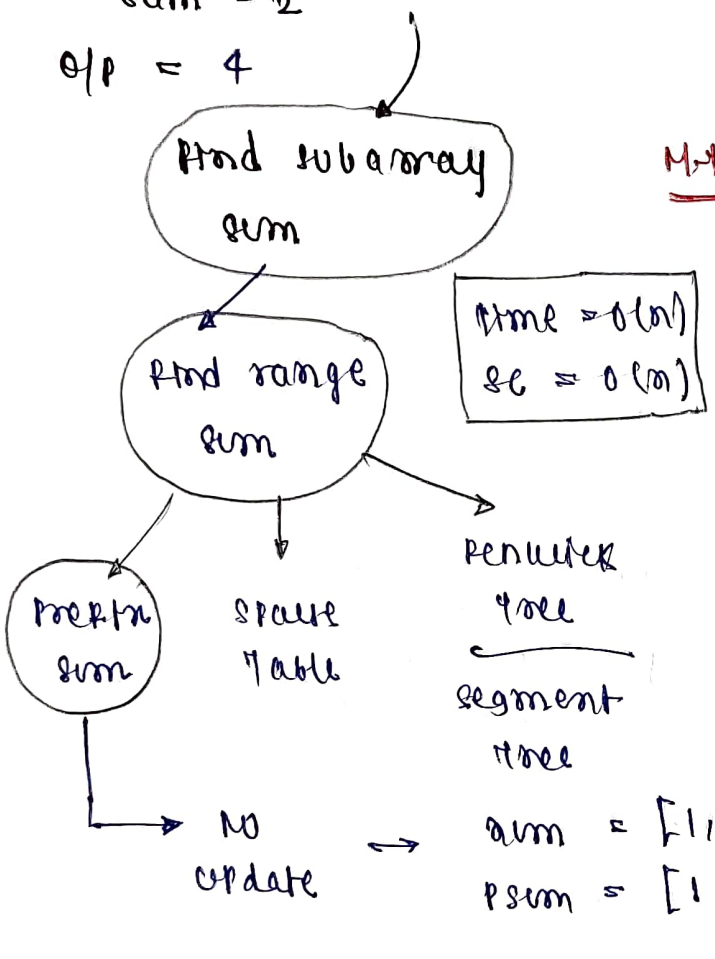
Two Pointer

1 Binary subarrays with sum

Eg. nums = [1,0,1,0,1]
sum = 2
O/P = 4

M-I Find out all subarrays $O(n^2)$
Find sum of that subarray $O(n)$
$O(n^3)$.

M-II



nums = [1,2,3] sum += 1
nums = [1,2,3] sum += 2
nums = [1,2,3] sum += 3

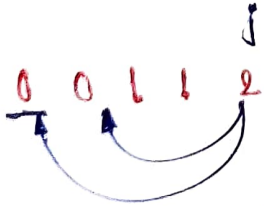
Each subarray is found simultaneously so $O(n^2)$.

arr = [1,2,3] and goal = 3.
psum = [1,3,6].

Eg. num[] = 0 1 0 1 0
psum[] = 0 0 1 1 2 2 3

- range sum = psum[j] − psum[i−1] = goal.
  - psum[j]: You at right now
  - goal: known
  - So, psum[i−1] = psum[j] − goal.

So,



- Ek zero extra add kr lenge or sath me sath jaha tak j pahuchega waha se pehle same element ko map me v dalte jayenge.

Eg 1 0 1 0 1 goal = 2
psum 1 1 2 2 3
total = 4

ans = ~~0~~ ~~1~~ ~~2~~ 4
psum = ~~0~~ ~~1~~ ~~2~~ ~~2~~ 3

mpp
0 → 1
1 → ~~1~~ 2
2 → ~~1~~ 2
3 → 1

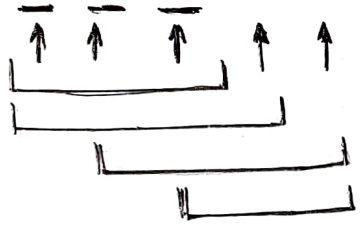
2 Subarray sum equals k

Same code for previous one will be done by it.

Now another way to solve 930 (Binary subarrays with sum)

Eg 1 0 1 0 1

goal = 2



ans = ~~0 1 1 2 2 3 2~~ 2

count = ~~1 2 3~~ (4)

so, 101
10 1 0
0 1 0 1
101

3 Number of zero filled subarrays [FB]

Eg nums = [1,3,0,0,2,0,0] Eg. nums = [0,0,0,2,0,0]

So, total = 6 So total = 9.

So, firstly we have to find no. of zeroes continuous

[0,0,0]

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

total subarrays.

M1 Brute force

$O(N^2) \cdot O(N) = O(N^3)$

Space = O(1).

M-II Now, use previous logic but don't store zeroes count.

direct compute.

M-II Find all zero subarrays i.e starts finding no. of subarrays only when starts with 0.

Time = $O(N^2)$

Space = O(1).

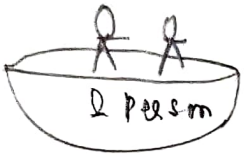
M-III Observe pehle group nikal lenge o ke groups ka

Eg. nums [0,0,0,2,0,0]

⇒ [3,2]

↳ $n(n+1)/2$.

4 Boats to save people



2 Person wt $\leq$ Limit

1 Person $\leq$ Limit

Eg people = [3, 2, 2, 1]

limit = 3

sort this so, 1 2 2 3

now l r

p[l] + p[r] <= limit

1st 1 + 3 <= 3 ✗ cnt++

so, only place right person. (r--)

2nd 1 + 2 <= limit

yes cnt++; r--, l++

3rd 2 + 2 <= limit

yes r++ cnt++;

TC = O(n log n)

SC = O(1).

5 Maximum Points you can obtain from cards

arr[] = [5, 2, 3, 4, 7, 2, 1, 8, 1]

k = 4

- Pickup either from front and also from back else nothing.

So, 6 2 8 1 = 12

6 2 8 4 = 18

6 2 1 7 = (16) yes best

Brute way

4	0
3	1
2	2
1	3
0	4

→ Good

lsum = ~~0~~ ~~18~~ ~~11~~ 9 6 0

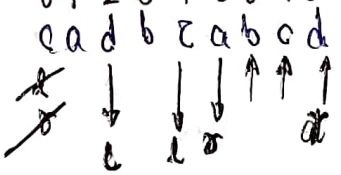
rsum = 0 1 8 9 11

sum 10 12 (16) 15 11

Store this and give o/p.

6 Longest substring without repeating characters.

Eg. s = c a d b c a b c d



c → 4
b → ~~3~~ 6
d → ~~7~~ 8
a → ~~1~~ 5
{c, a} ~~7~~

map · {char, index}

maxlen = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

```
func (string s) {
    hash [256] = [-1]

array hai ye

l=0 , r=0 , maxlen=0
n = s.size ();
while (r < n) {
    if (hash [s[r]] != -1) {
        if (hash [s[r]] >= l) {
            l = hash [s[r]] + 1;
        }
    }
    len = r-l+1;
    maxlen = max (len, maxlen);
    hash [s[r]] = r
    r++;
}
return maxlen;
```

→ TC = O(N)

→ SC = O(256).

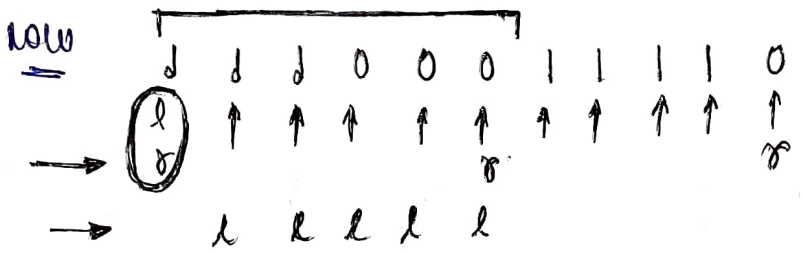
9 Max consecutive ones III

arr[] = [1 1 1 0 0 0 1 1 1 1 0] k = 2

allowed to flip at most k zeroes.

↳ o/p = 6

- Find longest subarray with max zeroes as k.



zeroes = 0 1 2 3 2 3 2

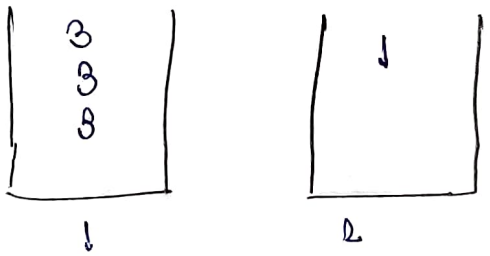
maxlen = 0 1 2 3 4 5 6

ANS

8 Fruits into Baskets.

arr[] = [3 3 3 1 2 1 1 2 3 3 4]

Eg

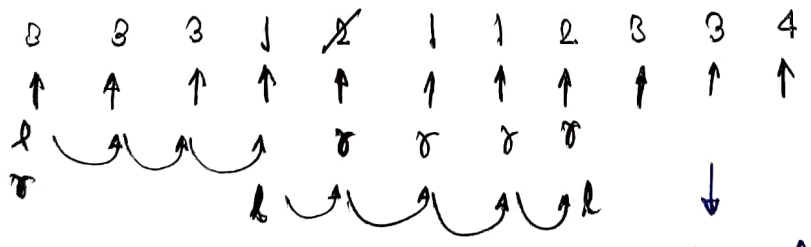


= 4.

only 2 buckets, that only stores similar types of thing.

- max length subarray with at most two types of numbers.

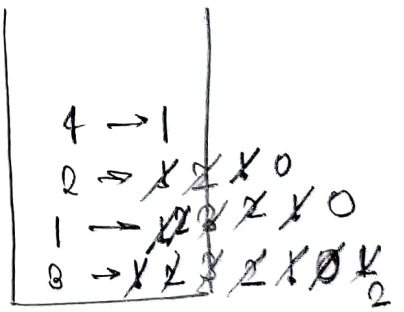
Eg 1 1 1 2 2 = 5



TC = O(N+N) ≅ O(N)

SC = O(3)

O(2N)



maxlen = 0 1 2 3 4 5

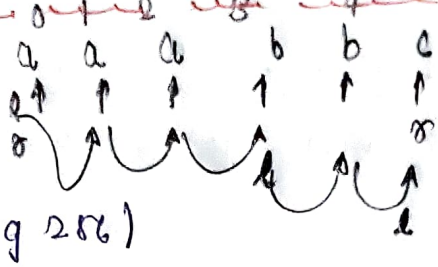
- One another way to optimise this will be simply do the subarray and just store in map coz map size = 3 so return ans.

↳ max consecutives ones II.

9 Longest substring with at most k distinct characters

Eg: s = a a a b b c c d

k = 2.



$TC = O(N) + O(\log 2K)$

$SC = O(2K)$.

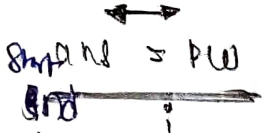
10 Number of substrings containing all the 3 characters.

s = b b a c b a len = 2

Sliding window

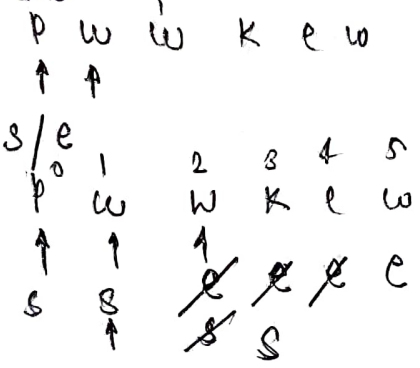
01 Longest substring without repeating chas.

Eg. p w w k e w



```
map
p → 1
w → 1
ans = 3
```

```
for ( ——— ) {
    while (mp[s[i]] > 0) {
        ans = i - j
        mp[s[j]]--;
        j--;
    }
    mp[s[i]]++;
```



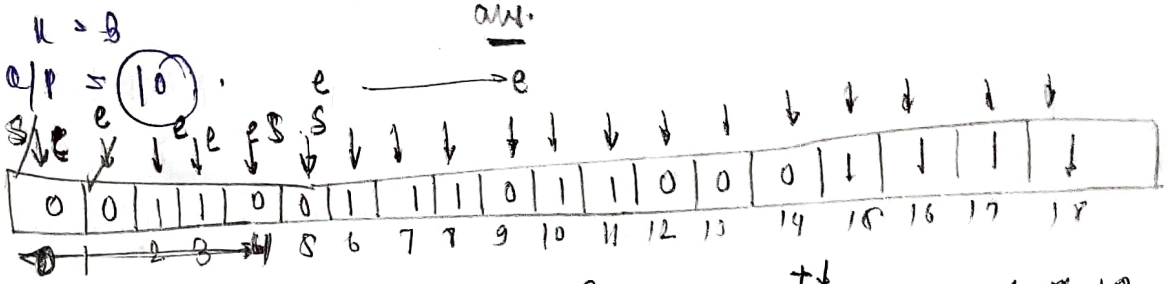
```
map
p → 1 0
w → 1 0 1 2
k → 1
e → 1
ans = 3
```

```
int Pene (string s) {
    int start = 0, end = 0, ans = 0;
    considered_map <char, int> mp;
    while (end < s.length()) {
        while (mp[s[end]] > 0) {
            ans = max(ans, (end - start));
            mp[s[start]]--;
        }       start++;
        mp[s[end]]++;
    }       end++;
    ans = max(ans, (end - start));
    return ans;
}
```

02 max consecutive ones.

Eg. nums = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1]

ans.



k = ~~0 1 2 3~~ 0 ~~1~~ 0 ans = (head - tail ~~+1~~) = ~~5 7 8~~ 10

```
int fone ( vector<int> nums , int k){
    int start = 0 , end = 0 , ans = 0 ;
    while ( end < nums.size() ){
        if ( nums[i] == 0 ) {
        } k-- ;
        while ( k<0) {
            if (nums [start] == 0) {
                k++ ;
            }
            start ++;
        }
        ans = max (ans , ( end - start +1)) ;
        end ++;
    }
}
return ans ;
```

03 Get equal substrings within budgets.

S = 'abcd'
t = 'bcdf'
maxCost = 3

```
int equalSubstring (string S, string t,
                    int maxCost) {
    int i = 0, j = 0 ;
    int cost = 0 ; int ans = 0 ;
    while (j < s.length ()) {
        cost = cost + abs (s[j] - t[j]);
        if (cost > maxCost) {
            cost = cost - abs (s[i] - t[i]);
            i++;
        }
        ans = max (ans, (j+1-i));
    } j++;
    return ans;
```

04 Subarray product less than K.

Yaha subarray ka logic check krna hai bs.

10	5	2	6

k = 100.

[10 ✓
10 5 ✓
5 ✓

[5 2 ✓
5 2 6 ✓
2 ✓

[2 6 ✓
6 ✓

Total 8 subarrays.

```
int cont (vector <int> nums,
          int k) {
    int i = 0, j = 0, ans = 0,
        prod = 1;
    if (k <= 1) return 0;
    while (j < nums.size ()) {
        prod = prod * nums[j];
        while (prod >= k) {
            prod = prod / nums[i];
            i++;
        }
        ans = ans + (j - i + 1);
        j++;
    }
    return ans;
```

Yeh subarray de rha hai.

5 Maximum erasure value

Eg [4 2 4 5 6]
17

Hme value erase kro jo unique hu array me uska maximum sum return krna hai.

```
int i = 0, j = 0, sum = 0, ans = 0;
unordered_map<int, int> mp;
while (j < nums.size()){
    while (mp[nums[j]] > 0){
        mp[nums[i]]--;
        sum = sum - nums[i];
        i++;
    }
    sum = sum + nums[j];
    ans = max(ans, sum);
    mp[nums[j]]++;
    j++;
}
return ans;
```

6 Longest repeating character replacement

Eg s/e
A A B A B B A
i=0 j=0
K = 1 0
ans = A A A / B (4)

```
int i = 0, j = 0, ans = 0;
vector<int> count(26,0);
while (j < s.length()){
    count[s[j] - 'A']++;
    while ((j-i+1) - (*max_element(count.begin(), count.end())) > k){
        count[s[i] - 'A']--;
        i++;
    }
    ans = max(ans, j-i+1);
    j++;
}
return ans;
```

good que
logic.

09 Minimum size subarray sum

Eg. nums = [2,3,1,2,4,3]
target = 7

```
int i=0, j=0, sum=0;
int ans = INT_MAX;
while (j < nums.size()){
    sum = sum + nums[j];
    while (sum >= target){
        ans = min(ans, (j-i+1));
        sum = sum - nums[i];
        i++;
    }
    j++;
}
if (ans == INT_MAX)
return 0;
return ans;
```

Q Minimum operations reduce to Zero.

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

5 − 3 = 2

2 − 2 = 0

Eg | 4 | 2 | 4 | 2 | 3 |

5 − 3 = 2

2 − 2 = 0

Eg | 6 | 2 | 1 | 1 | 5 | 5 |

x = 10

- 6 − 2 − 1 − 1 = 0

4 steps.

- 5 − 5 = 0.

- Yaha ye krn sbse bada size ka subarray chaiye jiska length max ho or sum total − k ho.

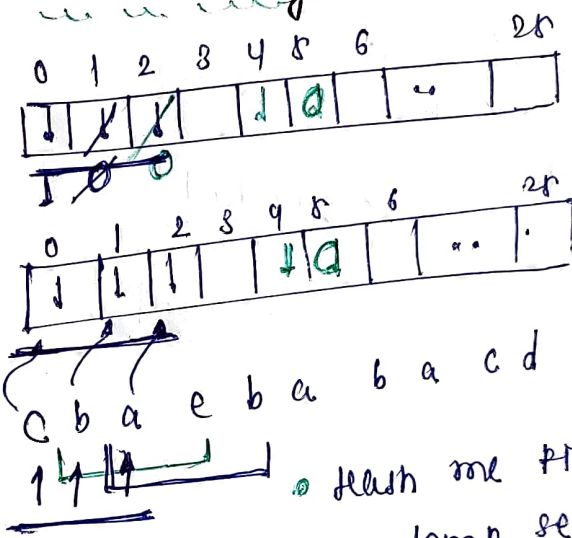
Q Find all anagrams in a string

s = C b a e b a b a c d

0 1 2 3 4 5 6 7 8 9

cba bac

so, [0, 6]



- Hash me final dal do, or temp se compare karte raho temp se 0/1 pe switch krte raho.

Q Longest subarray of 1s after deleting one element

Eg [1,1,0,1]

o/p = 3.

Same as max consecutive ones III

k = variable

Yaha k = 1 or

ans − 1.

Q Count subarrays with sum less than k

Eg. | 2 | 1 | 4 | 3 | 5 |

sum = sum + nums[j]

```
while (sum * (j-i+1) >= k) {
    sum = sum - nums[i];
    i++;
}
ans = ans + (j-i+1);
} j++;
```

qu fruit into baskets.

eg [1,2,1] we have only 2 baskets

o/p : 3

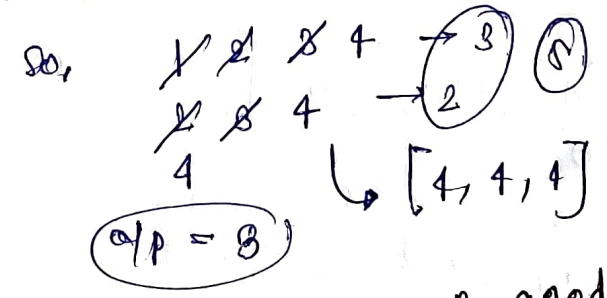
same se r maximum subarray chahiya hai jiska length sbse jada ho & me hi aa jaye.

qu Minimum consecutive cards to pick up.

almost same.

qu frequency of most frequent element

eg [1 2 4] k=0.



qu Number of continuous subarrays.

qu No. of smooth descent periods of stock.

same qu.

qu count the no. of good subarrays

SLIDING WINDOW QUESTIONS

1.) LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS

CODE

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int start=0;
        int end=0;
        int ans=0;
        unordered_map<char,int> mp;

        while(end<s.length()){
            while(mp[s[end]]>0){
                ans=max(ans,(end-start));
                mp[s[start]]--;
                start++;
            }
            mp[s[end]]++;
            end++;
        }
        ans=max(ans,(end-start));
        return ans;
    }
};
```

2.) MAX CONSECUTIVE ONES III

CODE

```
class Solution {
public:
    int longestOnes(vector<int>& nums, int k) {
        int i=0;
        int j=0;
        int ans=0;
        while(j<nums.size()){
            if(nums[j]==0)
            k--;
            while(k<0){
                if(nums[i]==0)
                k++;
                i++;
            }
            ans=max(ans,(j-i+1));
            j++;
        }
        return ans;
    }
};
```

3.) GET EQUAL SUBSTRING WITHIN BUDGETS

CODE

```
class Solution {
public:
    int longestOnes(vector<int>& nums, int k) {
        int i=0,j=0,ans=0;
        while(j<nums.size()){
            if(nums[j]==0)
            k--;
            while(k<0){
                if(nums[i]==0)
                k++;
                i++;
            }
            ans=max(ans,(j-i+1));
            j++;
        }
        return ans;
    }
};
```

4.) SUBARRAY PRODUCT LESS THAN K

CODE

```
class Solution {
public:
    int numSubarrayProductLessThanK(vector<int>& nums, int k) {
        int i=0;
        int j=0;
        int ans=0;
        int prod=1;

        // EDGE CASE
        if(k<=1)
        return 0;

        while(j<nums.size()){
            prod=prod*nums[j];
            while(prod>=k){
                prod=prod/nums[i];
                i++;
            }
            ans=ans+(j-i+1);
            j++;
        }
        return ans;
    }
};
```

5.) MAXIMUM ERASURE VALUE

CODE

```
class Solution {
public:
    int maximumUniqueSubarray(vector<int>& nums) {
        int i=0;
        int j=0;
        int sum=0;
        int ans=0;
        unordered_map<int,int> mp;

        while(j<nums.size()){
            while(mp[nums[j]]>0){
                mp[nums[i]]--;
                sum=sum-nums[i];
                i++;
            }
            sum=sum+nums[j];
            ans=max(ans,sum);
            mp[nums[j]]++;
            j++;
        }
        return ans;
    }
};
```

6.) LONGEST REPEATING CHARACTER REPLACMENT

CODE

```
class Solution {
public:
    int characterReplacement(string s, int k) {
        int i=0;
        int j=0;
        int ans=0;
        vector<int> count(26, 0);

        while(j<s.length()){
            count[s[j]-'A']++;
            while((j-i+1)-(*max_element(count.begin(),count.end()))>k){
                count[s[i]-'A']--;
                i++;
            }
            ans=max(ans,j-i+1);
            j++;
        }
        return ans;
    }
};
```

7.) MINIMUM SIZE SUBARRAY SUM

CODE

```
class Solution {
public:
    int minSubArrayLen(int target, vector<int>& nums) {
        int i=0;
        int j=0;
        int sum=0;
        int ans=INT_MAX;
        while(j<nums.size()){
            sum=sum+nums[j];
            while(sum>=target){
                ans=min(ans,(j-i+1));
                sum=sum-nums[i];
                i++;
            }
            j++;
        }
        if(ans==INT_MAX)
        return 0;
        return ans;
    }
};
```

8.) MINIMUM OPERATIONS TO REDUCE X TO ZERO

CODE

```
class Solution {
public:
    int minOperations(vector<int>& nums, int x) {
        int sum=accumulate(nums.begin(),nums.end(),0);
        int req_sum=sum-x;
        if(req_sum==0)
        return nums.size();
        if(req_sum<0)
        return -1;

        int i=0;
        int j=0;
        int max_len=0;
        int my_sum=0;
        while(j<nums.size()){
            my_sum=my_sum+nums[j];
            while(my_sum>req_sum){
                my_sum=my_sum-nums[i];
                i++;
            }

            if(my_sum==req_sum)
            max_len=max(max_len, j-i+1);

            j++;
        }
        if(max_len==0)
        return -1;
        else
        return (nums.size()-max_len);
    }
};
```

9.) FIND ALL ANAGRAMS IN A STRING

CODE

```
class Solution {
public:
    vector<int> findAnagrams(string s, string p) {
        vector<int> hash(26, 0),temp(26, 0);

        for(int i=0;i<p.length();i++){
            hash[p[i]-'a']++;
        }
        int i=0;
        int j=0;
        int n=p.length();
        vector<int> ans;
        while(j<s.length()){
            if(temp==hash)
            ans.push_back(i);
            while(j-i+1>n){
                temp[s[i]-'a']--;
                i++;
            }
            temp[s[j]-'a']++;
            j++;
        }
        if(temp==hash)
        ans.push_back(i);
        return ans;
    }
};
```

10.) LONGEST SUBARRAY OF 1'S AFTER DELETING ONE ELEMENT

CODE

Same as max consecutive ones III

```
class Solution {
public:
    int longestSubarray(vector<int>& nums) {
        int i=0;
        int j=0;
        int ans=0;

        // small change
        int k=1;

        while(j<nums.size()){
            if(nums[j]==0)
            k--;
            while(k<0){
                if(nums[i]==0)
                k++;
                i++;
            }
            ans=max(ans,(j-i+1));
            j++;
        }
        return ans-1;
    }
};
```

11.) COUNT SUBARRAYS WITH SCORE LESS THAN K

CODE

```cpp
class Solution {
public:
    long long countSubarrays(vector<int>& nums, long long k) {
        long long int ans=0;
        long long int i=0;
        long long int j=0;
        long long int sum=0;

        while(j<nums.size()){
            sum=sum+nums[j];

            while(sum*(j-i+1)>=k){
                sum=sum-nums[i];
                i++;
            }
            ans=ans+(j-i+1);
            j++;
        }
        return ans;
    }
};
```

12.) FRUITS INTO BASKETS

CODE

```
class Solution {
public:
    int totalFruit(vector<int>& items) {
        int i=0;
        int j=0;
        int count=0;
        unordered_map<int,int> mp;

        while(j<items.size()){
            mp[items[j]]++;
            while(mp.size()>2){
                mp[items[i]]--;

                // map m element jiski value zero h usko remove kr do
                if(mp[items[i]]==0)
                mp.erase(items[i]);

                i++;
            }
            count=max(count, j-i+1);
            j++;
        }
        return count;
    }
};
```

13.) MINIMUM CONSECUTIVE CARDS TO PICK UP

CODE

```
class Solution {
public:
    int minimumCardPickup(vector<int>& cards) {
        int i=0;
        int j=0;
        int ans=INT_MAX;
        unordered_map<int,int> mp;

        while(j<cards.size()){
            mp[cards[j]]++;

            while(mp[cards[j]]>1){
                ans=min(ans, j-i+1);
                mp[cards[i]]--;
                i++;
            }
            j++;
        }

        if(ans==INT_MAX)
        return -1;
        else
        return ans;
    }
};
```

14.) FREQUENCY OF MOST FREQUENT ELEMENT

CODE

```
class Solution {
public:
    using ll= long long int;
    int maxFrequency(vector<int>& nums, int k) {
        sort(nums.begin(), nums.end());
        ll i=0;
        ll j=0;
        ll sum=0;
        ll ans=0;

        while(j<nums.size()){
            sum=sum+nums[j];

            if((j-i+1)*nums[j]-sum>k){
                sum=sum-nums[i];
                i++;
            }
            ans=max(ans, j-i+1);
            j++;
        }
        return ans;
    }
};
```

15.) NUMBER OF ZERO FILLED SUBARRAYS

CODE

```
class Solution {
public:
    long long zeroFilledSubarray(vector<int>& nums) {
        using ll=long long int;
        ll i=0;
        ll j=0;
        ll ans=0;

        while(j<nums.size()){
            i=j;
            while(j<nums.size() && nums[j]==0){
                ans=ans+(j-i+1);
                j++;
            }
            j++;
        }
        return ans;
    }
};
```

16.) NUMBER OF SMOOTH DESCENT PERIODS OF A STOCK

CODE

```
class Solution {
public:
    long long getDescentPeriods(vector<int>& prices) {
        using ll=long long int;
        ll i=0;
        ll j=1;
        ll ans=0;

        while(j<prices.size()){
            while(j<prices.size() && prices[j]-prices[j-1]==-1){
                ans=ans+(j-i);
                j++;
            }
            i=j;
            j++;
        }
        return ans+prices.size();
    }
};
```

17.) COUNT THE NUMBER OF GOOD SUBARRAYS

CODE

```cpp
class Solution {
public:
    long long countGood(vector<int>& nums, int k) {
        using ll=long long int;
        ll i=0;
        ll j=0;
        ll count=0;
        ll ans=0;
        unordered_map<ll, ll> mp;

        while(j<nums.size()){
            count=count+mp[nums[j]];
            mp[nums[j]]++;

            while(i<j && count>=k){
                ans=ans+(nums.size()-j);
                mp[nums[i]]--;
                count=count-mp[nums[i]];
                i++;
            }
            j++;
        }
        return ans;
    }
};
```

18.) LONGEST NICE SUBARRAYS

CODE

```cpp
class Solution {
public:
    int longestNiceSubarray(vector<int>& nums) {
        int i=0;
        int j=0;
        int ans=0;
        int result=0;

        while(j<nums.size()){
            while((ans & nums[j])>0){
                ans=ans^nums[i];
                i++;
            }
            ans=ans|nums[j];
            result=max(result, j-i+1);
            j++;
        }
        return result;
    }
};
```

19.) MAXIMISE THE CONFUSION OF AN EXAM

CODE

```
class Solution {
public:
    int maxConsecutiveAnswers(string s, int k) {
        int i=0;
        int j=0;
        int countT=0;
        int countF=0;
        int ans=0;

        while(j<s.size()){
            if(s[j]=='T')
            countT++;
            if(s[j]=='F')
            countF++;

            while(min(countT, countF)>k){
                if(s[i]=='T')
                countT--;
                if(s[i]=='F')
                countF--;
                i++;
            }
            ans=max(ans, j-i+1);
            j++;
        }
        return ans;
    }
};
```

20.) BINARY SUBARRAYS WITH SUM

CODE

```
class Solution {
public:
    int func(vector<int>& nums, int goal) {
        long long int i=0;
        long long int j=0;
        long long int ans=0;
        long long int sum=0;

        while(j<nums.size()){
            sum=sum+nums[j];
            while(i<=j && sum>goal){
                sum=sum-nums[i];
                i++;
            }
            ans=ans+(j-i+1);
            j++;
        }
        return ans;
    }

    int numSubarraysWithSum(vector<int>& nums, int goal) {
        return func(nums,goal)-func(nums,goal-1);
    }
};
```

21.) COUNT NUMBER OF NICE SUBARRAYS

CODE

```
class Solution {
public:
    int func(vector<int>& nums, int k) {
        int i=0;
        int j=0;
        int ans=0;

        while(j<nums.size()){
            if(nums[j]&1)
            k--;
            while(k<0){
                if(nums[i]&1)
                k++;
                i++;
            }
            ans=ans+(j-i+1);
            j++;
        }
        return ans;
    }
    int numberOfSubarrays(vector<int>& nums, int k) {
        return func(nums, k)-func(nums, k-1);
    }
};
```

22.) SUBARRAYS WITH K DIFFERENT INTEGERS

CODE

```
class Solution {
public:
    int func(vector<int> &nums, int k){
        int i=0;
        int j=0;
        int ans=0;
        unordered_map<int,int> mp;

        while(j<nums.size()){
            mp[nums[j]]++;

            while(mp.size()>k){
                mp[nums[i]]--;
                if(mp[nums[i]]==0)
                mp.erase(nums[i]);
                i++;
            }
            ans=ans+(j-i+1);
            j++;
        }
        return ans;
    }
    int subarraysWithKDistinct(vector<int>& nums, int k) {
        return func(nums, k)-func(nums, k-1);
    }
};
```

23.) MINIMUM SWAPS TO GROUP ALL 1'S TOGETHER II

CODE

```
class Solution {
public:
    int minSwaps(vector<int>& nums) {
        int count=0;
        for(int i=0;i<nums.size();i++){
            if(nums[i]==1)
            count++;
        }

        int w=count;
        int countZ=0;

        // initially zero handelled
        for(int i=0;i<w;i++){
            if(nums[i]==0)
            countZ++;
        }

        int mini=countZ;
        for(int i=w;i<w+nums.size();i++){
            if(nums[i%nums.size()]==0)
            countZ++;
            if(nums[i-w]==0)
            countZ--;

            mini=min(mini, countZ);
        }
        return mini;
    }
};
```

24.) MINIMUM WINDOW SUBSTRING

CODE

```
class Solution {
public:
    string minWindow(string s, string t) {
        int i=0;
        int j=0;
        int mini=INT_MAX;
        int start=0;
        int size=0;

        vector<int> hash(128);
        for(auto it:t){
            hash[it]++;
        }

        while(j<s.length()){

            hash[s[j]]--;
            if(hash[s[j]]>=0){
                size++;
            }

            while(size==t.size()){
                if(mini>(j-i+1)){
                    mini=j-i+1;
                    start=i;
                }
                hash[s[i]]++;
                if(hash[s[i]]>0)
                size--;
                i++;
            }

            j++;
        }

        if(mini==INT_MAX)
        return "";
        return s.substr(start, mini);
    }
};
```

25.) MAXIMUM NUMBER OF VOWELS IN A SUBSTRING OF GIVEN LENGTH

CODE

```
class Solution {
public:
    int maxVowels(string s, int k) {
        int i=0;
        int j=0;
        int count=0;
        int ans=0;

        while(j<s.length()){
            if(s[j]=='a' || s[j]=='e' || s[j]=='i' || s[j]=='o' || s[j]=='u'){
                count++;
            }
            if(j-i+1==k){
                ans=max(ans, count);
                if(s[i]=='a' || s[i]=='e' || s[i]=='i' ||
                   s[i]=='o' || s[i]=='u'){
                    count--;
                }
                i++;
            }
            j++;
        }
        return ans;
    }
};
```

26.) NUMBER OF SUBSTRINGS CONTAINING ALL THREE CHARACTERS

CODE

```
class Solution {
public:
    int numberOfSubstrings(string s) {
        int i=0;
        int j=0;
        int ans=0;
        unordered_map<int, int> mp;

        while(j<s.length()){
            mp[s[j]]++;

            while(mp['a'] && mp['b'] && mp['c']){
                ans=ans+(s.length()-j);
                mp[s[i]]--;
                i++;
            }

            j++;
        }
        return ans;
    }
};
```

27.) COUNT SUBARRAYS WHERE MAX ELEMENT APPEARS AT LEAST K TIMES

CODE

```
class Solution {
public:
    long long countSubarrays(vector<int>& nums, int k) {
        long long int maxi=*max_element(nums.begin(),nums.end());
        long long int i=0;
        long long int j=0;
        long long int ans=0;
        long long int count=0;
        while(j<nums.size()){
            if(nums[j]==maxi)
            count++;

            while(count>=k){
                if(nums[i]==maxi)
                count--;
                ans=ans+(nums.size()-j);
                i++;
            }
            j++;
        }
        return ans;
    }
};
```

28.) LENGTH OF LONGEST ALPHABETICAL CONTINUOUS SUBSTRING

CODE

```
class Solution {
public:
    int longestContinuousSubstring(string s) {
        int j=1;
        int maxi=1;
        int ans=1;
        while(j<s.length()){
            if(s[j]==s[j-1]+1){
                ans++;
                maxi=max(maxi, ans);
            }
            else
            ans=1;
            j++;
        }
        maxi=max(maxi, ans);
        return maxi;
    }
};
```

29.) SLIDING WINDOW MAXIMUM

CODE

```
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        vector<int> ans;
        deque<int> dq;
        for(int i=0;i<nums.size();i++){
            if(!dq.empty() && dq.front()==i-k){
                dq.pop_front();
            }
            while(!dq.empty() && nums[dq.back()]<=nums[i]){
                dq.pop_back();
            }
            dq.push_back(i);
            if(i>=k-1)
            ans.push_back(nums[dq.front()]);
        }
        return ans;
    }
};
```