Arrays

Lect-1 (2/12/2020)

1. Seperate +ve and -ve numbers.
2. Seperate 0, and 1
3. 283 / Move zeroes.
4. Seperates 0, 1, 2.
5. 75 seperate colors
6. 189 / Rotate array.
7. MAX sum of array and its index.

Lect-2 (6/12/2020)

1. 3 / longest substrings without repeating characters
2. 76 / Min window substring.

Lect-3 (8/12/2020)

1. smallest window containing itself (gfg)
2. 159 / find k distinct element window }.
3. 340 /
4. Max vowel letter in any substring (1456)
5. Count Unique Substring with non-repeating characters.
6. 239 / sliding window maximum.

Lect-4 (9/12/2020)

1. longest subArray with equal no. of 0 and 1 } same
2. 525
3. count subArray with equal no. of 0 & 1 (2)
4. Longest subArray with sum divisible by k (3)
5. 974 SubArray sum divisible by k (4)
6. 781 Rabbit in forest (5)

Lect-5 (11/12/2020). (Notebook-3)

1. 930 / Count SubArray with sum equal to k
2. 1248 / Count no. of nice subArray.
3. 1004 / Max consecutive one - III (3 limitas.)
4. 904 / fruit into Basket // longest subString with atm 2 distinct character.
5. 209 / Min size subArray sum.

Lect-6 (13/12/2020).

1. Kadane algorithm.
2. k-concatenation. maximum sum (1191).

Lect-7 (14/12/2020).

1. 1074 // number of submatrix that sum to Target
2. 363 // max sum of submatrix. rect. sum not larger than k.
3. max product.

APC

Date: /

Arrays/String Lect-1 9/12/2020.

agenda-

1. Separate +ve and negative numbers.] 4 (5) (same)
2. Separate 0 and 1
3. separate 0, 1, 2] same.
4. 75 Sort colors. S: O(1), T: O(n).
5. 283./ Move zeroes.
6. 189 / Rotate Array.
7. Max sum of array and its index.

APCO
Date: / /

① Arrays:-

① Array/string → stack/que. } Basic
→ HM./Heap/Rec. }

→ dp. } advance.
→ Recursion }

② Sorting.
③ Binary Search.
④ Rotation.

1. Que Seperate +ve and negative numbers.

what? given array with mixed +ve and negative seperate them.

I/P. = {12, 11, -13, -5, 6, -7, 5, -3, -6}.

O/P = {-13, -5, -7, -3, -6, 12, 11, 6, 5}.

Approach 1.

keep left and right pointer and set your region.
say left for → -ve
right for → +ve.

(+) (-) (+)

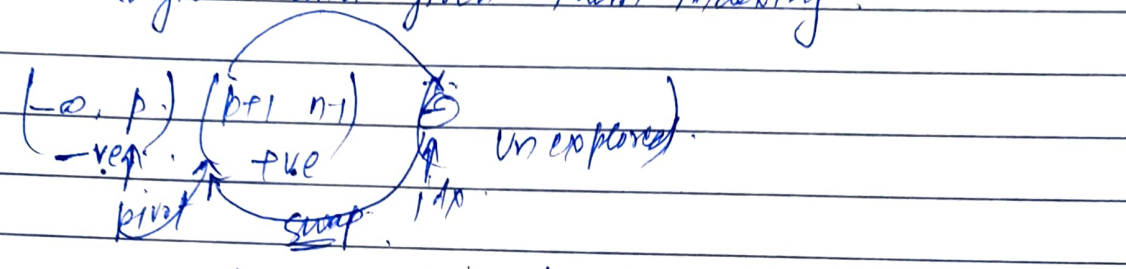
if get +ve on the left then left ++ when get -ve then swap
if +ve on right then right --
when -ve then swap

APCO
Date: / /

and swap (arr (left), arr (right)).

Approach 2:-

defined region and given them indexing.

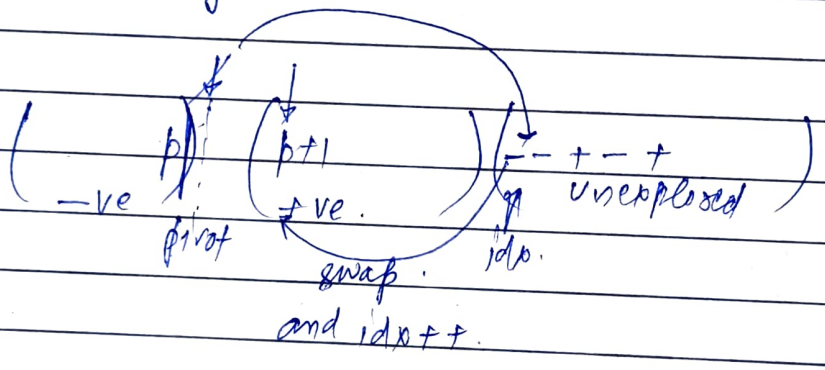


and -ve Region (yes).

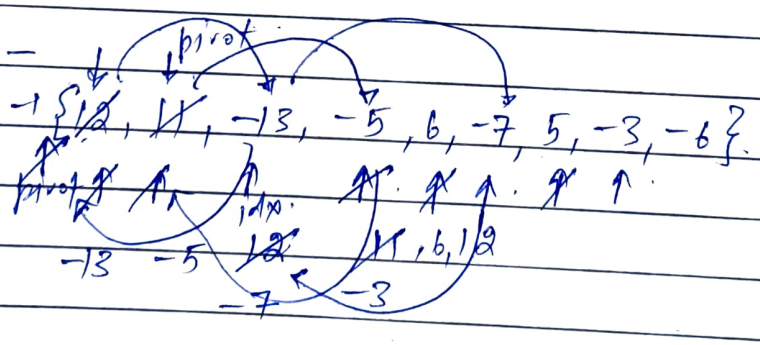
→ and +ve Region get a guranteed +ve so.

→ idx++.

→ Unexplored region yes.



dry Run:-



do this way.

APCO
Date: / /

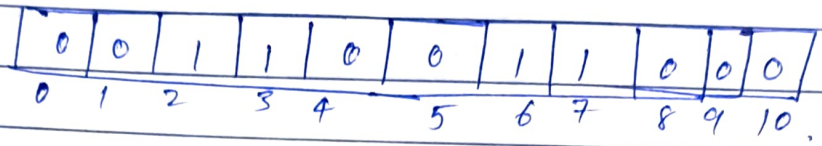
Question 1, 2, 3 and 5 done using this Method

Code.

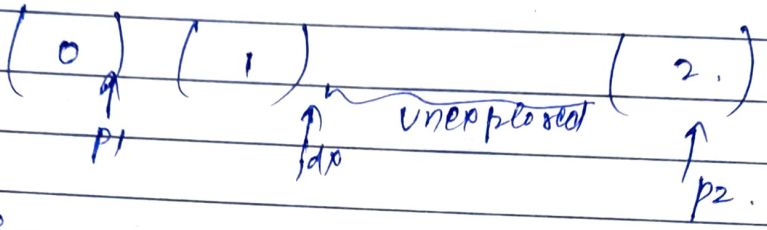
```
pivot = -1, idx = 0.
while (idx < n) {
    if (arr[idx] == 0) {
        swap(arr, pivot, idx);
    }
    idx++;
}
```

APCO
Date: / /

3. Que seperate 0, 1, 2 // lc-75



define Region:

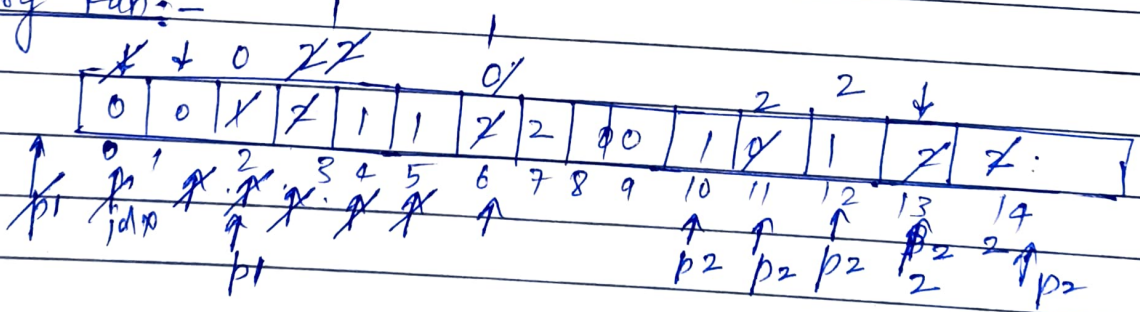


Algorithm:

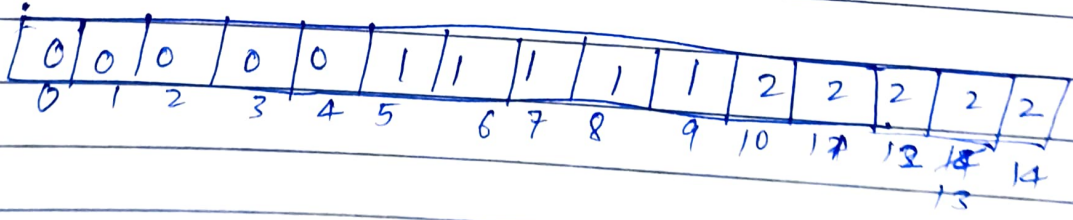
① if arr(idx) = 0 swap with p1 and ↑ idx because p1 is already ①
② if (arr(idx) = 2) swap with p2
but don't ↑ idx because don't know about p arr(p2) element.
③ arr(idx) = 1 then idx++;

↓ p2,

dry Run:-



we get



APCO
Date: / /

Code:

```
Public void sortColors(int[] arr){
    if(arr.length==0){
        return;
    int p1=-1, idx=0, n=arr.length, p2=n-1;
    while(idx<=p2){
        if(arr[idx]==0){
            swap(arr,++p1,idx++);
        else if(arr[idx]==2)
            swap(arr,idx,p2--);
        else idx++;
    }
}
```

APCO
Date: / /

⑥ Que. 189 Rotate Array.

What :- Rotate Array to right by k (k>0).

I/P = [1, 2, 3, 4, 5, 6, 7], k = 3.

O/P = [5, 6, 7, 1, 2, 3, 4].

How :- We do this in two steps.

first reverse $(0, n-k-1)$. and reverse $(n-k, n-1)$.

p_1' p_2'

Now Reverse $(p_1' \cdot p_2')'$

which will become nothing but.

$p_2 \cdot p_1$

$(0, n-k-1)$ & $(n-k, n-1)$.

eg. (4, 3, 2, 1, 7, 6, 5)

O/P (5, 6, 7, 1, 2, 3, 4).

eg. we have a b c d e

-1	b	c	d	e	a	→ same as k+l = -1+5 = ④
0	a	b	c	d	e	
1	e	a	b	c	d	
2	d	e	a	b	c	
3	c	d	e	a	b	
4	e	d	c	b	a	
5	a	b	c	d	e.	same.
6	e	a	b	c	d	

APCO
Date: / /

different cases for k.

c-1
① if k > a.l.
k = k % a.l.

c-2 if k < 0. (a = array size).
k = (k + a.length) % a.length.

Algorithm:
reverse (0, n-k-1) and rev(n-k, n-1).

① P1 and P2
② P1' and P2' (reverse individual).
③ P2'' and P1''
(P2 and P1) will be our (answer).

// code.

① Reverse. Public void (int[] arr, int i, int j) {
while (i < j) {
swap(i++, j--);
}
}
Public void swap(int arr[], int i, int j) {
int temp = arr(i);
arr(i) = arr(j).
arr(j) = temp.
}

APCO
Date: / /

```
Public void rotate( int[] arr, int k) {
    if(arr.length == 0) return;

    k = (k%n + n) % n; // it handle all k (+ve, -ve, all)
                       [0, n):
    int n = arr.length;
    revers (arr, 0, n-1);
    reverse(0, k-1);
    revers(k, n-1).

    }
}
```

if we do first reversal of (0, n) then. this algo work

if we do first part reversal then

① (0, n-k), (n-k, n-1).

② (0, n-1)'

APCO
Date: / /

7 Que: Max sum of $i * arr[i]$ among all rotation of a given array. (gfg)

What :-

arr[] = { 8, 3, 1, 2 }.

o/p = 29.

① { 8, 3, 1, 2 } = 8×0 + 3×1 + 1×2 + 2×3 = 11

② { 3, 1, 2, 8 } = 3×0 + 1×1 + 2×2 + 3×8 = 29.

③ { 1, 2, 8, 3 } = 1×0 + 2×1 + 8×2 + 3×3 = 27

④ { 2, 8, 3, 1 } = 2×0 + 8×1 + 3×2 + 1×3 = 17

Max(1, 2, 3, 4) = 29.

App-1 T : $O(n^2)$.

① Rotate array for all value 0 to n.
② Calculate sum for each rotation.
③ check if max sum is greater than current sum then update.

APCO
Date: / /

App-2 O(n)

Jump

We have a, b, c, d ⇒ a×0 + b×1 + c×2 + d×3.
→ b×0 + c×1 + d×2 + a×3.
or b×(1-1) + c×(2-1) + d×(3-1) + a×(4-1+4)
or (a×0 + b×1 + c×2 + d×3) − (a+b+c+d) + arr[i]×n

Eg. (a[0] + b[1] + c[2] + d[3]) − (a+b+c+d) + a[i]×4
Rotated sum — self sum

Algorithm :-
① calculate sum of all element
② Calculated value ×index for element
③ for first rotation sum.
rotated sum = (prev sum − sum + n×arr[i])

// Code

```
Public int maxSumInRotation(int arr[]) {
    int sum = 0, rotatedSum = 0, n = arr.length;
    for (int ele : arr) sum += ele;
    for (int i = 0; i < arr.length; i++) rotatedSum += arr[i]*i;
    int maxSum = rotatedSum;
    for (i = 0, i < n; i++) {
        rotatedSum = rotatedSum - sum + arr[i]*n;
        maxSum = max(rotatedSum, maxSum);
    }
    return maxSum;
}
```

APCO
Date: / /

Lecture-2 (Array) 06/12/2020

Agenda:

1. 3 // Longest substring without repeating character.
2. 76 // Minimum window substring.

3. Que: 3 // Longest substring without Repeat character.

what :- find the length of longest substring without Repeating character.

s: abc abc bb.
o/p = 3.
abc.

How :- These problem are like sliding window and done using 2 step.
take si, ei two pointers.

1. acquire till condition does not met (here count).
2. settle. till condition to make valid (here count).
3. release (↑ si)
   and count max length & set si and ei.

APCO
Date: / /

Dry Run:

si ↓
a b c a b c b b a b b c d e a
ei ↑

(to check repetation):
→ count = X ≠ 3
len =
n =
ei =

a - X ≠ 1
b - 1
c - 1

// Code:

```
Public int lengthOfLongestSubstring(String s){
    if(s.length() <= 1){
        return s.length();
    }
    int n = s.length(), si=0, ei=0, count=0, len=0;
    int maxSi = 0, MaxEi = 0;
    int[] map = new int[128];

    while(ei < n){
        if(map[s.charAt(ei++)]++ > 0) count++;
```

Note:-

1. Here all post ↑ run after in next line of if and evaluation done from right {.

means पहले count++ होगा,
then condition check
फिर si ↑
ei++ ↑ etc.

```
while ( count > 0 ){
    if (map[ s.charAt(si++) ]-- > 1) count--;
}

if( ei - si > len ){
    len = ei - si;
    MaxEi = ei;
    MaxSi = si;
}

return len;
}
```

(2) Qu 76. Leetcode-

APCO
Date: / /

② 76 // Minimum window Substring

what :- Return min window in 's' containing all character of 't'

I/P = ADOBECODEBANC t = ABC.

O/P = BANC.

How :- It is a kind of sliding window and work in ③ step.

① acquire, ② settle ③ Release.

Algorithm :-

① Make 4 variables. si, ei, head, length and requirement

② Make fq. map of array to store fq. of each character in string.

③ Now loop through the string.

① if. (freq(s(ei)) > 0) requirement --;

② while (require == 0) {

① length cal and updation.

② si को आगे, बढ़ाना है & check करना है whether it is part of answer or not

if not part the ↑ requirement.

APCO
Date: / /

dry Run.

S = O D D A B E C O D E B A N C. t = ABC.
0 1 2 3 4 5 6 7 8 9 10 11 12 13

A − 0.
B − 0.
C − 0.
D − 0 (-2).
E − 0 (-1).
N − 0
O. − 0 (-1)

head =
si =
ei =
len = 7
requirement = 3 2 0

fq map.

Here first acquired by ei then done settlement for min length.
then released when condition violated (req > 0).

APCO
Date: / /

Code

```
Public string minWindow (String s, string t){
    int ns = s.length();
    int nt = t.length();
    int si=0, ei=0, len = ns+1, head=0;
    int requirement = nt;
    int[] freq = new int[128];
    for(int i=0; i<nt; i++) freq[t.charAt(i)]++;

    while(ei<ns){
        if(freq[s.charAt(ei++)]-->0) requirement--;

        while(requirement==0){
            if(ei-si<len) len = ei-(head=si);
            if(freq[s.charAt(si++)]++==0)
                requirement++;
        }
    }

    return len == ns+1 ? "" : s.substring(head, head+len);
}
```

APCO
Date: / /

Lect-3 (Arrays). 8/12/2020.

Agenda.

1. Smallest window containing itself (gfg). } Same as 76.
2. 159 // locked
3. 340 // locked
4. Max Vowel letter in any substring 'k' with k length. (1456).
5. Count Unique substring with non-repeating char (gfg).
6. 239 / sliding window Max.

1. Q1. What Smallest window containing itself.

I/P = aabcbcdbca.
O/P = dbca.

How :-

Same as previous but here we keep ff. of each element one not more / less. because we want smallest window having all ~~repetation~~ characters & no repetation.

~~O/P~~ Same as 76 / prev.

APCO
Date: / /

dry run- abc bc cd bc a.

a - 1 0

b - 1 0

c - 1 0

d - 1 0

req = 4 - 1 = 3

do same-

Code:

```
public int minWindow(String s){
    int n = s.length();
    int[] freq = new int[128];
    for(int i=0 ; i<n; i++) freq[s.charAt(i)] = 1;

    int requirement = 0, si=0, ei=0, head=0, len=(int)1e8;
    for(int ele : freq) requirement += ele;

    while(ei<n){
        if(freq[s.charAt(ei++)]-- > 0) requirement--;

        while(requirement==0){
            len = (ei-si)<len ? len = ei-(head=si) : len;

            if(freq[s.charAt(si++)]++ == 0) requirement++;
        }
    }
    return len;
}
```

APCO
Date: / /

Q. 159 | Longest Substring with atmost two distinct character

What :- given a string 's' find the length of the longest substring 't' that contain atmost 2 distinct character.

I/P: eceba.
o/p: 3 → ece.

How: Algo: make a distinct count variable.
① if (freq of any char encounter == 0) then.
distinct count.

② while (distinctCount > 2) {.
① fq[char] --
② if (fq[char(ch)] == 1). then. distcount--.

③ Calculate length as well.

~~# Code~~

dryrun.

c c a a b b b a a c c c d d c d c d.
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17.

a - ✗ ✗ ✗
b - ✗ ✗ 3
c - ✗ ✗ ✗ 0 1
d -

distcount = ✗ ✗ ✗ ✗ 3.

similarly work goes on.

APCO
Date: / /

// Code:

Similarly for k distinct can can be done so do for k:

and check for k as well instead of '2'

k = 2

Code:

```
int n = s.length();
int si=0; ei=0, head=0, len=0, discount=0;
vector<int> freq(128,0);
while(ei<n){
    if( freq[s[ei++]]++==0)
        discount++;

    while( discount>k){
        if(freq[s[si++]]--==1)
            discount--;
    }

    len = (ei-si>len) ? ei-(head=si) : len;
}
return len;
}
```

APCO
Date: / /

④ Qn 1456 | Max vowel letters in Any substring s with. K length.

What :— Max Vowel in Window of K size.

s = abciijdef K=3.
o/p = 3

How :— do same as sliding window just make function to check vowel / not

① if vowel (then count++.

② during loop if. ei − si = K then. ~~↑ si and~~.
count max vowel.
↑ si and if s(si) is vowel.
then. vowel count − −

~~Dry Run~~.

si ei
a b c iij d e f.
0 1 2 3 4 5 6 7 8.
K=3 ei

VC = 0 1 2 2 3.

APCO
Date: / /

Code:

① bool isVowel(string s, int k){
    return ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u';
}

int maxVowels(string s, int k){
    int n = s.length();
    int si=0, ei=0, vc=0, maxVc=0;
    while(ei<n){
        if(isVowel(s[ei++]))
            vowelCount++;
        if((ei-si)==k)
            maxVowelsCount = max(maxVc, vc);
            if(isVowel(s[si++]))
                vc--;
    }
}
return maxVc;
}

APCO
Date: / /

(5) Qu. Count all unique substring with non-Repeating characters.

I/P = abba.

o/p = 4

a, b, ab, ba.

APCO
Date: / /

Leet 239. Sliding window Maximum.

Return max from 'k' size window.

nums = [1, 3, -1, -3, 5, 3, 6, 7] k = 3.
(0 1 2 3 4 5 6 7)

o/p = [3, 3, 5, 5, 6, 7].

How:

What Not do: Here I have thought k size के q. के top का element लेके आते but its wrong.

1, 3, -1 → Remove

but it will be part of next window.

Here poping done if (~~i-k~~ peek (index) <= i-k).

dry Run:

~~3 3 5 6 7~~

Ex: 1, ~~3~~, -1, -3, 5, 3, 6, 7

ans | 3 | 3 | 5 | 5 | 6 |

o(n log n) (for n element) to add

T: O(n log n)

APCO
Date: / /

Code:

```
Public int[] maxSlidingWindow(int[] nums, int k){
    if(nums.length == 1 || k == 1) return nums;

    PriorityQueue<Integer> pq = new PriorityQueue<>((a,b)->{
        return nums[b] - nums[a]; // acc. element based
    });                           - comparision (while index in pq.)

    int n = nums.length;
    int[] ans = new int[n-k+1];
    int idx = 0;

    for(int i=0; i<n; i++){
        while(pq.size() != 0 && pq.peek() <= i-k)
            pq.remove();

        pq.add(i);

        if(i >= k-1) ans[idx++] = nums[pq.peek()];
    }
}
```

for initial (k-1) will add into ans.

APCO
Date: / /

lecture-4 (9/12/2020).

Agenda:

1. Longest SubArray with equal no. of 0 and 1 } same ④
2. Leet code 525
3. Count SubArray with equal no. of 0 and 1 ⑤
4. Longest SubArray with sum divisible by k. } same ③
5. 974. subArray sum divisible by k.
6. 781 Rabbit in forest. ④

① Que: Longest SubArray with equal no. of 0 and 1:

eg.

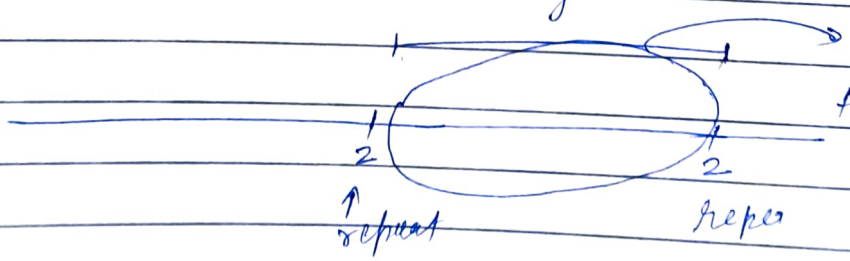
arr [] = { 1, 0, 1, 1, 1, 0, 0 }.

0 1 2 3 4 5 6.

o/p = 1 to 6.

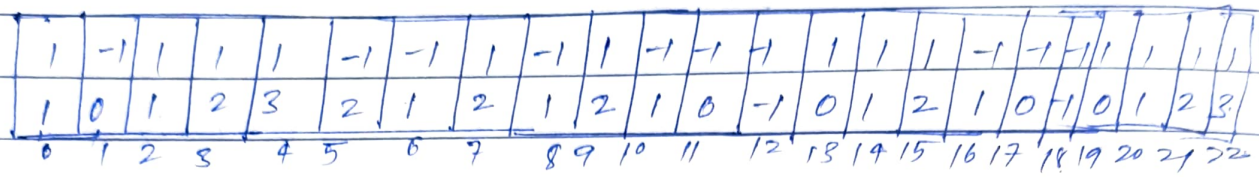
How:- Here we keep track of sum vs index.
if any sum get repeated that means.
इसको बीच के सारे वाले ⇔ till that element
का Contribution zero होगा.

eg.



APCO
Date: / /

eg.



Here we keep. sum of zero as -1 initially.
So that we have easy to calculate.
else have to manage count by adding (1).

eg for 0, 1 ⇒ 1 − 0 = 1 (len) ✗.

if 0 = -1 1 − (-1) = 2. ✓

psum.	idx.
0	-1
1	0
2	3
3	4
-1	12

len = ~~2 6 8 16 12 14 16~~
~~18~~ 20.

APCO
Date: / /

Code.

```
int maxLen(int[] arr, int N){
    if(N<1) return 0;

    HM<I,I> map = new HM<>();
    map.put(0,-1); // for easy calculation.
    int sum=0, len=0;
    for(int i=0; i<N; i++){
        int val = arr(i)
        if(val==0) val=-1;
        sum += val;

        if(map.containsKey(sum)) len= Max(len, i-map.get(
                                        sum));
        else map.put(sum,i);
    }
    return len;
}
```

APCO
Date: / /

(3) Count subArray with equal no. of 0 & 1

What :—

arr[] = {1, 0, 0, 1, 0, 1, 1};

o/p = 8.

(0,1), (2,3), (0,3), (3,4), (4,5), (2,5), (0,5), (1,6).

How :— → Here we will also do same as prev maintain psum Array.

→ make count of each sum and take all combination of ② eg $^{n}C_{2}$.

→ make defaul count of 0, as 1 as it will help in calculation.

or initial 0 की count 1 ही होगा है.

dryRun

0	+1	+1	+1	-1	-1	-1
1	+1	+2	+3	+2	+1	0.
	1	1	2	2	2	

0 — 2

1 — 2

2 — 2

3 — 0

∴ ans = $^{2}C_{2} + {}^{2}C_{2} + {}^{2}C_{2} = 3$.

APCO
Date: / /

// Code

```
int countEven(int[] arr, int N){
    if(N<=1) return 0;

    HM<I,I> map = new HM<>();
    map.put(0,1);

    int sum=0, count=0;
    for(int i=0; i<N; i++){
        int val = arr[i];
        if(val==0) val=-1;
        sum += val;

        count += map.getOrDefault(sum,0);
        map.put(sum, map.getOrDefault(sum,0)+1);
    }
/*  for(Integer key: map.keySet()){
        count += (map.get(key) * (map.get(key-1)))/2;
    }
*/
    return count;
```

APCO
Date: / /

(6) 781 // Rabbit in forest

What :- In a forest each rabbit has some color. Some subset of rabbit (possibly all of them) tell you how many other rabbit have the same colour as them. Those answer are placed in an array.

Return minimum number of rabbit that could be in the forest.

eg. I/P : [1, 1, 2].
O/P = 5.

How :- Here if 1 ⟶ means there could be map 2 Rabbit like o/ (itself) +1 and popped from map.

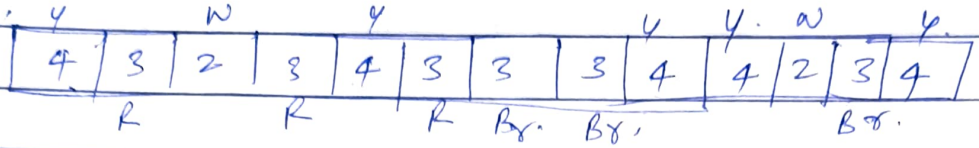
if 1 occurs 3 times then it will counted seperate.
↳ and remain till 2 in the map.

⟶ Note Any key remains in the map till its value <= key +1
if key +1 == remove it.
and insert again with 1 value.

ans = 2 + 3
= 5 Ans

X 1 — Y 2
2 — 1

APCO
Date: / /



4 → 5
3 → 4 (R).
2 → 3.
3 → 4 (Brown)

// Code

```
Public int numRabbits(int[] arr){
    if (arr.length == 0) return 0;
    int n = arr.length;
    HM<I,I> map = new HM<>();

    int ans = 0;
    for (int ele : arr){
        if (!map.containsKey(ele)){
            ans += ele+1;
            map.put(ele,1);
        } else {
            map.put(ele, map.get(ele)+1);
        }
        if(map.get(ele) == ele+1) map.remove(ele);
    }
    return ans;
}
```

APCO
Date: / /

(4) (5) Longest SubArray with sum divisible by k. (gfg)

What :- find longest subArray with its sum divisible by k

eg I/P = {2, 7, 6, 1, 4, 5}
k = 3.
O/P = 4 (size).

O/P = {7, 6, 1, 4}

Approach - 1

① Calculate the prefix sum.
sum(1, 4) = psum(4) - psum(0)
② Calculate ~~ps~~ sum of all subArray and compare. it will give all subarray sum. largest will be answer (index diff).

t : O(n^2).

dry run :-

{2, 7, 6, 1, 4, 5}

0	2	9	15	16	20	25
	1	2	3	4	5	6

Here we have shifted index to avoid diff calculation error.

eg. if not shifted psum(2, 4) = psum(4) - psum(1). (ei - si + 1)

now: psum(2, 4) = psum(4) - psum(1).
(ei - si) directly.

APCO
Date: / /

```
for(int i=0; i<n; i++){
    psum[i+1] = psum[i] + arr[i];
}

int len = 0;
for(int i=0; i<n; i++){
    for(int j=i; j<n; j++){
        if((psum[j+1]-psum[i])%k==0)
            len = max(j-i+1, len);
    }
}
return len;
```

→ ej − si directly
otherwise ej − (si−1) in normal case.

soln gives tle.

M-2 O(n) soln.

map of sum Vs index does not give us any thing but

eg.

sum	index.
2	
9	
15	
16	no meaning.
20	
25	

But if we store remainder but index.

then it gives some valuable information.

eg.

arr →	2	7	6	1	4	5
sum:	2	9	15	16	20	25
Rem.	2	0	0	1	2	1
	0	1	2	3	4	5

no addition

Repeatation of rem. show. there is not any valuable addition in given range.

and store 0 → (-1) 8

because we are assuming that our first 0 is at (-1) so that no need to add 'one' and easy calculation.

eg. if do not place (-1).

0 → + + + = 2.

zero first occurs at index 1
but length is 2 so need to add 1.
so. len = (1)
so need to add '1' (have to handle seperately)

if place 0 → -1 then just do simple calculation

0 → -1
1 → 3.
2 → 0.

if remainder has already occured then calculated len and compare from prev len).

APCO
Date: / /

Code :-

```
int LongestSubArraywithSumDivK(int arr[], int n, int k){
    if(n==0) return 0;
    int len = 0;

    unordered_map<int,int> map; // sum, index
                                // rem.
    map[0] = -1;

    int sum = 0;
    for(int i=0; i<n; i++){
        sum += arr[i];
        int SUM = (sum%k+k)%k;
        if(map.find(SUM) != map.end())
            len = max(len, i - map[SUM]);
        else map[SUM] = i;
    }

    return len;
}
```

Qu 974 // leetCode -

count all subArrays with sum divisible by k.

A = { 4, 5, 0, -2, -3, 1 } k = 5.

O/P = 7

How :-

APCO
Date: / /

M-1 Here we are first counting then removing that remainder count

M-2 Or we can do here the $^{n}C_{2}$ for that particular sum.

arr	4	5	0	-2	-3	1
psm	4	9	9	7	4	5
rem	4	4	4	2	4	0

, k = 5.

0 ⟶ 0 + 1 = 2.

$^{4}C_{2} = \frac{4 \times 3}{2} = \frac{12}{2} = 6.$

$^{2}C_{2} = \frac{2 \times 1}{2} = 1 = 1$

total = 7 | or Count at every level.

Code

```
Public int SubArraysDivByK( int[] arr, int k){
    if(arr.length == 0) return 0;
    int[] map = new int[k+1];
    map[0] = 1;  // Count if rem = map[0]=1

    int sum =0, int count= 0;
    for(int i= 0; i< arr.length; i++){
        sum += arr[i];
        int rem = (sum%k + k)%k;
        count += map[rem];
        map[rem]++;
    }
    return count;
}
```

APCO
Date: / /

Lect - 5 (11/12/2020)

Agenda -

1. 930 | Count Binary SubArray with Sum equal to K
2. 1248 | Count no. of nice subArray.
3. 1004 | Max Consecutive one - III
4. 904 | fruit into basket
5. 209 | Min size subArray Sum.

APCO
Date: / /

1. Que what?

An array of 0 and 1 how many non-empty subArray have sum s.

A = [1,0,1,0,1] s = 2.

O/P →

[1,0,1], (1,0,1,0), (0,1,0,1), (1,0,1).

How :— → Here just the array and make map of sum vs freq count.

→ if a sum not present then put, that and value 1. and ↑ment it.

→ ~~If~~ to add. Check whether sum is present or not just check.

(sum − k) of freq. into answer.

arr(ei) − arr(si) = k.

arr(ei) − k = arr(sj).

↑ sum (at ei index). ↑ starting index.

dry run :—

1, 0, 1, 0, 1

✗ ✗ ✗ ↑

k = 2

0 → 1	sum = ✗
1 — ✗ 2	1 − k → not contain.
2 — ✗ 2	2 − 2 → contain ↑1 = 1
3 — 1	3 − 2 → 2

APCO
Date: / /

// code

```
Public int numSubArraysWithSum(int[] arr, int s){
    int n = arr.length;
    if(n==0) return 0;

    int ei=0, count=0, psum=0;

    // int[] freq = new int[3000+1];
    // freq[0]=1

    HashMap<I,I> map = new HashMap<>();
    map.put(0,1);
    while(ei<n){
        psum += arr[ei++];
        // if(psum-s>=0) count += freq[psum-s];
        count += map.getOrDefault(psum-s,0);
        map.put(psum, map.getOrDefault(psum,0)+1);
        // freq[psum]++;
    }
    Return count;
}
```

APCO
Date: / /

My-2. Here fun calculate subarray with sum less than k/s
then take diff of
all subArray of (s) - all subArray of (s-1). Ans.

// code.

```
Public int numSubArrayWithAtmostSum(int[] arr, int s){
    int si=0, ei=0, count=0, sum=0, n=arr.length;

    while(ei<n){
        sum += arr[ei++];

        while(sum>k){
            sum -= arr[si];
        }
        count += ei-si;
    }

Public int numSubArraywithSum(int[] arr, int s){
    int n = arr.length;
    if(n==0) return 0;
    return numSubArray(arr,s) -
        (s>0 ? fn(arr, s-1) : 0);
```

APCO
Date: / /

2. Qn 1248. Count Number of Nice SubArrays

what

given array of integer nums. and integer k.
A subArray is called nice if there are 'k' odd numbers in it.
find numbes of nice sub-Arrays.

I/P = {1,1,2,1,1} k = 3.
o/p = 2.

How :- do same as done in previous.
① Count all subArrays with atmost 'k' odd numbers. here len then 'k' are also included.
② Count all subArrays with 'k-1' odd.
③ subtract ① − ②

// dryRun.

(2,2,2,1,2,2,1,2,2,2) k=2
odd count = 2

now. subtract (k-1) odd subArray

total subArray len than (k).
res = 1+2+3+4+5+6+7+8

2
2, 22
2, 22, 222
1, 21, 221, 2221
2 22 12, 2212, 212, 12, 2
22, 121, 22121, 2121, 121, 21, 1

APCO
Date: / /

// code:

```
Public int numberOfSubArrayAtmost(int[] arr, int k){
    int n = arr.length;
    int si=0, ei=0, oddCount=0, res=0;

    while(ei<n){
        if(arr[ei++]%2 != 0) oddCount++;

        while(oddCount>k){
            if(arr[si++]%2 != 0) oddCount--;
        }

        res += ei-si;
    }

    return res;
}

Public int numbers of subArrays(int[] arr, int k){
    int n = arr.length;
    if(n==0) return 0;

    return f(arr,k) - f(arr,k-1);
}
```

APCO
Date: / /

S.Qus 1004/ max Consecutive one - III

What? given Array of 0 and 1 we may change upto k value from 0 and 1
return length of longest subArray that contain only 1.

I/P = { 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0 } k = 2.

How:- ① make a zero count and sum loop from left make 2 pointer si and ei.
② if zero count > k then si++.
③ now calculate the len if max then update.

dryRun:

1 1 0 1 0 1 0 1 0 0 0 1 0 1
0 1 2 3

k = 3.

zeroCount = X ≠ 3

len = X ≠ 3 ≠ 5 ≠ 6 ≠ 8 9

APCO
Date: / /

Code-

```
Public int longestOnes(int[] arr, int k){
    int n = arr.length, ei=0, si=0, len=0, zcount=0;

    while(ei<n){
        if(arr[ei]==0) zcount++;

        while(zcount >k){
            if(arr[si++]==0) zcount--;
        }
        len= Math.max(len, ei-si);

    }
    return len;

}
```

APCO
Date: / /

Q. Min Size Subarray with Sum greater equal to k

Sum >= k/s.

s = 7

nums = [2, 5, 1, 2, 4, 3].

o/p = 2. [4, 3]

APCO
Date: / /

Lecture-6 (13/12/2020)

Agenda-

1. kadane algorithm / for maximum subArray sum.
2. 1191 - k Concatenation maximum sum.

1. Que Maximum SubArray Sum. (Kadane algorithm).

2	3	-4	1	-7	4	-1	-2	1	5	-3	6	-10

How :-

1. Maintain two variable ~~gsum~~, gmaxSum, and runningSum.
2. add element into runningSum till >=0 once it is less than '0' set it to '0'
3. Set gmaxSum, ~~Gmaxsum~~ by a max of runningSum and gmaxSum.

dry Run -

2, 3, -4, 1, -7, 4, -1, -2, 1, 5, -3, 6, -10.

gSum = 5, 9

runSum = 2 5 5 -1 0 0 4, -1 -2 0 1, 6 3 9

APCO
Date: / /

→ Here in this method if element is negative then set arr sum '0'

→ but in generic method if -ve element occurs then set that to that value.

// Code:-

```
Public static void KadaneAlgo(int[] arr){
    int gmaxSum = -(int)1e8, runningSum=0;
    for(int ele : arr){
        runningSum += ele;
        if(runningSum > gmaxSum)
            gmaxSum = runningSum;

        if(runningSum <=0)
            runningSum =0;
    }
    return gmaxSum;
}
```

APCO
Date: / /

Return ther g max sum subArray.

→ for that keep two extra variable
gsi, gei, (global starting index,
rsi (running of starting index to keep update temparorily).
and rei is keep on going
if runningSum > gmaxSum then update
→ gsi, gei → rei.
↳ rsi
update them.

// code:

```
Public static void KadanAlgoSubArray (int[] arr){
    int gMaxSum = -(int)1e8, runningSum = 0;
    int gsi = 0, gei = 0, rsi = 0; // gei = global si
    // running si = rsi.          // gei = global ei;

    for(int rei = 0; rei < arr.length; rei++){
        runningSum += arr[rei];
        if(runningSum > gmaxSum){
            gmaxSum = runningSum;
            gsi = rsi;
            gei = rei;
        }

        if(runningSum < 0){
            runnnysum = 0;
            rsi = rei+1;
        }
    }
    return gmaxSum;
}
```

APCO
Date: / /

T-2 Kadane algo

This algo set that -ve value as max arr sum if all are negative element but not '0' at as prev.

```
Public static long KadaneAlgoGeneric(int[] arr){
    long gmaxSum = arr[0], runningsum = arr[0];

    for(int i=1; i<arr.length; i++){
        runningSum = max(arr[i], arr[i]+runningSum);
        gmaxSum = max(runningsum, gmaxSum);
    }
    return gmaxSum;
}
```

APCO
Date: / /

2 Que. 1191. K-Concatenation maximum Sum.

What? given an integer arr and integer 'k' modify array by repeating it 'k' times.

eg arr = [1, 2], k = 3.

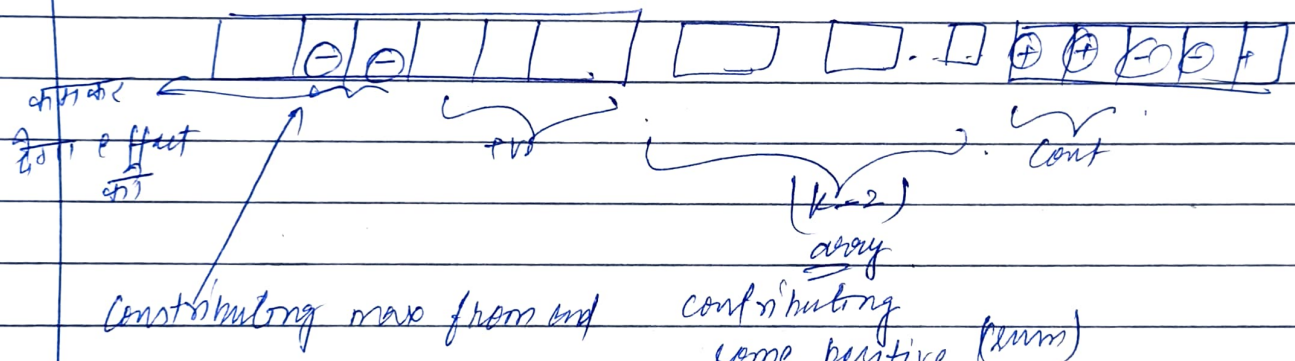
[1, 2, 1, 2, 1, 2].

max sub array sum = 9.

How :-

M-1. Here arises two situation if (arr sum > 0) and k-2 > 0

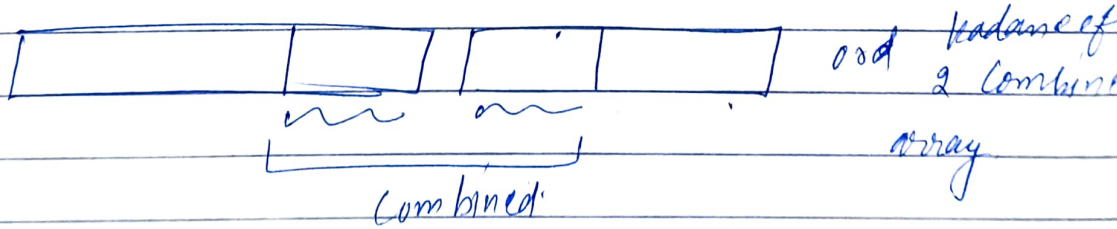
then situation will be like.



C-2 if arr sum < 0

then.

answer will be



APCO
Date: / /

c-2

if $k=1$ ans will be kadane of arr.

eg.

| 2 | 5 | -8 | 2 | 3 | -3 |

, $k=4$.

c-1 kadane Sum = 7.

prefix sum = 7.

suffix sum = 2

arr sum = 1

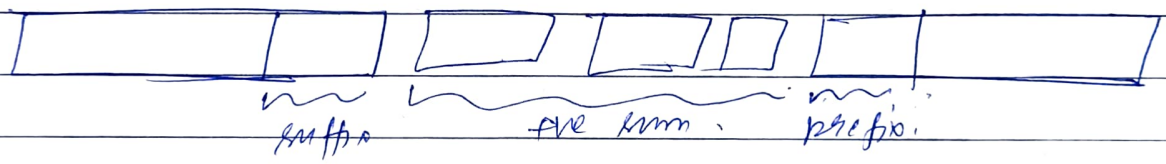
arr sum > 0.

ans = ps + ss + (k-2)×1

= 7 + 2 + (4-2)×1

= 9 + 2 = 11.

situation is like -



c-2 if

arr Sum < 0

| 2 | 5 | -10 | 2 | 3 | -3 |

, $k=4$

arr sum = -1

kadane Sum = 7

psum = 7

ssum = 2.

ans = max (ps + ssum, ks)

= (7, 9)

= 9 Ans.

APCO
Date: / /

Code M-1

long int mod = (int) 1e9 + 7;

① Kadane Algo. (int[] arr){.

int gmaxSum = 0, runningSum = 0;

for (int ele : arr){.

runningSum += ele;.

if (runningSum > gmaxSum) gmaxSum = runningSum;

if (runningSum < 0) runningSum = 0;

}.

return gmaxSum;.

}.

② prefixSum.

Public long prefixSum(int[] arr){.

long gSum = -(int) 1e9;.

long cSum = 0;

for (int ele : arr){.

cSum = (cSum + ele) % mod;.

gSum = max(cSum, gSum);.

}.

return gSum;.

}.

APCO
Date: / /

③ // Sum from last

```
Public long suffixSum(int[] arr){
    long gsum = -(int)1eq;
    long csum = 0;

    for(int i = arr.length-1; i>=0; i--){
        csum = (csum + arr[i]) % mod;
        gsum = max(gsum, csum);
    }
    return gsum % mod;
}
```

④

```
Public int k_concatenation MaxSum(int[] arr, int k){
    long ksum = KadaneAlgo(arr) % mod;
    if(k==1) return (int) kadane sum;

    long SumofArray = 0;
    for(int ele: arr) = sumofArray += ele;

    long prefixSum = prefixSum(arr);
    long suffixSum = suffixSum(arr);

    if(sumofarr > 0){
        long Apsum = ((k-2)* sumofarr + ps + ss) % m
        return (int) max(apsum, ksum);
    }
    else{
        return (int)(max(psum + ssum, kadane sum));
```

APCO
Date: / /

M-2.

what we are doing is: doing same thing, using one for loop till $k=3$. $(k>3)$.

Here prevsum = suffixsum + prefix sum.

How:
↳ it is kadane of two array and always become equal to individual sum.

eg.

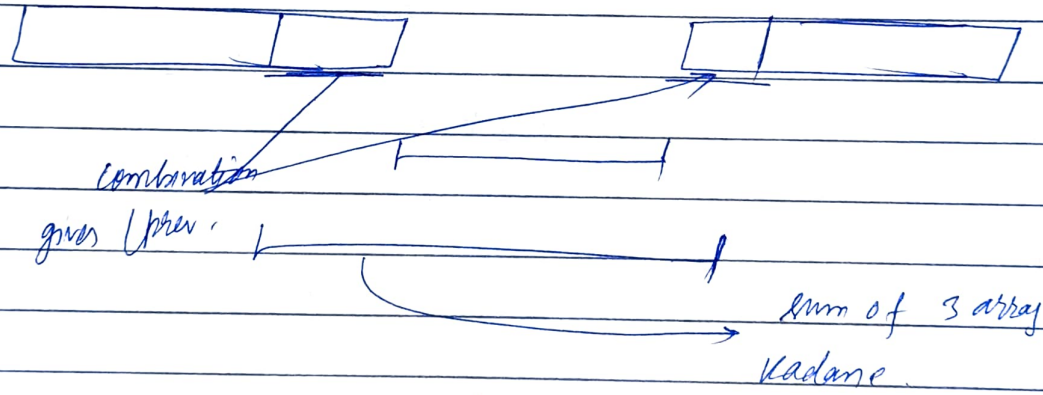
C-1.

2	5	-8	2	3	-3		2	5	-8	2	3	-3

ksum = 9. } = 9. } equal.
ss = 2.
ksum of two (prevsum) = 9.

C-2 and kadane of 3 array - prev (ss+ps) give sum of one array.

How:



so one array = kadane sum - prev

APC
Date: /

Lecture-7 11/12/2020

Agenda:

① 1074 - Number of submatrix that sum to target
② 363 - max sum of rectangle sum no larger than k.
③ 152 - Max product subArray

APCO
Date: / /

Leet 1074. Number of submatrices that sum to target

what? given a matrix and a target return number of non-empty submatrix that sum to target

eg

0	1	0
1	1	1
0	1	0

target = 0

o/p = 4.

How: Algorithm.

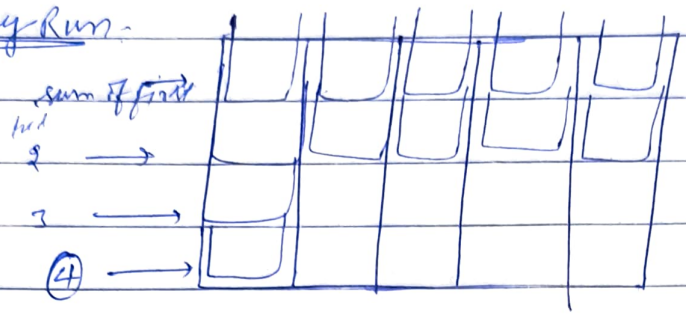
① do column wise sum and store them at each cell from top to bottom.
② Run a nested loop on row and column.
③ for each Row treat them as single row and count all sub-Array divisible by k.

Alternate :-

① we can do it as from bottom to top, left to right / right to left as well.
② do further steps accordingly

APCO
Date: / /

dry Run:-



set a base at top and slide them at one 1 after each iteration and if (base != 0) then subtract prev cell pe value from current

eg:-

b →

1	2	3	4	5
6 (6-1=5)	7 (7-2=5)	8 (8-3=5)	9 (9-4=5)	10 (10-5=5)
11 (11-1=10)	12 (12-2=10)	13 (13-3=10)	14 (14-4=10)	15

similarly do for other base when shift them.

APCO
Date: / /

// Code :-

```
Public int submatrixSumTarget(int[][] matrix, int target) {
    int n = matrix.length, m = matrix[0].length;

    for(int row = 1; row < n; row++){
        for(int col = 0; col < m; col++){
            matrix[row][col] += matrix[row-1][col];
        }
    }

    int count = 0;
    for(int base = 0; base < n; base++){
        for(int row = base; row < n; row++){
            HashMap<I, I> map = new HM<>();
            map.put(0,1);
            int sum = 0;
            for(int j=0; j<m; j++){
                sum += matrix[row][j] - (base != 0)
                    matrix[base-1][j] : 0);
                count += map.getOrDefault(sum-k, 0);
                map.put(sum, map.getOrDefault(sum, 0)+1);
            }
        }
    }
    return count;
}
```

APCO
Date: / /

2. Que. Max Sum of Rectangle Sum no larger than K.

what :-

given non-empty 2-d matrix and Integer 'K' find max. sum of Rectangle in matrix. Sum not larger than k.

1	0	1
0	-2	3.

$k = 2$

o/p = 2.

Rectangle means submatrix.

How :- Here do the same as previous qn.

same ?

① Calculate ars sum of each cell and put into cell.

② Make loop for base and sum row no. of time..

different ?

Here Have to Calculate Sum not greater than k.

if (Sum = K) found then Returen.

other wise. Ceil

floor ↪ val will be ceil

and max Sum = max(max Sum, Sum-Val)

APCO
Date: / /

→ Here instead of HM, tree map used because it is capable to give ceil or floor if value not found in map.

2, 4, 8, 10, 15

(A)

note - ceil of (16) will be null in ceiling function.

dryRun -

1	2	2	1	4	2	1	3	5

0	1	3	5	6	10	12	15	16	21

k = 11

sum = 1

val = 0 0 1

max = X X X 0 10

(10, 11) → 11

APCO
Date: / /

// code.

```
Public int maxSumSubmatrix(int[][] matrix, int k){
    int n = matrix.length, m = matrix[0].length;
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            matrix[i][j] += matrix[i-1][j];
        }
    }

    int maxSum = -(int)1e9;
    for(int base=0; base<n; base++){
        for(int row=base; row<n; row++){
            int gsum = -(int)1e9, rsum=0;
            for(int j=0; j<m; j++){
                int val = matrix[row][j] - (base!=0?
                    matrix[base-1][j]:0);
                rsum = max(val, rsum+val);
                gsum = max(gsum, rsum);
                if(gsum==k) return k;
            }
            if(gsum<k){
                maxSum = max(maxSum, gsum);
                continue;
            }
        TreeSet<Integer> map = new TreeSet<>();
        int sum=0;
        map.add(0);
```

part of hardone and make code faster

APCO
Date: / /

original code:

```
for (int j=0; j<m; j++){
    sum += matrix[row][j] - (base!=0? matrix[base-1][j]
                                    [j]:0);
    if(map.contains(sum-k)) return k;

    Integer val = map.ceiling(sum-k);
    if(val != null){
        maxSum = max(maxSum, sum-val);
    }
    map.add(sum);
}
```