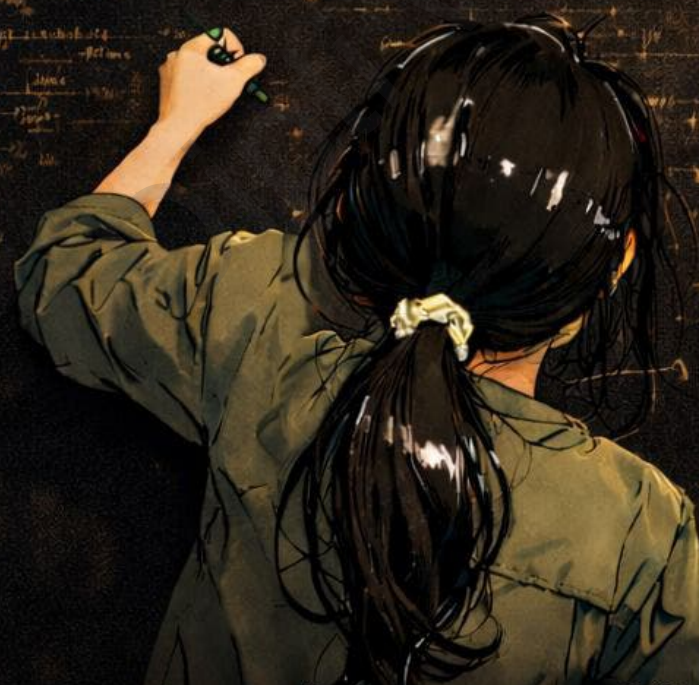
Sliding Window Pattern



Handwritten Notes

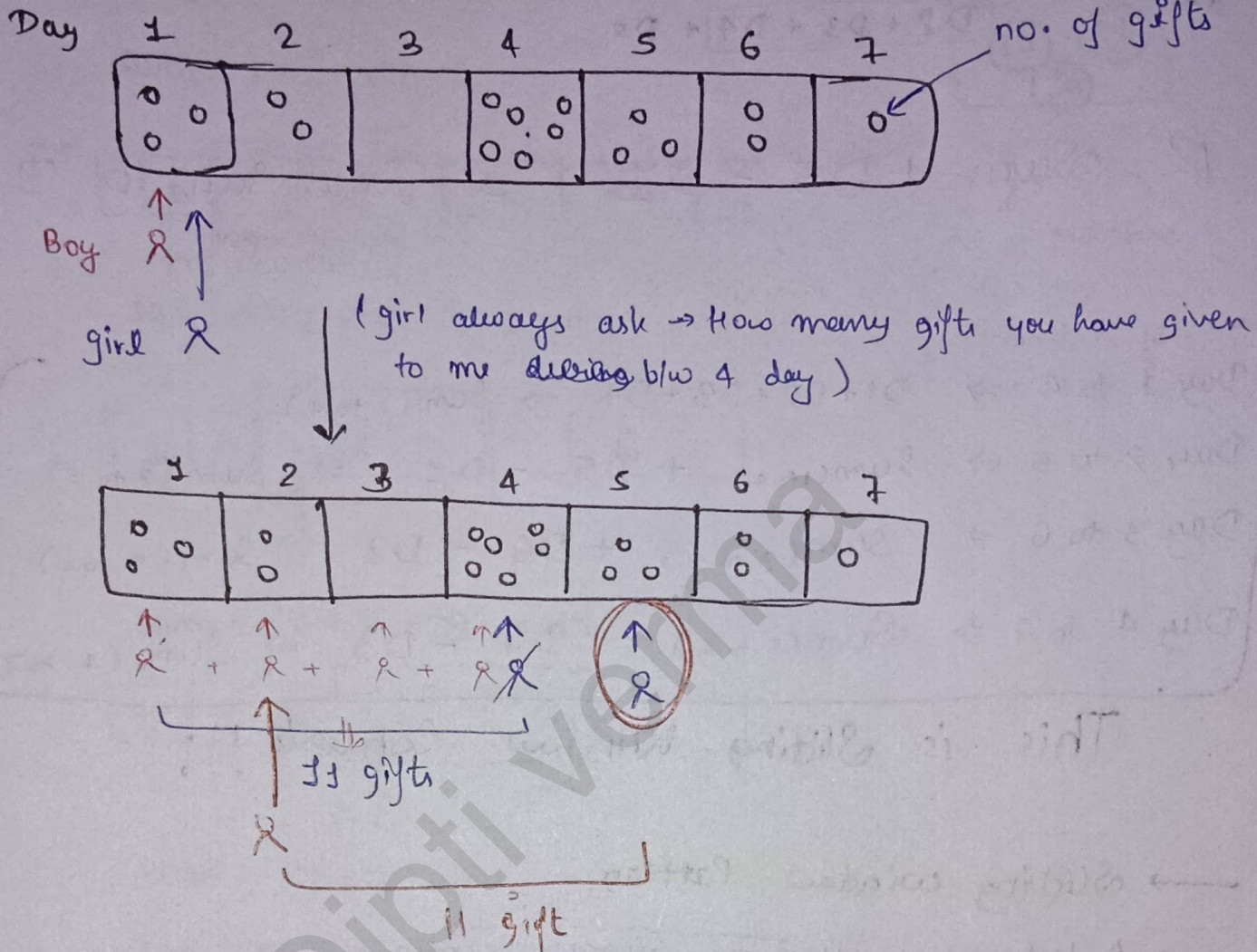
by **Dipti Verma**

learned from **CTO Bhaiya**

Day-13

Sliding Window Pattern

(Theory)



Problem: Boy check as I. D1 + D2 + D3 + D4
II. D2 + D3 + D4 + D5
⋮
again - again calculate.

Let

Iteration:

I ⇒ Sum = D1 + D2 + D3 + D4

II ⇒ D2 + D3 + D4 + D5

✗ Sum + D5 − D1 = Total Sum of II Iteration

Day 1 to 4 ⇒ D1+D2+D3+D4 ⇒ Sum(1 to 4)

Day 2 to 5 ⇒ Sum(1 to 4) + D5 − D1 ⇒ Sum(2 to 5)

Day 3 to 6 ⇒ Sum(2 to 5) + D6 − D2 ⇒ Sum(3 to 6)

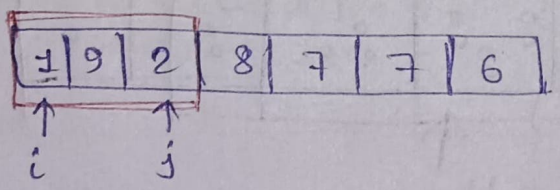
Day 4 to 7 ⇒ Sum(3 to 6) + D7 − D3 ⇒ Sum(4 to 7)

This is Sliding Window Concepts !!!

→ Sliding window Pattern

- A technique used to process contiguous chunks (subarrays or substrings) of sequential data efficiently.

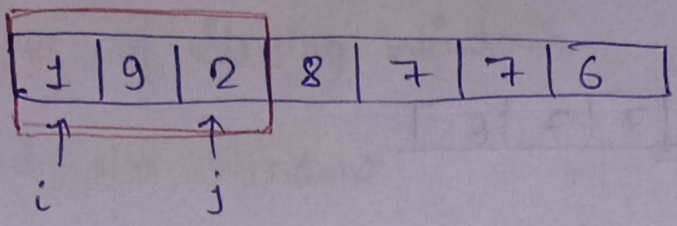
Eg,



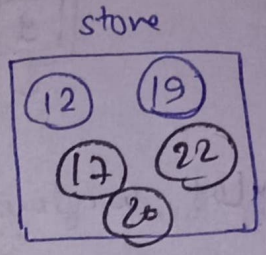
Sum = 1 + 9 + 2

= 12

Eg.

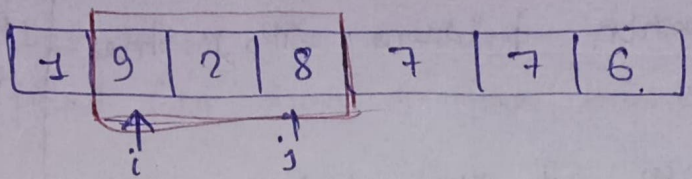


store



sum = 1+9+2 = 12

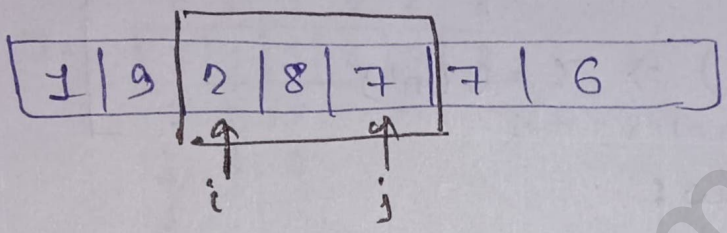
→



sum = 12 + 8 - 1 = 19

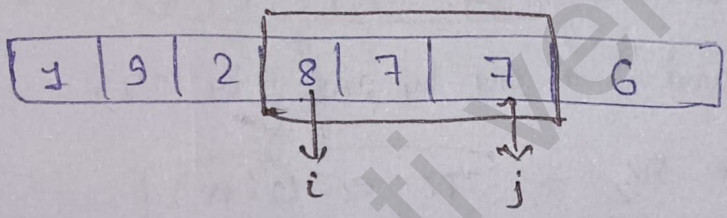
2 pointers on the boundary of window

→



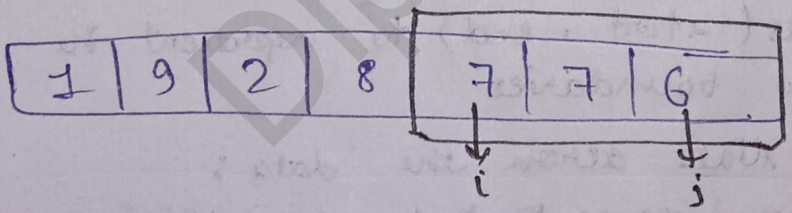
sum = 19 + 7 - 9 = 17

→



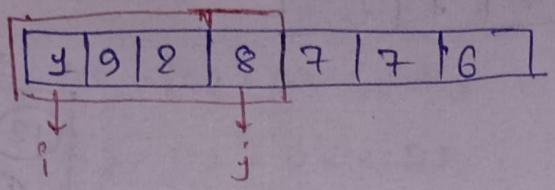
sum = 17 + 7 - 2 = 22

→



sum = 22 + 6 - 8 = 20

- Instead of recalculating from scratch everytime, the window "slides" — adding one element & removing an old element, so each update happen in constant time.



Let window size = $n/2$

So, in first iteration i runs $(n/2)$ times

then j++

in II iteration i runs $(n/2)$ time

$(n/2) \times (n/2) \Rightarrow TC = O(n^2)$

Brute Force

But, In sliding window:

first Sum calculate → time = $n/2$

and then, j++ and j++ one by one by $(n/2)$

$n/2 + n/2 \Rightarrow$ TC $\Rightarrow O(n)$

→ Use two pointers (start, end) to represent the current window boundaries

As the window slides across the data:

→ Add the new element that enters the window

→ Remove the element that leaves the window

⟹ Types of Sliding Window

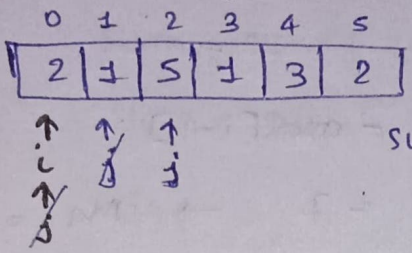
① Fixed - size Window

- Window size K is constant
- Both pointers move together, maintaining a fixed distance
- Used when the number of consecutive elements is fixed (eg. sum of three numbers).

Eg, Maxm Sum of Subarray of size K=3

Brute force

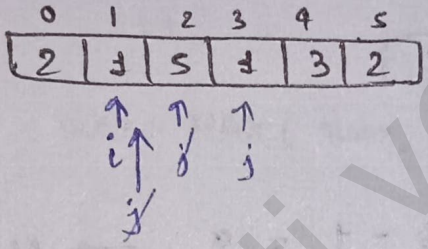
I.



Max = ∅ 8

sum = 2+1+5 = 8

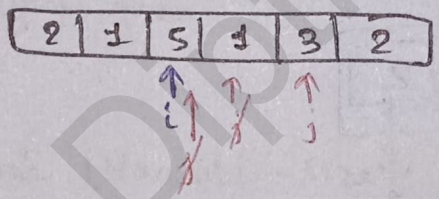
II.



sum = 1+5+1 = 7

Max = 8

III.



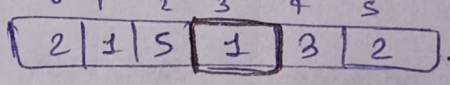
sum = 5+1+3 = 9

Max = 9

Code

```
max = 0
K = 3
for (i = 0 ; j <= n-K ; i++){
    sum = 0
    for( j = i ; j <= i+k-1 ; j++){
        sum = sum + arr[j]
    }
    max = MAX ( max, sum)
}
return max
```

'i' Kaha tak jayega ?



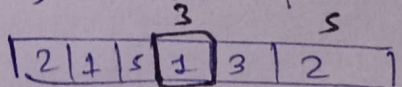
bcz, K=3

So, n = 6
K = 3

6-3 = 3 = index of last i

∴ n-K

'j' Kaha tak ?



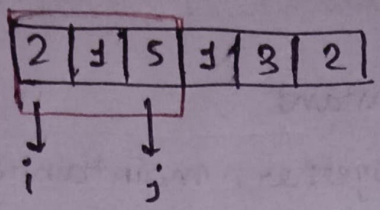
i=3 , j=3 he reached to 5th index

∴ i+k-1

3+3 = 6-1 = 5

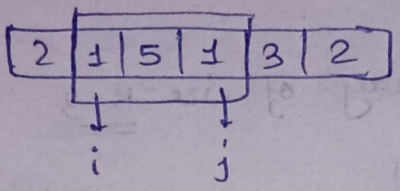
optimized

Max^n sum of subarray of size K=3



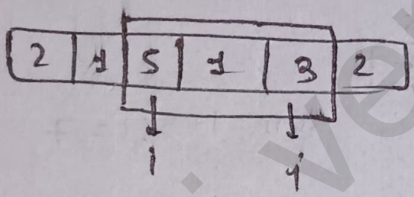
sum = 8 → Max = 8

i++
j++

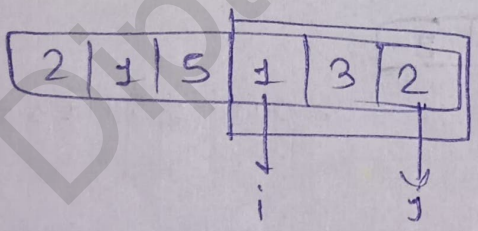


sum = 8 + arr[j] - arr[i-1]

= 8 + 1 - 2 = 7 → Max = 8



sum = 7 + 3 - 1 = 9 → Max = 9

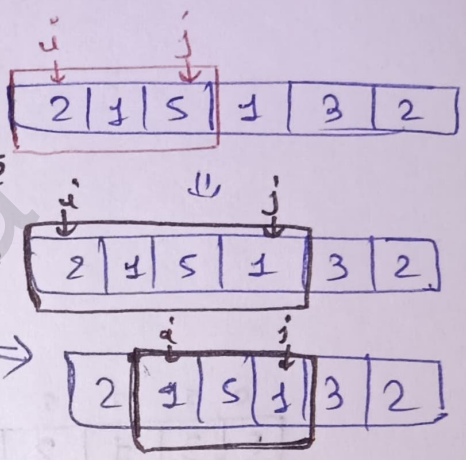


sum = 9 + 2 - 5 = 6

⇒ Max = 9

Code :

```
max = 0
sum = 0

K = 3
i = 0
for (j=0 ; j<k ; j++){
    sum = sum + arr[j]
}

max = Max ( max , sum )

while ( j<n ):
    sum = sum + arr[j] & arr[i]
    sum = sum - arr[i]
    i++
    j++

max = MAX ( max, sum )

return max
```



T(= O(n)

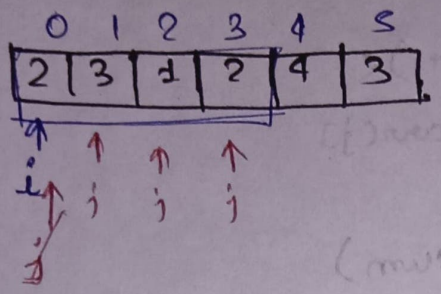
(2) Dynamic (Variable-size) Window

- Windows expands or contract based on conditions
- Common for problems like :
  - Longest substring without repeating characters
  - Smallest subarray with sum >= target.

Eg Smallest subarray with sum >= target

target = 7

Brute force



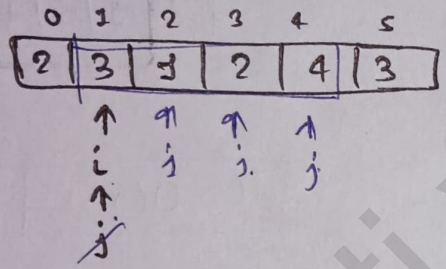
sum =

S = 2+3 = 5+1 = 6+2 = 8 > target

sum = 8

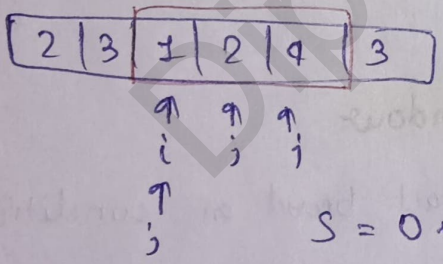
size = (j - i + 1) = (3 - 0 + 1) = 4

min_size = min (min, size) = 4



S = 0+3 = 3+1 = 4+2 = 6+4 = 10 > 7

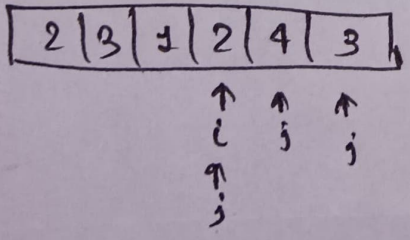
size = 4



S = 0+1 = 1+2 = 3+4 = 7

size = 3

min_size = 3 → sum = 7



sum = 2+4+3 = 9

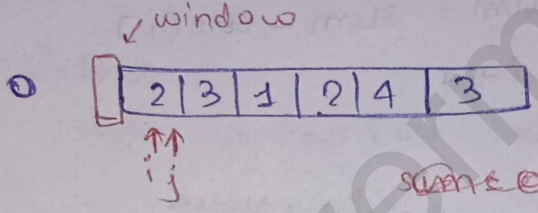
size = 3

Code:

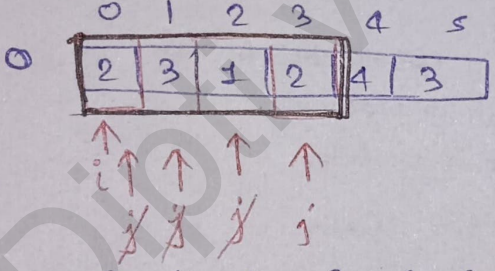
```
min = infinity
target = 7
for (i=0 ; i<n; i++){
    sum = 0
    for( j=i ; j<n ; j++){
        sum = sum + arr[j]
        if(sum >= target){
            min = MIN( min , j-i+1 )
            break
        }
    }
return min
```

Optimized

target = 7

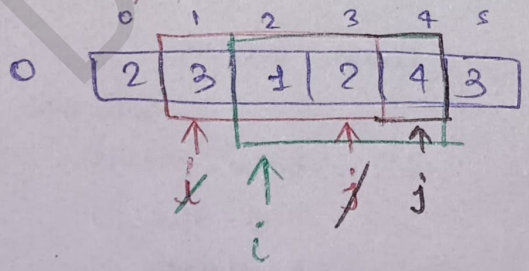


sum = 0 s = 0
min = ∞

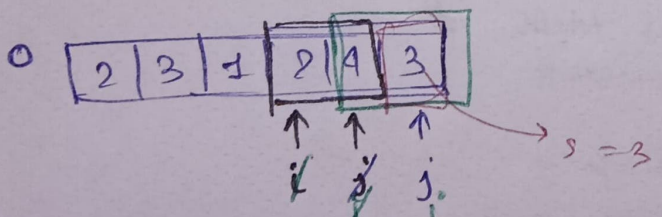


s = ~~0~~ ~~2~~ ~~5~~ ~~6~~ 8 >= target
min = 4 (j-i+1)

Now, i++



s = ~~8~~ ~~10~~ >= target
But min already = 4
So, i++



s = ~~9~~ ~~7~~ = target
min = ~~3~~ 2
i++

s = 7 >= target ✓
min = 3
Now, i++

i++

∴ Size = 2 min sum = 7.

Code :

```
min = infinity
target = 7
sum = 0

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

        while ( sum >= target){
                min = MIN ( min, j-i+1)
                sum = sum - arr[i]
                i++
        }

    j++
}
return min
```

⇒ Identify Sliding Window

- The ip is contiguous (array / string)
- Question says "subarray" or "substring"
- I. Try Brute force
  II. Check the condition of "Brute force" convert into Sliding window or not.
- Asked for max / min / count with a continguous range
- You can express the problem as "window slides one step" each iteration

⇒ Template

- Fixed Window

```
sliding_window (arr, condition):
    result = window

    for i in range (k, len(arr)):
        window = add arr[i] to window
        window = subtract arr[i-k] from window
        result = best (result, window)

    return result
```

- Dynamic Window

```
sliding_window (arr, condition):
    left = 0
    result = ???

    for right in range (len(arr)):
        # expand window

        while condition satisfied:
            result = process result b/w left & right
            shrink window

    return result
```

Day-13

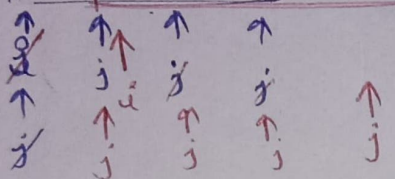
[Sliding Window Pattern]

(Medium level Question)

- → Max Sum Subarray of size K (GFG)
- → 2461. Max Sum of Distinct Subarrays with length k (LC)
- → 209. Minm Size Subarray Sum (LC)

Q. Max Sum Subarray of size K

Eg, [1, 4, 2, 10, 23, 3, 1, 0, 20] K = 4



sum = 1+4+2+10 = 17 sum = 4+2+10+23 = 39

Max = ~~0~~ 17 39

TC = $O(n^2)$

What we have observe?

I. sum = 1 + [4 + 2 + 10]

II. sum = [4 + 2 + 10] + (23)

repeat the calculation

Optimized [We get it as: (1 + 4 + 2 + 10) + 23 − 1 = II. sum

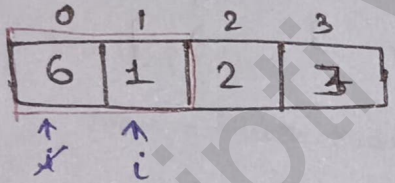
Code :

```
public int maxSubarraySum( int arr[], int k) {
    int sum = 0 , max = 0 ;

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

    max = Math.max( max , sum)

    for ( int i = k ; i < arr.length ; i++ ){
        sum = sum + arr[i]
        sum = sum - arr[i-k]

        max = Math.max( max, sum)
    }

    return max
}
```

TC = O(K + N-1)

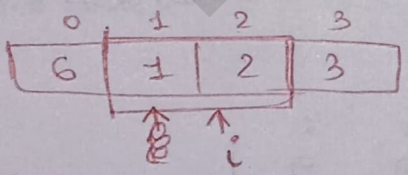
∴ TC = O(n)
SC = O(1)

Eg



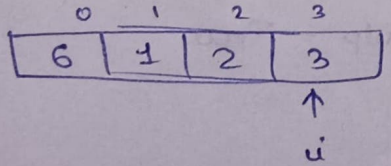
I. sum = ø ø 7
max = ø 7

II.



Sum = 7 + 2 = 9
Sum = 9 - arr[i-K] = 9 - 6 = 3
max = 7

III.

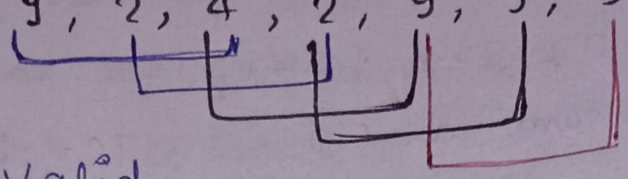


Sum = 3 + 3 = 6
Sum = 6 - arr[i-K] = 6-1 = 5
max = 7

Ans = 7

0·(2461) Max Sum of Distinct Subarrays with Length K

nums = [1, 2, 4, 2, 9, 9, 9] k = 3



Valid

- 1 2 4 ✓
- (2) 4 (2) ✗ (same, same)
- 4 2 9 ✓
- 2 (9) (9) ✗
- (9) (9) (9) ✗

[1 2 4] sum = 7

[4 2 9] sum = 15 ✓ (Max)

Brute Force

eg, [1, 2, 4, 2, 9, 9, 9] (indices 0 1 2 3 4 5 6)

I. Set = {1, 2, 4} → No duplicate isDuplicate = False

sum = ✗ ✗ (7)

eg, [1, 4, 4, 2, 9, 9, 9] (indices 0 1 2 3 4 5 6)

Set - [1, 4]

'4' already → isDulp = True

~~sum~~ sum = ✗ ✗ 7

~~sum = 0~~

Check (isDulp = T) ✓

ignore

Code:

```
Public long maximumSubarraySum( int nums[], int k){
    long max = 0, sum = 0;
    int n = nums.length;
    for( int i = 0; i<= n-k; i++){
        Set<Integer> set = new HashSet<>();
        boolean isDup = False
        sum = 0;
        for( int j = i; j<= i+k-1 ; j++){
            if( set.contains(nums[j]) ){
                isDup = true
                break
            }
            set.add(nums[j])
            sum = sum + nums[j]
        }
        if( ! isDup){
            max = Math.max( max, sum)
        }
    }
    return max
}
```

0 1 2 3 4
⇒ 1 2 3 4 5
K = 2 (end)
i kaha tak go?
⇒ index = 3
n = 5
and k = 2
∴ n-k

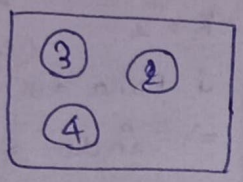
T(n) = O(n²)
S(n) = O(n)

Optimized

nums = [3, 2, 4, 3, 4, 3, 9] K = 4

i j j j j →

- Set



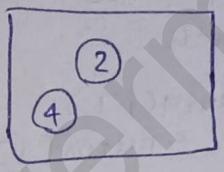
present → invalid

s = 12

max ⇒ no update
(bcz. of dup)

- 3 remove and 4 add [2, 4, 3, 4]

→ remove '3' from set



but window show '3'

→ and we delete (3)

that's why we use (Map), not set

2	1
4	(2) → invalid
3	~~2~~ 1

Now, use Map:

nums = [3, 2, 4, 3, 4, 3, 9] K = 4

i j j j j → j j

-

3	~~1~~ ~~2~~ ~~2~~
2	~~1~~ 0
4	~~1~~ ~~2~~ 1
9	1

Eg. [3, 2, 4, 3, 4, 3, 9] K = 4

i i i i i i i

Map

3 : ~~1~~ 2 ~~1~~ 2
2 : ~~1~~ 0
4 : ~~1~~ ~~1~~ 1
9 : 1

- S = 3+2+4+3 = 12
  max = No update (3 ≥ 2 dup)
- S = 12 + 4 - 3 = 13
  max = No update (4 : 2 dup)
- S = 13 + 3 - 2 = 14
  max = X (3 : 2)
- S = 14 + 9 - 4 = 19
  max = X (3 : 2)

In this case:
We check the whole map again & again
⇒ TC = O(n²)

★ How to optimised this map case? ★★★★

Eg [T, T, P, P]

i j i i

T : ~~1~~ 2 ⟶ T = 2 > 1 ⟶ duplicate = ~~0~~ 1
P : ~~1~~ 2 ⟶ P = 2 > 1 ⟶ duplicate = ~~1~~ 2

~~dupli~~ in one window ⇒ dup = 2

nums = [3, 2, 4, 3, 4, 3, 9] k = 4

I: Map :

- 3 : 1
- Check count of 3:

  ~~1~~ = 1 ⟶ dup = 0
- then, traverse i to 4 times

  3 2 4 3
  ↓ ↓ ↓ ↓
  i i i i

  3 : ~~1~~ (2) ⟶ dup = 1
  2 : 1
  4 : 1

  sum = 12
  max = no update = 0

II: add 4 and remove 3

nums = [3, 2, 4, 3, 4, 3, 9]
                        ↓ i

3 : ~~1~~ 1
2 : 1
4 : ~~1~~ 2 ⟶ duplicate = ~~1~~ 2

- add 4 (first)
- remove 3
  ⟶ 3 ⇒ count = ~~2~~ 2
  2 > 1
  duplicate = ~~1~~ 1
  freq --
  ⟶ 3 : 1

So, dup = 1

sum = 13

max = 0

III. num = [3, 2, 4, 3, 4, 3, 9]

• add 3

3 : ~~1~~ 2

2 : 1

4 : 2

⟶ dulp = ~~1~~ 2

• remove 2

3 : 2

2 : 0

4 : 2

⟶ dup = 2

sum = 14

max = 0

IV. num = [3, 2, 4, 3, 4, 3, 9]

• add 9

3 : 2

2 : 0

4 : 2

9 : 1

⟶ dup = 2

sum = 19

• remove 4

3 : 2

4 : 1

9 : 1

⟶ dup = ~~2~~ 1

sum = max = 0

V: num = [3, 2, 4, 3, 4, 3, 9, 6]

• add 6

3 : 2

4 : 1

9 : 1

6 : 1

⟶ dup = 1

• remove 3

3 : 1

4 : 1

9 : 1

6 : 1

⟶ dup = 0

sum = 19 + 6 - 3 = 22

max = 22

Code:

```
public long maximumSubarraySum( int nums[], int k ){
    long sum = 0, max = 0;
Map< Integer, Integer > map = new HashMap<>();
    int dups = 0
for( int i = 0; i < k; i++ ){
    if( ! map.containsKey( nums[i] ) ){
        map.put( nums[i], 0)
    }
    map.put( nums[i], map.get( nums[i] + 1 )
    sum = sum + nums[i]
    if( map.get( nums[i] ) > 1 ){
        dups = dups + 1
    }
}
if( dups == 0 ){
    max = Math.max( max, sum)
}
for( int i = k; i < nums.length; i++ ){
    int numToAdd = nums[i];
    int numToRemove = nums[i-k]
    if( ! map.containsKey( numToAdd ) ){
        map.put( numToAdd, 0)
    }
    map.put( numToAdd, map.get(numToAdd) +1)
```

```
if( map.get(numToAdd) > 1 ){
        dups = dups + 1
}

sum = sum + numToAdd

if( map.get( numToRemove ) > 1 ){
        dups = dups - 1
}

map.put( numToRemove, map.get(numToRemove) - 1 )

Sum = sum - numToRemove

if( dups == 0 ){
  max = Math.max( sum, max )
}

}

return max

}
```

TC = O(n)

SC = O(n)

Q(209). Min^n Size Subarray Sum

Eg target = 7

nums = [2, 3, 1, 6, 4, 3]

> Brute force pattern always help to recognise the sliding window pattern.

Brute Force

[2, 3, 1, 6, 4, 3]

i ↓

j ↓

TC = $O(n^2)$

SC = $O(1)$

I. [2, 3, 1, 6, 4, 3]
(0 1 2 3 4 5)

i j j j

j = 3

~~length = 6~~

i = 0

~~size = 6 - 3 + 1 = 4~~

S = 12 > target

size = 4 size = j - i + 1

II. [2, 3, 1, 6, 4, 3]
(0 1 2 3 4 5)

i j j

j

j = 3

i = 1

~~length = 6~~

~~size = 6 - 3 + 1 = 4~~

size = 3 - 1 + 1

S = 10 > target = 3

size = 3

III. [2, 3, 1, 6, 4, 3]

i j

j

S = 7 = target

size = 2

IV. [2, 3, 1, 6, 4, 3]

i j

j

S = 10 > target

size = 2

V. [2, 3, 1, 6, 4, 3]

i j

j

S = 7 = target

size = 2

VI. [2, 3, 1, 6, 4, 3]

i

j

Stop.

Observe :

→ j loop stop ⇒ on a condition if 'j' value >= target hai toh stop bcz, bade no. mein add karne pe bada hi hoga.

→ size value = j - i + 1

→ Recognise : How sliding window applied ?

Break condition → if (sum >= target) {
min = Math.min(min, j-i+1)
break;
}

[2 , 3 , 1 , 6 , 4 , 3]

S = 2 + 3 + 1 + 6
S = 3 + 1 + 6
S = 1 + 6

☆ Calculation redundant (again-again)

What we do
- Sum get → break
i++
- again calculate

In sliding window:
- Sum get → remove one item & add one item

same

[2 , 3 , 1 , 6 , 4 , 3]
↑ i
↓ j

- S = 12 > Target
Size = 4

- [2 , 3 , 1 , 6 , 4 , 3]
↑ i ↑ j

S = 12 − 2 = 10 > Target
Size = 3

- [2 , 3 , 1 , 6 , 4 , 3]
↑ i ↑ j

S = 10 − 3 = 7 = target
size = 3

- [2 , 3 , 1 , 6 , 4 , 3]
↑ i ↑ j

- S = 7 − 1 = 6
- S = 6 + 4 = 10 > target
size = 2

- [2 , 3 , 1 , 6 , 4 , 3]
↑ i ↑ j

S = 10 − 6 = 4
S = 4 + 3 = 7 >= target
size = 2

∴ Answer = 2

Brute Force Code :

```
min = Integer.MAX_VALUE
for ( i = 0 ; i < num.length ; i++ ){
    sum = 0
    for ( j = i ; j < num.length ; j++ ){
        sum += num[j]
        if ( sum >= target ){
            min = Math.min ( min , j - i + 1 )
            break
        }
    }
}
return min == Integer.MAX_VALUE ? 0 : min ;
```

TC = O(n²)
SC = O(1)

Sliding window Code :

```
size = Integer.MAX_VALUE
sum = 0
i = 0 , j = 0
while ( j < nums.length ){
//expand window
sum = sum + nums[j]
    while ( sum >= target ){
        size = Math.min ( size , j - i + 1 )
        sum = sum - nums[i]
        i++
    }
    j++
}
return min == Integer.MAX_VALUE ? 0 : size
```

TC = O(n)
SC = O(1)

Day-14

Sliding Window Pattern

(Med Level Ques)

219. Contains Duplicate II (easy) LC

643. Max Average Subarray I (easy) LC

3. Longest substring without repeating characters (medium) LC

Q. (219) Contain Duplicate II

Condition

- num[i] = num[j]
- abs | i-j | <= K (Boundary)

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

o/p = True

```
 0  1  2  3  4
[1, 2, 3, 4, 2]
```

num[i] = 1
num[j] = 4 ✗
no any dup

num[i] = 2
num[j] = 2 ✓ → True

Brute Force

```
 0  1  2  3  4
[1, 2, 3, 4, 2]
 i  ↑  ↑  ↑
 j  j  j  j
```

(set)

1	2
3	4

no dup

```
 0  1  2  3  4
[1, 2, 3, 4, 2]
    i  ↑  ↑  ↑
    j  j  j  j
```

2	3
4	

already have → dup

return True

Code :

```
for( i=0 ; i< nums.length ; i++ ){
    Set<Integer> set = new HashSet<>();
    for( j=i ; j<=(i+k) ; j++ ){          → error biz of constraint:
        if( set.contain(nums[j]) ){
            return true
        }
        set.add(nums[j])
    }
}
return false
```

$0 <= K <= 10^5$

So, change cond^n :

$j <= Math.min(i+k, nums.length-1)$

Eg. $i+K = 32$

length of array = 5

$TC = O(n^2)$

$SC = O(n)$

Optimization

Eg. [1, 2, 3, 4, 2] (0, 1, 2, 3, 4) K = 2

(-1) (1, 2, 3, 4) + (2)

(2, 3, 4, 2)

] Window concept

~~Fixed size~~

Let's check, fixed size window concept fitted or not ⇒

○ [1, 2, 3, 4, 2] K = 3

You have to check 4 elements at a time

abs | j - i | <= K

(0 - 3) <= 3

But here = 3

= size = 4

set

1	2
3	

So,

⇒ [1, 2, 3, 4, 2] , K = 3

then, what we do in fixed size ⇒ i = K ⟶ then check

[1, 2, 3, 4, 2]

check (4) is in set or not

⟶ that means window ⇒ size = 4 (4 elements process at a time)

1	4
3	2

○ Now, add and remove element

[~~1~~, 2, 3, 4, 2]

⇒ Check (2) is in set or not

~~1~~	4
2	3

· Yes

· True

Code:

```
Set<Integer> set = new HashSet<>()
for( i=0 ; i< Math.min( k, nums.length ) ; i++){
    if( set.contains( nums[i] ){
        return true
    }
    set.add( nums[i] )
}
for ( i=k ; i< nums.length ; i++){
    if ( set.contains( nums[i] ){
        return true
    }
    set.add( nums[i]
    set.remove( nums[i-k] )
}
return false
```

TC = O(n)
SC = O(n)

eg, 1 2 3 1 K = 2 (0 1 2 3)

① 1 2 3 1 (0 1 2 3)

ret = [1, 2]

then, [1 2 3 1]

ret = [1, 2, 3]

remove 1

ret = [2, 3]

② 0 1 2 3 1

ret = [2, 3]

add 1

remove = 2

ret = [3, 1]

→ False

An this eg, dulpticate is repeaent but on In the range of abs|j-i| <= k

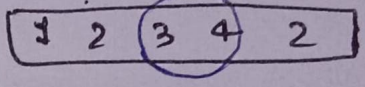
Q (643) Maximum Average Subarray I

Eg. nums = [1, 12, -5, -6, 50, 3] K = 4

(Which 4 no's average sum is Max)

Max Subarray sum concept = Max Average subarray

How? K = 2



Max sum = [3, 4]

= 7

same avg =

~~7~~ $\frac{7}{2}$ = (3.5) Max

Max sum = Max Avg

[1, 12, -5, -6, 50, 3]

first 4 elements = ~~1~~ $\frac{2}{4}$ = 0.5

$\frac{51}{4}$ = 12.75

$\frac{42}{4}$ = 10.50

Max = 12.75

Brute Force

[1, 12, -5, -6, 50, 3]

sum = 0
max = 0

sum = ~~2~~ ~~51~~ 42

max = ~~2~~ 51

→ max = $\frac{51}{4}$ = 12.75

Optimized

- nums = [1, 12, -5, -6, 50, 3] k = 4

  i i i i

  sum = ~~0~~ ~~1~~ ~~13~~ ~~8~~ 2 max = ~~0~~ 2

- [~~1~~, 12, -5, -6, 50, 3]

  i

  sum = ~~2~~ ~~52~~ 51 max = ~~2~~ 51

- [1, ~~12~~, -5, -6, 50, 3]

  i

  sum = ~~51~~ ~~54~~ 42 max = 51

Now, $\frac{max}{4} = 12.75$

Code 11 :

```
double max = 0, sum = 0

for ( i=0; i< k ; i++ ){
    sum = sum + num[i]
}

max = Math.max( max, sum)   sum

for ( int i=k ; i< num.length ; i++ ){
    // expand window
    sum = sum + num[i]
    // contract window from left
    sum = sum - num[i-k]

    max = Math.max (max, sum)
}
return max/k
```

TC = O(n)
SC = O(1)

Error for case : num = [-1] and K = 1

expected => -1.0000
o/p get => 0.0000

Why,

bcz. of max = Math.max (max, sum)

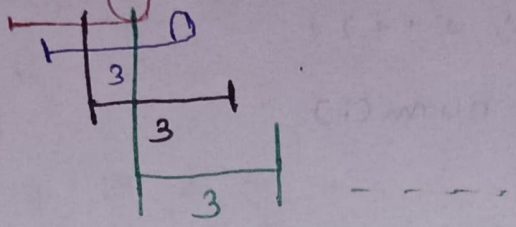
Update :

max = sum

Means, how we update max once.

(3) Longest Substring Without Repeating Characters

Eg s = " a b c a b c b b "
0 1 2 3 4 5 6 7



len = 3

Brute Force

```
max = 0
for( i=0 ; i< s.length() ; i++ ){
    let<Character> set = new HashSet<>()
    for( j=i ; j < s.length() ; j++ ){
        char c = s.charAt( j )
        if( set.contains(c) ){
            break
        }
        set.add(c)
        max = Math.max( max, j-i+1 )
    }
}
return max
```

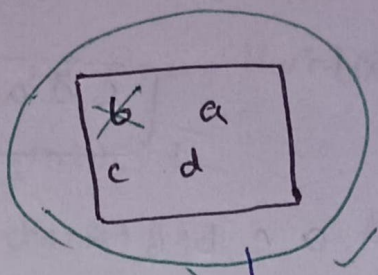
T.C = O(n²)
S.C = O(n)

Optimized

#dynamic - window

- s = " b a c d a c b b "

i j j j j j



empty

repeating char add in ret

↓

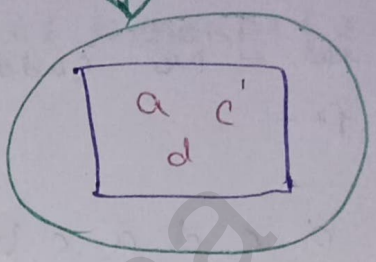
redundant

↓

sliding window

- s = " b a c d a c b b "

i j j j j

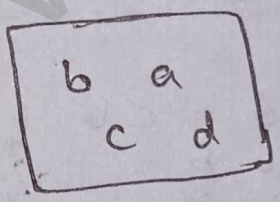


- s = b a c d a c b b

i j j j j j

len

max = ~~1~~ 2 ~~3~~ 4



- s = b a c d a c b b

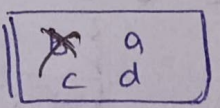
i j

same (no, again add ✓)

~~dele~~

- remove 'b'
- Check again 'a' in ret

→ Yes



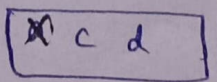
i++

- s = b a c d a c b b

0 1 2 3 4 5 6 7

i j

- remove 'a'
- check 'a' → NO



~~max~~ len = j - i = 2 → max = 4

- s = b a c d a c b b
  i j

→ c in set
break

cda

- s = b a c d a c b b
  i j

xda

→ c in set = No (add c)
~~i++~~ j++

s = b a c d a c b b
  i j

→ b in set → No (add b)
j++

dacb

len = 4

s = b a c d a c b b
  i j

b in set → Yes
i++

dacb

- s = b a c d a c b b
  i j

'remove' d'
check 'b' in set → Yes
again i++

~~d~~acb

Code :

```
max = 0
i = 0, j = 0
Set<Character> set = new HashSet<>()
while ( j < s.length() ){
    char c = s.charAt(j)
    while ( set.contain(c) ){
        set.remove ( s.charAt ( i ) )
        i++
    }
    set.add(c)
    max = Math.max ( max , j - i + 1 )
    j++
}
return max
```

T.C = O(n)

S.C = O(n)

Day-15

Sliding Window Pattern

(Rabin Karp Algo)

187. Repeated DNA Sequences leetcode (Medium)

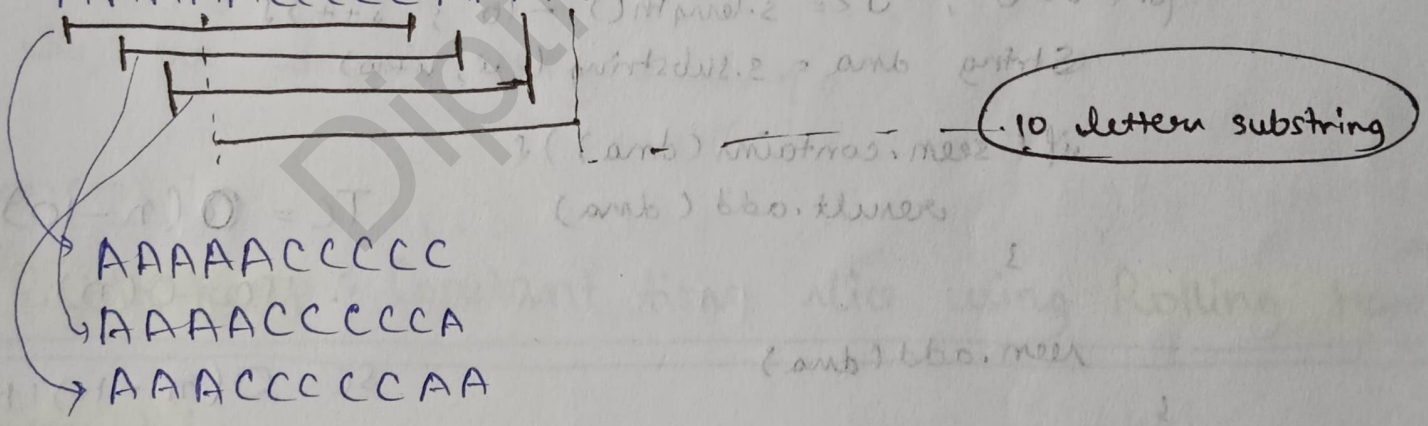
- DNA sequence composed of 'A', 'C', 'G' and 'T'

  Eg, "ACGAATTCCG"
- repeated sequences
- return all the 10-letter long sequences that occur more than once in a DNA molecule.

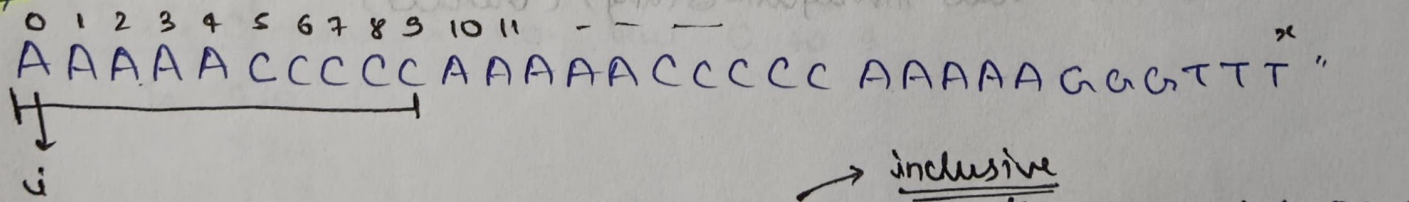
Eg, "AAAAACCCCCAAAAACCCCCAAAAAGGGTTT"
(same (repeated))

→ o/p = "AAAAACCCCC" and "CCCCCAAAAA"

Eg. "AAAAACCCCCAAAAACCCCCAAAAAGGGTTT"

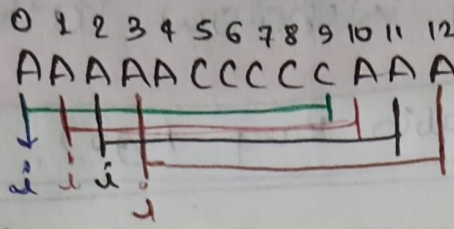


Logic:



- s.substring(i, i+10) → inclusive → substring from (index 0 to 9)
- check the string contain in set
  if it already in set, then add in result ~~set~~ set

⇒ i kaha tak jayega?



~~0 to~~ An this case : i moves 0 to 3

We have,

length = 13

size = 10 (window)

Hame chahiye i moves to '3'

∴ i = 0

$i <= (length - size)$

Code:

```
Set<String> seen = new HashSet<>();
Set<String> result = new HashSet<>();

for( i=0 ; i<= s.length() - 10 ; i++){
    String dna = s.substring( i, i+10)

    if( seen.contains(dna)){
        result.add(dna)
    }

    seen.add(dna)
}

return new ArrayList<String>(result)
```

$TC = O(n-10)(10)$

$\therefore TC = O(n)$

$SC = O(n-10)(10)$

$\therefore SC = O(n)$

Optimized

S = "AAAAACCCCCAAAAACCCCCAAAAAGGGTTT"

AAAAACCCCC → repeated calculation

AAAACCCCCA

(use sliding window as):

AAAAACCCCC + A − A

= AAAACCCCCA

★ We have string → Add / Remove → give new string

So, we don't have constant time on these operations bcz of string.

So, we convert string → ??

Concept:

Rabin-Karp : Constant time slice using Rolling Hash

Prerequisites:

Base 10 → means no. blw 0 to 9

Eg. 950
320
329 ...

4 digit → 0000 − 9999

0368
3980
9210 → Unique (each digit in a no. are unique)

→ Relate this with DNA sequence :

DNA → 10 len
which only have 4 characters

A, C, G, T

Let,

Base 4 → 0, 1, 2, 3

Assum A = 0
C = 1
G = 2
T = 3

Eg,

AACG → 0012

mean, all length of string represent in no.

As we know, Binary to Decimal

(binary no) $101_{2\,1\,0}$ = $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$ (decimal no.)

(decimal no. always unique of binary no.)

So, For Base 4

AACG → 0012

$0\ 0\ 1\ 2 \longrightarrow 0 \times 4^3 + 0 \times 4^2 + 1 \times 4^1 + 2 \times 4^0$

$4^3\ 4^2\ 4^1\ 4^0$

$= 6$

AACG = 6 (this string represent with unique value)

Eg. AACGC

we know, AACG = 6

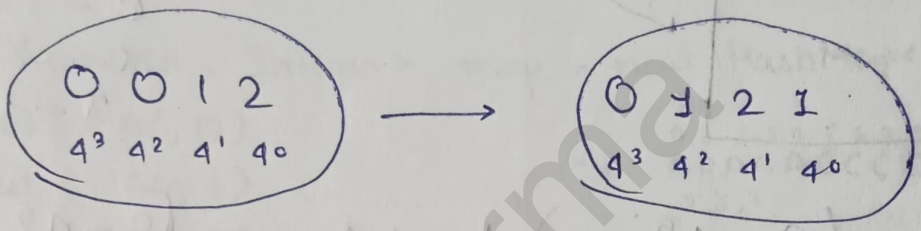
A → 0
C → 1
G → 2
T → 3

We want,

AACG + C − A = ACGC

Before : AACG (0012)

After : ACGC (0121)



Let

a	b	c	d
0	0	1	2
4^3	4^2	4^1	4^0

→

b	c	d	e
0	1	2	1
4^3	4^2	4^1	4^0

Remove

$\Rightarrow 4^3 \cdot a + 4^2 \cdot b + 4^1 \cdot c + 4^0 \cdot d$

+1 +1 +1

$4^3 \cdot b + 4^2 \cdot c + 4^1 \cdot d + 4^0 \cdot e$ add

power + 1

So, $4^3 \cdot a + 4^2 \cdot b + 4^1 \cdot c + 4^0 \cdot d = \text{Sum}$

$\Rightarrow (\text{Sum} - 4^3 \cdot a)4 + e$

$= 4^3 \cdot b + 4^2 \cdot c + 4^1 \cdot d + e$

Sum always unique

for understand binary to decimal always unique

A → 0 G → 2
C → 1 T → 3

Eg.

S = "AAAAACCCCCAAAAACCCCCAAAAAGGGTTT"

i

AAAAACCCCC = Sum

$4^9\ 4^8\ 4^7 \ldots 4^0$

add

Set

A → 0
C → 1
G → 2
T → 3

Formula: $(S - 4^9 \cdot A)\,4 + A = [S - 4^9 \cdot (0)]\,4 + 0$

Symmetry:

String → Numerical representation in constant time

Code:

```
public List<String> findRepeatDnaSequences(String s) {
    int k = 10
    if (s.length() <= k) {
        return new ArrayList<>();
    }
    Set<Integer> seen = new HashSet<>();
    Set<String> result = new HashSet<>()

    int rep = 0   //sum / representation of string

    Map<Character, Integer> map = new HashMap<>();
    map.put('A', 0)
    map.put('C', 1)
    map.put('G', 2)
    map.put('T', 3)

    for(int i = 0; i < k; i++) {
        int pow = k - i - 1
        rep = rep + (int)(Math.pow(4, pow) * map.get(s.charAt(i)))
    }

    seen.add(rep)

    for(i = k; i < s.length(); i++) {
        rep = rep - (int)(Math.pow(4, k-1) * map.get(s.charAt(i-k)))
        rep = 4 * rep
        rep = rep + map.get(s.charAt(i))

        if(seen.contain(rep)) {
            result.add(s.substring(i-k+1, i+1))
        }
        seen.add(rep)
    }
    return new ArrayList<String>(result);
}
```

TC = O(n)
SC = O(n)

Eg. 0 1 2 3 4 5 6 7 8 9
A A A A A C C C C C
$4^9\ 4^8\ 4^7 \ldots 4^0$

rep = $4^9 \times 0 + 4^8 * 0 + \ldots$

rep = $4^9 \times 0 + \ldots + 4^0 \times 1$

Day-16

Sliding Window Pattern

(2 Impt Interview Ques)

3. Longest Substring Without Repeating Characters — HARD

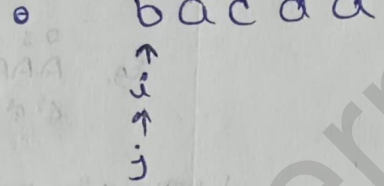
904. Fruit Into Baskets — Medium-Hard

Q. (3) Longest Substring Without Repeating Characters

#More Optimized

in case of jumps

Previous logic :



max = 0

- b a c d a
  i j i i i

> b not in set → add → b
then j++

> again a not in set → b a
j++

> b a c → > b a c d

> a in set
→ remove 'b'
→ i++

~~b~~ ~~a~~c d

- b a c d a
  i j

> a in set
→ remove 'a'
i++

~~a~~ c d

0 1 2 3 4

- b a c d a
  i j

j not in set

c d a

max = 4 − 2 + 1
= 3

j++ stop

return 3

Intuition:

b e f a c d a
i j

What we do?

> remove one-one element from set and check.

See:

b e f a c d a
i j

a ko remove karne se 'a' repeat nhi kar raha hai set mein.

Toh, kya ham

a ko add karne se 'a' repeat kar raha hai " toh ham direct 'a' ke aage hii jump kar lete hai" → isse hame extra remove 'bef' karne ki need nhi hoti !!!

→ Use Map !!

Code:

```
int i=0, j=0, max = 0
Map<Character, Integer> map = new HashMap<>();
while ( j < s.length() ) {
    char c = s.charAt(j)
    if( map.containsKey(c) ){
        i = map.get(c) + 1
    }
    map.put(c, j)
    max = Math.max( max, j-i+1)
    j=j+1
}
return max.
```

Eg. 0 1 2 3 4 5 6
b e f a c d a
i
j

- j = b
  > b → not in map
  b:0
  > max = 1
  j++

- b e f a c d a
  i j
  > j = e → not in map
  b, e
  max = 2
  j++

- b e f a c d a
  i j
  > j = f → not in map
  b e f
  max = 3
  j++

- b e f a c d a
  i j
  > j = a → not
  b e f a
  max = 4
  j++

- b e f a c d a
  i j
  > j = c → x
  b e f a c
  max = 5
  j++

- b e f a c d a
  i j
  > j = d ✓
  max = 6
  j++
  b e f a c d

- b e f a c d a
  i j
  > j = a ✓
  i = 3 + 1 = 4

b:0 d:5
e:1
f:2
a:3
c:4

0 1 2 3 4 5 6
- b e f a c d a
  i j

max = (3, 6)
j++ atop

return max = 6

befacda

but that code not run for this case :

s = " a b b a "
0 1 2 3
i, j

- s = "a b b a"
  i, j
  j = a → x
  a : 0
  max = 1
  j++

- s = "a b b a"
  i, j
  a : 0
  b : 2
  max = 2
  j++

- s = a b b a
  i j
  j = b → x
  a : 0
  b : 1
  max = 2
  j++

- s = "a b b a"
  i j
  a : 0
  b : 2
  j = a ✓
  i = 0 + 1 = 1
  "i again go on 1 index"

  a b | b a
  i j
  and b is not exist in the window

- s = a b b a
  i j
  j = b ✓
  So, j = 1 + 1 = 2
  max = 2

Toh ham :

```
if (map.containsKey(c)) {
    if (map.get(c) >= i) {
        i = map.get(c) + 1
    }
}
```

- s = a b b a
  j i
  → j = b ✓
  a : 0
  b : 1
  i = 1 + 1 = 2
  again

(i ke value choti hai agar current i se toh → ignore)

Updated code :

TC = O(n)
SC = O(n)

```
int i = 0, j = 0, max = 0
Map<Character, Integer> map = new HashMap<>();
while ( j < s.length() ) {
    char c = s.charAt(j)
    if (map.containsKey(c)) {
        if (map.get(c) >= i) {
            i = map.get(c) + 1
        }
    }
    map.put(c, j)
    max = Math.max(max, j - i + 1)
    j = j + 1
}
return max
```

Q. (904) Fruit Into Baskets

- Trees are represented by array fruits[] where fruits[i] = type of fruit the ith tree produces.
- We want to collect as much fruits as possible with strict rules :
  - Only 2 baskets, each basket can hold a single type of fruit. No limit on amount.
  - We must pick exactly one fruits from every tree
- Once you reach a tree with fruit that cannot fit in your basket, Stop.

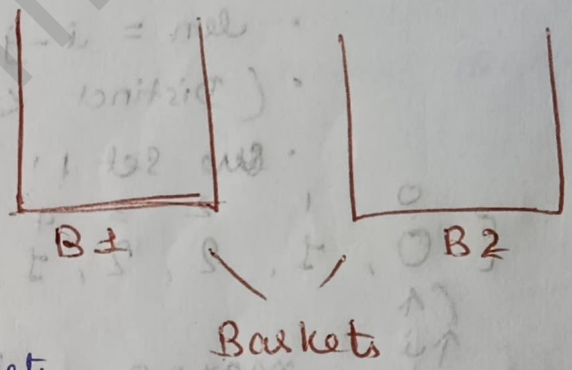
- return, the max no. of fruits you pick.

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

Assume: 0 → type of tree of △ fruit
1 → type of tree of □ "
2 → " " " " ◎ "
3 → " " " " ⬭ "

[0, 1, 2, 2, 1, 1, 3, 2, 2, 1]

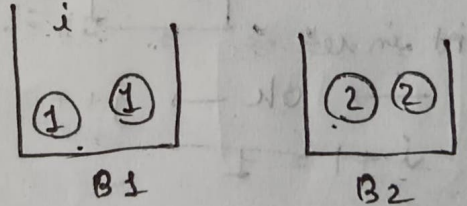
↳ [△, □, ◎, ◎, □, □, ⬭, ◎, ◎, □]



> B1 ~~and~~ basket mein ek hi type ka fruits unlimited dal sakte hai
> same for B2 basket
> but, jese hi 3rd type of fruit milega, toh hum B1 & B2 basket mein nahi dal skte hai → Stop and ~~calcu~~ count the no. of fruits in B1 and B2.

Eg, [0, 1, 2, 2, 1, 1, 3, 2, 2, 1]
0 1 2 3 4 5 6 7 8 9

Let i on index 2

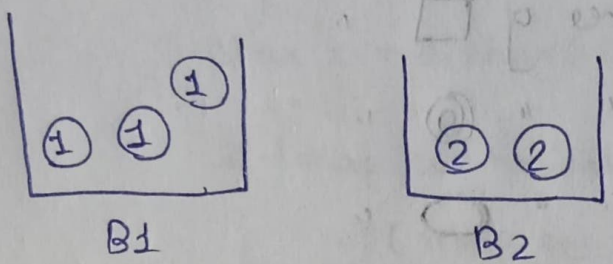


on index 6
i get 3 fruits
→ Stop bcz, B1 have (1)
B2 have (2)
→ count = 4 fruits

Eg. [0, 1, 2, 2, 1, 1, 3, 2, 2, 1] o/p = 5

How? o/p = 5

stop



Count = 5

Len ? → i = 5, j = 1 → (i - j + 1) = 5

Initiation:

- move until, we have 2 type of distinct fruits
- len = i - j + 1
- (Distinct <= 2) → move
- Store set !!

Eg, [0, 1, 2, 2, 1, 1, 3, 2, 2, 1]

max = 0

set

- [0, 1, 2, 2, 1, 1, 3, 2, 2, 1]

> 0 → put in set

> ask, element in set = 1 → ok → j++

> size = j - i + 1 = 1

> max = 1

- [0, 1, 2, 2, 1, 1, 3, 2, 2, 1]
  ↑ ↑
  i j

  > 1 → put in set [0, 1]
  > ask, size of set = 2 → ok → j++
  > size = 2
  > max = 2

- [0, 1, 2, 2, 1, 1, 3, 2, 2, 1]
  ↑ ↑
  i j

  > 2 → set [0, 1, 2]
  > size of set = 3 X → j++
  and j = i and reset the set

- [0, 1, 2, 2, 1, 1, 3, 2, 2, 1]
  ↑
  i
  ↑
  j

  > 1 → set [1]
  > size = 1 ✓ → j++

- [0, 1, 2, 2, 1, 1, 3, 2, 2, 1]
  ↑ ↑ ↑
  i j

  > 2 → set [1, 2]
  > size = 2 ✓ → j++

- [0, 1, 2, 2, 1, 1, 3, 2, 2, 1]
  ↑ ↑ ↑ ↑ ↑
  i j j j

  → 2 → set already ✓
  → size = 2 → j++
  max = 2 3 4 5 (j - i + 1)

- [0, 1, 2, 2, 1, 1, 3, 2, 2, 1]
  ↑ ↑
  i j

  → 3 → set [1, 2, 3]
  → size of set = 3 → X → i++ max = 5 //

Code :

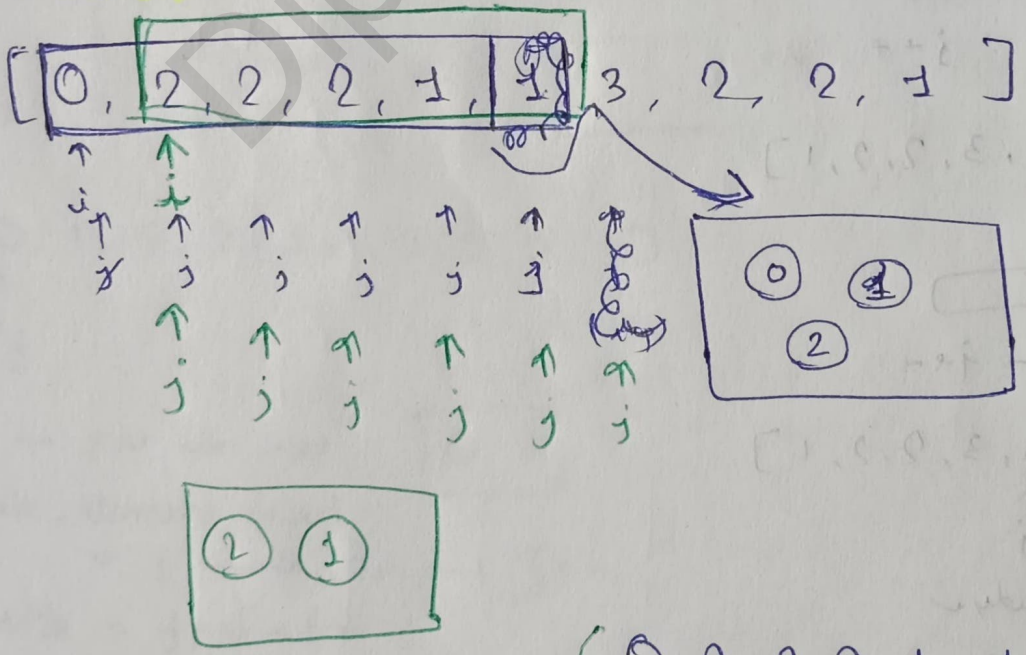
```
max = 0
for( i=0 ; i< fruits.length ; i++){
    Set<Integer> set = new HashSet<>()
    for ( j=i ; j< fruits.length ; j++){
        set.add ( fruits[j] )
        if ( set.size() > 2 ){
            break
        }
        max = Math.max ( max , j-i+1 )
    }
}
return max
```

TC = $O(n^2)$

SC = $O(3)$

Here, SC = constant
biz, hum set ki value 2 se jada nhi kar rahe.

Optimized

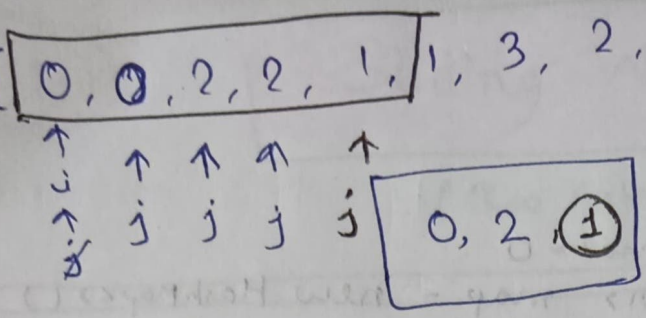


repeating !!

0 2 2 2 1

2 2 2 1 ...

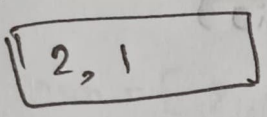
Eg. [0, 0, 2, 2, 1, 1, 3, 2, 2, 1]

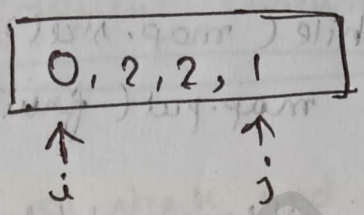


max = ~~0~~ ~~1~~ 2 ~~3~~ 4

size > 2
↳ i++
↳ remove 0 from set

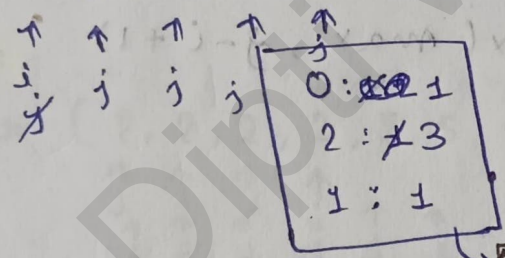
But, this window is not represent by set





* That's why we don't use Set
→ Use Map (that store count)

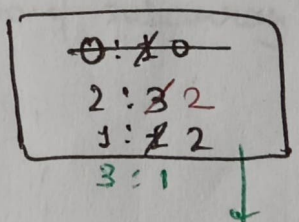
Eg. [0, 2, 2, 2, 1, 1, 3, 2, 2, 1]



max = ~~0~~ ~~1~~ 2 ~~3~~ 4

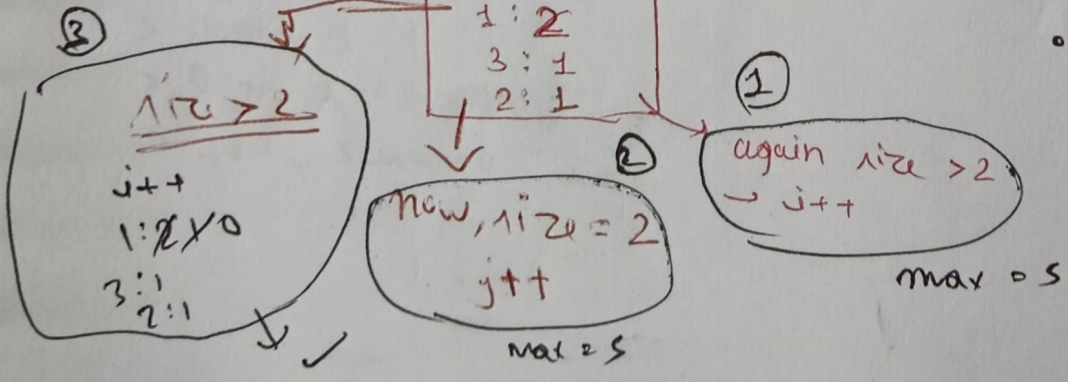
size > 2
- i++
- remove from set

[0, 2, 2, 2, 1, 1, 3, 2, 2, 1]



max = ~~4~~ 5

size > 2
- i++



Code :

```
int i=0, j=0, max=0
Map<Integer, Integer> map = new HashMap<>()
while ( j < fruits.length ){

    map.put ( fruits[j] , map.getOrDefault ( fruits[j] , 0) +1)

    while ( map.size() > 2 ){
        map.put( fruits[i] , map.get(fruits[i]) -1)

        if(map.get(fruits[i]) == 0){
            map.remove( fruits[i] )
        }

        i++
    }

    max = Math.max( max , j-i+1)
    j++
}

return max
```

TC = O(n)

SC = O(1)

Day - 17

Sliding Window Pattern

(Too Interview Ques)

1343. No. of sub-arrays of size k and average greater than or equal to threshold (Medium)

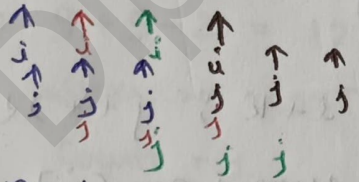
1838. Frequency of the most frequent element (medium - hard)

(1343). No of sub-arrays of size k and average greater than or equal to threshold

Eg, arr = [2, 2, 2, 2, 5, 5, 5, 8] k = 3 threshold = 4

Brute Force :

arr = [2, 2, 2, 2, 5, 5, 5, 8]



> sum = 6
> 6/3 = 2 >= 4 X
> i++, sum = 0, j = j

> sum = 6
> 6/3 = 2 >= 4 X
> j++, sum = 0, i = j

> sum = 9
> 9/3 = 3 >= 4 X
> i++, sum = 0, i = j

> sum = 2+5+5 = 12
> 12/3 = 4 >= 4 ✓
> No. of subarray = 1

etc.

Let a case arr:

arr = [2, 2, 2, 3, 5, 5, 5, 8] K=3 threshold = 4

sum = 10 ⟶ $\frac{10}{3}$ = 3.3333 >= 4 ?

Let $\frac{x}{y}$ >= t

⟶ x >= (t x y) [No need to use float data type]

~~Ans~~

10 >= (3 x 4) ✗

Code:

```
int maxT = K * threshold
int count = 0

for ( i=0 ; i < arr.length - K ; i++ ) {
    int sum = 0

    for ( j=i ; j < i+k ; j++ ){
        sum = sum + arr[j]
    }

    if ( sum >= maxT ) {
        count = count + 1
    }
}

return count
```

TC = O(n²)
SC = O(1)

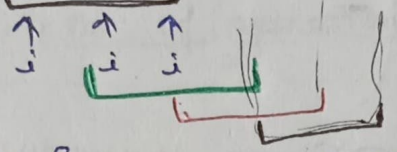
Optimization :

arr = [2, 2, 2, 2, 5, 5, 5, 8]

2 * (2 + 2)
2 (2) + 2
2 + 2 + 5

] repetation calculation !!!

eg. arr = [0 1 2 3 4 5 6 7
2, 2, 2, 2, 5, 5, 5, 8] K = 3 t = 4



→ sum = 6 maxT = 12

sum >= maxT ✗

→ sum = 6 + 2 − 2 = 6 maxT = 12

sum >= maxT ✗

→ sum = 9 maxT = 12

sum >= maxT ✗

→ sum = 9 + 5 − 2 = 12 maxT = 12

sum >= maxT ✓

count = 1

Code:

```
int sum = 0
int maxT = k * threshold

for( i=0 ; i<k ; i++ ){
    sum = sum + arr[i]
}
int count = 0
if( sum >= maxT ){
    count++
}

for( i=k ; i< arr.length ; i++ ){
    sum = sum + arr[i]
    sum = sum - arr[i-k]
    if( sum >= maxT ){
        count++
    }
}
return count
```

TC = O(n)
SC = O(1)

Q. (1838) Frequency of the Most Frequent Element

Eg nums = [1, 2, 4] (k = 5) → operation

(0, 1, 2)

on index 0 → do 3 operation, it becomes 4 — Use atmost operations

on index 1 → do 2 operation, it becomes 4

Let

∴ nums = [4, 4, 4] ⟶ o/p = 3

Approach-1

Eg, nums = [1, 10, 19, 17, 16] K = 5

⟶ Let we take (17)

- (17) se chota (16)
- ab (16) +1 (17) → 1 operation = (17) ✓

⟶ Let we take (19)

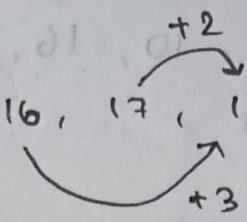
- (19) se chota (17)
- ab (17) + 2 ⟶ 2 operation = (19) ✓
- (19) se chota (16)
- ab (16) + 3 ⟶ 3 operation = (19) ✓
- 0 operation left
- we get 19 → frequency = 3

for this, we want sorting

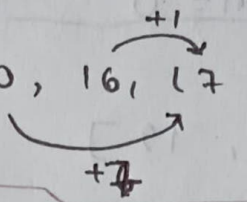
Let nums = [1, 10, 16, 17, 19]

nums = [1, 10, 16, 17, 19] k = 5

Let take subarray = [16, 17, 19] ⇒ Operation = 5



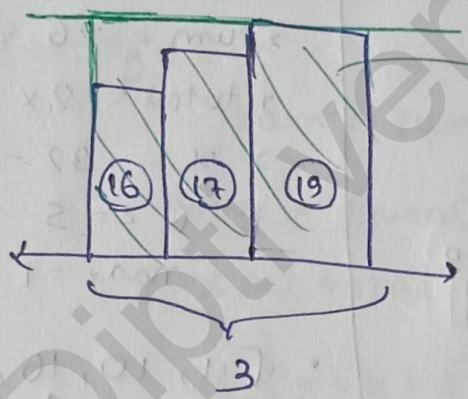
Let take subarray = [10, 16, 17] ⇒ Operation = 8 ✗



Means, if we analysis all subarray after sorting, and check the no. of operation $=< k$, then count frequency.

subarray → x ?? (no. of operations)

Let subarray = [16, 17, 19]



19 × 3 = 57

but actually = 16 + 17 + 19 = 53

and, 57 − 53 = 4

we have

x = 4 and k = 5

4 =< 5 ✓

- sorted
- Take subarray and calculate sum
- take last element of subarray (let e) × size
- Sum − (e × siz) = x
- Check (x =< k) ✓
  - → valid subarray
  - → calculate frequency

size calculate ?

0	1	2	3
1	16	18	19
	↑		↑
	i		j

3

j − i + 1 = size

Eg, nums = [1, 10, 19, 17, 16] k = 5

0 1 2 3 4

Sort → [1, 10, 16, 17, 19]

i j

max = 0

sum = 0

total size = j - i + 1

total = total size × last element

x? → total - sum

- [1, 10, 16, 17, 19]
  i j

> sum = 1 , total size = 1
> total = 1 - 1 = 0 (x)
> 0 <= 5 ✓ j++
> max = 1

- [1, 10, 16, 17, 19]
  i j

> sum = 11 , total size = 2
> total = 2 × 10 = 20
> x = 20 - 11 = 9
> 9 <= 5 X → i++
max = 1

- [1, 10, 16, 17, 19]
  i j

> sum = 10 total size = 1
> total = 1 × 10 = 10
> x = 10 - 10 = 0
> 0 <= 5 ✓ j++
max = 1

- [1, 10, 16, 17, 19]
  i j

> sum = 26 T_size = 2
> total = 2 × 16 = 32
> x = 32 - 26 = 6
> 6 <= 5 X i++
max = 1

- [1, 10, 16, 17, 19]
  i j

> sum = 16 , T_size = 1
> total = 1 × 16 = 16
> x = 0
> 0 <= 5 ✓ max = 1

- [1, 10, 16, 17, 19]
  i j

> sum = 33 , T_size = 2
> total = 2 × 17 = 34
> x = 34 - 33 = 1
> 1 <= 5 ✓ max = 2

- [1, 10, 16, 17, 19]
  i j

> sum = 52, Tsiz = 3
> total = 3 × 19 = 54
> x = 54 − 52 = 2
> 2 <= 5 ✓ max 3 j++
Stop.

return = 3

Code:

```
max = 0
Arrays.sort(nums)
for(i=0; i<nums.length; i++){
    sum=0
    for(j=i; j<nums.length; j++){
        sum = sum + nums[j]
    total = nums[j] * (j-i+1)
    x = total - sum
    if(x>k){
        break
    }
    max = Math.max(max, j-i+1)
}
return max
```

TC = O(n²)
SC = O(sorting space complexity)
O(n log n)

Optimized

nums = [1, 10, 1, 19, 17, 16] K = 5

sort → 1, 1, 10, 16, 17, 19

0 1 2 3 4 5

```
if(nums[j] x (j-i+1) - sum > k){
    break
}
```

sum = 0 ~~1~~ ~~2~~ 12

max = ~~1~~ 2

Check: ~~total~~ = (10 x 3) - 12 > 5 ✓

break i++

window = [1, 1, 10]

~~Since~~

and, in 2nd iteration:

1 [1 10] 16 17 19

i
↑
j

Simply: sum -1 =

(No need to again calculation)

Dynamic size window

Eg, nums = [1, 10, 1] K = 5

sort → [1 (0), 1 (1), 10 (2)]

- sum = 1
  total = 1
  x = 0
  0 > K ✗
  j++ max = 1
- sum = 2
  total = 2 × 1 = 2
  x = 0
  0 > k ✗
  j++ max = 2
- sum = 12
  total = 3 × 10 = 30
  x = 30 - 12 = 18
  18 > 5 ✓ → break
  j++ max = 2
- Sum = 12 - 1 = 11
  total = 2 × 10 = 20
  x = 20 - 11 = 9
  9 > 5 → break
  i++ max = 2
- sum = 11 - 1 = 10
  total = 1 × 10 = 10
  x = 0
  0 > 5 → ✗
  j++ stop. max = 2

Code:

```
int max = 0
Array.sort(nums)
long sum = 0
int i = 0, j = 0

while ( j < nums.length) {
    sum = sum + nums[j]
    (long)
    while ( nums[j] * (j-i+1) - sum > k){
        sum = sum - nums[i]
        i++
    }
    max = Math.max( max, j-i+1)
    j++
}
return max
```

TC = O(n log n + n)
SC = O(n log n)

This code little bit wrong bcz of constraint:

$1 <= nums.length <= 10^5$

$1 <= nums[i] <= 10^5$

nums.length = 100000
element = 100000
sum = 10000000000 (10^{10})

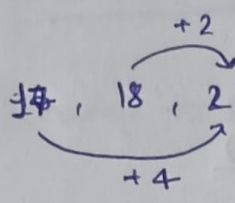
But max int value = 2147483647
→ which is smaller than 10^{10}

So, change the data type.

Follow up Question

freq of the most frequent element

1, 1, 10, 17, 18, 20, 29 K = 6

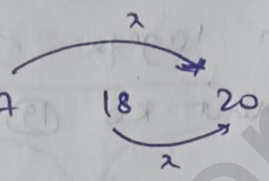


→ 20, 20, 20

freq = (3)

But, here we take assumption that last element of subarray is present in the main array (eg. 20)

Let take (21)

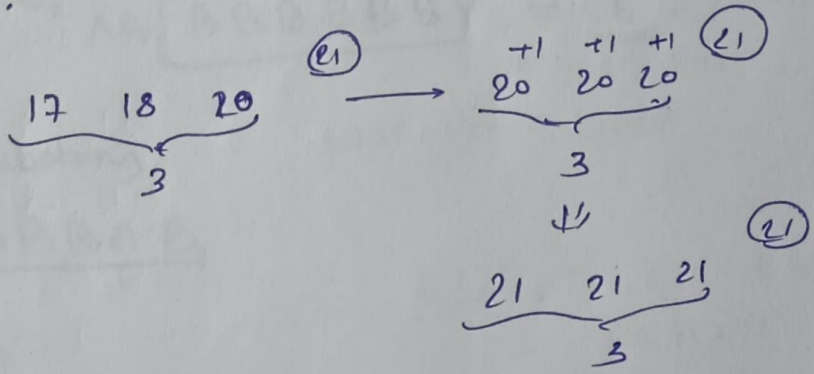


x operation = 5

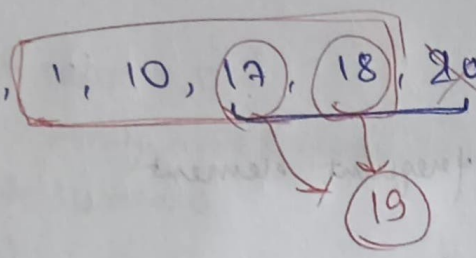
take x+1 which is not exists (eg. 21)

→ Hamne tin element ko 20 kiya tha with 5 operation

→ Toh hame only 3 more operation se ham 21 bana skte hai.



Eg. [1, 1, 10, 17, 18, 20, 29] K = 5



freq = 3

20 → x = 3

19 → x - 2 = 2

- Agar hame 20 ko banane mein x operation lag raha hai.
- Toh ham 20 se chota let 19 ko banaye, toh hame x operation se bhi kam lagega
- Toh ham aur bhi element ko 19 bana skte hai.

But, But, But ------

- Agar 19 se already chota element array mein exist karta hai, toh ham simply bol skte hai ki hame hamara frequent element apne array mein hi milega...

Day-18

Sliding Window Pattern

Q. (424) Longest Repeating Character Replacement (Medium)

eg. s = "AABDBBAB" (K = 2) Operation

Take any character, and change it to diff upper case character

4, D and A change to B then,

AABBBBBB = (6)

Brute Force :

o AABDBBAB

D, A → B

substring → (B = 4) max char in substring of BDBBAB

length = 6

6 - 4 = 2 and 2 = K ✓(valid)

then, (2) char ko convert into B

So, AA BBBBBB = 6

o Let another substring:

AABDBBAB

length = 7

max value char (B) = 4

7 - 4 = 3

but 3 > k (invalid substring)

Eg. S = "AABDBBAB" K = 2

0 1 2 3 4 5 6 7

max = 0

- A : 1
  maxC = 1
  and size = j - i + 1 = 1
  maxC - size = 1 - 1 = 0
  No need to change
  j++
- A : 2
  maxC = 2
  size = j - i + 1 = 2
  2 - 2 = 0
  j++
- {A : 2, B : 1
  maxC = 2
  size = 3
  3 - 2 = 1
  mean, 1 operation we can do
  and 1 < k
  valid ⟶ max = 3
  j++
- {A : 2, B : 1, D : 1
  maxC = 2
  size = 4
  4 - 2 = 2
  2 = k (2) operation
  valid ⟶ max = 4 (j - i + 1)
  j++
- {A : 2, B : 2, D : 1
  maxC = 2
  size = 5
  no. of operation 5 - 2 = 3 > k ⟶ invalid
  max = 4
  i++ and j = i

"0 1 2 3 4 5 6 7"
A A B D B B A B
i j

- A = 1
  maxC = 1
  size = 1
  1 - 1 = 0 ✓ j++
- {A = 1, B : 1
  maxC = 1, size = 2
  2 - 1 = 1 < k ✓
- {A : 1, B : 1, D : 1
  maxC = 1, size = 3
  3 - 1 = 2 = k ✓
- {A : 1, B : 2, D : 1
  maxC = 2, size = 4
  4 - 2 = 2 = k ✓
- {A : 1, B : 3, D : 1
  maxC = 3, size = 5
  5 - 3 = 2 = k ✓ max = 5
- {A : 2, B : 3, D : 1
  maxC = 3, size = 6
  6 - 3 = 3 > k ⟶ X

max = 5

Code:

```
max = 0
for ( i=0; i < s.length() ; i++ ) {
    maxC = 0
    int[] count = new int[26]
    for ( j = i ; j < s.length() ; j++ ) {
        char c = s.charAt(j)
        count[c - 'A'] = count[c - 'A'] + 1
        maxC = Math.max ( maxC, count[c - 'A'] )
        ops = (j - i + 1) - maxC
        if ( ops > k ) {
            break
        }
        max = Math.max ( max, j - i + 1 )
    }
}
return max
```

convert
A → 0 B → 1 C → 2 D → 3 ...

T.C = O(n²)
S.C = O(1)

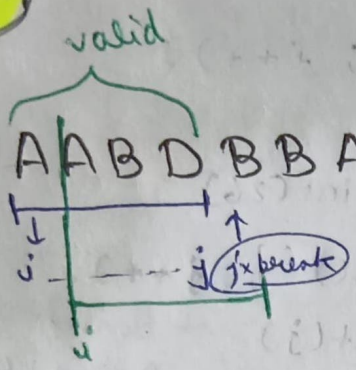
Eg, A B C B , k = 1

i, j, max = 0, maxC = 0

- A : 1 maxC = 1
  ops = 1 - 1 = 0
  0 > 1 ✗
  max = 1
- {A : 1, B : 1} maxC = 1
  ops = 2 - 1 = 1
  1 > 1 ✗ break ✗
  max = 1
- {A : 1, B : 1, C : 1} maxC = 1
  ops = 3 - 1 = 2
  2 > 1 ✓ break
  i++
- B : 1 maxC = 1
  ops = 0
  No break, Max = 1
- {B : 1, C : 1} maxC = 1
  ops = 1
- {B : 2, C : 1, B : 3} maxC = 2
  ops = 1
  2 ≤ 1 valid max = 2

Max = 2

Optimized



S = " A A B D B B A B " K = 2

in brute force →

we get,

max = 2

{ A:2, B:1, D:1 } 3

max = 2

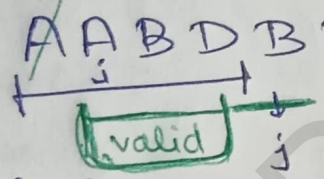
A A B D B B A B

recalculate

~~aa~~ → A A B D ⇒ valid

if ~~A~~ A B D (remove) → A B D valid ✓

Variable - size sliding window



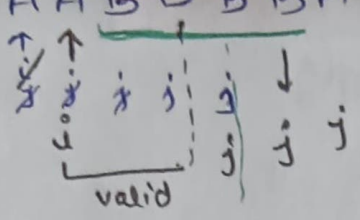
max = 2

{ A: ~~2~~ 1, B: ~~2~~ 1, D: 1 } 3

~~max 2~~

op = ~~4~~ - 2 = ~~2~~ = k → valid

Eg, s = "AABDBBAB" , (K = 2)



max = 0
maxC = 0

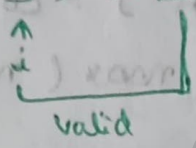
- {A: 1
  maxC = 1
  ops = 0 > k ✗
  max = 1
- {A: 2
  maxC = 2
  ops = 2-2 = 0 > k ✗
  max = 2
- {A: 2, B: 1
  maxC = 2
  ops = 3-2 = 1 > k ✗
  max = 2
- {A: 2, B: 1, D: 1
  maxC = 2
  ops = 4-2 = 2 > k ✗
  max = 2
- {A: 2, B: 2, D: 1
  maxC = 2
  ops = 5-2 = 3 > k ✓
  break, i++

∴ AABD valid

- and, ABD after removing 'A' is also valid !!
  > remove from freq map
  {A: 1, B: 2, D: 1
  maxC = 2
  ops = 4-2 = 2 > 2 ✗
- {A: 1, B: 3, D: 1
  maxC = 3
  ops = 5-3 = 2 > 2 ✗
- {A: 2, B: 3, D: 1
  maxC = 3
  ops = 6-3 = 3 > K ✓
  break, i++

ABDBB valid

- AABDBBAB



- A: 1, B: 3, D: 1
  maxC = 3
  ops = 5-3 = 2 > k ✗
- A: 1, B: 4, D: 1
  maxC = 4
  ops = 6-4 = 2 > k ✗

BDBBAB valid

How we terminate ?

s = "AABDBBAB"

j j j

j j j j j j j j j

ab agar hum i ko ++ kiye toh bhi. kya valid substring with max value milega ⟶ Never.
bcz, length toh chota hote jaa raha hai.

Isliye, ~~termination~~ depend on j

jab j end mein jayega → stop.

Code:

```
max = 0
i = 0, j = 0
maxC = 0
int count[] = new int[26]

while ( j < s.length() ){
  char c = s.charAt(j)
  count[c - 'A'] = count[c - 'A'] + 1
  maxC = Math.max( maxC, count[c - 'A'] )

  while ( (j-i+1) - maxC > k ) {
      char d = s.charAt(i)
      count[d - 'A'] = count[d - 'A'] - 1
      i++
  }
  for ( d=0; d<26; d++){
      maxC = Math.max( maxC, count[d] )
  }
  max = Math.max( max, j-i+1 )
  j++
}
return max
```

→ when invalid remove i element

update the code:

Eg, AAAAAADDD , k = 1

i, j

AAAAAADDD

valid

i

j

here, maxC = 6

invalid (break)

8 - 6 = 2 > k

then, remove 'A' from front.

→ So, freq of A now, = 5

But, we don't update maxC