Todays Content:

1. length of longest subarrays with sum = 0;
2. longest subarray with equal 1's & 0's.
3. Count of subarrays with sum = 0;

18. Find the length of longest Subarray with sum = 0:

	0	1	2	3	4	5	6	7	8	9	10	11	12		
Ex: ar[] = {	3	3	4	-5	-2	2	1	-3	3	-1	5	-4	-1	}	len=10

#idea:

Generate all subarrays & calculate sum & check if sum==0 & get max len

Way1: TC: $O(N^3)$

```
ans = 0;
s=0; s<N; s++) {
    e=s; e<N; e++) {
        # [s.. e];
        sum = 0;
        i=s; i<=e; i++) {
            sum = sum + ar[i]
        }
        if( sum == 0) {
            ans = max(ans, e-s+1);
        }
    }
}
return ans;
```

Way2: TC: $O(N + N^2 * 1) = O(N^2)$ SC: $O(N)$

```
Generate pf[N];
ans = 0;
s=0; s<N; s++) {
    e=s; e<N; e++) {
        # [s.. e];
        sum = 0;
        if(s==0) { sum = pf[e] }
        else { sum = pf[e] - pf[s-1] }
        if( sum == 0) {
            ans = max(ans, e-s+1);
        }
    }
}
return ans;
```

#idea2: Apply pf[] & for every element store it's 1st occuren

if pf[i] == x & pf[j] == x: sum[i+1.. j] = 0 : len = j - i

	0	1	2	3	4	5	6	7	8	9	10	11	12
ar[] = {	3	3	4	-5	-2	2	1	-3	3	-1	5	-4	-1 }
pSum[] = {	3	6	10	5	3	5	6	3	6	5	10	6	5 }

obs:

Tracing

<3,0>
<6,1>
<10,2>
<5,3>

ele	ind	1st occu	l = ind - 1st occu	m
3	4	0	l = 4 - 0 = 4	4
5	5	3	l = 5 - 3 = 2	4
6	6	1	l = 6 - 1 = 5	5
3	7	0	l = 7 - 0 = 7	7
6	8	1	l = 8 - 1 = 7	7
5	9	3	l = 9 - 3 = 6	7
10	10	2	l = 10 - 2 = 8	8
6	11	1	l = 11 - 1 = 10	10
5	12	3	l = 12 - 3 = 9	10

Edge Case:

```
              0    1    2    3    4
arr[] =    {  4   -3   -1    2   -2 }

Psum[5] = 0 { 4    1    0    2    0 }
         -1
```

Tracing	ele	ind	1st occ	l = ind - 1st occ	m
<4,0>	0	4	2	l = 4 - 2 = 2	2.
<1,1>					
<0,2>					
<2,3>					

Issue: Above trace is not working Expected ans = 5.

why? For 0 at 4th Index we compare with 0 at 2nd Index, It is wrong because we have a 0, before starting pf(). Assume 0 at -1st Index

How to handle edge case:

```
              0    1    2    3    4
arr[] =    {  4   -3   -1    2   -2 }

Psum[5] = 0 { 4    1    0    2    0 }
```

Tracing	ele	ind	1st occ	l = ind - 1st occ	m
<0,-1>	0	2	-1	l = 2 - (-1) = 3	3
<4,0>	0	4	-1	l = 4 - (-1) = 5	5
<1,1>					
<2,3>					

Note: When we take pf(), always assume there is a zero at start = 0.

```
int maxlength( vector<int> &arr){ TC: O(N+N) = O(N)  SC: O(N+N) = O(N)

    long pf[N];
    long sum=0;
    for( int i=0; i<N; i++){
        sum = sum + arr[i])
        pf[i]= sum;
    }

    unordered_map<long, int> um;
    um[0]=-1;   # <0,-1>
    int ans=0;
    for( int i=0; i<N; i++){
        if( um.find(pf[i]) == um.end()){
                um[pf[i]] = i;
        }
        else{
                # pf[i] repeating
                int l= i- um[pf[i]]
                ans= max(ans,l);
        }
    }
    return ans;
}
```

TODO: Count of Subarrays with sum=0

28: Given an array contains only 0's & 1's find max length subarray which contains equal 1's & 0's

Ex:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
arr[] = {	0	0	1	0	1	0	1	1	1	0	1	0	0	1	0	0	0	1	1	0 }

Q: Max len subarray with equal 1's & 0's

Hint: Replace 0's with -1;

Q: Max len subarray with equal <u>1's & -1's</u>

Q: Max len subarray with sum = 0;

Solution: 1. Replace all 0's with -1;

2. In update arr[]: Calculate length of longest subarray with sum = 0

Count Pair Sum:

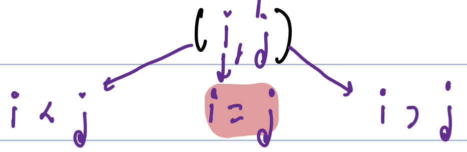
Given an ar[N] & k.

Count no: of pairs (i, j) such that ar[i]+ ar[j] = k && i! = j && (i, j) = (j, i)

	0	1	2	3	4	5	
Ex1: ar[ ] = {	7	3	2	3	7	8	}

k=10 : (0, 1) (0, 3) (1, 4) (2 5) (3 4)

Idea1: Generate all pairs & calculate sum, if sum==k: Inc cnt



Note: while generating all pairs either generate i<j or i>j

TC: O(N^2) SC: O(1)

```
int pairSum( vector<int> &ar, int k ){
    int c=0;   # i>j
    for( int i=0; i< ar.size(); i++){
        for( int j=0; j<i; j++){ # j: [0.. i-1]
            if( ar[i]+ ar[j]==k){
            }   c++;
        }
    }
    return c;
}
```

Dry Run:

ar[i]+ar[j]==k && i>j:

k=10	0	1	2	3	4	5
ar[ ] = {	7	3	2	3	7	8 }
cnt 3 = 0	i					
cnt 7 = 1		i				
cnt 8 = 0			i			
cnt 7 = 1				i		
cnt 3 = 2					i	
cnt 2 = 1						i

ans = 5.

Obs: For every ar[i]:

Count frequency of k-ar[i] in left of i : [0..i-1]

j

Opt: Using Hashmap

At ar[i]: We only search from [0.. i-1]

Note: At i^th Index: Hashmap should only contain ele from [0.. i-1]

Dry Run:

```
                 0   1   2   3   4   5
k=10 ar[6] = {   7   3   2   3   7   8  }

Target =         3   7   8   7   3   2
HashMap          0   1   0   1   2   1  = 5.
```

<7:2>
<3:2>
<2:1>
<8:1>

```
int countSum(vector<int> &ar, int k){ TC: O(N) SC: O(N)
    unordered_map<int, int> um;
    int c=0;
    for( int i=0; i<ar.size(); i++){
        if( um.find( k-ar[i]) != um.end(){
            c= c+ um[k-ar[i]];
        }
        #Insert ar[i];
        um[ar[i]]++;
    }
    return c;
}
```