 || j<0) return 0;
    IF (dp[i][j] != -1) return dp[i][j];
    up = f(i-1, j)
    left = f(i, j-1)
    dp[i][j] =
    return up + left;
}
```

Memorization:

Tabulation:

```
for (i=0 → m-1) {
    for (j=0 → n-1) {
        IF (a[i][j] == -1) dp[i][j] = 0;
        else if (i==0 && j==0) dp[i][j] = 1;
        ~~If (i>=0 && j>=0 &&~~ 0
        Else:
        if (i>0) up = dp[i-1][j];
        if (j>0) left = dp[i][j-1]
        return left + right;
```

Problem Minimum path sum in Triangular grid:

1 → Starting point is fixed.

2 3

3 6 7

8 9 6 10

→ Represent (i,j)

→ Explore all paths (↓↘)

→ Minimum of all path

(i+1) (j+1)

There are four recurrence point ⑧ ⑨ ⑥ ⑩

So Here we will start from Starting point which is fixed destination (n-1)

```
f(i,j) {
    // Base case
    if (i == n-1) return a[n-1][j];
    q = a[i][j] + f(i+1, j);
    dq = a[i][j] + f(i+1, j+1);
```

return min (d, dg);
}

T.C of this recursion ⇒ 2^n

As there is overlapping problems SC: O(N)

Memoization:

```
f(i,j) {
    if (i==n-1) return a[n-1][j];
    if (dp[i][j] != -1) return dp[i][j];
    d = a[i][j] + f(i+1, j);
    dg = a[i][j] + f(i+1, j+1);
    return dp[i][j] = min(d, dg)
```

T.C = O(N×N)
S.C = O(N×N)
Triangular shape

Tabulation Method: