Problems on Array

Question:

Given a strings of lower case characters,

Find no of subsequence of string "ag" in (cgag)

Ex: S = 

(ag)
(ag)
(ag)
(3)

"bc" ✓
"ba" ✓
"cb" X
(Not in order)

Ans = 3

Ex: g a g = 1

Order Matters

No. of occurrence of 'a' = 2
& 'g' = 2

((2x2))

Brute Force:

Brute Force

O(

Total No. of subsequences?

S = (a b c d e)

Random subseq = {2 2 2 2 2 2} = 2 x 2 x 2 x 2 x 2 x 2 = 2^6

n elements

=) 2 x 2 x 2 ... n times = 2^n

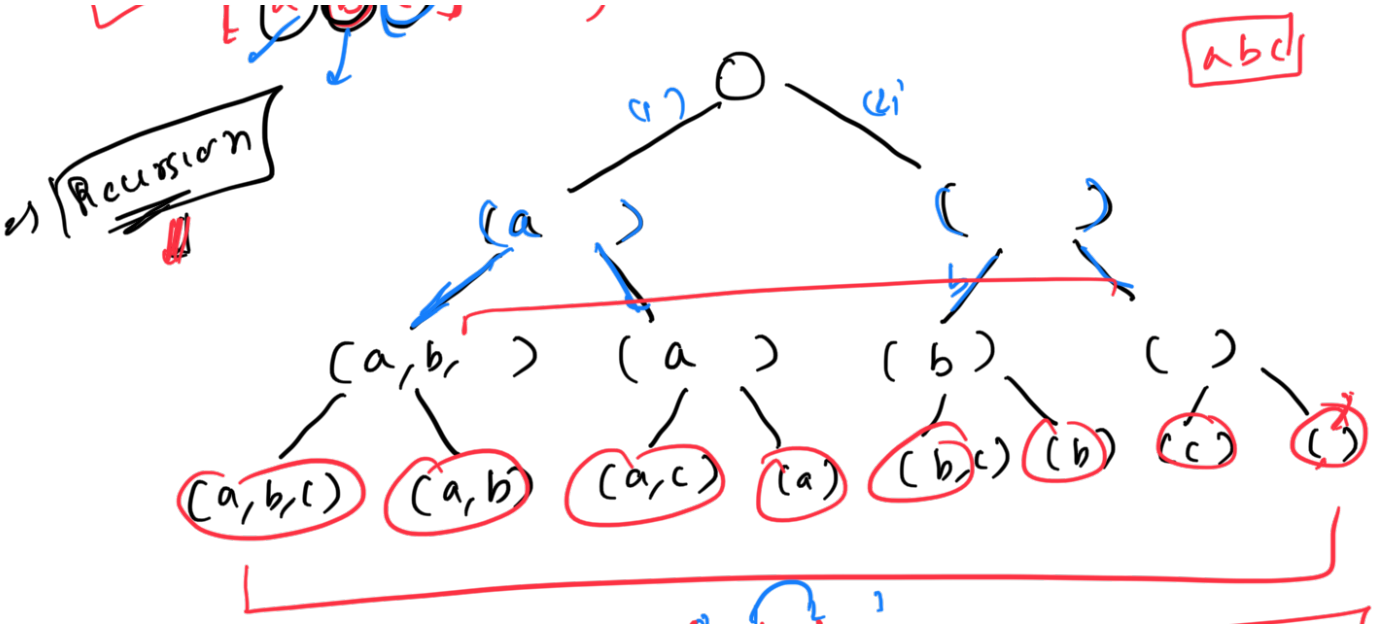
(Recursion)

For 'a', 2 choices

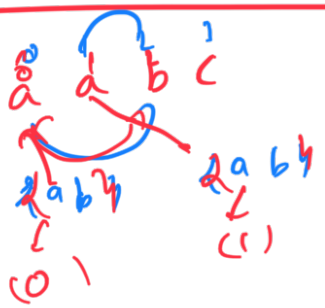
→ Included in subseq

→ Exclude

$2^{3 \text{ to } 8}$



S = a a b c



⇒ b a b

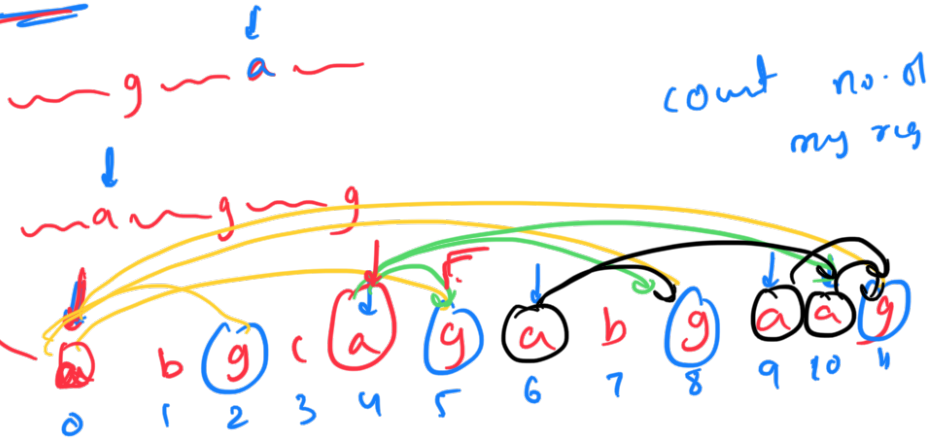
Exponential t.c

generate(2^n)

'ag'

count++;

Approach 2:



count no. of g's to my right

4

4 g's to the right

ans = 4 + 3 + 2 + 1 + 1

⇒ 11

ans = 4 + 3 + 2 + 1 + 1 = 11

'ag'

count = 0

for (i = 0 to N-1) {

(4 loops) ∧ (Recursion/DP)

for(i = 0 ...

Approach:

if(s[i] == 'a') {

for(j = i+1 to N-1) {

if(s[j] == 'g')

count = count+1;

}

}

T.C: $n + (n-1) + (n-2) + \dots + 1$

$= \frac{n(n+1)}{2}$

$= O(n^2)$

Ex: a a a a a a a g

Efficient Approach

a a g g g

Atleast 3 subseq.

'a', 'g'

count_g = 0

For any 'a', No. of g's on the right

s = a g b c a a g g x y z a d g a

No. of a's to my left

count_g = 1 x 3 4

ans = 1 + 3 + 3 + 4

= 11

→ Traverse from Left to Right

count_a;

ans = count_a;

agg, abc

string's length > 3

S_1 = a b c d . . . . (10^5)

S_2 = x y z . . (10^5)

Recursion / Dynamic Programming

```
count_g = 0;
ans = 0;
for(int i = n-1; i >= 0; i--) {
    if(s[i] == 'g')
        count_g++;
    if(s[i] == 'a')
        ans = ans + count_g;
}
```

2 operations

T.C: O(n)

$\Rightarrow O(2n) \Rightarrow O(\frac{n}{2})$

ans = ans+0

Question: Print / Generate a pattern

A = 5

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}_{5 \times 5}$$

primes size $\leq N$ (10^5)

A =

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 2 & 3 \end{bmatrix}_{3 \times 3}$$

1

1 2

1 2 3

1 2 3 4

Question: Maximum Element

9 8 7 6 1 20

Ex: A2 [3 4

max = a[0]

max = -INF 3 4

9

20

1) Naive Approach:

→ sort the array ⇒ O(nlogn)

→ Return the last ele ⇒ O(1)

⇒ O(nlogn)

Efficient Approach:

→ Linear search

T.C: O(n)

ans = INT_MAX

Question: Closest min max

→ Given an array, find the smallest subarray which contains maximum and minimum element.

Ex: 1 2 3 1 3 4 4

[1, 3, 4] → [4, 1] [2]

max = 4

min = 1

O(n)

length = 3

[1, 4]

1

Multiple mins and maxs

→ Smallest subarray with min & max

[1.....4]

1) min ~~ max

2) max ~~ min [4.... 1]

corner elements are min and max

min ~~ max ~~ min

max ~~ max

(Find a max)

For all min's

Approach

min — First max element (ans) INT_MAX

Find first max

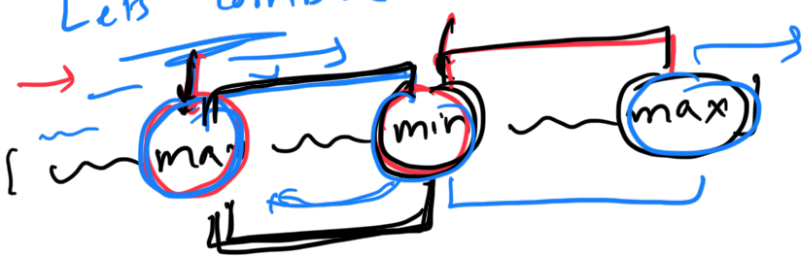
max — Right

first-max — first min element

max / min

left → Right

Lets Combine



Iterate and get

(min = 1, max = 7) => O(n)

max — min — max — max — min

A = 1 7 5 1 7 4 7 1

0 1 2 3 4 5 6 7

A[0...1]

(2)

ans = (INT_MAX)

(2) (2) => break

(if max) look for min

(if min) look for max

a[i...j] = (j - i + 1)

max = T, min

1 2 3 7 7 1 2 3 4

// O(n)

max_ min

ans = INT_MAX

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

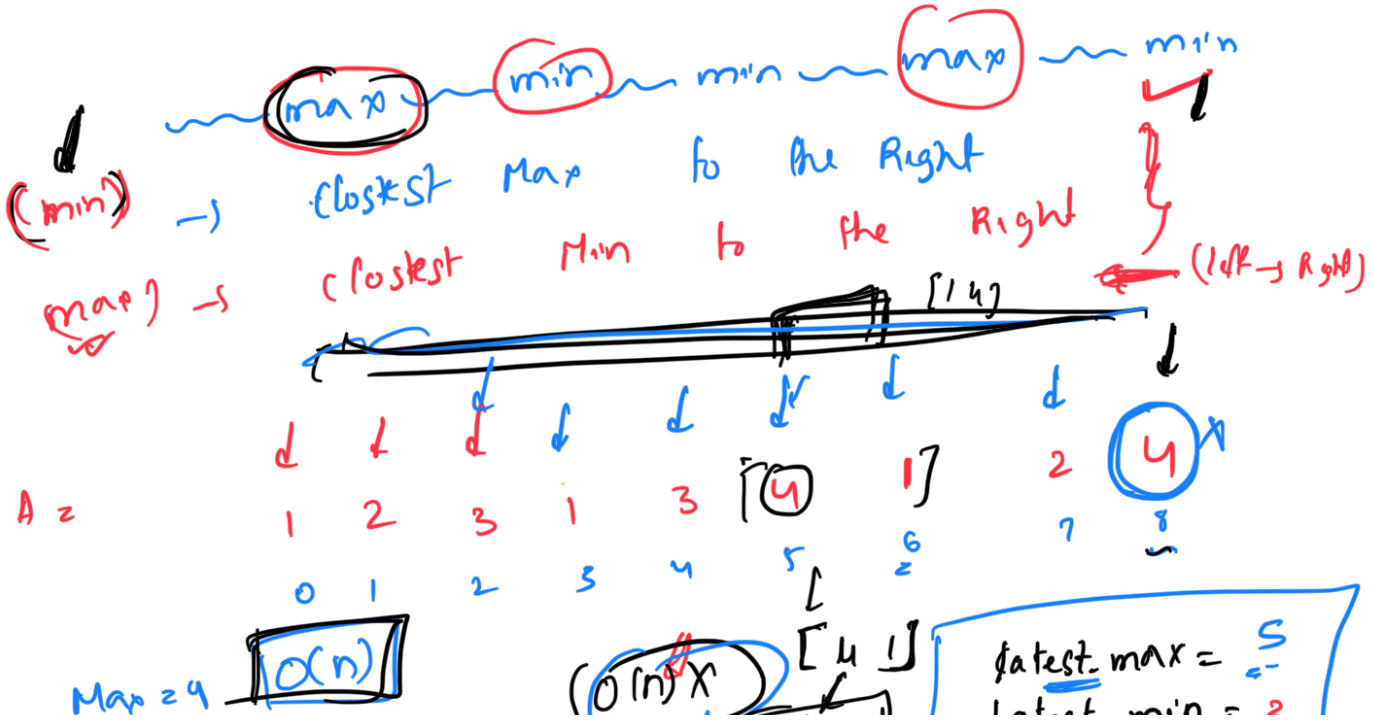
```
if( a[i] == max) {
    for (j = i+1; j<n; j++) {
        if (a[i,j] == min) {
            length = j-i+1;
            ans = min (ans, length);
            break;
        }
    }
}
if (a[i] == min)
```

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

Efficient Approach:

For every max/min element, Count No. of g's ← $O(n)$ → $O(1)$?

g g g g g g g → $O(n) \rightarrow O(1)$?



Min = 1

'n' elements

T.C: O(n) ⇒ O(n)

O(n)

```
for(i = n-1; i >= 0; i--) {

}
```

ans = 2

(L = R =)

[1, 4]

max min

Question: Max length consecutive 1's

→ Array of 0's and 1's

→ Swap atmost one '0' with a '1'.

→ Length of largest contiguous subarray made up of 1's

Ex: 1 1 1 0 1 1 1 0 1 ⇒ 1 1 1 1 1 1 1 0 0

7

0 1 1 1 1 0 0

Ex: 1 1 1 1 0 0 0 ⇒ 4

Ex: 1 1 1 0 1 1 1 ⇒ 1 1 1 1 1 1 0 ⇒ 6

Ex:

swap = 3

1 1 0 1 0 0 1 1 1 0 1 1 1 ⇒ 7

0 1 2 3 4 5 6 7 8 9 10 11 12

d

⇒ T.C:

(swap = 3)

arr[i-1] = 0 && arr[i+3] = 0

1 1 1 0 1 1 1

Ex: 1 1 0 1 0 0 1 1 1 0 1 1 1
0 1 2 3 4 5 6 7 8 9 10 11 12

swap = 3 | swap = 3 | Break when violet

1 1 0 1 0 0 1 1 1 0 1 1 1

end = 3

any = 3

Not allowed to swap → largest

1 1 1 0 1 0 0 1 1 0 0 1

(i)

50%

No. of ones on left + no. of ones on right

for(j = i-1; j >= 0; j--)

for(j = i+1 ; j < n; j++)

1 0 1 1 1 1 1

(i)

Can we optimise this a little?

0 1 1 1 1 1 1 1 1 0

2 | 3+1 | 1+1+2 | 1

(2+1)

O(1)

Leftlid as No. of consecutive 1's to the left of 'i' including i

if(s[i] = 1) left[i] = 1

i = 8

0 1 2 3 4 5 6 7 8 9 10 11 12

s = 1 1 0 0 1 0 1 1 1 1 0 1

if(s[0] = 0)

left: 1 2 0 0 1 0 1 2 3 4 5

{ Right: 2 1 0 0 1 0 5 4 3 2 1 0 1

right[i+1]

mins of [left+right)

[left[i] + right[i] - 1)

left[i] = 3

right[i] = 3

=> [left[i] + right[i] - 1)

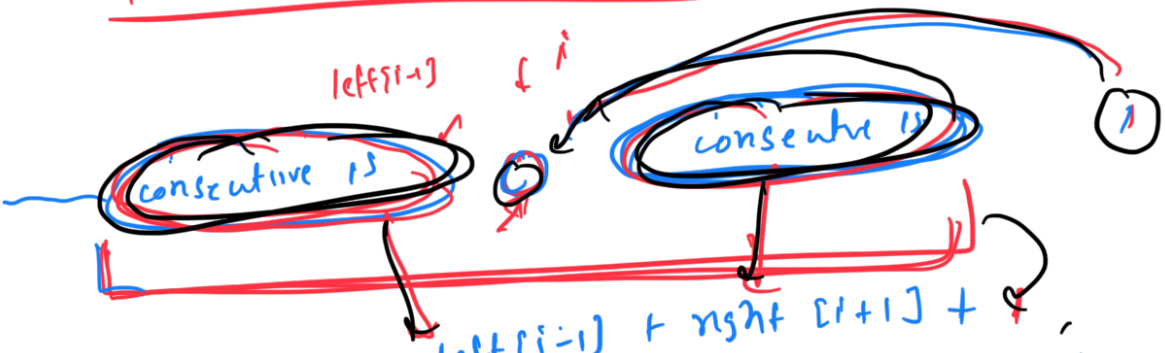
if(A[i] == 0)
left[i] = 0
else
left[i] = left[i-1] + 1

if(A[i] == 0)
left[

1 swap is allowed (A[i] = 0)

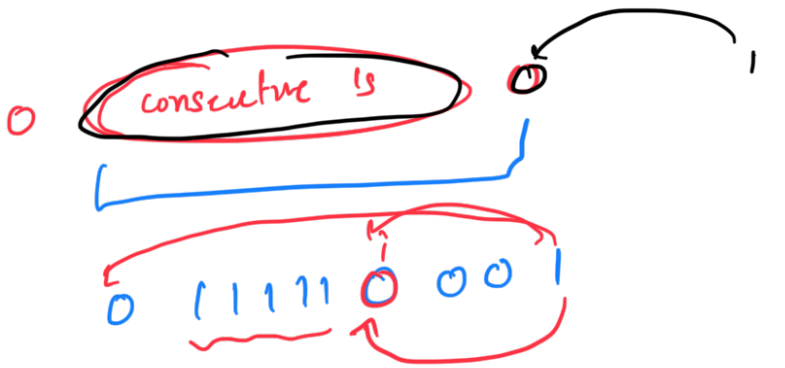
Case 1

left, right



(1 swap)

Case 2:



left[i-1] = 5

right[i+1] = 0

5 + 0 + 1 = 6

[take care of boundary]

count_ones;

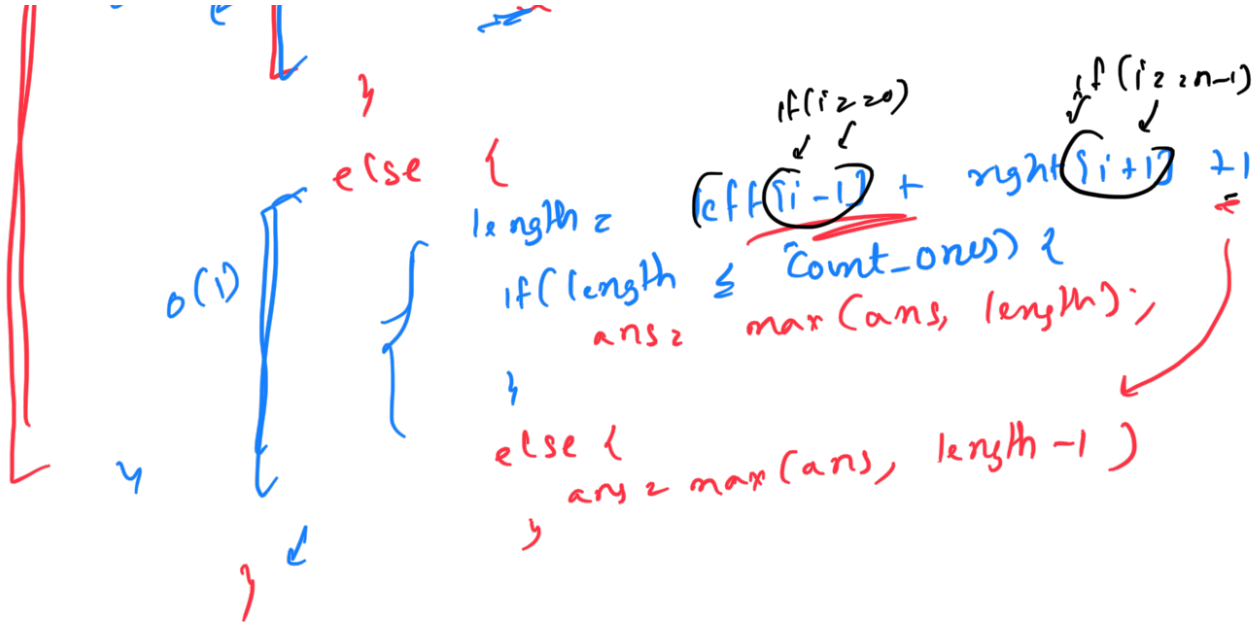
ans = INT_MIN;

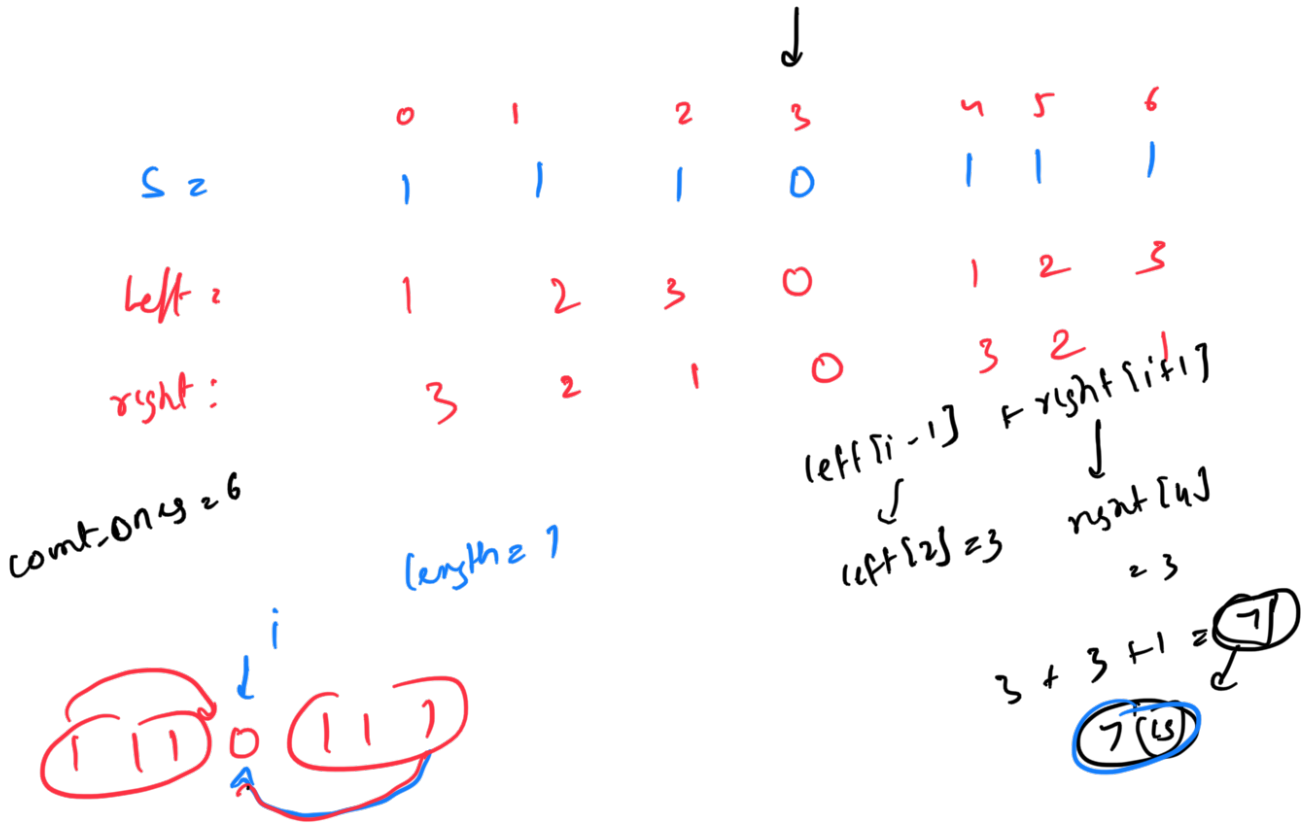
for(int i = 0; i < N; i++) {

O(1)

if(A[i] == 1) {

ans = max(ans, left[i] + right[i] - 1));





T.C:

-> left, Right = O(n)

-> O(n)

T.C: O(n)

left & Right Array

S-C:

O(n x O(1)) = O(n)

O(n) + O(n) = O(2n) = O(n)

Ex:

max -> closest max to right

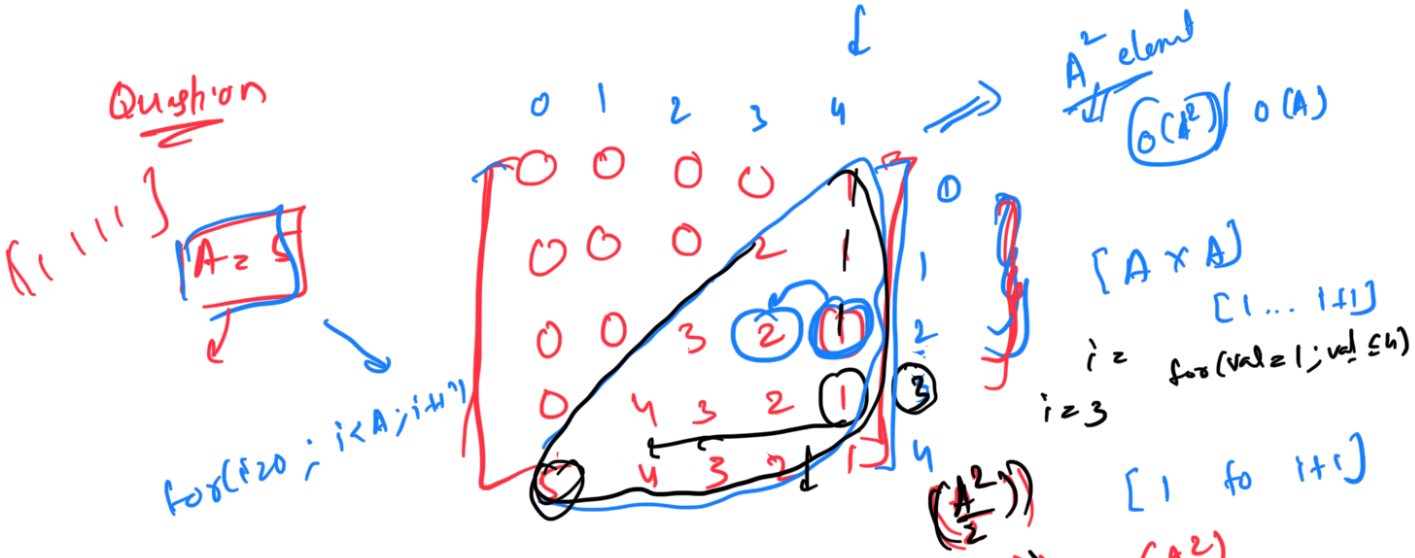
min -> closest min to right

(5 - 0 + 1) = 6

(5 - 3 + 1) = 3

6 - 5 + 1 = 2

Question



[ ]mn

O(m.n)

(-1 ... i+1)

col = i - 1;
for(val = 1; val ≤ i+1; val++) {
    mat [i] [col] = ___;
    col--;
}

mat [A] [A] = {0};

[A] [A]

(Pascal Triangle)

([[0] for i in range (5)] for j in range(5)].

(m x n)

a[i][j] = a[i-1]+-

Ex: [8 7 6 7 8 8 8]

Hashing?

for(i=0; i<n; i++) {
    for(j=i+1; j<n; j++) {

T.C: O(n²)

}

key
Array element are key.

}

for(int i=0; i<n; i++) {
    if(a[i] exists in map) {
        mp[a[i]]++;
    }

map = {
1 :
8 :
7 :

No. of subsequences 'ag' in the given string

```
int num_subsequences_ag(string s){
      ans = 0;
      count_g = 0;
      for(i = s.size()-1; i >=0; i--){
            if(s[i] == 'g')
                  count_g++;
            if(s[i] == 'a')
                  ans += count_g;
      }
return ans;
}
```

Generate Pattern

```
for(int i = 0; i < A; i++){
      col = A-1;
      for(val = 1; val <= i+1; val++){
            mar[i][col] = val;
            col--;
      }
}
```

Brute Force : Smallest subarray with min and max

```
max, min;
ans = INF;
for(int i = 0; i < n; i++){
      if(A[i] == min){
            for(int j = i+1; j < n; j++){
                  if(A[i] == max){
                        ans = min(ans, j - i + 1);
                        break;
                  }
            }
      }
      if(A[i] == max){
            for(int j = i+1; j < n; j++){
                  if(A[i] == min){
                        ans = min(ans, j - i + 1);
                        break;
                  }
            }
      }
return ans;
```

Efficient Approach : Smallest subarray with min and max

```
latest_max = -1;
latest_min = -1;
ans = INT_MAX;
for(int i = n-1; i >=0 ;i--){
     if(A[i] == min){
           latest_min = i;
           if(latest_max != -1)
                ans = min(ans, latest_max - latest_min + 1);
     }
     else if(A[i] == max){
           latest_max = i;
           if(latest_min != -1)
                ans = min(ans, latest_min - latest_max + 1);
     }
}
```

Maximum consecutive 1s with at most 1 swap

```
// Generate Left array
int left[] = {0};
if(A[0] == 1) left[0] = 1;
for(int i = 1; i < N; i++){
      if(A[i] == 0)
            left[i] = 0;
      else
            left[i] = left[i-1] + 1;
}

// Generate right array
int right[] = {0}
if(A[n-1] == 1) right[n-1] = 1;
for(int i = N-2; i >= 0; i--){
      if(A[i] == 0)
            right[i] = 0;
      else
            right[i] = right[i+1] + 1;
}

for(int i = 0; i < N; i++){
      if(A[i] == 1){
            ans = max(ans, left[i] + right[i] - 1);
      }
      else{
            // Take care of boundary cases i = 0, n-1
            length = left[i-1] + right[i+1] + 1;
            if(length <= count_ones)
                  ans = max(ans, length);
            else
                  ans = max(ans, length-1);
      }
}
return ans;
```