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 sums & ==0 & get overall max subarray length.

```
Way1: TC: O(N3) SC: O(1)
int 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;
```

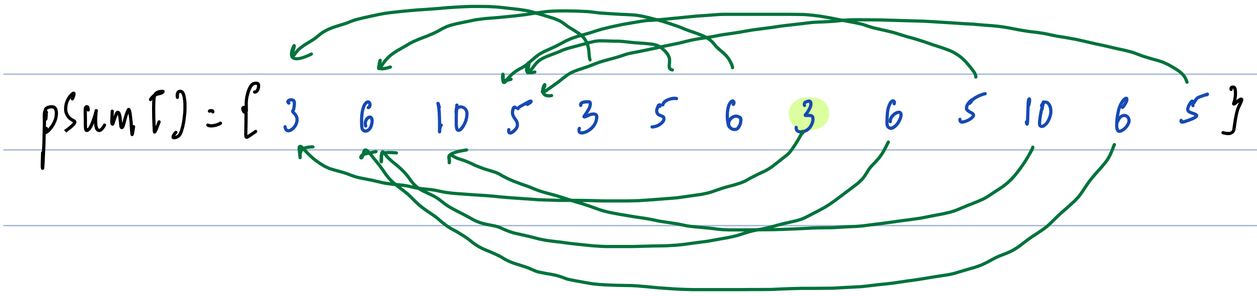
```
TC: O(N2) SC: O(N)
int ans=0;
Create pf[N];
s=0; s<N; s++){
    e=s; e<N; e++){
        # [s.. e)
        sum=0;
        if(s==0){ sum= Pf[e] }
        else{ sum = Pf[e]- Pf[s-1] }
        if( sum==0){
        }   ans = max(ans, e-s+1);
    }
}

return ans;
```

#idea2: Apply pf[]

a b b-a+1

pf[i] == pf[j] #sum[i+1..j] = 0 ; len = j-(i+1)+1 = $\underline{\underline{j-i}}$

	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 }

$\underline{\underline{\text{obs}}}$: Every pf[i]; We compare it with it's 1st occurrence.

To optimize we store $\underline{\underline{\text{1st occurrence}}}$ of each element in hashmap.

Dry Run:

	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 }

Tracing

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

ele	ind	1st occ	l: ind - 1st occ	max
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 return max = 10

Edge Case:

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

Tracing	ele	ind	1st occ	l = ind - 1st occ	m
{4:0}	0	4	2	l = 4 - 2 = 2	2 ≠ Expected 5?
{1:1}					
{0:2}					
{2:3}					

Reason?

0 at 4th index we are comparing with 0 at 2nd index, which is wrong, we already have a 0 at start before pf[], we assume that 0 is at index -1

How to handle edge case: Insert {0, -1} in hashmap

			0	1	2	3	4	
arr[] =		{	4	-3	-1	2	-2	}
Psum[] =	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 do subarray sum related problems using pf[], keep in mind that, there is 0 at start itself # Need it to avoid edge cases.

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

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

    unordered_map<long, int> hm;
    hm[0] = -1;
    int ans = 0;
    for( int i=0; i<N; i++){
        if( hm.find(pf[i]) == hm.end()){
                hm[pf[i]] = i;
        }
        else{              1st occu of pf[i];
            int l = i - hm[pf[i]];  # i: current index of pf[i]
            ans = max(ans, l)
        }
    }
    return ans;
}
```

2Q: 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} l = 18

Q: Calculate length of longest subarray with equal 1's & 0's

Hint: Replace all 0's with -1

Q: Calculate length of longest subarray with equal 1's & -1's

sum = 0

Q: Calculate length of longest subarray with sum = 0.

Sol:

1. Replace 0 with -1's
2. On 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)

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

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

Idea: Generate all pairs & calculate sum & if sum == k : c++

(i, j) → (i > j)   (i == j)   (i < j) : (i > j) & (i < j) are same pairs

#Generating all pairs (i > j)

```
int pairSum(vector<int> &ar){ (i > j)
    int N = ar.size();
    int c=0;
    for( int i=0; i<N; i++){
        for( int j=0; j<i; 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

cnt 7 : 1

cnt 8 : 0

cnt 7 : 1

cnt 3 : 2

cnt 2 : 1   ans = 5

#Obs: For every ar[i]:

Count frequency of k - ar[i] on left of i = [0..i-1]

Opt: Use hashmap:

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

Note: At i^{th} index Hashmap should only contains elements from [0..i-1]

Dry Run:

	0	1	2	3	4	5	
k=10 ar[6] = {	7	3	2	3	7	8 }	
	i	i	i	i	i	i	i : return cnt=5
Target =	3	7	8	7	3	2	
cnt =	0	1	0	1	2	1	

HashMap

<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> hm;
    int c=0;
    for( int i=0; i< ar.size(); i++){
        # ar[i] : Target k-ar[i];
        if( hm.find(k-ar[i]) != hm.end()){
            c= c+ hm[k-ar[i]];
        }

        # Insert ar[i] in hm;
        hm[ar[i]]++;
    }
    return c;
}
```